IOT APPLICATION DEVELOPMENT & CASE STUDY

- Santosh Katti
Asst. Prof., Dept. of MCA
PES University
Bangalore, INDIA

OUTLINE – IOT APPLICATION DEVELOPMENT

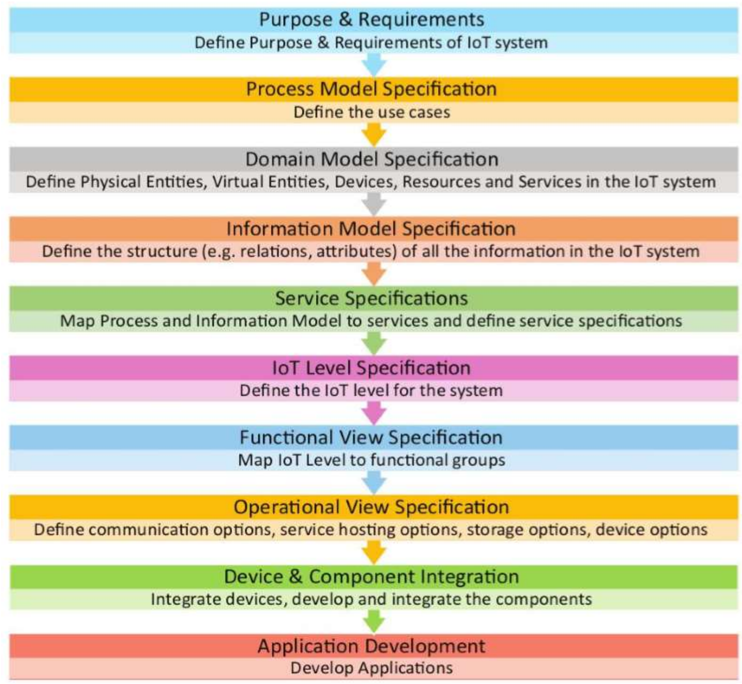
IoT Design Methodology that includes:

- Purpose & Requirements Specification
- Process Specification
- Domain Model Specification
- Information Model Specification
- Service Specifications
- IoT Level Specification
- Functional View Specification
- Operational View Specification
- Device & Component Integration
- Application Development

Mr. Santosh Katti, Asst. Prof.

PES UNIVERSITY

IOT DESIGN METHODOLOGY: STEPS



Mr. Santosh Katti, Asst. Prof.

PES UNIVERSITY

STEP 1: PURPOSE & REQUIREMENTS SPECIFICATION

The first step in IoT system design methodology is to define the purpose and requirements of the system. In this step, the system purpose, behaviour and requirements (such as data collection requirements, data analysis requirements, system management requirements, data privacy and security requirements, user interface requirements, ...) are captured.

Mr. Santosh Katti, Asst. Prof.
PES UNIVERSITY

STEP 2: PROCESS SPECIFICATION

The second step in the IoT design methodology is to define the process specification. In this step, the use cases of the IoT system are formally described based on and derived from the purpose and requirement specifications.

Mr. Santosh Katti, Asst. Prof.
PES UNIVERSITY

STEP 3: DOMAIN MODEL SPECIFICATION

The third step in the IoT design methodology is to define the Domain Model. The domain model describes the main concepts, entities and objects in the domain of IoT system to be designed. Domain model defines the attributes of the objects and relationships between objects. Domain model provides an abstract representation of the concepts, objects and entities in the IoT domain, independent of any specific technology or platform. With the domain model, the IoT system designers can get an understanding of the IoT domain for which the system is to be designed.

Mr. Santosh Katti, Asst. Prof.
PES UNIVERSITY

STEP 4: INFORMATION MODEL SPECIFICATION

The fourth step in the IoT design methodology is to define the Information Model. Information Model defines the structure of all the information in the IoT system, for example, attributes of Virtual Entities, relations, etc. Information model does not describe the specifics of how the information is represented or stored. To define the information model, we first list the Virtual Entities defined in the Domain Model. Information model adds more details to the Virtual Entities by defining their attributes and relations.

Mr. Santosh Katti, Asst. Prof.
PES UNIVERSITY

STEP 5: SERVICE SPECIFICATIONS

The fifth step in the IoT design methodology is to define the service specifications. Service specifications define the services in the IoT system, service types, service inputs/output, service endpoints, service schedules, service preconditions and service effects.

Mr. Santosh Katti, Asst. Prof.
PES UNIVERSITY

STEP 6: IOT LEVEL SPECIFICATION

The sixth step in the IoT design methodology is to define the IoT level for the system.

The various levels in IoT has already been discussed in previous classes.

