Design Report

Cloud1

Archit Mendiratta, Tam Nguyen, Max Parelius, Etienne Urbain-Racine, Corey Wilson

4/20/15

Table of Contents

Introduction

Project Overview

The IoT team will be developing a cloud based system to connect devices in a standard way in order to create the internet of things. The components of the product will be a core platform, a GUI, a database, and APIs for each device to be connected to the cloud. The GUI will be a web based application that allows the user to add a new device to the cloud network. The user will be able to easily add a device to the network by using the interface to specify information about the device and the network it is to be added to. The database will contain information about each device in the network. The APIs will be unique to each device in the network. The APIs will include functionality for contacting the device and communicating between other devices. The core platform will be the central server point that interfaces with the GUI, database, and APIs. The core platform will generate API code based on user input about a device in the GUI, and store device information in the database. This system will ultimately provide a standard environment for connecting devices and will be scalable in order to integrate a number of APIs for new devices.

Clients and Community Partners

There are two clients with whom we are serving on this project. The first of which is Simon Koo, an established professor at Santa Clara University. The second client is Shivakumar Mathapathi, the COO of Dew Mobility Solutions in San Jose. All parties in question, clients and developers, will benefit from this project. The clients will at the end

receive a working product as well as research and development that provides insight into how to proceed with the product in the future. The developers will receive funding to develop the product and gain knowledge and insight into many new skills of development. The clients will receive whatever we have developed at the end of the six month process. We will present the clients with adequate documentation on the work that was done as well as input on future possibilities for the product.

Stakeholders

Other than our clients, Simon Koo and Shivakumar Mathapathi, as well as their company, there are no direct stakeholders. However, depending on the success of our project, quite a few people may be interested. The IoT field is rapidly expanding and our project could further others' endeavors in the field. For example, Nest Labs may be interested in our software as it relates to their goals and purposes. We would like for our project to succeed and become a tool that other companies can use. Our project will be a small stepping stone in the advent of IoT technology. If our project follows what we envision then this will allow any individual developer to easily connect devices together. Our software will open the IoT field to large companies as well as individual consumers.

Framed Insights and Opportunities

The project clients initially came to the IoT team with the general idea of building a developer-friendly, cloud-based IoT platform for connecting, managing, and integrating heterogeneous devices from different vendors running different standards. The design requirements were specified to have an Amazon Web Services (AWS) core

for system integration built in Scala, a database for device records and identity, several APIs for connecting the smart devices to the core (in C, Java, or Scala), and a GUI for programming and managing the devices.

After the first meeting with the clients, it is clear that the team is given a lot of freedom regarding the direction of this project, given that everyone on the team must agree to a specific design. However, the clients did request several things from the team, which are as follows: give each team member a responsibility, dedicate more focus and resources towards the API, maintain weekly conversations with the client (or more frequently as needed), look into some of the other existing IoT platforms (a few links were provided by the Shiva), and review/learn as much as possible about Scala. The team was also given the option to use GitHub for version control and project management, and to use Dropbox or Google Docs for collaboration. The clients mentioned a good starting point for the team would be to figure out how to use basic APIs, specifically regarding social networks (Facebook, Twitter, etc).

After further meetings with the clients, the discussion shifted towards other existing IoT projects. As the team asked questions about what aspects the Cloud1 platform should have, different ideas from different IoT projects were pulled together to try and come to a consensus about which direction Cloud1 should be heading towards. Finalized amongst the team, Cloud1 will ultimately be built into a more developer-based (rather than consumer-based) tool, creating APIs for open-sourced development. The clients requested a block diagram of our initial system design architecture, for which we

created and emailed to them with further details on where the team believed the project was going. The clients will be constantly giving us feedback as we continue to communicate with them through Skype or Cal Poly's teleconferencing room. In the meantime, the clients have given us a list of a total of eighteen different types of sensors they can send us to work with.

Goals and Objectives

The IoT team's project goals are:

1. Create a program that can be easily expanded upon in the future and scale with any number of devices.
2. Build a developer-friendly, cloud-based Internet of Things platform to connect different products.
3. Have an easy-to-learn graphical user interface for programming and managing a variety of devices.
4. Use a relational database management system to keep track of each device connected over a network.
5. Make the platform adaptable and capable of linking with numerous different items.

The IoT team's project objectives are:

1. The system needs to be able to communicate with at least two devices across a local network.
2. The user should be able to control the connected devices through from external networks.

3. The system needs to support usernames and passwords for security purposes.
4. At least two devices need to be able to communicate with each other based on programmable logic (e.g. a temperature sensor that turns on the A.C. when it's over 70 degrees Fahrenheit).
5. Finally, APIs for every device need to be available for developers to utilize through open-source.

Outcomes and deliverables

Our project results will be an easy-to-integrate Internet of Things platform that not only manages different devices (such as sensors, electrical outlets, etc.) but also allows each of them to communicate with one another based on programmable logic. This platform can be utilized anywhere with a wireless network such as a home environment or at a workplace. Also, the platform will be very easy for other developers to use and adapt to their own projects through the power of open-source. When the project is finished, the IoT team will leave behind a scalable platform for both developers to extend and consumers to enjoy.

Team Mission and Team Objectives

Mission Statement:

To create scalable software that facilitates the communication between any number of smart devices using different architectures and designs.

Objectives:

- To create hard deadlines and stick to a schedule
- To maintain communication with client and group members
- To keep progress updated on each members' work
- To create a program that can be easily expanded upon in the future and scale with any number of devices

Team membership and roles

Archit Mendiratta - Procurement, Software Designer, System Interface

Tam Nguyen - Software Designer, Development Tools Specialist

Max Parelius - Liaison, Software Designer, Product Reliability and Serviceability

Etienne Urban-Racine - Project Manager, Software Designer

Corey Wilson - Financial Officer, Software Designer, System Architect

Related Work

1. "Hello, Smart Home." SmartThings. SmartThings, n.d. Web. 17 Oct. 2014. Smart things is an easy-to-use app that turns your smartphone into a remote to control all of the smart devices in your home. It is very similar to what we will be building with our IoT platform in that it connects devices in a standard way.
2. "Ayla Networks." Ayla Networks. Ayla Networks, n.d. Web. 17 Oct. 2014. Ayla Networks provides a cloud-based application enablement platform that makes it easy and cost effective for people to connect any device to the Internet. This product uses some of the same ideas that we are planning on implementing into our system.

