Smart Home Energy Conservation System

Graduation Project

2017

Supervisors

Dr. Mohamed Badawy

Dr. Salah Eldin Shaban

Acknowledgement

First and foremost we would like to thank

Dr. Mohamed Badawy

Dr. Salah Eldin Shaban

For their support, outstanding guidance and encouragement throughout our senior project. We would like to thank our family, especially our parents, for their encouragement, patience and assistance over the years. We are forever indebted to our parents, who have always kept us in their prayers.

watt?

Teamwork

Ahmed Abdeltwab Elzoughby

Basant Essam

Abdelraouf Sabry

Mohammed Ibrahim Bazazo

Ahmed Maher Tawfik

Mahmoud Omar

Contents

Chapter 1

Introduction

Abstraction

More more embedded devices and sensors are connected to each other and to the Internet. Then the devices can be remotely controllable and their data accessible any where using any kind of devices the users might have. This is the Internet of Things.

Watt? is a simple implementation of the the Internet of Things. It features these smart devices all equipped with high current sensors connected with an embedded devices. The devices are connected to a cloud server where it streams the data along with storing it locally in the device. From the cloud server you can access the devices data from any Android mobile device.

Watt? is a system to measure the power consumption, calculate the energy consumption and control some appliances of the home remotely. We are trying to smarten up some existing physical things like electricity meters and electricity plugs by connecting them wireless to the *Internet* and this is the Internet of Things.

The three major components of our product can be described as the following:

- **Watt? Smart Meter**

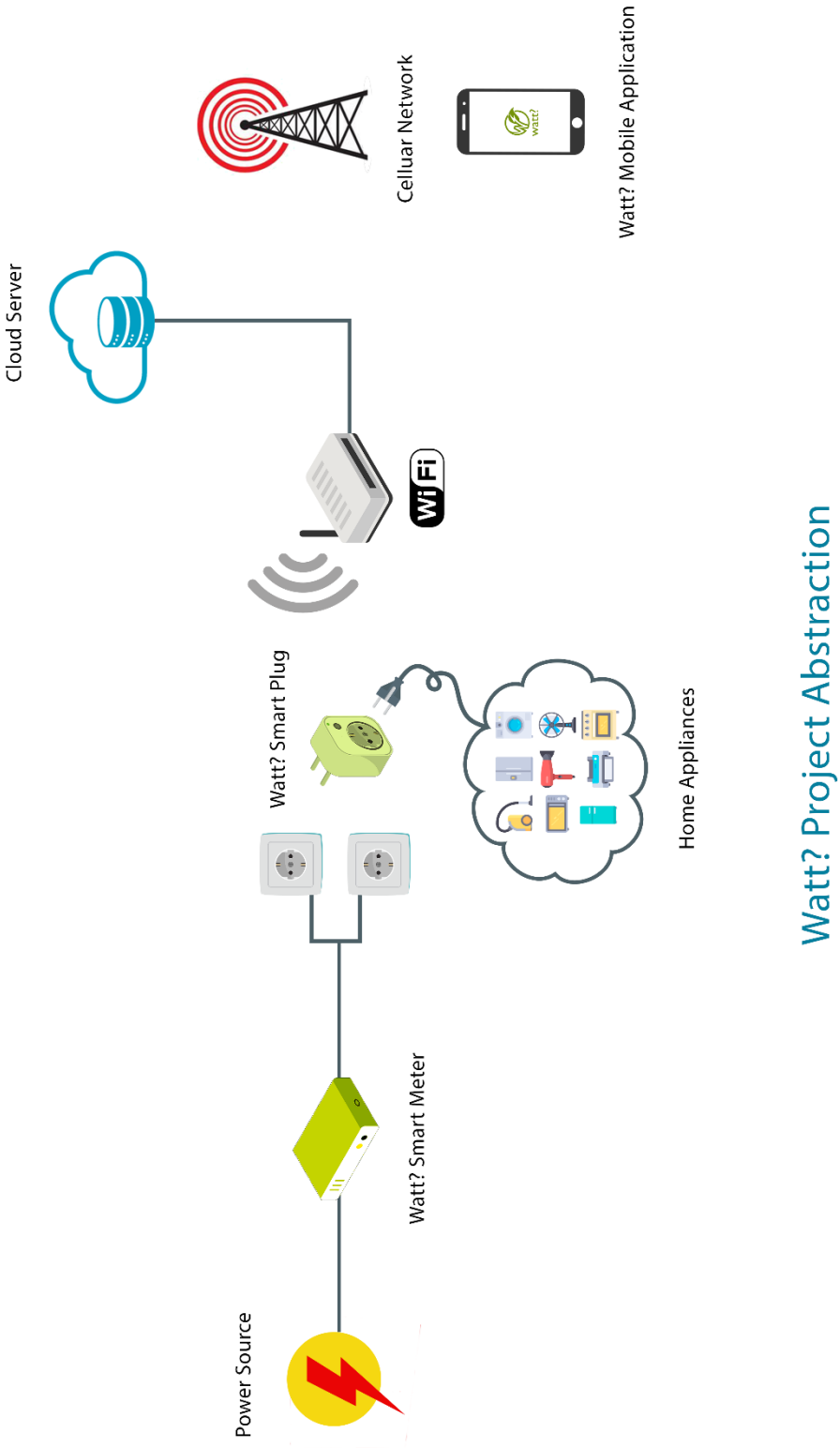
  An Internet connected electricity meter to measure the overall power consumption and the cumulative energy consumption of the home. Then to send that data to a back-end cloud server through the Internet.

- **Watt? Smart Plug**

  An external plug to measure the realtime power consumption of the plugged appliance and to make it controllable remotely through an Internet connected mobile device.

- **Watt? Mobile Application**

  The main Interface for the users to deal with all the system features. It makes them able to monitor their power and energy consumptions, control the plugged appliances and expect the bill cost. all of that can be accessed remotely any where.


Watt? Project Abstraction
Cloud Server
Celluar Network
Watt? Mobile Application
Wi Fi
Watt? Smart Plug
Home Appliances
Watt? Smart Meter
Power Source

Internet of Things

The closest to an official definition of the Internet of Things comes from the Internet of Things Global Standards Initiative: The Internet of Things (IoT) is the network of physical objects or “things” embedded with electronics, software, sensors, and network connectivity, which enables these objects to collect and exchange data. That’s a high-level definition, and the IoT goes by other names as well.

Today the Internet has become ubiquitous, has touched almost every corner of the globe, and is affecting human life in unimaginable ways. However, the journey is far from over. We are now entering an era of even more pervasive connectivity where a very wide variety of appliances will be connected to the web. We are entering an era of the “Internet of Things”. The Internet of Things refers to a new kind of world where almost all the devices and appliances that we use are connected to a network. We can use them collaboratively to achieve complex tasks that require a high degree of intelligence.

For this intelligence and interconnection, IoT devices are equipped with embedded sensors, actuators, processors, and transceivers. IoT is not a single technology; rather it is an agglomeration of various technologies that work together in tandem.

Sensors and actuators are devices, which help in interacting with the physical environment. The data collected by the sensors has to be stored and processed intelligently in order to derive useful inferences from it.

The storage and processing of data can be done on the edge of the network itself or in a remote server. If any preprocessing of data is possible, then it is typically done at either the sensor or some other proximate device. The processed data is then typically sent to a remote server. The storage and processing capabilities of an IoT object are also restricted by the resources available, which are often very constrained due to limitations of size, energy, power, and computational capability. As a result the main research challenge is to ensure that we get the right kind of data at the desired level of accuracy. Along with the challenges of data collection, and handling, there are challenges in communication as well. The communication between IoT devices is mainly wireless because they are generally installed at geographically dispersed locations. The wireless channels often have high rates of distortion and are unreliable. In this scenario reliably communicating data without too many retransmissions is an important problem and thus communication technologies are integral to the study of IoT devices.

Now, after processing the received data, some action needs to be taken on the basis of the derived inferences. The nature of actions can be diverse. We can directly modify the physical world through actuators. Or we may do something virtually. For example, we can send some information to other smart things.

The process of effecting a change in the physical world is often dependent on its state at that point of time. This is called context awareness. Each action is taken keeping in consideration the context because an application can behave differently in different contexts. For example, a person may not like messages from his office to interrupt him when he is on vacation.