Mr. Santosh Katti, Asst. Prof.
PES UNIVERSITY

STEP 7: FUNCTIONAL VIEW SPECIFICATION

The seventh step in the IoT design methodology is to define the Functional View. The Functional View (FV) defines the functions of the IoT systems grouped into various Functional Groups (FGs). Each Functional Group either provides functionalities for interacting with instances of concepts defined in the Domain Model or provides information related to these concepts.

Mr. Santosh Katti, Asst. Prof.
PES UNIVERSITY

STEP 8: OPERATIONAL VIEW SPECIFICATION

The eighth step in the IoT design methodology is to define the Operational View Specifications. In this step, various options pertaining to the IoT system deployment and operation are defined, such as, service hosting options, storage options, device options, application hosting options, etc

Mr. Santosh Katti, Asst. Prof.
PES UNIVERSITY

STEP 9: DEVICE & COMPONENT INTEGRATION

The ninth step in the IoT design methodology is the integration of the devices and components.

Mr. Santosh Katti, Asst. Prof.
PES UNIVERSITY

STEP 10: APPLICATION DEVELOPMENT

The final step in the IoT design methodology is to develop the IoT application.

HOME AUTOMATION CASE STUDY

Mr. Santosh Katti, Asst. Prof.

PES UNIVERSITY

STEP:1 - PURPOSE & REQUIREMENTS

Applying this to our example of a smart home automation system, the purpose and requirements for the system may be described as follows:

• Purpose : A home automation system that allows controlling of the lights in a home remotely using a web application.

• Behavior : The home automation system should have auto and manual modes. In auto mode, the system measures the light level in the room and switches on the light when it gets dark. In manual mode, the system provides the option of manually and remotely switching on/off the light.

• System Management Requirement : The system should provide remote monitoring and control functions.

• Data Analysis Requirement : The system should perform local analysis of the data.

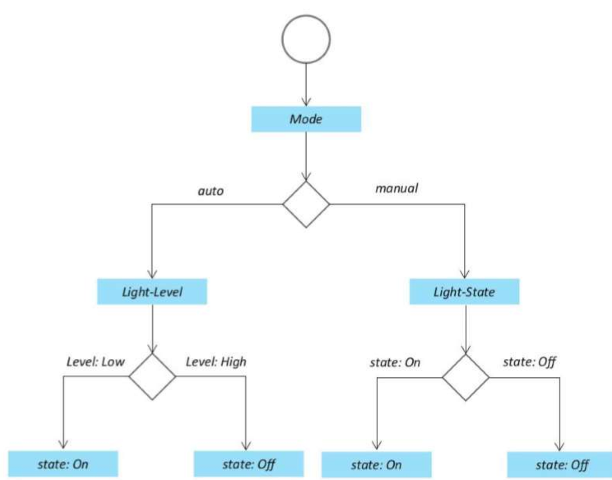
• Application Deployment Requirement : The application should be deployed locally on the device, but should be accessible remotely.

• Security Requirement : The system should have basic user authentication capability.

Mr. Santosh Katti, Asst. Prof.

PES UNIVERSITY

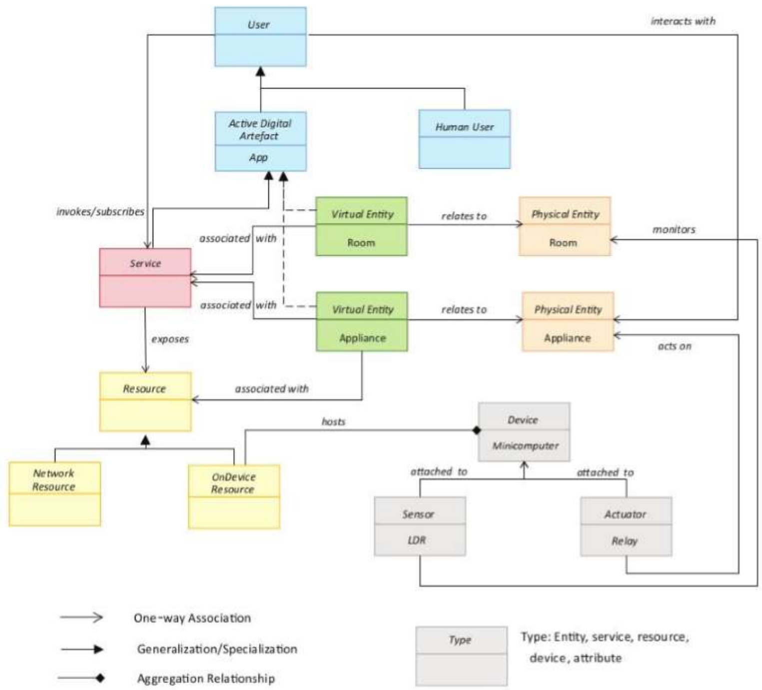
STEP:2 - PROCESS SPECIFICATION



Mr. Santosh Katti, Asst. Prof.