3. "Technology - EVRYTHNG." EVRYTHNG. N.p., n.d. Web. 17 Oct. 2014. The EVRYTHNG Engine provides technology to create and serve digital identities for any device. Each device has an online profile creating a unique digital presence on the Web. It is essentially a Facebook for things where individual devices, just like people on social networks, have their own unique digital profiles that enable communications, apps and services.

4. "Projects." Web of Things. WebOfThings.org, n.d. Web. 17 Oct. 2014. Web of things is a useful blog to find information about new technologies that people are using and developing relating to IoT. It will be a useful source of ideas and technologies during our development.

5. "Get Connected." Connect2me. Plasma Business Intelligence, n.d. Web. 17 Oct. 2014. Connect2.me is a large IoT educational ecosystem for the developer community. It is a centralized location for developers to find APIs and smart devices for integrating into their projects and product offerings. Will be very useful during our development.

6. "Revolv." Revolv Devices. Revolv Inc., n.d. Web. 17 Oct. 2014. Revolv makes it easy to integrate and connect wireless smart devices. It can do things like allowing you to control Philips Hue lights, Yale and Kwikset locks, Sonos Hi-Fi speakers, Belkin WeMo, Honeywell thermostats and Insteon sensors and switches from one convenient app. This app does a lot of what we will be trying to do with our platform.

Formal Product Definition

Intro to the formal product definition

The Internet of Things platform is a scalable software that facilitates the communication between any number of wireless-capable devices using different architectures and designs.

Marketing Requirements

- Product should be user friendly
- Product should be reliable and accurate
- Product should ease the integration of APIs that manage an array of IoT devices
- Product should be responsive to user action
- Product should be easy to install and start using

Engineering Requirements

- System will initially contain 3 compatible devices but will be able to support any number of devices and APIs
- System should have a core platform that will act as an interface between the users and devices
- The core platform will also hold APIs for all existing types of devices in the network
- System should have a database to store device information
- APIs should all contain standard protocols for communication between devices

Constraints

- System should have a GUI to add devices to the cloud network and control existing devices on the network

Criteria

- Device registration: The platform needs to be able to register and add new devices.
- Device logic: The platforms needs to be able to manipulate devices through programmable logic.
- External data gathering: The platform should be able to gather information from publicly accessible devices.
- Local data gathering: The platform should be able to gather device information and identify devices through the local network.
- Device sharing: The platform should have an option to allow the users to share their devices across the web for other users to use, either publicly or privately.
- Connecting devices: The platform needs to allow devices to connect through it for either data gathering, device configuration, or both.

The User Experience

Overview of the User Experience

Use-Case	AddDevice
Actors	User, Database, Webserver
Description	This use-case occurs when a user of a computer or smart device wants to add a new device using the AddDevice. The AddDevice prompts the user to enter information about the devices such as the device name, its MAC address, etc. and stores the new information into a database so the server could connect the device to the main platform.
Stimulus	User adds a new device through NewDevice.
Response	Verify all of the details entered are correct and are able to find the device on the network.

Use-Case	CodeFunctions
Actors	User, Devices, GUI
Description	This use-case occurs when a user wants to add functionality to all of the connected devices in the network. All the user has to do it simply use the GUI to connect visual representations of devices together and insert a programmable logic statement to auto-generate code. Then, the devices will act based on the logic the user entered.
Stimulus	User connects visual representations of devices together in the GUI.
Response	Make sure that the logic entered by the user is programmable and if it is, then follow through.

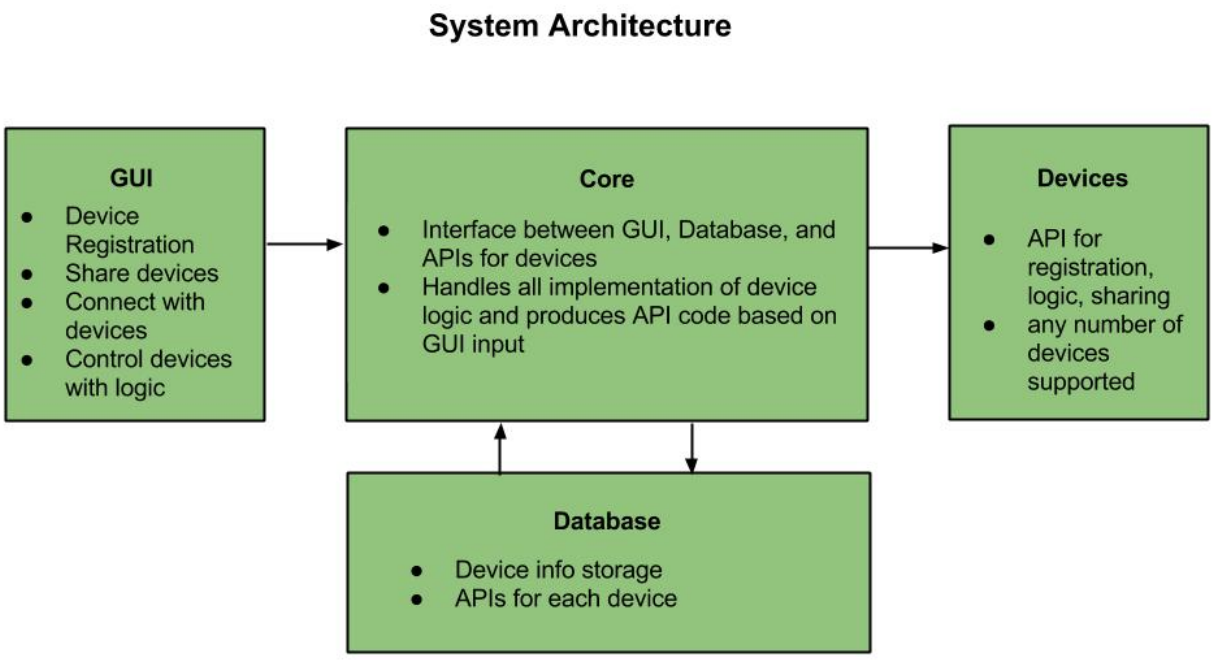
Personas

This platform is accessible and useful for people from varying backgrounds, including:

- Developers: Those who have a strong technical experience and/or a degree in Computer Science/Engineering will find the platform very useful as it allows the user to be able to code for the devices added to the network.
- Average person: Those who have little to no experience in coding but can still use everyday computer application will also find the platform useful as it contains a GUI that allows to add and control various devices.
- Corporations: Due to the platform's scalability, everyone (or only specific people) can use the platform.