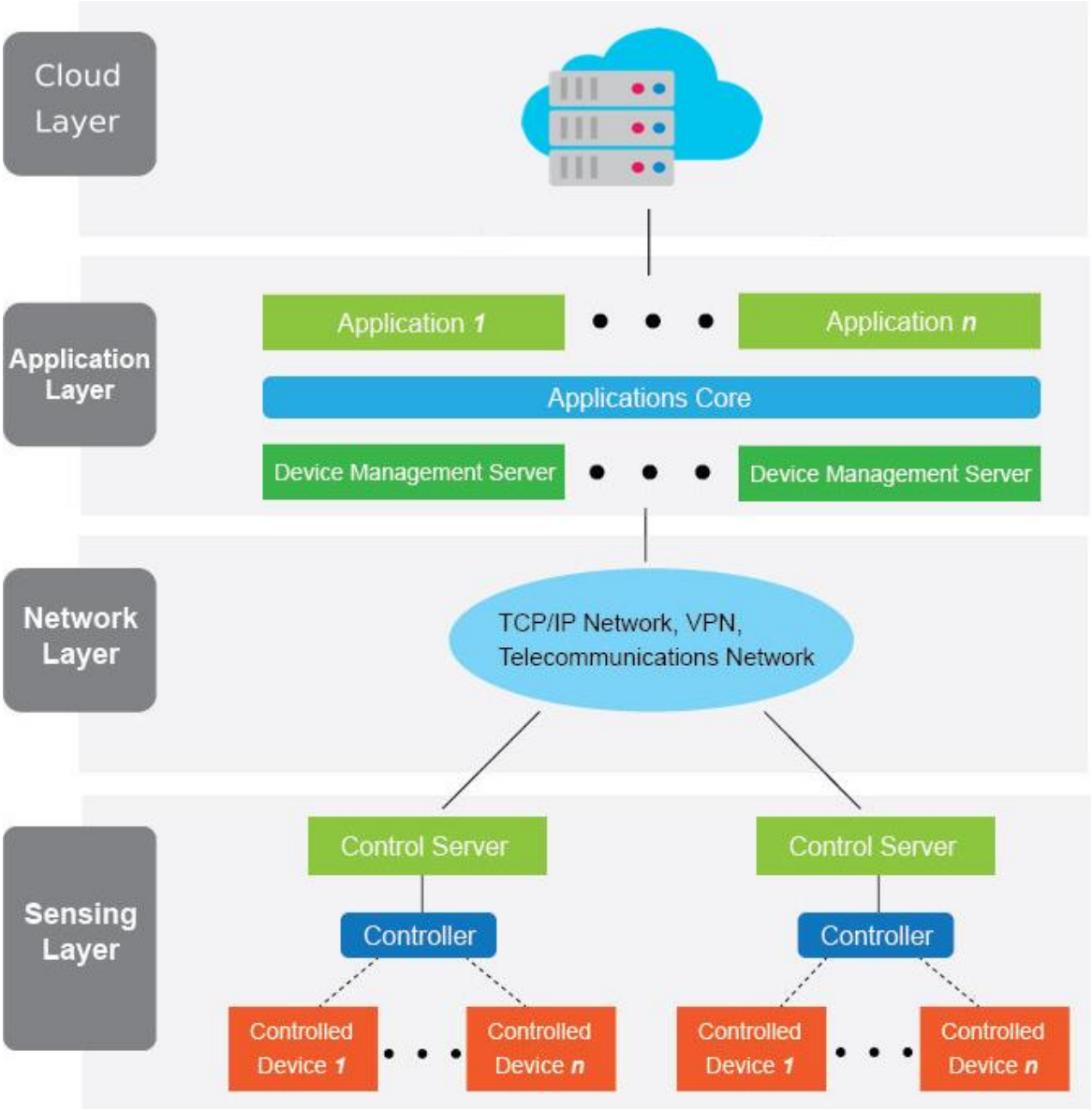
Sensors, actuators, compute servers, and the communication network form the core infrastructure of an IoT framework. However, there are many software aspects that need to be considered. First, We need a lot of standardization to connect many different devices.

The Internet of Things finds various applications in health care, fitness, education, entertainment, social life, energy conservation, environment monitoring, home automation, and transport systems. We see that, in all these application areas, IoT technologies have significantly been able to reduce human effort and improve the quality of life. [18]

Advantages of IOT

- **Communication** IoT encourages the communication between devices, also famously known as Machine-to-Machine (M2M) communication. Because of this, the physical devices are able to stay connected and hence the total transparency is available with lesser inefficiencies and greater quality.
- **Automation and Control** Due to physical objects getting connected and controlled digitally and centrally with wireless infrastructure, there is a large amount of automation and control in the workings. Without human intervention, the machines are able to communicate with each other leading to faster and timely output.
- **Information** it is obvious that having more information helps making better decisions. Whether it is mundane decisions as needing to know what to buy at the grocery store or if your company has enough widgets and supplies, knowledge is power and more knowledge is better.
- **Monitor** The second most obvious advantage of IoT is monitoring. Knowing the exact quantity of supplies or the air quality in your home, can further provide more information that could not have previously been collected easily. For instance, knowing that you are low on milk or printer ink could save you another trip to the store in the near future. Furthermore, monitoring the expiration of products can and will improve safety.
- **Time** As hinted in the previous examples, the amount of time saved because of IoT could be quite large. And in today's modern life, we all could use more time.
- **Money** The biggest advantage of IoT is saving money. If the price of the tagging and monitoring equipment is less than the amount of money saved, then the Internet of Things will be very widely adopted. IoT fundamentally proves to be very helpful to people in their daily routines by making the appliances communicate to each other in an effective manner thereby saving and conserving energy and cost. Allowing the data to be communicated and shared between devices and then translating it into our required way, it makes our systems efficient.

- **Automation of daily tasks leads to better monitoring of devices** The IoT allows you to automate and control the tasks that are done on a daily basis, avoiding human intervention. Machine-to-machine communication helps to maintain transparency in the processes. It also leads to uniformity in the tasks. It can also maintain the quality of service. We can also take necessary action in case of emergencies.
- **Efficient and Saves Time** The machine-to-machine interaction provides better efficiency, hence; accurate results can be obtained fast. This results in saving valuable time. Instead of repeating the same tasks every day, it enables people to do other creative jobs.
- **Saves Money** Optimum utilization of energy and resources can be achieved by adopting this technology and keeping the devices under surveillance. We can be alerted in case of possible bottlenecks, breakdowns, and damages to the system. Hence, we can save money by using this technology.
- **Better Quality of Life** All the applications of this technology culminate in increased comfort, convenience, and better management, thereby improving the quality of life.[1]



Internet of Things Architecture

The layered architecture for Internet Of Things in Watt? Project depends on four layers as the traditional layered architecture. This architecture is illustrated in this figure:

- **Cloud layer**

  IoT systems frequently involve the use of cloud computing platforms. Cloud computing platforms offer the potential to use large amounts of resources, both in terms of the storage of data and also in the ability to bring flexible and scalable processing resources to the analysis of data. Cloud computing is given primacy because it provides great flexibility and scalability. It offers services such as the core infrastructure, platform, software, and storage. Developers can provide their storage tools, software tools, data mining, and machine learning tools, and visualization tools through the cloud.

- **Application layer**

  includes the IoT application. This layer is responsible for delivery of various applications to different users in IoT. The applications can be from different industry segments such as: manufacturing, logistics, retail, environment, public safety, health-care, food and drug etc.

- **Network layer**

  performs the following functions; Gateway – Routing & Addressing – Network Capabilities – Transport Capabilities – Error detection & Correction. Also, it takes care of message routing, publishing and subscribing. With demand needed to serve a wider range of IOT services and applications such as high speed transactional services, context-aware applications, etc, multiple networks with various technologies and access protocols are needed to work with each other in a heterogeneous configuration. These networks can be in the form of a private, public or hybrid models and are built to support the communication requirements for latency, bandwidth or security.

- **Hardware layer**

  includes sensors, other hardware such as; embedded systems, RFID tags and readers and others. The sensors enable the interconnection of the physical and digital worlds allowing real-time information to be collected and processed. The miniaturization of hardware has enabled powerful sensors to be produced in much smaller forms which are integrated into objects in the physical world. There are various types of sensors for different purposes. The sensors have the capacity to take measurements such as temperature, air quality, movement and electricity. In some cases, they may also have a degree of memory, enabling them to record a certain number of measurements. A sensor can measure the physical property and convert it into signal that can be understood by an instrument. Sensors are

grouped according to their unique purpose such as environmental sensors, body sensors, home appliance sensors and vehicle telemetric sensors, etc. Many of these hardware elements provide identification and information storage (e.g. RFID tags), information collection (e.g. sensors), and information processing (e.g. embedded edge processors). [19]

In the next chapters, We will discuss in details the development fields of Watt? Project from the perspective of the Internet Of Things architecture layers.