PES UNIVERSITY

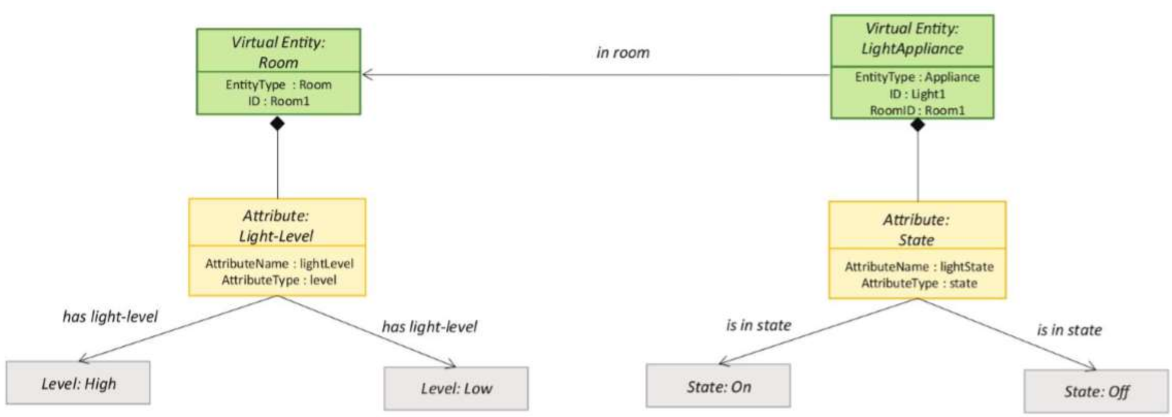
STEP 3: DOMAIN MODEL SPECIFICATION



Mr. Santosh Katti, Asst. Prof.