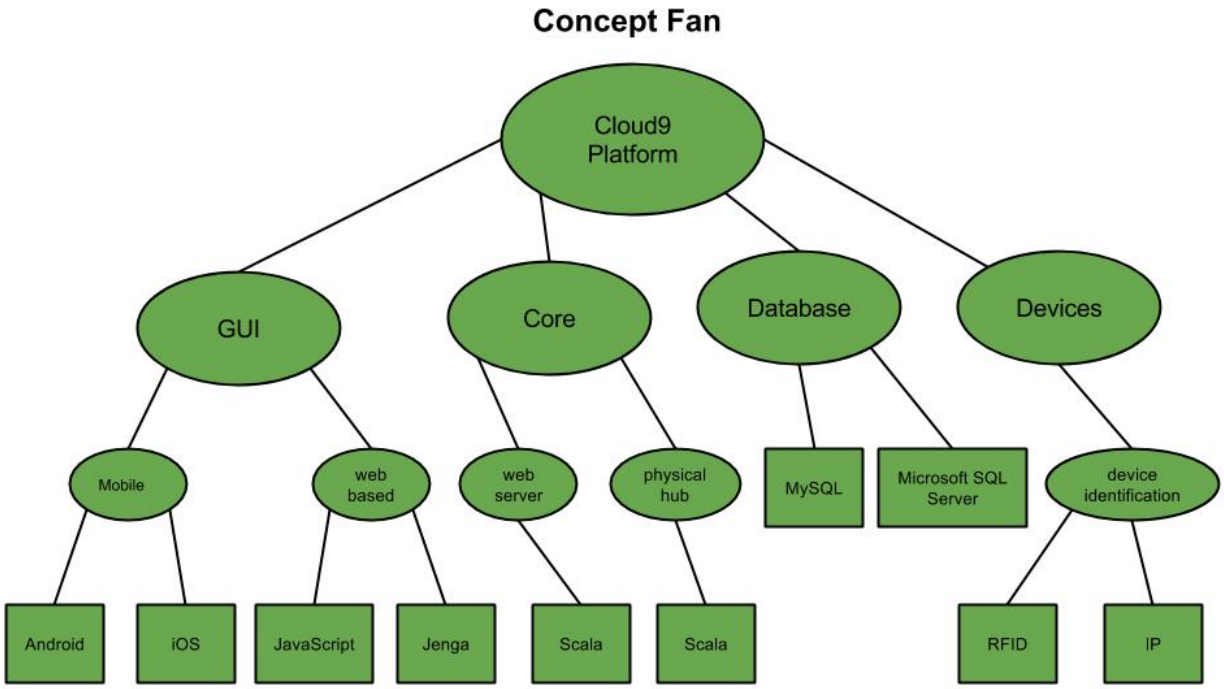
Design and Justification

The first step of our design process was to define the overall high level architecture of our system. The figure below shows the system architecture of our product and the components that comprise it.



Our system consists of a GUI, a Core, a Database, and Devices and the diagram above gives descriptions of what each component's purpose is.

Once the system components were well defined at a high level we proceeded to generate a concept fan that outlines different possibilities and solutions for implementing each component of the system. The concept fan is shown below.



The way the concept fan works is that we have subtrees of the root representing different components of the system. In our case the Cloud1 platform is the root. Each component of the Cloud1 platform is a subtree which has additional subtrees representing different possible courses of action for implementing that component. The leaves of the concept fan represent different possible tools for implementing each course of action of each component.

After creating the concept fan with all possible ideas for each component, we put our ideas into a decision matrix to come up with a solution that best fits each of our criteria for system requirements. The decision matrix is shown below.

Decision Matrix

Criteria	Wt	% Importance	Idea1	Idea2	Idea 3	
Device Registration	3	22	7	3	1	<- enter a criterias score 1-10
Device Logic	4	30	6	6	5	<- enter a criterias score 1-10
External data gathering	1	7	8	6	8	<- enter a criterias score 1-10
local data gathering	1.5	11	9	9	9	<- enter a criterias score 1-10
Device Sharing	2	15	5	5	5	<- enter a criterias score 1-10
Connecting devices	2	15	7	6	4	<- enter a criterias score 1-10
Totals	13.5	100	6.7	5.5	4.6	<- Highest score total 10 lowest 1

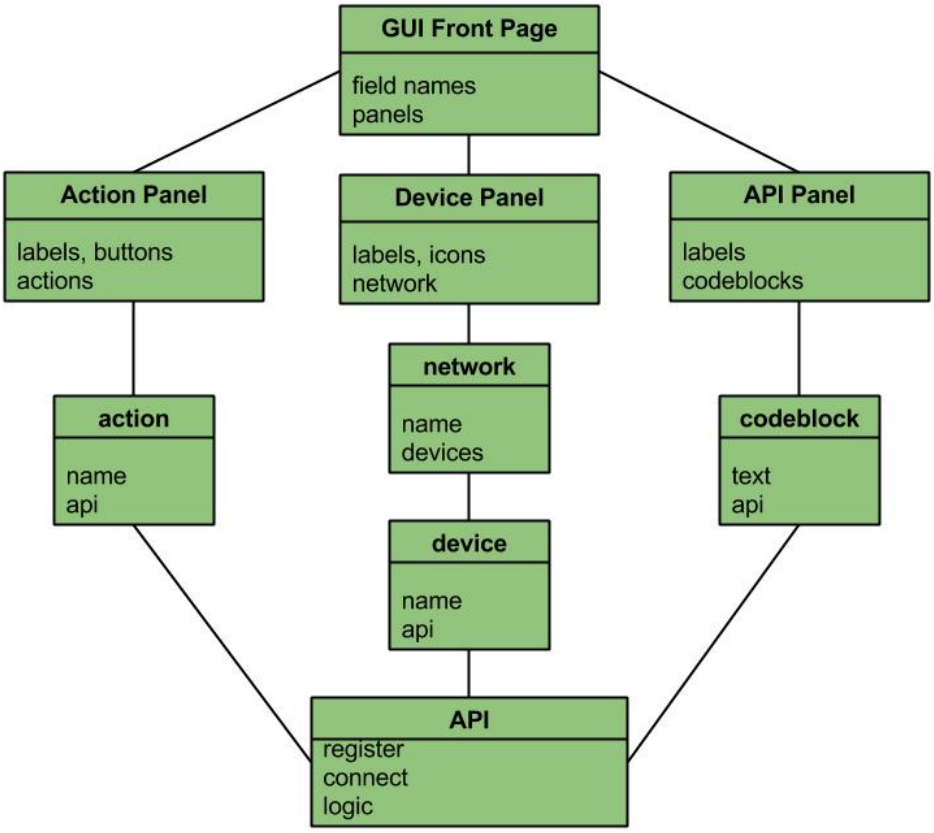
Idea 1	Web GUI (JavaScript), Web Core (Scala), MySQL Database
Idea 2	Mobile GUI (Android/iOS), Physical Core (Scala), MySQL Database
Idea 3	Mobile GUI (Android/iOS), Web Core (Scala), MySQL Database