Chapter 2

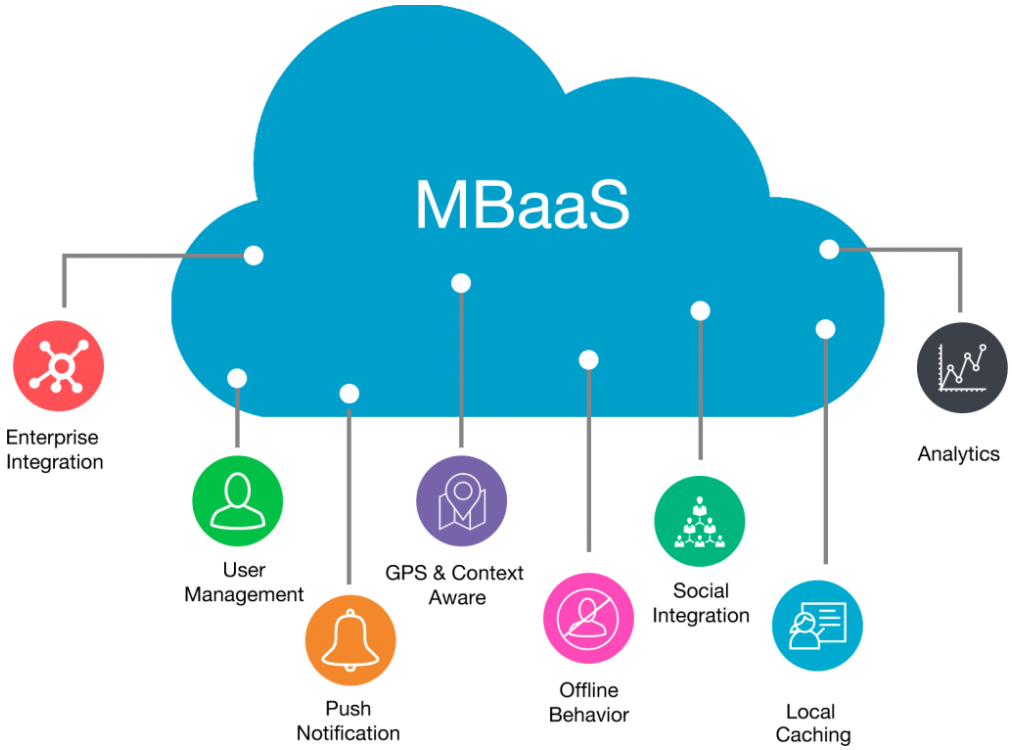
Firebase

The backend, The Firebase

There are two divergent trends happening in mobile development right now. The most common one is the mobile-first approach strategy. You construct a landing page website and then build a product in iOS, if targeting the U.S. market, or Android, if targeting the European market. Then, you push a single version and develop for other markets down the line.

The other trend is the API-first approach, in which the underlying construct — the application programming interface (API) — is built first. This strategy allows the website and apps on various platforms to be built on top of the same basic conditions. If your sole target audience is iOS users, perhaps a mobile-first strategy works for you. However, the downside is this hinders quick development for future audiences, including Windows, BlackBerry, Android, and a web-app for your non-mobile users.

The API-first solution allows app developers to quickly reach subscribers on many different devices. With this strategy, you can build, deploy and manage the whole mobile lifecycle from one source using an API Backend as a Service (BaaS). Perhaps most importantly, BaaS allows you to save time and money while scaling your business rapidly. And, according to Markets and Markets market research firm, BaaS will be a $7.7 billion market by 2017, so we see BaaS as a safe bet.[5]



NoSQL Database

NoSQL encompasses a wide variety of different database technologies that were developed in response to the demands presented in building modern applications:

- Developers are working with applications that create massive volumes of new, rapidly changing data types — structured, semi-structured, unstructured and polymorphic data.
- Long gone is the twelve-to-eighteen month waterfall development cycle. Now small teams work in agile sprints, iterating quickly and pushing code every week or two, some even multiple times every day.
- Applications that once served a finite audience are now delivered as services that must be always-on, accessible from many different devices and scaled globally to millions of users.
- Organizations are now turning to scale-out architectures using open source software, commodity servers and cloud computing instead of large monolithic servers and storage infrastructure.

Relational databases were not designed to cope with the scale and agility challenges that face modern applications, nor were they built to take advantage of the commodity storage and processing power available today.

NoSQL Database Types

- **Document databases** pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.
- **Graph stores** are used to store information about networks of data, such as social connections. Graph stores include Neo4J and Giraph.
- **Key-value stores** are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or 'key'), together with its value. Examples of key-value stores are Riak and Berkeley DB. Some key-value stores, such as Redis, allow each value to have a type, such as 'integer', which adds functionality.
- **Wide-column stores** such as Cassandra and HBase are optimized for queries over large datasets, and store columns of data together, instead of rows.[6]

Benefits of NoSQL

NoSQL databases offer enterprises important advantages over traditional RDBMS, including:

- **Scalability**: NoSQL databases use a horizontal scale-out methodology that makes it easy to add or reduce capacity quickly and non-disruptively with commodity hardware. This eliminates the tremendous cost and complexity of manual sharding that is necessary when attempting to scale RDBMS. Performance: By simply adding commodity resources, enterprises can increase performance with NoSQL databases. This enables organizations to continue to deliver reliably fast user experiences with a predictable return on investment for adding resources—again, without the overhead associated with manual sharding.

- **High Availability**: NoSQL databases are generally designed to ensure high availability and avoid the complexity that comes with a typical RDBMS architecture that relies on primary and secondary nodes. Some "distributed" NoSQL databases use a masterless architecture that automatically distributes data equally among multiple resources so that the application remains available for both read and write operations even when one node fails.
- **Global Availability**: By automatically replicating data across multiple servers, data centers, or cloud resources, distributed NoSQL databases can minimize latency and ensure a consistent application experience wherever users are located. An added benefit is a significantly reduced database management burden from manual RDBMS configuration, freeing operations teams to focus on other business priorities.
- **Flexible Data Modeling**: NoSQL offers the ability to implement flexible and fluid data models. Application developers can leverage the data types and query options that are the most natural fit to the specific application use case rather than those that fit the database schema. The result is a simpler interaction between the application and the database and faster, more agile development.

Firebase

Realtime Database

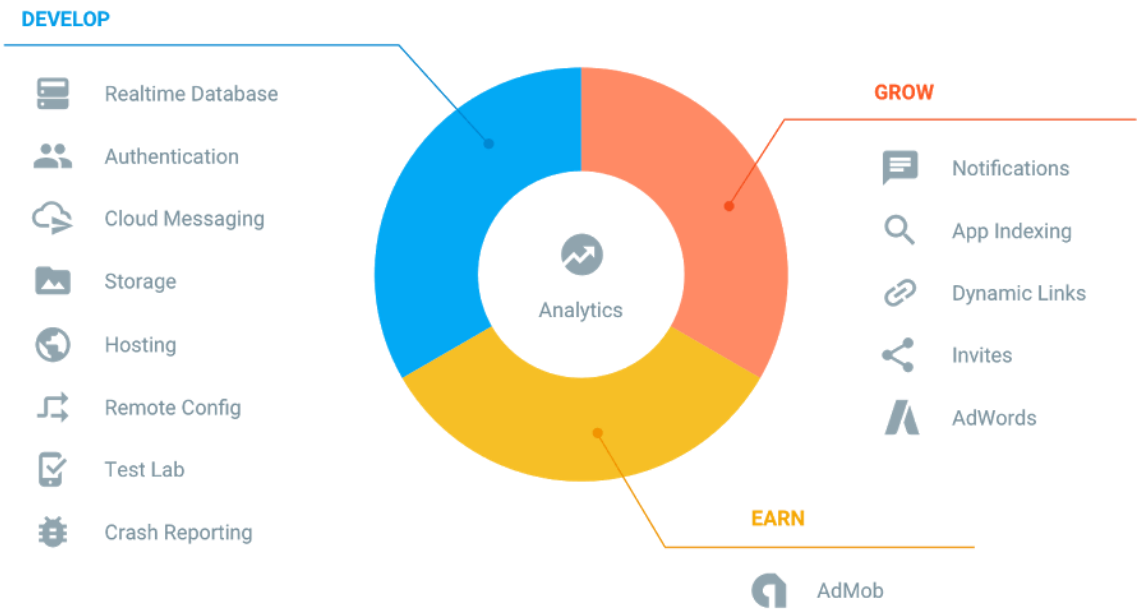
The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client. When you build cross-platform apps with our iOS, Android, and JavaScript SDKs, all of your clients share one Realtime Database instance and automatically receive updates with the newest data.