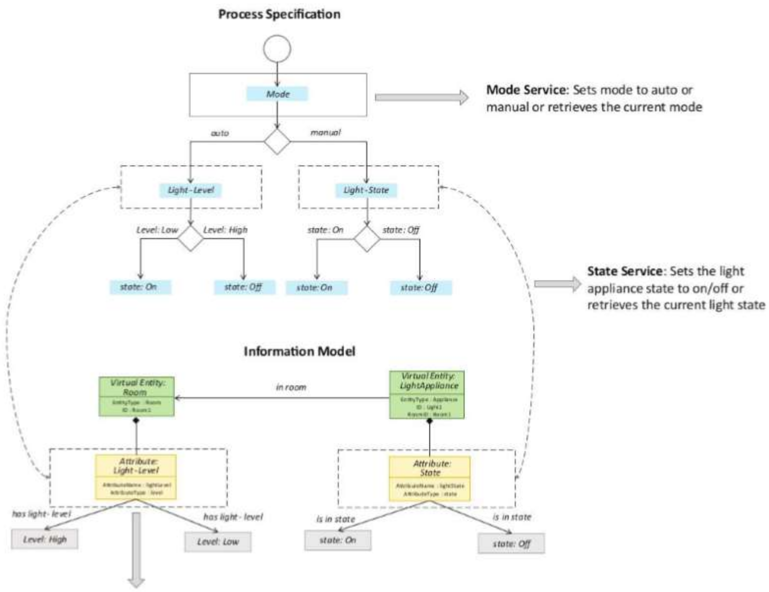
PES UNIVERSITY

STEP 4: INFORMATION MODEL SPECIFICATION



Mr. Santosh Katti, Asst. Prof.
PES UNIVERSITY

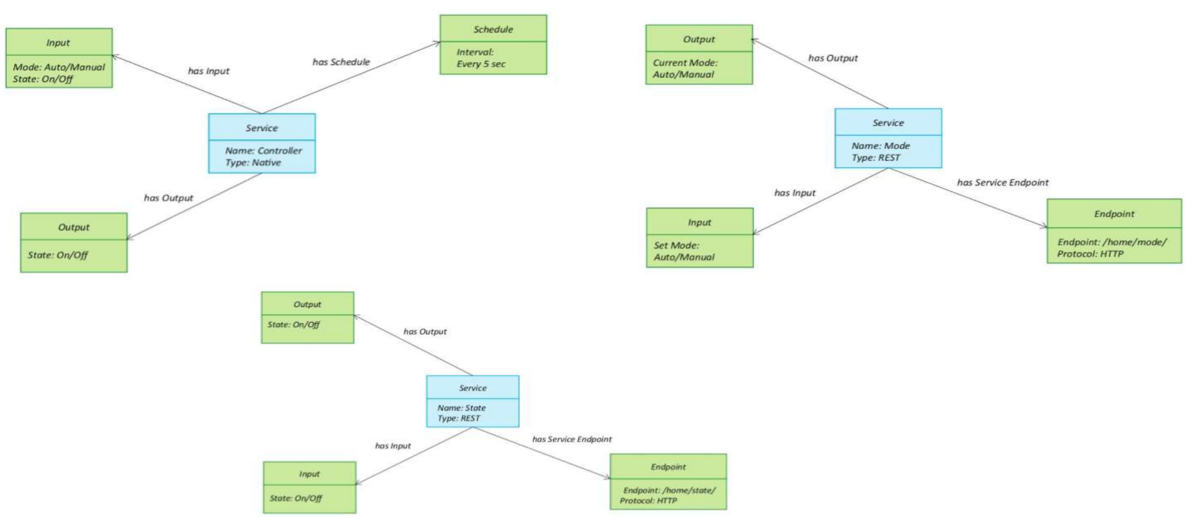
STEP 5: SERVICE SPECIFICATIONS



Controller Service: In auto mode, the controller service monitors the light level and switches the light on/off and updates the status in the status database. In manual mode, the controller service, retrieves the current state from the database and switches the light on/off.

Mr. Santosh Katti, Asst. Prof.
PES UNIVERSITY

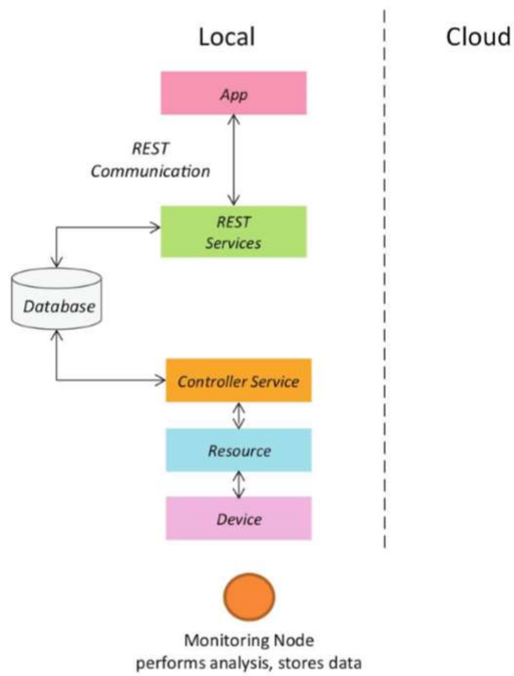
STEP 5: SERVICE SPECIFICATIONS CONTD..



Mr. Santosh Katti, Asst. Prof.

PES UNIVERSITY

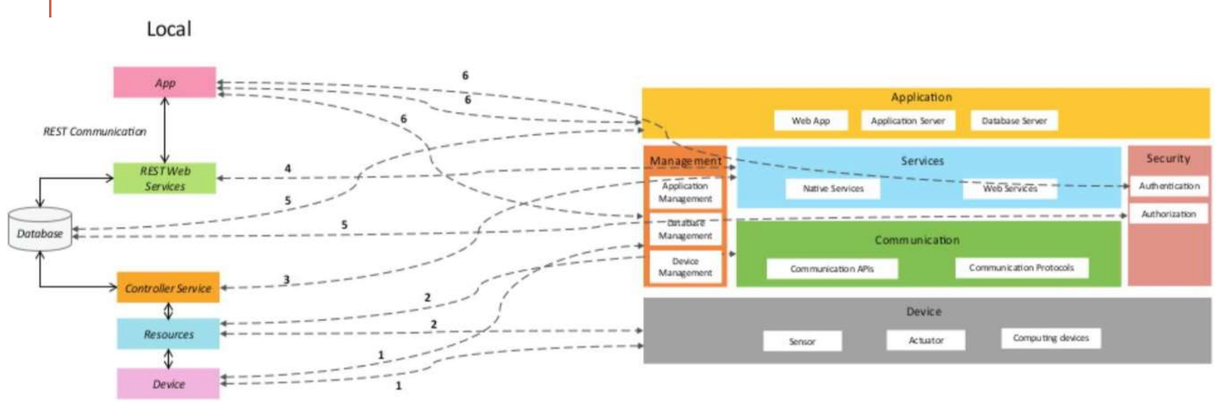
STEP 6: IOT LEVEL SPECIFICATION



Mr. Santosh Katti, Asst. Prof.