The way the decision matrix works is we have a list of criteria on the left column which comprise engineering requirements of the system. We then give weights for each criteria and then extrapolate the criteria weights over percentages totalling 100. We then score each idea for each criteria on a scale of 1 - 10 where 1 means the idea will not work well with the criteria and 10 meaning the idea will work really well with the criteria. The idea with the best score is the optimal idea in the end. Our idea of implementing a Web GUI, Web Core, and MySQL Database turned out to be the most optimal solution according to the decision matrix and is the solution we plan to implement.

After the decision matrix is created and an implementation solution is chosen, we proceeded to created low level diagrams for each component of the system. These diagrams show how each component will be implemented and how the code will be

structured to implement the component. Below are the component architecture diagrams.

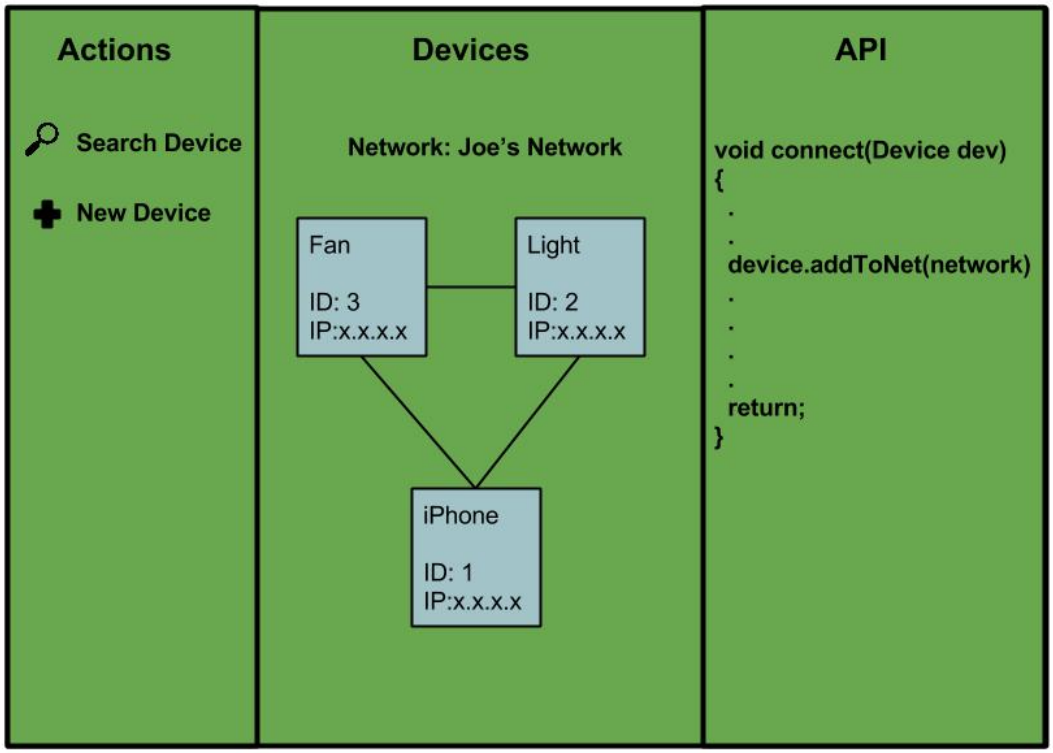
GUI Backend Architecture



The code for the GUI will be broken up into three main objects: an Action Panel, a Device Panel, and a API Panel. Each of the three main panel objects will consist of other objects as depicted in the diagram.

The figure below gives a graphical depiction of what the GUI will look like at the front end which will use the above components as the underlying infrastructure.

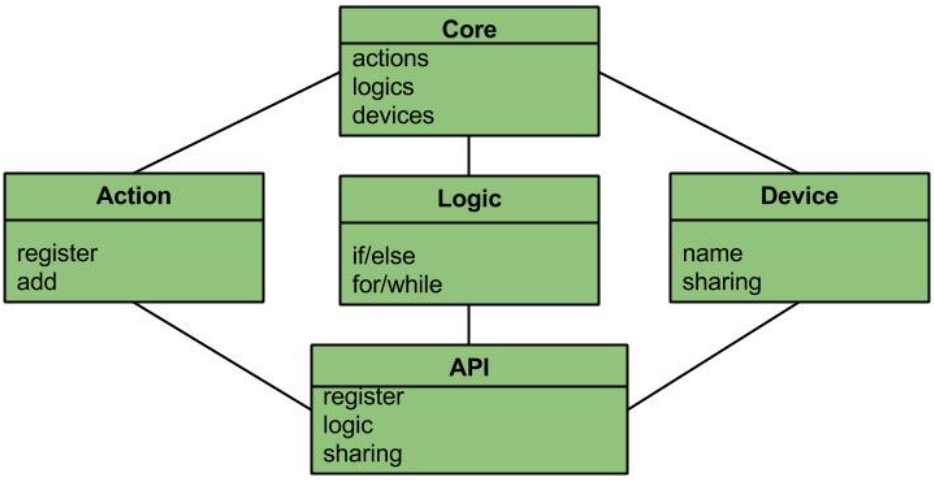
GUI Frontend Design



The GUI mimics the underlying code structure in that there is a Actions panel, a Devices panel, and an API panel. The Actions panel is where a user can search for existing devices in a network and add new devices to a network. The Devices panel shows a graphical depiction of the users current devices in the network. In this panel a user can create logical dependencies between devices as well as set sharing preferences of each device. The API panel is where a user can see the code that is generated from a device API depending on what action a user is doing on a device.