How it works?

The Firebase Realtime Database lets you build rich, collaborative applications by allowing secure access to the database directly from client-side code. Data is persisted locally, and even while offline, realtime events continue to fire, giving the end user a responsive experience. When the device regains connection, the Realtime Database synchronizes the local data changes with the remote updates that occurred while the client was offline, merging any conflicts automatically.

The Realtime Database provides a flexible, expression-based rules language, called Firebase Realtime Database Security Rules, to define how your data should be structured and when data can be read from or written to. When integrated with Firebase Authentication, developers can define who has access to what data, and how they can access it.

The Realtime Database is a NoSQL database and as such has different optimizations and functionality compared to a relational database. The Realtime Database API is designed to only allow operations that can be executed quickly. This enables you to build a great realtime experience that can serve millions of users without compromising on responsiveness. Because of this, it is important to think about how users need to access your data and then structure it accordingly.



How to implement

1. Integrate the Firebase Realtime Database SDKs: Quickly include clients via Gradle, CocoaPods, or a script include.
2. Create Realtime Database References: Reference your JSON data, such as "users/user:1234/phone_number" to set data or subscribe to data changes.
3. Set Data and Listen for Changes: Use these references to write data or subscribe to changes.
4. Enable Offline Persistence: Allow data to be written to the device's local disk so it can be available while offline.
5. Secure your data: Use Firebase Realtime Database Security Rules to secure your data.

Authentication

Most apps need to know the identity of a user. Knowing a user's identity allows an app to securely save user data in the cloud and provide the same personalized experience across all of the user's devices. Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to your app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more.

Firebase Authentication integrates tightly with other Firebase services, and it leverages industry standards like OAuth 2.0 and OpenID Connect, so it can be easily integrated with your custom backend.

How it works?

To sign a user into your app, you first get authentication credentials from the user. These credentials can be the user's email address and password, or an OAuth token from a federated

identity provider. Then, you pass these credentials to the Firebase Authentication SDK. Our backend services will then verify those credentials and return a response to the client.

After a successful sign in, you can access the user's basic profile information, and you can control the user's access to data stored in other Firebase products. You can also use the provided authentication token to verify the identity of users in your own backend services.

Storage

Cloud Storage for Firebase is a powerful, simple, and cost-effective object storage service built for Google scale. The Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for your Firebase apps, regardless of network quality. You can use our SDKs to store images, audio, video, or other user-generated content. On the server, you can use Google Cloud Storage, to access the same files.

How it works?

Developers use the Firebase SDKs for Cloud Storage to upload and download files directly from clients. If the network connection is poor, the client is able to retry the operation right where it left off, saving your users time and bandwidth.

Cloud Storage stores your files in a Google Cloud Storage bucket, making them accessible through both Firebase and Google Cloud. This allows you the flexibility to upload and download files from mobile clients via the Firebase SDKs, and do server-side processing such as image filtering or video transcoding using Google Cloud Platform. Cloud Storage scales automatically, meaning that there's no need to migrate to any other provider. Learn more about all the benefits of our integration with Google Cloud Platform.

The Firebase SDKs for Cloud Storage integrate seamlessly with Firebase Authentication to identify users, and we provide a declarative security language that lets you set access controls on individual files or groups of files, so you can make files as public or private as you want.[7]

Firebase as a backend

Firebase is a BaaS (Backend as a service), which provides RTDB (real time database) services for mobile apps and web developers. It was established in 2011 and was purchased by Google in 2015.

Firebase is a technology that allows us to create web apps without any server-side programming, so that development becomes quicker and easier. With Firebase, we don't need to worry about provisioning servers or building REST APIs—with just a little bit of configuration, we can let Firebase do the work: storing data, authenticating users, enforcing access rules and most importantly providing realtime database access.

Watt? uses Firebase as in the following use cases:

1. Stores consumption data
2. Connection between hardware plugs and user mobile phones
3. Authenticates users
4. Enforces access restrictions on stored data.
5. Stores per-user private files (images, and profile pictures)
6. Providing APIs for remote configuration change.

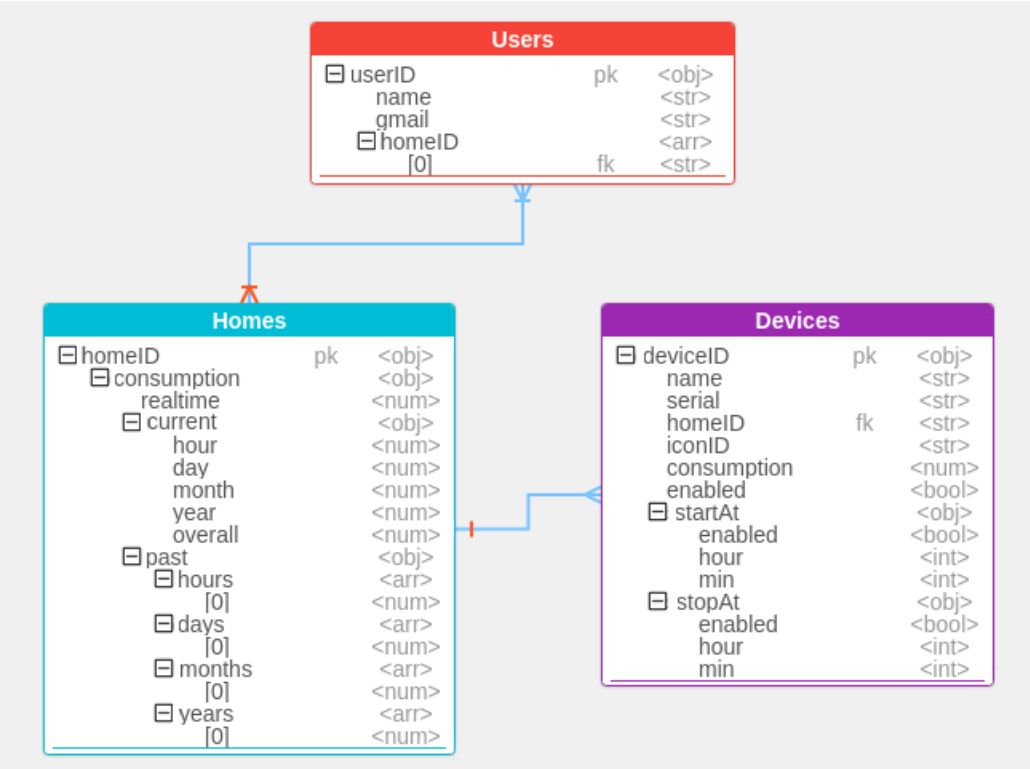
Watt? Database

One of the services that Firebase provides is the realtime database. it is a cloud database where data is synced across all clients in realtime, and remains available when their apps goes offline. Firebase Realtime Database is a Key-value NoSQL database which emphasizes simplicity and is very useful in accelerating an application to support high-speed read and write processing of data. The application has complete control over what is stored in the value, making this the most flexible NoSQL model. Stored values has the type of object which can be string, integer, double, boolean, array or even another object and are accessed via a key.

Watt? Database Diagrams

The Unified Modeling Language (UML) didn't stand powerless in front of the NoSQL databases, its Entity Relationship Diagram (ERD) still can be used to represent the NoSQL database models.

Entity Relationship Diagram



ERD is a way of mapping the relationships between different entities in the database. We still need ERD to know how exactly we are going to store data in our database, but things are little different here. Firebase Realtime Database is a Key-value NoSQL database, This means there are no Tables and Joins and we need to decide whether we are going to *embed* or *refer* an object in another object. For instance, we are creating a power monitoring system. So, we are going to store smart meters data (Homes) and consumption history (past). In this case it's a good idea to **embed** past object inside Homes object. This is one of the benefits of Key-value database i.e. you can get all the data related to the *home* with a single query. As *past* consumptions will always appear in the context of the Homes it's almost always a good idea to embed *past* in *home*. But what about the owners (Users) of the home? In this case we need to store a **reference** to the Homes inside Users objects.