PES UNIVERSITY

STEP 7: FUNCTIONAL VIEW SPECIFICATION



1. IoT device maps to the Device FG (sensors, actuators devices, computing devices) and the Management FG (device management)

2. Resources map to the Device FG (on-device resource) and Communication FG (communication APIs and protocols)

3. Controller service maps to the Services FG (native service). Web Services map to Services FG (web services)

4. Web Services map to Services FG (web services)

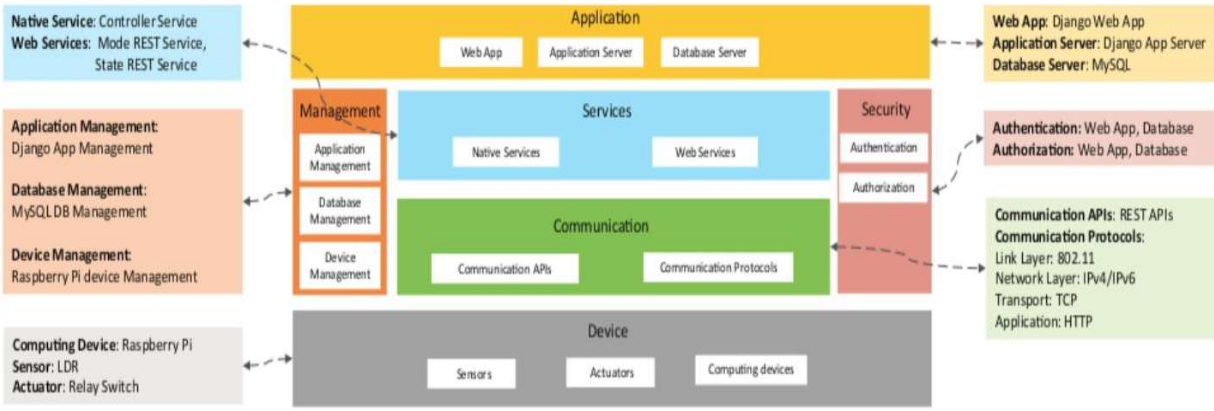
5. Database maps to the Management FG (database management) and Security FG (database security)

6. Application maps to the Application FG (web application, application and database servers), Management FG (app management) and Security FG (app security)

Mr. Santosh Katti, Asst. Prof.

PES UNIVERSITY

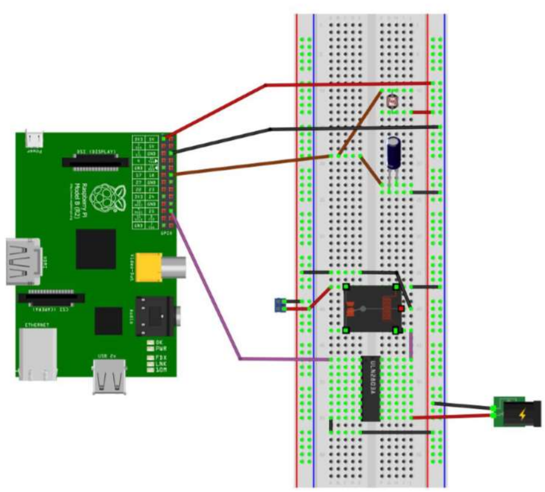
STEP 8: OPERATIONAL VIEW SPECIFICATION



Mr. Santosh Katti, Asst. Prof.

PES UNIVERSITY

STEP 9: DEVICE & COMPONENT INTEGRATION



Mr. Santosh Katti, Asst. Prof.
PES UNIVERSITY

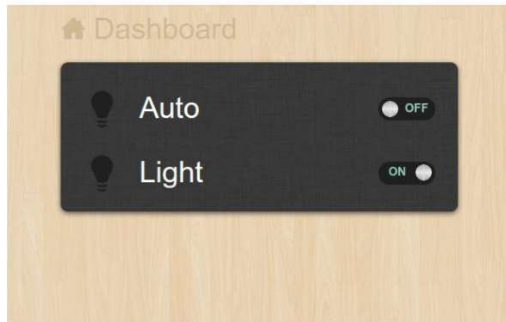
STEP 10: APPLICATION DEVELOPMENT

Auto

- Controls the light appliance automatically based on the lighting conditions in the room

Light

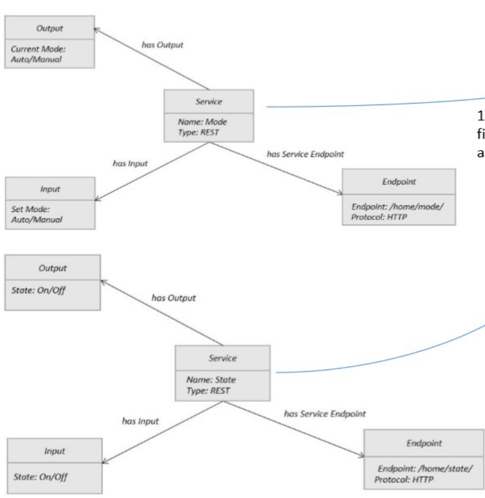
- When Auto mode is off, it is used for manually controlling the light appliance.
- When Auto mode is on, it reflects the current state of the light appliance.