The following diagram depicts the code structure of how the core will be implemented. It has similar aspects as the GUI since it will be an interface between the GUI and devices.

Core Architecture



Now that we have our component designs and decisions in place we put together a course of action for implementing the product. We divided the development into three distinct phases as follows:

Phase 1
GUI and Devices Only

- Use the GUI to control a device (turn on a fan or light)
- Collect data from online sources and configure devices depending on online data

Phase 2
Core and API integration

- Create APIs for devices that allow for registration, sharing and logic.
- Create core platform to interface between GUI and devices
- Modify GUI to communicate with the core and use device APIs

Phase 3
Database integration and platform scalability

- Integrate the database into the system to store device information
- Create support for nonstatic number of devices on the platform

System Integration and Testing

Test Plan

Objective

This is the High Level Functional Verification Test Plan for ***Cloud1 Platform.*** The test plan outlines the following aspects of a Functional Verification Test:

- Use case coverage
- Assumptions
- Scope
- Test methodology
- Entry & exit criteria
- Exclusions
- Dependencies & responsibilities
- Schedule
- Risks management
- Status tracking mechanism
- Test resources
- Defect management for functional test activities

These aspects should be reviewed and approved by stake holders before the beginning of functional verification test execution.

Assumptions

The following assumptions are made for the functional verification test of this project:

- The boundary conditions are met by the users.
    - Maximum and minimum length of a field
        - Mostly in case of the server for the database reaching a maximum
    - Whether a value entered into a field is valid or invalid.
        - We expect GUI users to know the values they enter in for devices.
        - Once a device entered has a mistake the only way to change this will be to remove the device and recreate it. No editing single database values.
- The Web App will be used on a browser such as chrome that can use Javascript and php.
- Large scale data performance will behave the same way as our small test cases.

Functional Verification Test

Functional verification testing is a type of black box testing that uses the solution specifications, design document and use case document as a point of comparison to validate that the product is functioning properly. Functional scenario testing includes main flow test, exceptional flow test and exception handling test. FVT also needs to ensure the system is working properly and not leading to an undesired behaviour when an end user uses the features in a way not described in the use cases. FVT testing is not limited to positive flow tests described in use cases.

Functional Verification Test Scope

The tests are responsible for the validation of functional correctness for the following aspects of the scope:

- Functional scenario testing
    - Validate use case input and output
    - Validate exception handling
    - Validate error flows
    - Validate functional end-to-end scenarios (differs from system end-to-end scenarios)
- Access control testing
    - Positive access control testing
    - Negative access control testing
- Globalization Verification Testing
- Exploratory testing / Ad hoc testing
- Large scale data testing

Functional Scenario Testing

Functional scenario testing is a type test that verifies the use cases are functioning as described in the use case document. Functional scenario testing is the major part of FVT coverage and it includes main path testing, error handling testing and exceptional flow testing.

Each use case is validated to ensure the functionalities are working as intended. In addition to testing each use cases individually, FVT is also responsible for testing scenarios where multiple use cases are put together to validate functional end to end scenarios.

The following examples are the type of scenarios that need to be covered in your tests:

Main Flow

- User logs in to the website – verify that the user can successfully log in and see their list of devices, if he or she has any.

Exception Flow

- User attempts to log in with the incorrect password.

Error Flow

- User submits their registration profile without filling in all the mandatory fields.

Functional End to End Flow

- User registers from the website, logs in successfully, add devices to his profile, choose settings for the device, adds more devices if necessary, and connects the devices through wireless.

Access Control Testing

Access control testing is performed to ensure only the authorized roles can perform actions on authorized business objects. The web interface provided by the Cloud1 team uses coarse grained access control determined by access control policies.

The access control policies need to be tested for positive and negative access control. For positive access control testing, the functional verification test is responsible for ensuring

roles can perform actions on all allowed objects. For negative access control testing, the functional verification test is responsible for ensuring roles cannot perform actions on unauthorized objects.

Access control policies need to be tested through user interactions such as the main website to ensure users with web page access are not accessing the administrative tools.

Examples:

Positive Access Control

- Registered users can view his/her account information and list of devices he or she owns, are shared, or are public.

Negative Access Control

- Registered users cannot view another user's device information without his or her permission.

Negative Access Control

- Hit Web URLs that the user doesn't have access to

Roles	Actions on objects	What to validate
Registered User	*Adds a device*	*Validate the action that registered users can add a device*
Registered User	*Modify device information*	*Validate the action that registered users can modify device information*

Large Scale Data Testing

Large scale data function testing verifies that the system will continue to function correctly when a large number of business objects are present. Large scale data response time test should be executed by performance test in conjunction with this functional test.

Large scale data boundary is defined with configurable limits for business objects. If the data size boundaries are undefined, large scale data testing with various data sets can help to determine the boundaries.

The goals of large scale data testing is to ensure the system is functioning correctly and as expected in the following conditions:

- Reach the data size boundary for an object
- Exceed the data boundary value by at least one
- Extremely large data (much higher than typical data set size)

Examples of large scale data testing scenarios include:

- Viewing a list of objects and scrolling through the list
  - Large list of devices with 50 unique items.
- Creating a new object
  - When 1000 users are already registered, register a 1001st user.
- Updating an existing object
  - Update an item from a large list of devices.
- Deleting an existing object

    - Delete a device from a large list that has more than 1000 devices.
- Sorting objects
    - Sort a large list of devices by location near the user.
- Viewing/searching and refreshing
    - Perform a search that results in a very large result set and view the results and refresh the resulting page.
- Input logic on devices
    - Be able to insert logic into devices
- Exceeding a defined boundary condition.
    - Add a device where the maximum boundary for the devices per user is already reached.
- Data scaling
    - If the page limit is 10 devices, then when 100 devices are added, 10 pages are displayed with 10 devices in each page.