We can still use our ERD to keep track of how we are going to store data in Firebase Realtime Database. But, in the end it comes down to - Embed or Refer? Here we have made a *modified* form of ERD which shows the embedded and referenced objects as a *tree* of objects. This diagram is built using a specialized software for modeling NoSQL databases called Hackolade.

In Watt? Database we have three main entities: Users, Homes and Devices. Users refer to the Watt? Application Users, Homes refer to the homes that has at least the Watt? Smart Meter and Devices refer to the appliances that are connected with Watt? Smart Plugs. Users can own multiple Homes and Homes can have multiple Devices.

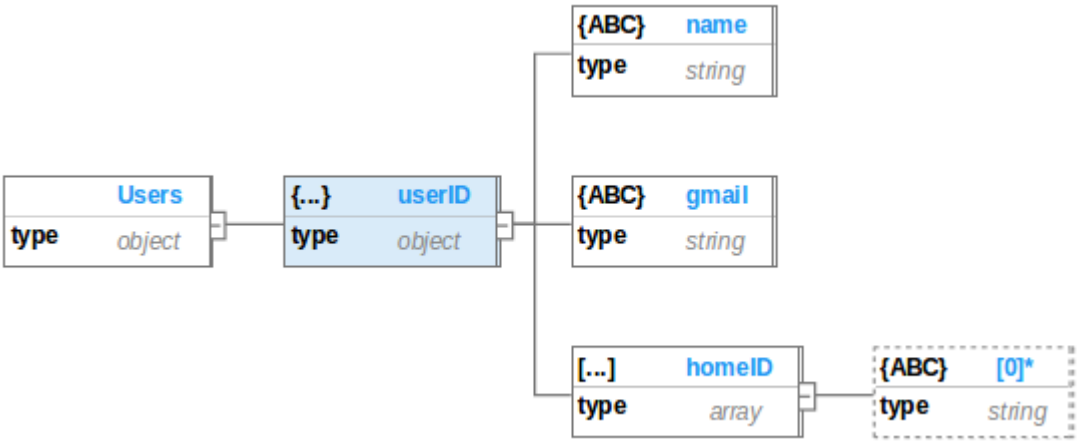
Tree Diagram

Tree diagram is a way of representing the hierarchical nature of a structure in a graphical form. The schema for a hierarchical database consists of :

- **Boxes**, which correspond to object key, value and type.
- **Lines**, which correspond to links between objects.

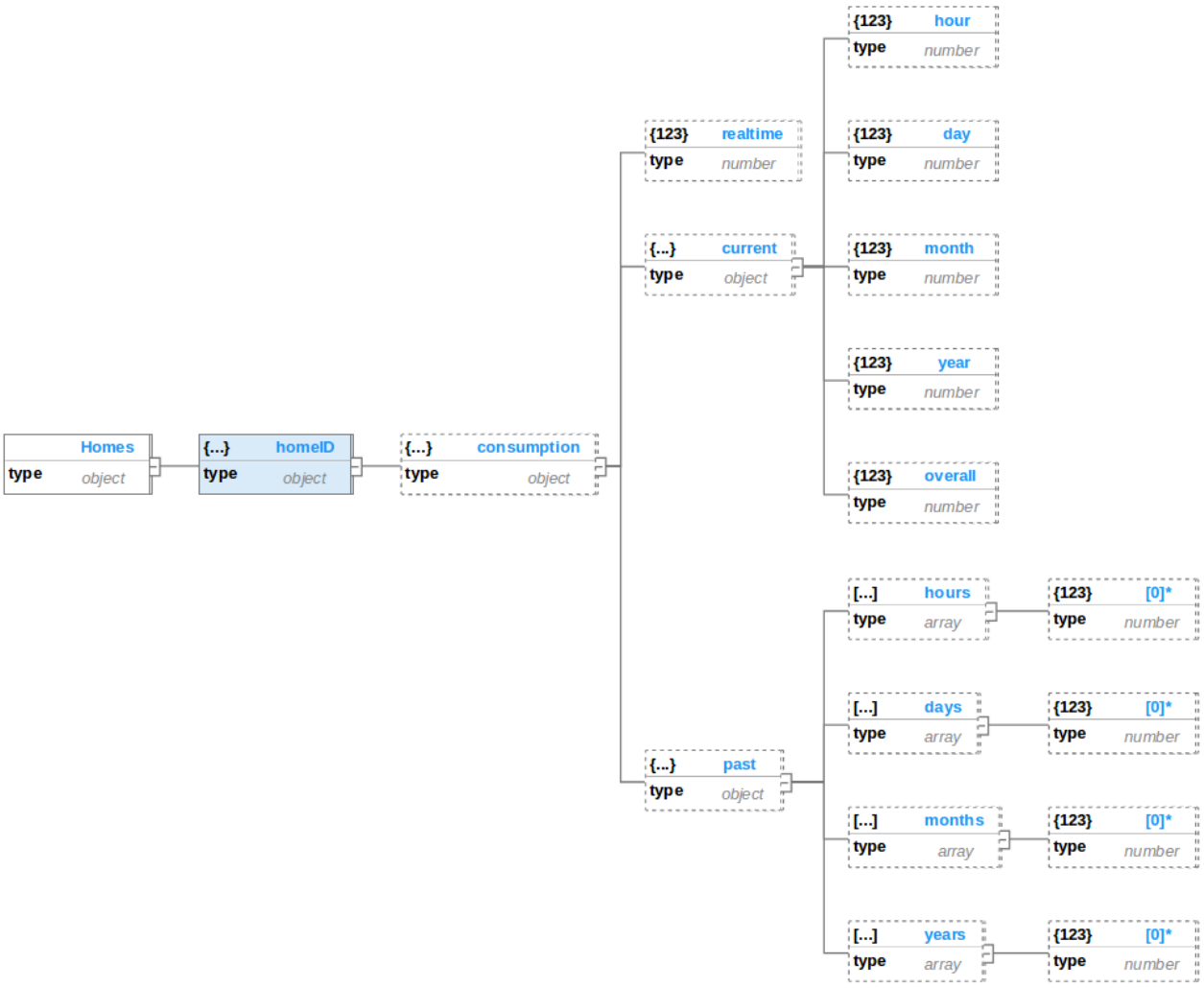
Relationships formed in the graph must be such that only *one-to-many* or *one-to-one* relationships exist between a parent and a child objects. The tree diagram of Watt? Database can be represented as the following three separated trees.

- **Users Tree**



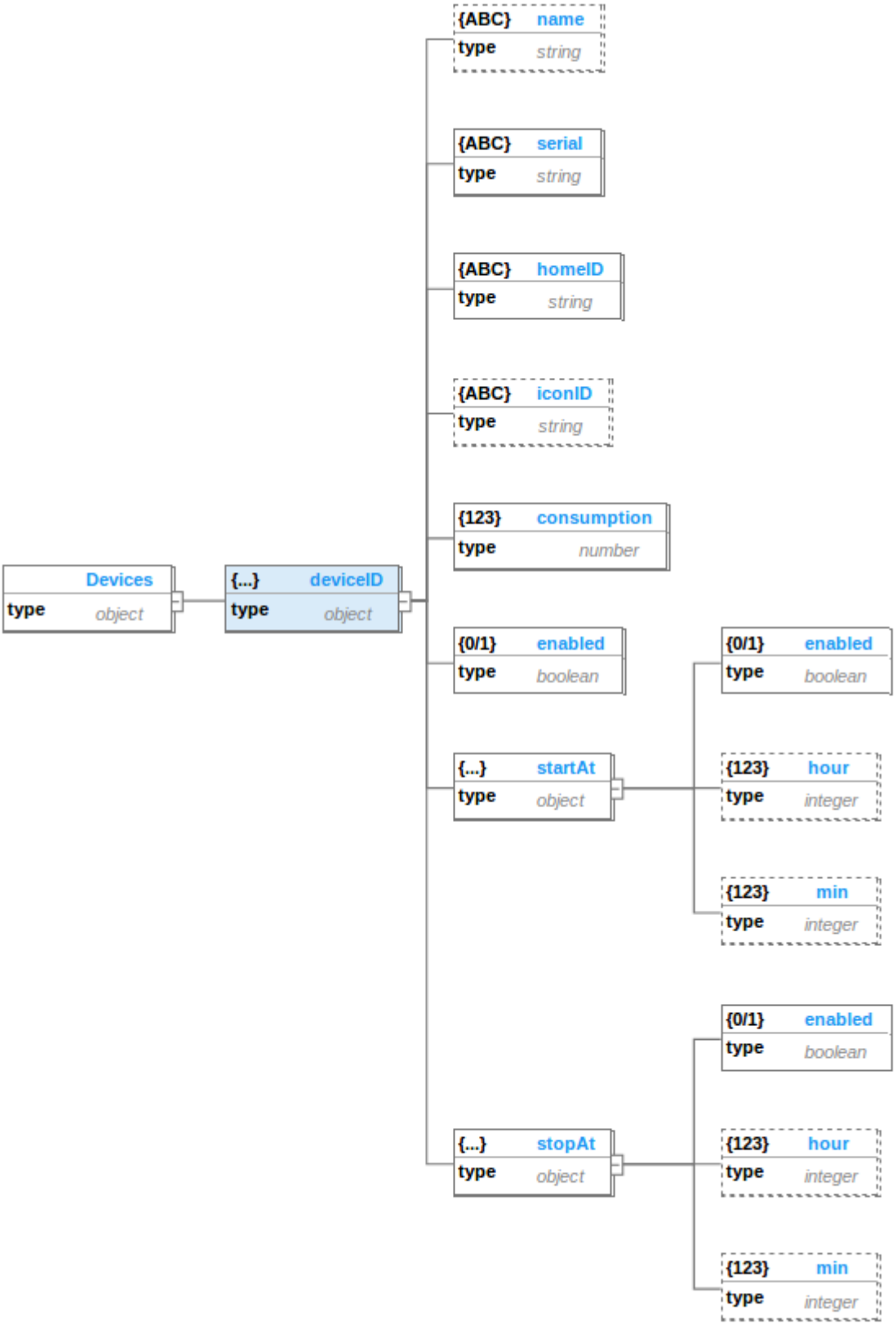
When a new user starts using our Watt? Mobile Application, He will be asked to register an account using his *Google Account*. The app will create an object under the root node (Users) with the key of a generated *userID* and the value of an object containing user info (name and email) and an empty array. Then the app will ask the user to define his Watt? Smart Meters and push a references to them in the *homeID* array.

