Project Documentation for Smart water Fountains

Objectives

The use of remote switching and monitoring of irrigation system using smart phones to address the need of automatic control of the water is presented. The data about soil moisture, temperature and humidity is sent to the smart phone for the user to make the decision.

IoT Sensor setup

Sensors play an important role in creating solutions using IoT. Sensors are devices that detect external information, replacing it with a signal that humans and machines can distinguish.

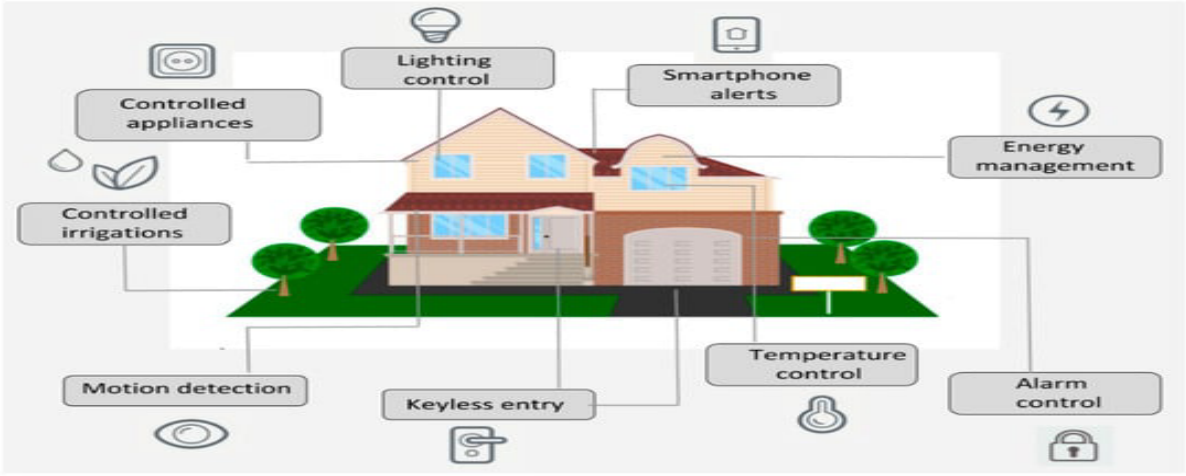
How to create mobile app with IoT?

8 Steps to Develop an IoT App

1. Define the Requirements (What?) .
2. Choose an IoT Platform. .
3. Get the Correct Hardware Components (With?)
4. Choose the Network Protocol (Which?)
5. Develop the Software.
6. Connect the Hardware and Software.
7. QA – Test the app.
8. Launch the IoT App.

IoT sensors are pieces of hardware that detect changes in an environment and collect data. They're the pieces of an IoT ecosystem that bridge the digital world to the physical world.

The key parameters of the water fountain were derived from the sensors and transmitted the encrypted data by GPRS or WIFI net to central server automatically. The Access database was established according to the real time running data of each water fountain fixed everywhere



Mobile app an IoT:

With the improvement of IoT, the world has got a wide range of new concepts. such as smart cities, connected vehicles, and connected toys. Mobile applications play a significant role in the Internet of Things. They are an interface that allows users to interact with IoT-enabled physical devices.

Reasons to Build a Custom Mobile App with IoT

- Enhanced User Experience. The IoT allows developers to enhance their product's user experience. ...
- Increased Efficiency. IoT-based mobile apps can make things more efficient by reducing manual processes. ...
- Cost Savings. ...
- Scalability. ...
- Security. ...

- Healthcare. ...
- Manufacturing. ...
- Agriculture.

Impact of IoT on mobile app development:

The impact of iot platform allows mobile applications to be cross-platform compatible, resulting in consistent and seamless user experiences regardless of the device used. This increases user satisfaction and engagement resulting in the app's popularity among a wide range of users.

IoT-based apps give users more control over home automation systems and health monitoring devices. These devices leverage cloud computing to enable faster data transfer speeds and better security. Cloud technology makes it easier to create mobile applications over the IoT network.

Enhanced User Experiences: IoT can enable mobile apps to provide personalized and context-aware experiences. By leveraging data from IoT devices and sensors, apps can deliver tailored content, recommendations, and notifications based on a user's preferences, location, and real-time data

IoT mobile app development allows mobile apps to collect data from connected devices to provide more personalized features and recommendations. It helps improve the UX and end-user engagement.

IoT Mobile Applications Play an Important Role in Enhancing the User Experience. The IoT devices let users make life simpler with the help of features like real-time data exchange, remote control, and automation. These features come into the picture by leveraging the capabilities of immersive technologies.