Globalization Verification Testing

The objective of Globalization Verification Test (GVT) is to prevent potential problems that could inhibit making the features globally available. GVT makes sure that the features can handle all international support without breaking functionality that may cause data loss or display issues. GVT ensures proper functionality of the features with any of the supported languages, cultures and locale settings using international inputs.

Performing GVT prior to doing translation increases the likelihood of having successful, on time translation tests with minimal globalization problems. GVT also reduces the cost and issues related to fixing errors after the application has been deployed.

Functional scenarios can be executed with GVT guidelines to verify globalization. They should be selected in such a way that all major flows and pages are tested. At the least, a test bucket containing the project's basic functionality must be run with GVT guidelines.

Globalization guidelines include the globalization checkpoints. Examples of globalization testing check points include:

Verify that double-byte characters can be entered into fields.

- Register a user with Korean inputs for User ID, First Name, Last name, and so on.

Verify that international numeric formats are followed.

- In French, commas are used instead of decimal points. For example, 124,34 to indicate 124.34 in US English)

Verify the language fall back works.

- A catalog entry is created with a German name only. When a user browses the website in French, the device's name should appear in German instead of displaying an error or blank.

GVT Check Point	Objects/Pages Covered	Areas to Validate	Remarks

Users from across the world should be able to add device typing their langauge	N/A	Device information	The website should accommodate these needs.

Exploratory Testing / Ad hoc Testing

Exploratory testing / ad hoc testing is a part of functional verification testing performed without planning or documentation. The tests are intended to be run only once and are performed to identify unexpected errors in the project. If an unexpected error is discovered, additional exploratory testing of that area should be performed. Testers are encouraged to achieve the exploratory test objective through various deviated paths.

What is NOT within the Functional Verification Test Scope

The following are NOT part of FVT scope:

- Boundary testing – covered by Unit Test
    - E.g. Large Strings
- Performance including response time for large scale data testing – covered by Performance Verification Test
    - E.g. 10000001st user can be registered within 3 seconds
- Accessibility testing

Test Methodology

High level test planning is done prior to formal detailed test planning and execution. As part of the high level test planning following are determined:

- FVT Criteria
    - FVT entrance criteria
    - FVT exit criteria
    - FVT Exception handling for entry and exit criteria
- Regression
    - Definition and test case selection guideline
    - When to execute
- Automation
    - Automation considerations
    - Automation tools

A detailed test case document is prepared based on the project specifications and the use case document. The test case document is reviewed and approved as part of the functional test preparation.

Prior to formal entry into the test execution phase, the Lab environment will be setup with appropriate operating systems, protocols, and any other pre-requisite software and hardware.

Entry and Exit Criteria

Entry Criteria

The development is based on Agile development model.

The following is the Sprint entry criteria for FVT:

- Functional verification test preparation can begin once:
    - Solution specifications are reviewed for the whole project or the sprint
    - Use case document is reviewed for the whole project or the sprint
- Functional verification test execution can begin once:
    - Functional verification high level test plan (this document) is reviewed and approved
    - Functional verification test case document for the sprint is available for review
    - Code is complete and has been unit tested for the area to be tested

Exit Criteria

The development is based on Agile development model.

The sprints are time boxed and end at the end of the duration. A sprint exit checklist is prepared with the following details:

- Status report

- Technical debt including the work remaining in PDs.

Exception Handling for Entry / Exit Criteria

A document is prepared by the test team lead with the exceptions and associated reason, action plan, and risks. The document is then reviewed by the solution owner/manager, development lead, test manager and project manager. The exceptions must be approved before officially claiming functional verification test entry or exit.

Regression Testing

The purpose of a regression test is to ensure that as code is checked in and defects are fixed, existing code is not regressed.

The regression test buckets are grouped into 2 categories:

1) Sanity bucket (approximately 10% of the test cases)
   The goal of this test bucket is to verify that the major functionalities are working correctly. A small subset of test cases is selected to ensure the critical paths are tested quickly.
2) Full regression bucket (approximately 30% of the test cases)
   The goal of this bucket is to verify the majority of functional flows are working correctly. The regression bucket selection also depends on the areas of code changes made and the importance of functionality.

Test Automation

Once test cases (generally the full regression bucket) have been selected for automation, they are grouped together in a batch and run. Automating your test cases will reduce the effort required to perform them in the future.

Automation Considerations

The following aspects are to be considered when automating your test cases:

- Ensure the test logic is kept separate from the test data. For example, input data should not be incorporated into the method that automates a test case, but rather stored in and referenced from a separate data file.
- Test scripts can be run one after the other. Therefore, the test cases must clean up the test environment prior to termination. If information from the previous run remains, the same test case may encounter an issue if it is run successively.
- When possible, automated test cases should be written so that their results are independent of one another. Otherwise, if test cases are interrelated, a test case may fail simply because a related test case failed. For example, an account is created in test case 1 and used in test case 2. If test case 1 fails then test case 2 will also fail, even if there are no problems with the flows in test case 2.

Test Tool

Full regression bucket (Version 1) is used for automating test cases. This automation tool is selected for automation for the following reasons:

- To verify that the majority of functional flows are working correctly.

- Depends on the importance of functionality.

Server Environment

The team is using the Bitnami LAMP stack with Amazon Web Services (AWS) in order to host a cloud server. Both the web server and database will be running on an EC2 instance that is hosted using AWS. The Bitnami LAMP stack provides the software package that makes it easier to deploy, manage, and monitor this instance. The EC2 instance will running on Ubuntu with a MySQL database server.

Software	Version Supported	Remarks
Cloud Server		AWS
Database	5.5.40 - MySQL Community Server (GPL)	
Operating System	Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-39-generic x86_64)	
Management Package/Tools		Bitnami LAMP stack

Client Environment

The main client environment will be a web browser. This is the only means of reaching the web app and using the platform. As a result, the majority of browser types/versions should be properly tested and supported.