- **Homes Tree**



Every newly manufactured Watt? Smart Meter has an object in the *Homes* tree. the key of this object is a 20 character string that is hard-coded in the controller of the smart meter. The Watt? Smart Meter is programmed to measure the realtime power consumption, calculate the cumulative energy consumptions and keep track of the past consumptions of the home. It will push all these data to the Firebase Realtime Database under its *homeID* object as shown in the previous diagram.

- **Devices Tree**



The Watt? Smart Plug doesn't differ from the Watt? Smart Meter a lot. It is also has a hard-coded *deviceID* to store its data under the object of its key. The smart plug is programmed to store some info about the device that is plugged into it like *name* and *iconID*, measure the realtime consumption of the plugged device and control the plugged device by switching it on or off.

The user can control the plugged device as the following:

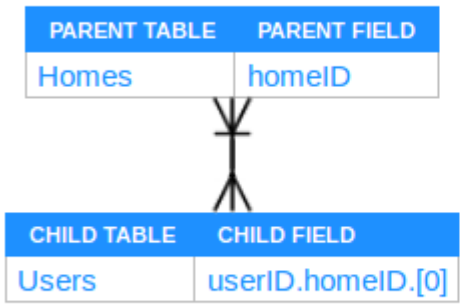
- **Immediately** by pushing the button on the smart plug or by using the mobile application.
- **At a specific time** by using the control timer feature in the mobile application.

So the *startAt* and *stopAt* objects are needed under the *Devices Tree* to handle the timer feature.

Relationships

As we said before, Users can own multiple Homes and Homes can have multiple multiple Devices. Those are the two relationships that are shown above in the ER diagram. They can be described as :

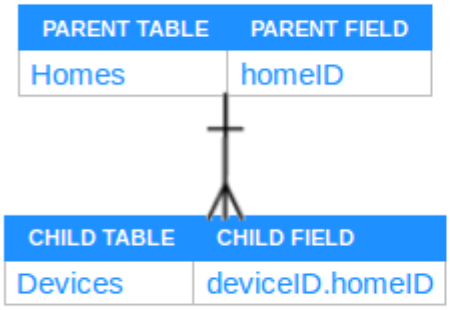
- **Belongs To**



PROPERTY	VALUE
Name	belongsTo
Description	
Parent Document	Homes
Parent field	homeID
Parent Cardinality	1..n
Child Document	Users
Child field	
Child Cardinality	n
Comments	

Each user in the *Users Tree* has an array to hold his *homeIDs*. This indicates that these homes belong to that user. As the user must have at least one home (Watt? Smart Meter) and the home can belong to zero or more users, this is a many-to-many relationship.

- **Has**



PROPERTY	VALUE
Name	has
Description	
Parent Document	Homes
Parent field	homeID
Parent Cardinality	1
Child Document	Devices
Child field	homeID
Child Cardinality	n
Comments	

Each home (Watt? Smart Meter) in the *Homes Tree* can have a zero or more Devices (Watt? Smart Plugs) and every device must be connected with one home. So *Devices* must have a foreign key to the *Homes* and this should be a one-to-many relationship.

Chapter 3

Mobile Application

Introduction

A mobile app is a software application developed specifically for use on small, wireless computing devices, such as smart phones and tablets, rather than desktop or laptop computers.

Mobile apps are designed with consideration for the demands and constraints of the devices and also to take advantage of any specialized capabilities they have.

Currently 77% of the world's population are online. With the rapid adoption of smart phones and tablets businesses are faced with more and more opportunities every day that will radically change how their service or product is delivered and accessed.

Watt? is a native application

Perhaps the simplest way to understand why we developed the application in Android is understanding the difference between 3 categories of applications used in today's development.

Different types of mobile applications

Native apps

Such apps are developed for a single mobile operating system exclusively, therefore they are "native" for a particular platform or device. App built for systems like iOS, Android, Windows phone, Symbian, Blackberry can not be used on a platform other than their own. In other words, you won't be able to use Android app on iPhone.

Main advantage of native apps is high performance and ensuring good user experience as developers use native device UI. Moreover, an access to wide range of APIs that puts no limitation on app usage. Native applications are distinctly accessible from app stores of their kind and have the clear tendency to reach target customers.

Some cons to native apps are higher cost compared to other types of apps – due to the need of create app duplicates for other platforms, separate support and maintenance for different types of apps resulting in bigger product price.

Examples

- iOS on Objective-C or Swift
- Android on Java
- Windows Phone on Net

Hybrid apps

They are built using multi-platform web technologies (for example HTML5, CSS and Javascript). So called hybrid apps are mainly website applications disguised in a native wrapper. Apps possess usual pros and cons of both native and web mobile applications.

Hybrid multi-platform apps are fast and relatively easy to develop – a clear advantage. Single code base for all platforms ensures low-cost maintenance and smooth updates. Widely used APIs, like gyroscope, accelerometer, geolocation are available.

On the other hand, hybrid applications lack in performance, speed and overall optimization in comparison to native apps for instance. Also, there are certain design issues due to app inability to look in exactly same way on two or more platforms.

Examples

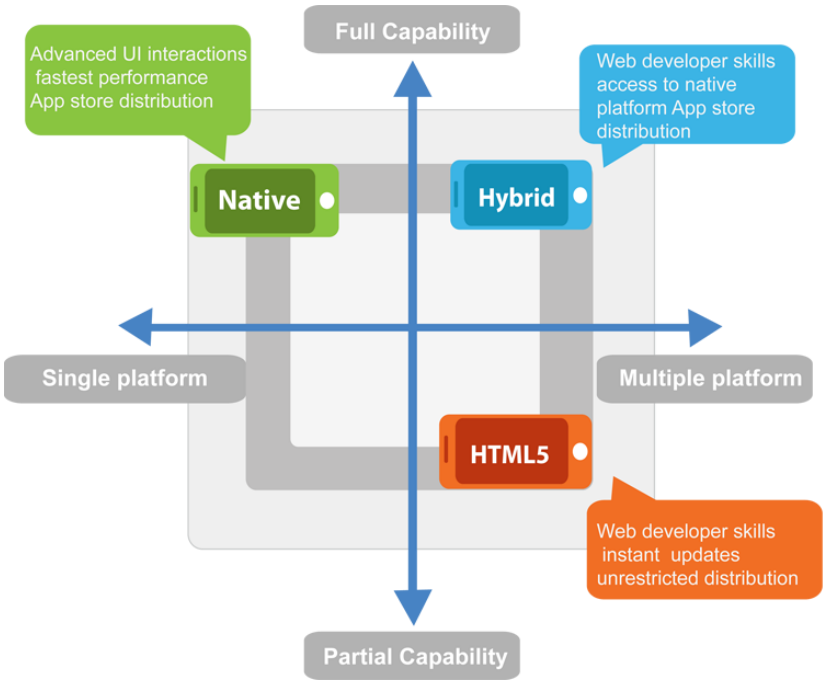
- Xamarin
- React Native
- Ionic
- Angular Mobile Sencha Touch

Web apps

These are software applications that behave in a fashion similar to native applications. Web apps use browser to run and are usually written in HTML5, JavaScript or CSS. These apps redirect a user to URL and offer “install” option by simply creating a bookmark to their page.