IoT (Internet of Things) is a technology of the future that is already becoming a reality in recent years. Thanks to the help of IoT, the way humans function on a daily basis has changed.

Instead of performing all tasks manually, with just an Internet connection, people can now simplify complicated tasks, track the status of connected devices and improve their quality of life.

Not only enclosed with various physical objects such as sensors, actuators, vehicles, etc., but IoT technology is also far more out of the curve when combined with mobile devices. IoT mobile app development has opened up a myriad of possibilities and taken technology to new heights. Mobile devices act as a gateway to control and interact with IoT systems, making them an integral part of the IoT ecosystem.

How IoT is used in mobile?

IoT technologies allow smart sensors and connected devices to automatically collect information and share it. Adding a layer of mobile technology means that you can access shared information from wherever you happen to be, as long as you have an internet connection.

How can I create my own mobile app?

How to create an app for mobile devices

1. Get your app idea on paper.
2. Build a Native app or a PWA, based on your needs.
3. Make your app by selecting the most suitable method for your business.
4. Create an app with an app builder (no-code option)
5. Test your app on iOS and Android devices.
6. Submit and Publish your app on the stores.

IoT Using the Raspberry Pi

The process of connecting sensors and objects with one another and with your applications and databases. Once connected, you can implement end-to-end automations that help you make full use of your equipment.

Raspberry Pi computers feature a set of General Purpose Input Output (GPIO) pins that provide connections to external electronic devices and therefore the development of IoT solutions. These GPIO pins can be connected to external sensors using either jumper wires or a ribbon cable.

The Raspberry Pi has a 40-pin GPIO (General Purpose Input/Output) connection, which makes it very easy to connect to the outside world. To connect the GPIO to external sensors, you can: Connect the sensors directly to the GPIO pins using jumper wires.

Connect a Raspberry Pi or other device

1. Set up your device.
2. Install the required tools and libraries for the AWS IoT Device SDK.
3. Install AWS IoT Device SDK.
4. Install and run the sample app.
5. View messages from the sample app in the AWS IoT console.

The Sense HAT is an add-on board that gives your Raspberry Pi an array of sensing capabilities. The on-board sensors allow you to monitor pressure, humidity, temperature, colour, orientation, and movement.

Hardware Components:

1. Raspberry Pi 3 Model B or later
2. Water pump
3. Relay module
4. Water level sensor (optional)
5. Power supply for the pump
6. Tubing and fountain nozzle
7. Waterproof container for the water reservoir
8. Various cables, connectors, and a breadboard

Software Components:

1. Raspbian OS (or a suitable Raspberry Pi OS)
2. Python for programming
3. IoT platform (e.g., MQTT, AWS IoT, or Google Cloud IoT Core)
4. Libraries for GPIO control (e.g., RPi.GPIO)
5. Optional: Web server and HTML/CSS/JavaScript for a web-based user interface

Steps to Create the Smart Water Fountain:

1. **Set Up Raspberry Pi:**
   - Install Raspbian OS on your Raspberry Pi.
   - Update the system packages using `sudo apt-get update` and `sudo apt-get upgrade`.

2. **Hardware Setup:**
   - Connect the water pump to a relay module. The relay will allow the Raspberry Pi to control the pump's power.
   - Connect the relay module to the GPIO pins on the Raspberry Pi. Make sure to connect it properly, following the GPIO pinout for your model.
   - Optionally, you can connect a water level sensor to the Raspberry Pi to monitor the water level in the reservoir.

---

3. **Install Required Libraries:**
   - Install the necessary Python libraries for GPIO control, such as RPi.GPIO.
   - You may also need additional libraries for sensor data (if using a water level sensor) and for connecting to your chosen IoT platform.

4. **Code the Fountain Control:**
   - Write a Python script that reads sensor data (if used), controls the water pump through the relay, and sends status updates to your chosen IoT platform.
   - You'll need to program the logic for when to turn the pump on and off based on sensor data or a predefined schedule.
   - Make sure to handle exceptions and errors in your code.