Software	Version Supported	Version Tested	Percentage Covered	Remarks
IE	IE 8 on Windows XP/7	Not Tested	0%	
	IE 11 on Windows 7/8.x	Not Tested	0%	
Firefox	Firefox 3.0, 3.5	Not Tested	0%	
Safari	Safari 4.0 on Mac	Not Tested	0%	
Chrome	Chrome 40.x	40.0.2214.111 m (64-bit)	100%	May need testing for Chrome on Android

Exclusions

This test plan only covers user interaction within the web app. Registration, login, and user-device management should all work accordingly from a web browser. However, device connection and hardware compatibility are outside the scope of these tests. Users will be able to manually add their own devices via certain identifiers (e.g. IP address and device type), but the platform cannot automatically detect the connection of an IoT device. However, there are test cases that can verify if a device type has an existing API that is manageable using the platform. Therefore, it is up to the user to know the specifications and validity of their devices when using the platform.

Dependencies

Following dependencies must be met on time for successful FVT.

Dependency	By (date)	By Whom	Risk if Not Met
Unit test completion	03/06/2015	All developers	Unit test defects may be caught by FVT and takes more turn around time to fix.
Frequent build images	02/16/2015	Build team	FVT cannot execute test cases without build images
Large scale data Test data	02/23/2015	System test	Large scale test data must be available for large scale data testing
Quick turn around on blocking defects	Fixes within 24 hours	Defect owning developer	Blocking defects delay the FVT execution

Responsibilities

Role	Responsibilities	Remarks
Project Manager	Maintaining contact with clients and ensuring them the project is remaining on schedule for completion. Facilitating team meetings and reminding all team members of the tasks that they need to have done and when.	The project manager should be the overarching facilitator of the project. The project manager ensures each team member has what they need to finish their tasks.
Solution Architect	Make sure all development environments are up and running and that the project infrastructure is well	Our environment consists of GitHub account, AWS server, and Web server all working together.

	maintained and that all components are well organized as they start working together.	
Developers	Complete development of the Cloud1 platform software including the client, core, and database. Other tasks in development include device APIs so that we can test the Cloud1 functionality adequately.	Developers are responsible for completing all software implementation for the Cloud1 platform as well as a few APIs for devices so that we can test Cloud1 functionality.
FVT Lead	Ensures test schedule is adheared to and that test goals are met. Also makes sure that all areas of the product have testing procedures planned out and are executed	The FVT lead should make sure all of the testing procedures are done on time and that the product is deliverable by the deadline.
FVT Testers	Responsible for carrying out the test procedures and meeting the scheduled test deadlines.	Testers are to carry out the testing tasks that the test lead has prescribed.
Build Team	Completing deployment of the Cloud1 Client, Core, and deployment. Completing integration of Cloud1 platform with OpenIoT infrastructure.	Our software (Cloud1) needs to integrate successfully with OpenIoT software by the end of the quarter for a successful project completion.

Schedule

This section of the FVT Plan outlines the testing schedule week by week and describes the testing tasks to be done each week.

Week 7	Web client and Database: Ensure login feature works properly and users get added correctly to the database. Ensure adding of devices works properly and that device information gets stored in the database appropriately
Week 8	Web Client and Core: Ensure that web client is interacting with the core and sending correct data and requests to the core. Ensure that the Core is handling device registration and interaction appropriately and that the core can communicate with the registered devices and send and receive information

	correctly with the devices.
Week 9	Cloud1 and OpenIoT: Ensure that the cloud1 platform integrates properly with the OpenIoT environment. This means being able to register devices, contact devices, and gather data from devices through OpenIoT. OpenIoT will be running on the same AWS as Cloud1 so both environments need to work together seamlessly.
Week 10	All components: Ensure that all components of the Cloud1 platorm are working together properly and that the OpenIoT environment is facilitating conection between devices and Cloud1 appropriately. This is the final testing stage so all components must be working properly and users should be able to log in, register a device, and start viewing data from devices and managing their devices.

Risk Management

Risks

The following risks have been identified in this project:

- Functional verification testing assumes the following tests are covered by the development team:
  - Web client testing
  - Core testing
  - If these areas are not properly unit tested, they could potentially be missed altogether.
- Supported browsers – The tested browsers are a subset of the supported browsers and therefore potential problems may exist with the browsers that have not been tested. For example the Web client may not work properly on Safari
- OpenIoT environment - OpenIoT is not guaranteed to work well with our Cloud1 platform. We must provide adequate time for testing this connection and encounter problems as early as possible to mitigate them.
- Supported devices - The tested devices are a subset of the supported devices and therefore potential problems may exist with the devices that have not been tested.

Risk Tracking and Management

Risk management meetings are scheduled once in a week and lead by the FVT lead. The following issues are tracked from meeting-to-meeting:

- OpenIoT integration - mitigating build issues, integration issues, and functional issues
- Core development - any issues that arise with core development

- Web Client - user login and device registration
- Database - properly storing user and device information

Deviations

In order to manage project deviations, team members will fill out a deviation request specifying what the deviation is and the reason for it. The FVT lead will then review the deviation request and decide whether or not the schedule permits the deviation.

Deviation Request Template

Deviation Request: <Specify a title for the deviation>
What is deviated: <Give details about the deviation, ie what is being deviated>
The reason for the deviation: <Give valid reason for deviation>
Risk Assessment on the deviation: <Potential risks that could result from the deviation>
Justification for the deviation: <Give adequate reason for the deviation>
Deviation Reviewers: <FVT lead>
Deviation Approver: <FVT lead>

Test Tracking

Tracking Tools

To track our testing, we will be using the JIRA issue tracking software. This system will contain all the bugs we have found as well as the tasks scheduled for each member. When a member completes a testing task he logs on to JIRA and indicates he performed the task. We will also keep a simple Google Doc with a testing log at:

https://docs.google.com/document/d/1as9cW1jnulzvYx4g7USNGdY5V8xrZK3S9BQAdGt3m2c/edit?usp=sharing

Status Reporting

As mentioned above the JIRA software will be used for status reporting. Each member will be given testing tasks they must complete. When the task is completed, the team member marks the task as completed and then the task will be taken off the to-do list, however a log of completed tasks still stores the task info.