Web applications require minimum of device memory, as a rule. As all personal databases are saved on a server, users can get access from any device whenever there is internet connection. That is why the use of web apps with poor connection would result in bad user experience.

The drawback is access to not that many APIs for developers, with exception of geolocation and few others.



Going native

Obviously native apps won the battle at least for our requirements. Since native apps work with the device's built-in features, they are easier to work with and also perform faster on the device. Native apps get full support from the concerned app stores and marketplaces. Users can easily find and download apps of their choice from these stores. Because these apps have to get the approval of the app store they are intended for, the user can be assured of complete safety and security of the app. They also work out better for developers, who are provided the SDK and all other tools to create the app with much more ease.

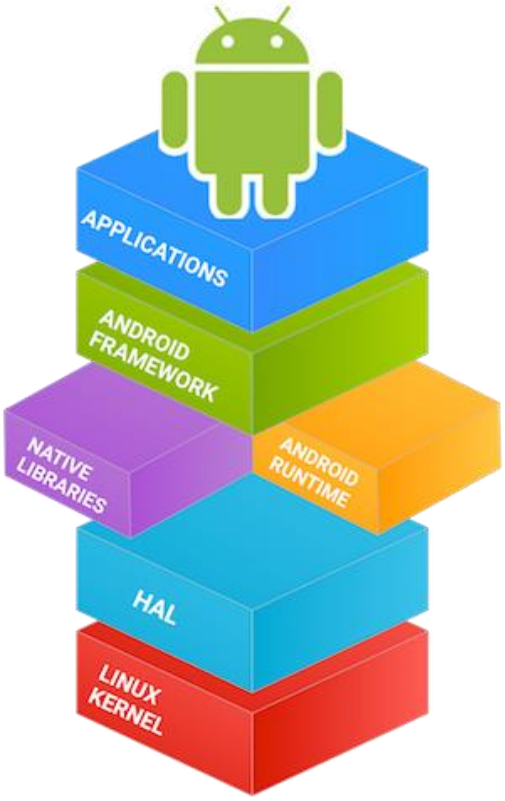
Android

Thanks to Google for introducing Android applications - the world of mobile apps will never be the same. The core reason behind instant success of Android apps is the open source platform that completely eliminates the restrictions common to other platforms. This makes Android apps development a whole lot more fun for even newbie users who have interestingly joined this field.

Believe it, some developers even as young as 14 years of age have successfully developed and sold Android apps. All you need to know are the development codes that help in modifying the apps as per requirements. Android phone users are given the discretion to manage OS for its effectiveness. This provides ample opportunity for Android users to have an OS in their phones that can accommodate any type of Android app according to their wish list while other phone, OS and mobile apps users are limited to the specifications of their phone-set and its features only.

Why choosing Android?

Since Android provides a more agile environment to perfect and test your app design, it is a safer bet to go for Android app development instead of iOS app development.



The main reasons that motivated us to develop the application in android:

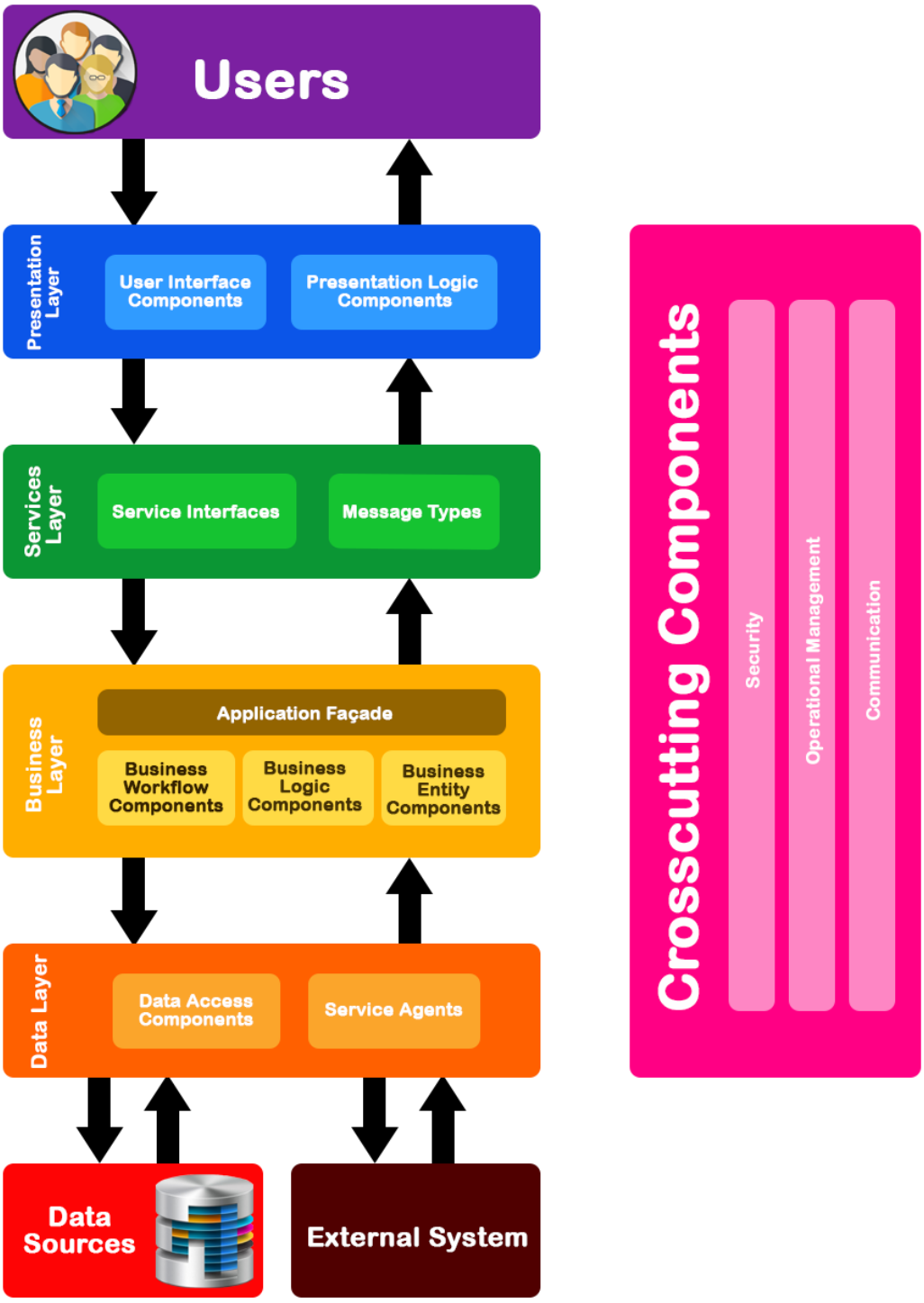
- **The Java Advantage** Since Android uses Java as its programming language, it has almost all the advantages of Java. Specially in terms of a lot of free and open source java libraries that can be used to develop any kind of app you are thinking of. Though not all java libraries are compatible with Android, many of them can be used as is and some of them can be easily translated into Android echo system. I was recently developing a Android app and required some of the utility functions for encoding, I sneaked into the Apache commons code and hand picked the useful classes for my app.
- **Easy to Find Help** Android is a popular platform and thousands of experienced developers are already developing on it. Communities like stackoverflow are filled up with many experts who are willing to answer your questions immediately. Unlike other platforms, android also has a advantage of a lot of Java experts who can suggest you alternatives in case you are not able to figure out a way to do things in Android.
- **Cost-effectively Marketing your App** Google Play provides its users over 12% more ad inventory than iOS which makes it relatively easier and cheaper to advertise your app. Imagine this power and how it can be used to devise a workable marketing strategy to make your app really sell.
- **Standardized Marketing Attribution Mechanism** Android has a standard mechanism for marketing attribution that is well understood and under practice for more than 15 years. Android automatically provides referrer information that anonymously identifies the source of a download, so the solution for tracking performance is clear and unambiguous.
- **Highly Reliable Ad Attribution** Unlike iOS marketing attribution that requires several methods for database data matching, Android keeps it simple with a referrer-based mechanism.
- **Quicker Turnaround** There is no perfect application - meaning that there is always room for improvement. Lucky for you, Google Play provides a very helpful platform for beginners and their apps. Developers can react to the feedback against their app and have an app

update available in a matter of hours. This helps in adaptability and ensures that app users remain hooked to your app.