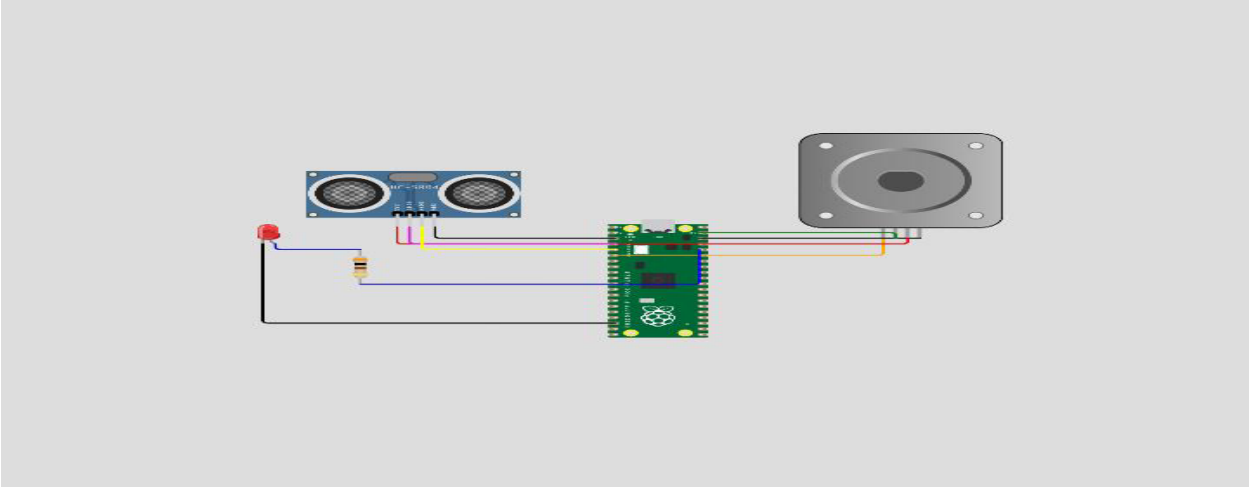
5. **IoT Integration:**
   - Set up your chosen IoT platform (e.g., AWS IoT, MQTT broker, Google Cloud IoT Core) and create a device or topic to which your Raspberry Pi can publish data.
   - Modify your Python script to connect to the IoT platform and publish status updates.

6. **Optional: Web Interface**
   - Create a web-based user interface for controlling the water fountain remotely using HTML, CSS, and JavaScript.
   - You can use libraries like Flask to create a web server on your Raspberry Pi.

7. **Assemble and Test:**
   - Assemble all the components in a waterproof container with the water reservoir, pump, and tubing.
   - Test the system to ensure that it turns the pump on and off correctly based on your code and sensor inputs.
   - Test remote control and monitoring via your IoT platform.

8. **Finalize and Deploy:**
   - Once everything is working as expected, secure the components and deploy your smart water fountain in the desired location.

Remember to follow safety precautions when working with water and electrical components, and consult relevant datasheets and documentation for your specific hardware components.



Raspberry pi integration

The Raspberry Pi integration uses the agent to collect metrics related to the operating system (Linux-based), including aspects like CPU usage, load average, memory usage, and disk and networking I/O. It also supports system logs being scraped by the agent using promtail.