Test Resources

Hardware

The following machines were used for testing:

Hardware	Located	Hardware Specification	Used For	Contact Person

Corey's PC	Corey's house	FX6300 hex-core, 1 TB hard drive, 8 GB RAM	All browser testing	Corey
Linux Lab	CSL	Unknown	All browser testing	N/A

Software

Software	Version	License Type	Where to download	Contact Person
Internet Explorer	10, 11	Free	preinstalled	Microsoft
Firefox	latest	Free	mozilla.com	Mozilla
Chrome	latest	Free	google.com	Google
JIRA	latest	Paid	web app	Atlassian

Staffing

Archit Mendiratta

Email: amendira@calpoly.edu

Tam Nguyen

Email: tnguy173@calpoly.edu

Max Parelius

Email: mpareliu@calpoly.edu

Etienne Urban-Racine

Email: eurbainr@calpoly.edu

Corey Wilson

Email: cwilso@calpoly.edu

Defect Management

Defect

A defect in our project is an unintended behavior in the software. This can be something as simple as a graphical glitch or something as severe as a database failure. If the

defect negatively impacts our software and hinders the customer's experience, we must fix it immediately.

Defect Severity

Severity 1:

These issues are the top priority for fixes. These bugs generally cause or product to fail. This can be devices not communicating, database failure, etc… The team will come together and decide a course of action on a per case basis.

Severity 2:

All other bugs will be considered severity 2. These bugs will not cause our software to fail, such as a graphical issue between browsers. These defects will be added to a list and will be fixed in a timely manner.

Defect Reporting System

All bugs must be submitted in a report containing the following information:

- Name + Description
- Date + Time occurred
- Steps to reproduce the bug
- Version of code

With this information, we should be able to track and fix any defects.

Defect Reporting Data

We will be using the JIRA issue and project tracking software. It is a useful tool the will store the defect data and help us track the defects. Using JIRA, creating defect reports and assigning them to a team member to fix are a simple task.

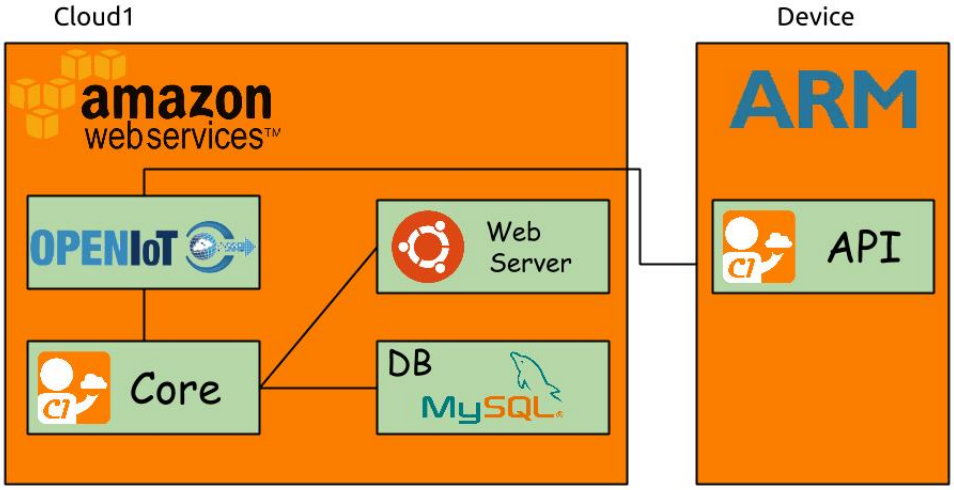
Defect Validation

Once the code for a bug has been fixed, the tester will go back to the original bug listing and follow the steps listed to reproduce the bug. If after a few tries the bug does not appear, the bug can be considered validated.

Conclusion

The figures below show the final schematic of Cloud1. The web server is where all of the user and device registration happens as well as device management. The devices have REST APIs running on them to allow for interfacing and communication with Cloud1. Cloud1 is running on an AWS that contains the database, the web server, and OpenIoT. Our results consisted of the REST API interfacing with the web server without the help of OpenIoT. Our plan was to implement OpenIoT to do all of the device connections but what we got was the

devices hosting a similar API that OpenIoT uses (REST). Transitioning from the REST API on the devices to using OpenIoT is an easy transition because the web server just needs a REST API to interface with it does not care who is hosting.



References

http://www.smartthings.com/index.php
https://www.aylanetworks.com/
https://evrythng.com/
http://www.webofthings.org/projects/
https://www.connect2.me/
http://revolv.com/devices/

Appendices

Bill of materials

1xSpark microprocessor antenna

Line Items:
1x Spark Core for $39.00 each
Core (chip antenna)

Subtotal: $39.00
Taxes: $2.93
Shipping: $6.91
Total: $48.84

AWS one of which included a T2.Small for open IoT

February $16.59

March $15.41

Gantt Chart

Title	%	Duration	Start Date	End Date
Alpha Prototype	0	31 days	11-06-2014	12-06-2014
HTML and CSS document creation	0	4 days	11-07-2014	11-10-2014
Javascript Implementation	0	8 days	11-07-2014	11-14-2014
Connecting Device to the Internet	0	12 days	11-12-2014	11-23-2014
Turning on/off a device from input	0	7 days	11-17-2014	11-23-2014
Give GUI output through button	0	8 days	11-24-2014	12-01-2014
Connect device to output	0	5 days	12-01-2014	12-05-2014

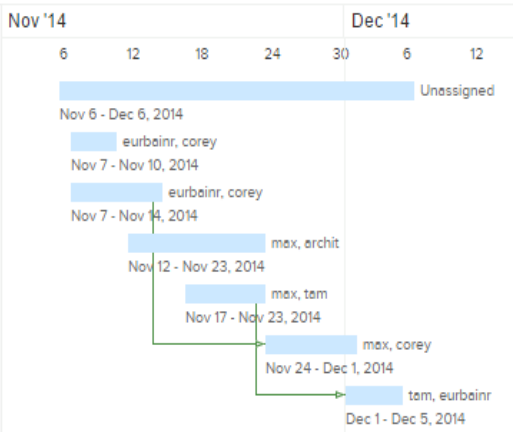


Chart One: Fall 2014

Open IoT Implementation	0	68 days	01-04-2015	03-13-2015
login screen	0	21 days	01-05-2015	01-25-2015
Set up a web server and allow communication	0	67 days	01-05-2015	03-13-2015
platform architecture	0	42 days	01-05-2015	02-15-2015
Create the platform base with Scala.	0	14 days	01-05-2015	01-18-2015
Create Database with MySQL to store new devices and	0	53 days	01-19-2015	03-13-2015
Link Database with GUI Device creation	0	25 days	02-16-2015	03-13-2015
User Creation	0	21 days	01-26-2015	02-15-2015