Mr. Santosh Katti, Asst. Prof.
PES UNIVERSITY

IMPLEMENTATION: RESTFUL WEB SERVICES

REST services implemented with Django REST Framework



1. Map services to models. Model fields store the states (on/off, auto/manual)

```
# Models – models.py
from django.db import models

class Mode(models.Model):
        name = models.CharField(max_length=50)

class State(models.Model):
        name = models.CharField(max_length=50)
```

2. Write Model serializers. Serializers allow complex data (such as model instances) to be converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types.

```
# Serializers – serializers.py
from myapp.models import Mode, State
from rest_framework import serializers

class ModeSerializer(serializers.HyperlinkedModelSerializer):
        class Meta:
                model = Mode
                fields = ('url', 'name')

class StateSerializer(serializers.HyperlinkedModelSerializer):
        class Meta:
                model = State
                fields = ('url', 'name')
```

Mr. Santosh Katti, Asst. Prof.

PES UNIVERSITY

IMPLEMENTATION: RESTFUL WEB SERVICES

```
# Models – models.py
from django.db import models

class Mode(models.Model):
        name = models.CharField(max_length=50)

class State(models.Model):
        name = models.CharField(max_length=50)
```

3. Write ViewSets for the Models which combine the logic for a set of related views in a single class.

```
# Views – views.py
from myapp.models import Mode, State
from rest_framework import viewsets
from myapp.serializers import ModeSerializer, StateSerializer

class ModeViewSet(viewsets.ModelViewSet):
        queryset = Mode.objects.all()
        serializer_class = ModeSerializer

class StateViewSet(viewsets.ModelViewSet):
        queryset = State.objects.all()
        serializer_class = StateSerializer
```

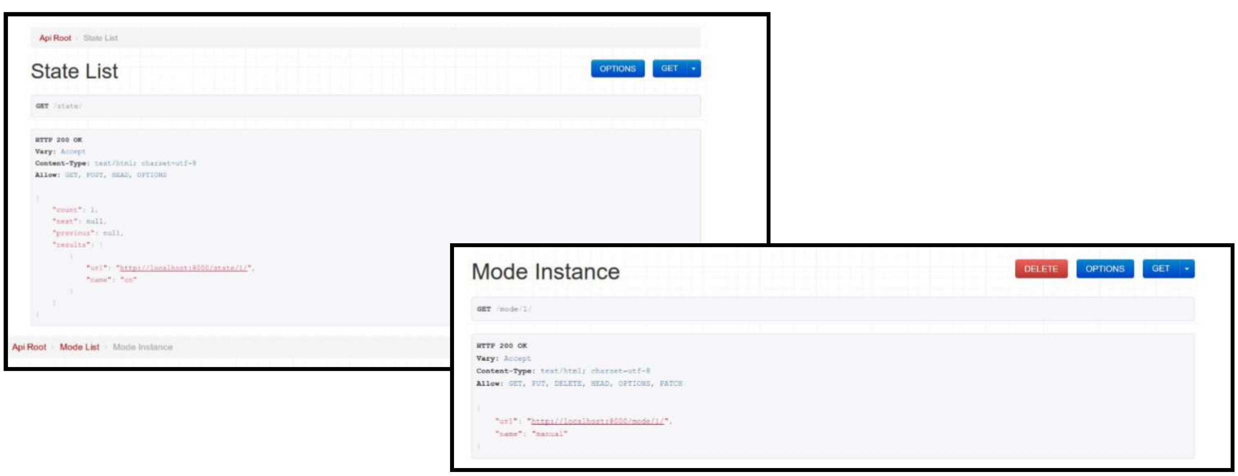
4. Write URL patterns for the services.
Since ViewSets are used instead of views, we can automatically generate the URL conf by simply registering the viewsets with a router class.
Routers automatically determining how the URLs for an application should be mapped to the logic that deals with handling incoming requests.

```
# URL Patterns – urls.py
from django.conf.urls import patterns, include, url
from django.contrib import admin
from rest_framework import routers
from myapp import views
admin.autodiscover()
router = routers.DefaultRouter()
router.register(r'mode', views.ModeViewSet)
router.register(r'state', views.StateViewSet)
urlpatterns = patterns('',
   url(r'^', include(router.urls)),
   url(r'^api-auth/', include('rest_framework.urls', namespace='rest_framework')),
   url(r'^admin/', include(admin.site.urls)),
   url(r'^home/', 'myapp.views.home'),
)
```

Mr. Santosh Katti, Asst. Prof.