Raspberry code implementation

```
1.#!/bin/sh
2. ### BEGIN INIT INFO
3. # Provides:          fountain
4. # Required-Start:    $remote_fs
5. # Required-Stop:     $remote_fs
6. # Default-Start:     2 3 4 5
7. # Default-Stop:      0 1 6
8. # Short-Description: Interrupt-based GPIO LED/pushbutton daemon
9. # Description:       Listens for button events and lights up LEDs
10. ### END INIT INFO
11.
12. # Do NOT "set -e"
13.
14. # Install this script instructions:
15. #   chmod a+x /home/pi/fountain.py
16. #   sudo ln -s /home/pi/fountain /etc/init.d
17. #   sudo /etc/init.d/fountain start
18. #   sudo ln -s /etc/init.d/fountain /etc/rc5.d/S01fountain
19. #   This file name/location: /home/pi/fountain which is linked to /etc/init.d/fountain:
20.
21.
22. # PATH should only include /usr/* if it runs after the mountnfs.sh script
23. PATH=/sbin:/usr/sbin:/bin:/usr/bin
24. DESC="fountain daemon"
```

```
25. NAME=fountain
26. #DAEMON=/usr/bin/$NAME
27. DAEMON=/home/pi/fountain.py
28. ARGS=""
29. PIDFILE=/var/run/$NAME.pid
30. SCRIPTNAME=/etc/init.d/$NAME
31.
32.
33. export QUIET=1
34.
35. # Exit if the package is not installed
36. [ -x "$DAEMON" ] || exit 0
37.
38. # Read configuration variable file if it is present
39. [ -r /etc/default/$NAME ] && . /etc/default/$NAME
40.
41. # work out daemon args
42.
43. # Load the VERBOSE setting and other rcS variables
44. . /lib/init/vars.sh
45.
46. # Define LSB log_* functions.
47. # Depend on lsb-base (>= 3.2-14) to ensure that this file is present
48. # and status_of_proc is working.
49. . /lib/lsb/init-functions
```

```
50.
51. #
52. # Function that starts the daemon/service
53. #
54. do_start()
55. {
56.       # Return
57.       #   0 if daemon has been started
58.       #   1 if daemon was already running
59.       #   2 if daemon could not be started
60.       start-stop-daemon --start --quiet --pidfile $PIDFILE --user root --exec $DAEMON --test
> /dev/null \
61.                 || return 1
62.
63.       start-stop-daemon --start --quiet --pidfile $PIDFILE --user root --make-pidfile
--background --no-close \
64.     --exec $DAEMON -- \
65.                 $ARGS \
66.                 || return 2
67.      sleep 1
68. }
69.
70. #
71. # Function that stops the daemon/service
72. #
73. do_stop()
```

```
74. {
75.       # Return
76.       #   0 if daemon has been stopped
77.       #   1 if daemon was already stopped
78.       #   2 if daemon could not be stopped
79.       #   other if a failure occurred
80.       start-stop-daemon --stop --retry=TERM/30/KILL/5 --pidfile $PIDFILE --user root
#--exec $DAEMON # don't pass --exec since is /usr/bin/python for scripts
81.       RETVAL="$?"
82.       [ "$RETVAL" = 2 ] && return 2
 83.       sleep 1
84.       # Many daemons don't delete their pidfiles when they exit.
85.       rm -f $PIDFILE
86.       return "$RETVAL"
87. }
88.
89. case "$1" in
90.  start)
91.       [ "$VERBOSE" != no ] && log_daemon_msg "Starting $DESC" "$NAME"
92.       do_start
93.       case "$?" in
94.                 0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
95.                 2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
96.       esac
97.       ;;
```

```
98.  stop)
99.      [ "$VERBOSE" != no ] && log_daemon_msg "Stopping $DESC" "$NAME"
100.     do_stop
101.     case "$?" in
102.             0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
103.             2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
104.     esac
105.     ;;
106. status)
107.     status_of_proc "$DAEMON" "$NAME" && exit 0 || exit $?
108.     ;;
109.  restart|force-reload)
110.     log_daemon_msg "Restarting $DESC" "$NAME"
111.     do_stop
112.     case "$?" in
113.       0|1)
114.             do_start
115.             case "$?" in
116.                     0) log_end_msg 0 ;;
117.                     1) log_end_msg 1 ;; # Old process is still running
118.                     *) log_end_msg 1 ;; # Failed to start
119.             esac
120.             ;;
121.       *)
122.             # Failed to stop
```

```
123.                log_end_msg 1
124.                ;;
125.    esac
126.    ;;
127.  *)
128.    echo "Usage: $SCRIPTNAME {start|stop|status|restart|force-reload}" >&2
129.    exit 3
130.    ;;
131. esac
132.
133. :
```

Raspberry fountain service

```
1.[Unit]
2. Description=RPi Fountain
3. After=network.target
4.
5. [Service]
6. ExecStart=/usr/bin/python3 -u /home/pi/rpi-fountain/fountain.py
7.WorkingDirectory=/home/pi/rpi-fountain
8. StandardOutput=inherit
9. StandardError=inherit
10. # The following "file:" option requires systemd version 236 or newer
11. # 2019-04-08-raspbian-stretch-lite has version 232
12. # To show version run: systemctl --version
13.#StandardOutput=file:/var/log/fountain.log
```

14.#StandardError=file:/var/log/fountain.log

15. Restart=always

16.User=root

17.

18.[Install]

19.WantedBy=multi-user.target

Stepup run from raspberry pi

1.#!/bin/bash

2.