PES UNIVERSITY

IMPLEMENTATION: RESTFUL WEB SERVICES - SCREENSHOTS



Mr. Santosh Katti, Asst. Prof.

PES UNIVERSITY

IMPLEMENTATION: CONTROLLER NATIVE SERVICE

1. Implement the native service in Python and run on the device

```
#Controller service
import RPi.GPIO as GPIO
import time
import sqlite3 as lite
import sys

con = lite.connect('database.sqlite')
cur = con.cursor()

GPIO.setmode(GPIO.BCM)
threshold = 1000
LDR_PIN = 18
LIGHT_PIN = 25

def readldr(PIN):
        reading=0
        GPIO.setup(PIN, GPIO.OUT)
        GPIO.output(PIN, GPIO.LOW)
        time.sleep(0.1)
        GPIO.setup(PIN, GPIO.IN)
        while (GPIO.input(PIN)==GPIO.LOW):
                reading=reading+1
        return reading

def switchOnLight(PIN):
        GPIO.setup(PIN, GPIO.OUT)
        GPIO.output(PIN, GPIO.HIGH)

def switchOffLight(PIN):
        GPIO.setup(PIN, GPIO.OUT)
        GPIO.output(PIN, GPIO.LOW)
```

```
def runAutoMode():
        ldr_reading = readldr(LDR_PIN)
        if ldr_reading < threshold:
                switchOnLight(LIGHT_PIN)
                setCurrentState('on')
        else:
                switchOffLight(LIGHT_PIN)
                setCurrentState('off')

def runManualMode():
        state = getCurrentState()
        if state=='on':
                switchOnLight(LIGHT_PIN)
                setCurrentState('on')
        elif state=='off':
                switchOffLight(LIGHT_PIN)
                setCurrentState('off')

def getCurrentMode():
        cur.execute('SELECT * FROM myapp_mode')
        data = cur.fetchone()          #(1, u'auto')
        return data[1]

def getCurrentState():
        cur.execute('SELECT * FROM myapp_state')
        data = cur.fetchone()          #(1, u'on')
        return data[1]

def setCurrentState(val):
        query='UPDATE myapp_state set name="'+val+'"'
        cur.execute(query)

while True:
        currentMode=getCurrentMode()
        if currentMode=='auto':
                runAutoMode()
        elif currentMode=='manual':
                runManualMode()
        time.sleep(5)
```

Mr. Santosh Katti, Asst. Prof.

PES UNIVERSITY

IMPLEMENTATION: APPLICATION

Implement Django Application View

```
# Views – views.py
def home(request):
    out=''
        if 'on' in request.POST:
                values = {"name": "on"}
                r=requests.put('http://127.0.0.1:8000/state/1/', data=values, auth=('username', 'password'))
                result=r.text
                output = json.loads(result)
                out=output['name']
        if 'off' in request.POST:
                values = {"name": "off"}
                r=requests.put('http://127.0.0.1:8000/state/1/', data=values, auth=('username', 'password'))
                result=r.text
                output = json.loads(result)
                out=output['name']
        if 'auto' in request.POST:
                values = {"name": "auto"}
                r=requests.put('http://127.0.0.1:8000/mode/1/', data=values, auth=('username', 'password'))
                result=r.text
                output = json.loads(result)
                out=output['name']
        if 'manual' in request.POST:
                values = {"name": "manual"}
                r=requests.put('http://127.0.0.1:8000/mode/1/', data=values, auth=('username', 'password'))
                result=r.text
                output = json.loads(result)
                out=output['name']

        r=requests.get('http://127.0.0.1:8000/mode/1/', auth=('username', 'password'))
        result=r.text
        output = json.loads(result)
        currentmode=output['name']
        r=requests.get('http://127.0.0.1:8000/state/1/', auth=('username', 'password'))
        result=r.text
        output = json.loads(result)
        currentstate=output['name']
        return render_to_response('lights.html',{'r':out, 'currentmode':currentmode, 'currentstate':currentstate},
context_instance=RequestContext(request))
```