3.# Set the fountain python program to start as a service

4.sudo cp fountain.service /etc/systemd/system/fountain.service

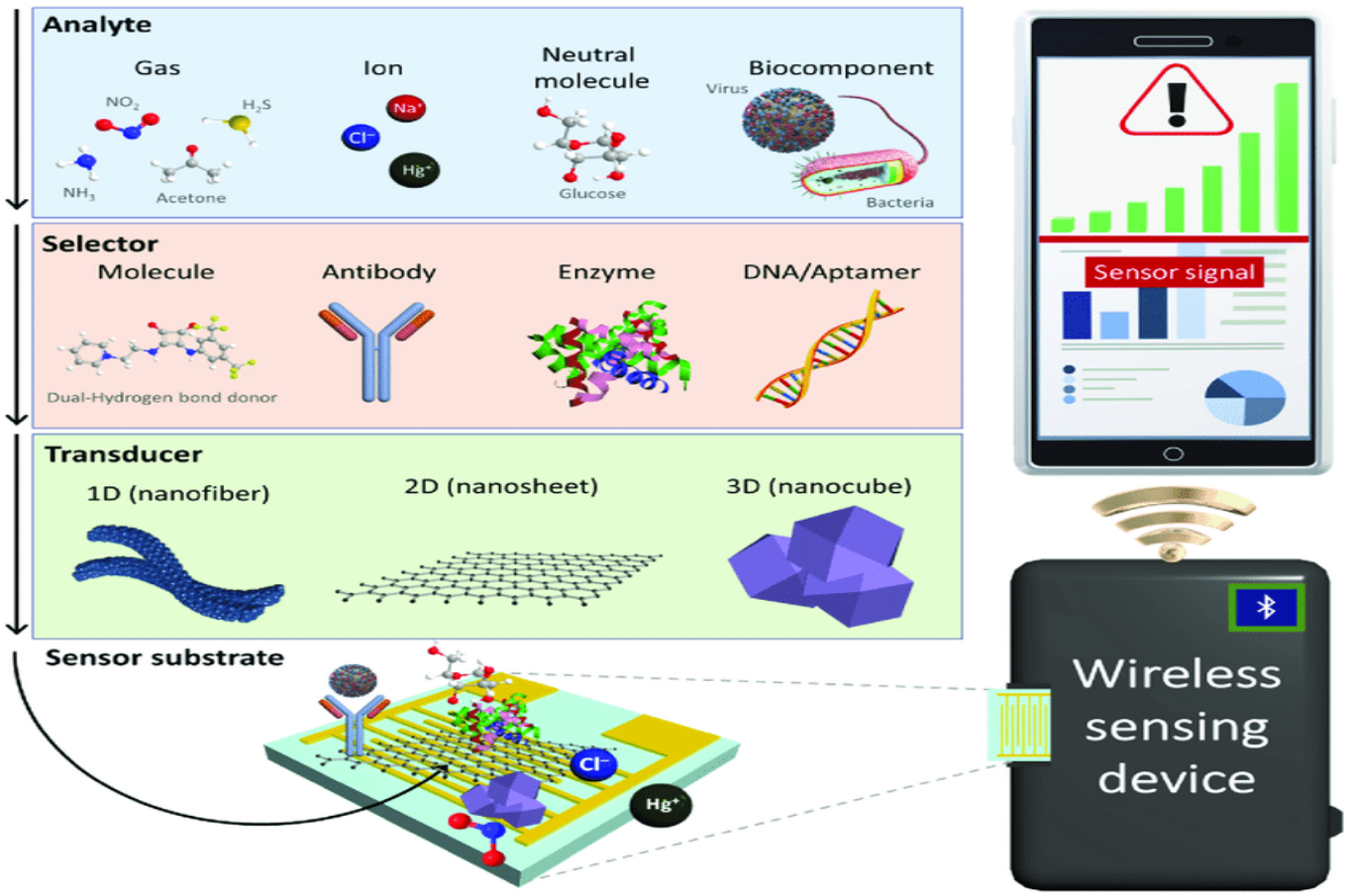
5.sudo systemctl daemon-reload

6.sudo systemctl enable fountain.service

7.sudo systemctl start fountain.service

8.sudo systemctl status fountain.service

Schematic



Schematic illustration of an IoT sensing platform comprising a chemical sensor, a wireless sensing device, and a smartphone for POCT application. Electrochemical sensors using nanomaterials consisting of selectors and transducers produce electrical signals upon chemical interactions with various analytes such gases, ions, neutral molecules, and biocomponents.

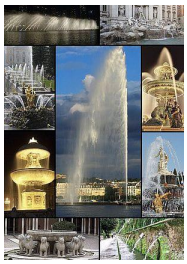
Screenshots of IOT sensors and Application


12:15
light
Connected to 192.168.43.207
960
920
880
840
800
760
12:11:59
12:12:26
12:13:16
12:13:39
12:14:00
12:14:21
12:14:43
12:15:05


100%
오전 5:46
MySensorGuard
Network Interfaces
Bluetooth
ZigBee
Update Interval
1
10
1
Sec
Sensing Interfaces
RGB
Hum Temp
Accel
UV
UV
Light
Prox
IR
Pres sure
HR
Home
Comfort Level
Food Quality
Config.

Real time water fountain status

A water fountain or drinking fountain is designed to provide drinking water and has a basin arrangement with either continuously running water or a tap. The drinker bends down to the stream of water and swallows water directly from the stream.



Public awarness

Conclusion

Public awareness stimulates innovation in water conservation technologies. As more individuals become conscious of the value of water, the demand for innovative solutions like smart water meters, leak detection systems, and efficient irrigation techniques increases.