Mr. Santosh Katti, Asst. Prof.
PES UNIVERSITY

IMPLEMENTATION: APPLICATION

Implement Django Application Template

```
<div class="app-content-inner">
<fieldset>
<div class="field clearfix">
<label class="input-label icon-lamp" for="lamp-state">Auto</label>
<input id="lamp-state" class="input js-lamp-state hidden" type="checkbox">
{% if currentmode == 'auto' %}
<div class="js-lamp-state-toggle ui-toggle " data-toggle=".js-lamp-state">
{% else %}
<div class="js-lamp-state-toggle ui-toggle js-toggle-off" data-toggle=".js-lamp-state">
{% endif %}
<span class="ui-toggle-slide clearfix">
<form id="my_form11" action="" method="post">{% csrf_token %}
<input name="auto" value="auto" type="hidden" />
<a href="#" onclick="$(this).closest('form').submit()"><strong class="ui-toggle-off">OFF</strong></a>
</form>
<strong class="ui-toggle-handle brushed-metal"></strong>
<form id="my_form13" action="" method="post">{% csrf_token %}
<input name="manual" value="manual" type="hidden" />
<a href="#" onclick="$(this).closest('form').submit()"><strong class="ui-toggle-on">ON</strong></a>
</form></span>
</div></div>
<div class="field clearfix">
<label class="input-label icon-lamp" for="tv-state">Light</label>
<input id="tv-state" class="input js-tv-state hidden" type="checkbox">
{% if currentstate == 'on' %}
<div class="js-tv-state-toggle ui-toggle " data-toggle=".js-tv-state">
{% else %}
<div class="js-tv-state-toggle ui-toggle js-toggle-off" data-toggle=".js-tv-state">
{% endif %}
{% if currentmode == 'manual' %}
<span class="ui-toggle-slide clearfix">
<form id="my_form2" action="" method="post">{% csrf_token %}
<input name="on" value="on" type="hidden" />
<a href="#" onclick="$(this).closest('form').submit()"><strong class="ui-toggle-off">OFF</strong></a>
</form>
<strong class="ui-toggle-handle brushed-metal"></strong>
<form id="my_form3" action="" method="post">{% csrf_token %}
<input name="off" value="off" type="hidden" />
<a href="#" onclick="$(this).closest('form').submit()"><strong class="ui-toggle-on">ON</strong></a>
</form>
</span>
{% endif %}
{% if currentmode == 'auto' %}
{% if currentstate == 'on' %}
<strong class="ui-toggle-on">    ON</strong>
{% else %}
<strong class="ui-toggle-on">    OFF</strong>
{% endif %}{% endif %}
</div>
</div>
</fieldset></div></div></div>
```

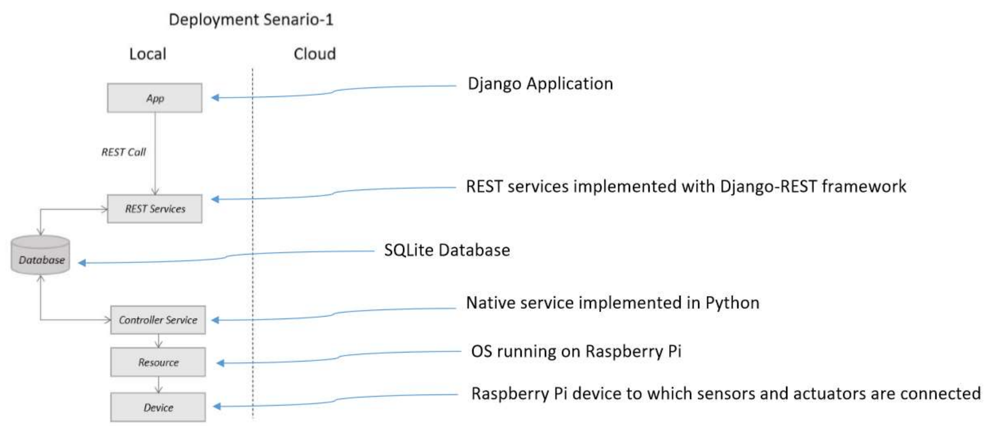
Mr. Santosh Katti, Asst. Prof.
PES UNIVERSITY

FINALLY - INTEGRATE THE SYSTEM

- Setup the device
- Deploy and run the REST and Native services
- Deploy and run the Application
- Setup the database

Mr. Santosh Katti, Asst. Prof.
PES UNIVERSITY

FINALLY - INTEGRATE THE SYSTEM



Mr. Santosh Katti, Asst. Prof.
PES UNIVERSITY

PROJECT WORK BEGINS…. BEST WISHES