- **Perfect Learning Platform** With faster development turnaround time and lower advertising costs, you can actually gain a lot from Android implementation. The Android platform is built upon the concepts of operability, allowing its users to learn a lot through experience.
- **Huge Free Learning Resources** Android community is vibrant and big. It has contributed thousands of resources for quickly learning the development on this platform. You can find your favorite media to learn android including eBooks, HTML, Videos and more absolutely free of cost.
- **Google's Ranking Algorithm** Ranking system in Google's ranking algorithm is not just about a huge number of downloads for your app. It ranks the app on basis of user retention. So, any start-up company can be sure to enter their app in Google play and then work on steadily building a user base.
- **Monetization** Until now, the research illustrates the payment mechanism as a huge factor behind why developers prefer iOS apps development to Android app development. However, with the integration of Google Wallet and Google Checkout, this gap between the two is also closing.
- **Scope for More Devices** Android platform has been adapted by all the big mobile hardware vendors and this will change the type of devices in future too. The hardware vendors like Samsung, HTC, Motorola etc are spending less on software and more on hardware now, this will allow them to come up with more innovative devices that can be much better than today’s smart phones.

Watt? Mobile application

It was back in August 2016 since we began the development of the mobile application. With a goal of high quality delivery of the application. We began the process of the development with requirements analysis. And by putting agile methods in mind (illustrated in the a subsequent section of this chapter), We could successfully reach many minor releases with production ready quality.


Users
Presentation Layer
User Interface Components
Presentation Logic Components
Services Layer
Service Interfaces
Message Types
Business Layer
Application Façade
Business Workflow Components
Business Logic Components
Business Entity Components
Data Layer
Data Access Components
Service Agents
Data Sources
External System
Crosscutting Components
Security
Operational Management
Communication

Architecture matters

Thinking of android as the development platform, Our android developer had to make long term plans for the application architecture, yet he still working on it till the moment. Reaching a better state of application architecture has been always the driving power.

Architecture is shape that a system adopts, to meet it's use cases. Software architecture matters because a good one is a key element of your long- term success. Here are some of the ways that architecture influences success. Not all of them are equally important, but all are related to your architecture.

- **Longevity** Most architectures live far longer than the teams who created them. Estimates of system or architectural longevity range from 12 to 30 or more years, whereas developer longevity (the time a developer is actively working on the same system) ranges from 2 to 4 years .
- **Stability** Many benefits accrue from the longevity of a well-designed architecture. One of the biggest is stability. Architectural stability helps ensure a minimum of fundamental rework as the system's functionality is extended over multiple release cycles, thus reducing total implementation costs. It provides an important foundation for the development team. Instead of working on something that is constantly changing, the team can concentrate on making changes that provide the greatest value.
- **Degree and Nature of Change** Architecture determines the nature of change within the system. Some changes are perceived as easy; others are perceived as hard. When easy correlates strongly with the desired set of changes that improve customer satisfaction or allow us to add features that attract new customers, we usually refer to the architecture as "good." In one application I worked on, the development team had created a plug-in architecture that would extend the analytic tools that manipulated various data managed by the system. Adding a new tool was relatively easy, which was a good thing because it was a major goal of product management to add as many tools as possible.
- **Profitability** A good architecture is a profitable one. By "profitable," I mean that the company that created the architecture can sustain it with an acceptable cost structure. If the costs of sustaining the architecture become too great, it will be abandoned . This does not mean that a profitable architecture must be considered elegant or beautiful. One of the most profitable architectures of all time is the Microsoft Windows family of operating system seven though many people have decried it as inelegant. It is important to recognize that the profitability of a given technical architecture often has little to do with the architecture itself. Take Microsoft, which has enjoyed tremendous advantages over its competitors in marketing, distribution, branding, and so forth. All of these things have contributed to Windows' extraordinary profitability. This is not an argument for poorly created, inelegant architectures that cost more money than necessary to create and sustain! Over the long run, simple and elegant architectures tend to be the foundation of profitable solutions.
- **Social Structure** A good architecture works for the team that created it. It leverages their strengths and can, at times, minimize their weaknesses. For example, many development teams are simply not skilled enough to properly use C or C++ . The most common mistake

is mismanaging memory, resulting in applications that fail for no apparent reason. For teams that do not require the unique capabilities of C or C++, a better choice would be a safer language such as Java, Visual Basic, Perl, or C#, which manage memory on the developer's behalf . Once created, the architecture in turn exhibits a strong influence on the team. No matter what language you've chosen , you have to mold the development team around it because it affects such as things as your hiring and training policies. Because architectures outlast their teams, these effects can last for decades (consider the incredible spike in demand in 19971999 for COBOL programmers as the industry retrofitted COBOL applications to handle the Y2K crisis).

How many people will it cost to replace this thing?

One of the most difficult decisions faced by senior members of any product development team is when to replace an existing architecture with a new one.

Many factors play into this decision, including the costs associated with sustaining the old architecture, the demands of existing customers, your ability to support these demands, and the moves of your competitors. Creating a formal set of rules for knowing when to replace an architecture is impossible , because there are so many factors to consider, and each of these factors is hard to quantify. The following rules of thumb have served me well.

If you feel that your current architecture could be replaced by a development team of roughly half the size of the existing team in one year or less, you should seriously consider replacing your current architecture with a new one. In very general terms, this rule allows you to split your team, putting half of your resources to work on sustaining the old system and half of your resources on creating the new system, with minimal increases in your cost structure.

The resources assigned to create your architecture may not all come from your current team. Bluntly, creating a new architecture with fewer people often requires radically different development approaches. Your current team may not have the required skills, and may be unwilling or unable to acquire them. Thus, every manager must be ready to substantially change the composition of his or her team before undertaking an architectural replacement.

While replacing one or two people is often required, it is lunacy to try to replace everyone. The older and more complex the system, the more likely you'll need the services of one or two trusted veterans of its development to make certain that the new system is faithfully implementing the functionality of the old. There will be skeletons, and you're going to need these veterans to know where these skeletons are buried.

All of the traditional adages of "being careful" and "it is harder and will take longer than you think" apply here. You should be careful, and it probably will be harder and take longer than you estimate. People usually forget to estimate the total cost of replacement, including development, QA, technical publications , training, and upgrades. Economically, a good rule of thumb is that it costs at least 20 percent of the total investment in the old architecture to create a functionally equivalent new one if you are using experienced people, with no interruptions, and a radically

improved development process. If you're on your fourth iteration of a system that has been created over five years with a total investment of $23 million, you're probably fooling yourself if you think that you can create the first version of the new system for less than $3$5 million. More commonly, the cost of creating a functionally equivalent new architecture is usually between 40 and 60 percent of the total investment in the original architecture, depending heavily on the size of the team and the complexity of the system they are replacing.

Watt? Services provided by the mobile application

Trying to control the energy consumption and costs of your home, we started the development of the application with many features in mind. The goal was to provide people with all the information they need to track their energy usage and understand the costs associated to that usage.

Feature list:

1. Get the whole realtime value of the power consumption of your home
2. Get the cumulative value of the consumption since the beginning of the current month till now.
3. Get the power consumption for each month of the current year.
4. Get the realtime value of power consumption of each of the connected devices.
5. Get the cumulative value of power consumption of each of the connected devices since the beginning of the current month.
6. Switch on/off any device connected to the system
7. Set timer to control the startup/shutdown of each device.
8. The price from the beginning of the month till now.
9. The expected price of the consumption till the end of current month.
10. Price saved by the system.
11. Account for each user.
12. Add new homes to one account.

Security

Mobile app security refers to the enforcement of access and data protection measures for individual apps. Examples of such app security policies include mobile app data encryption and authentication. These security policies and others can be applied during app development, later with software development kits (SDKs), or after the app is compiled with app wrapping.

Watt? applies many security polices:

1. **Authentication** – Dynamically requires users to enter their credentials before the app will open. This utilizes Google integration for single sign-on (SSO).
2. **Self-updating App** – Checks for new versions of an app at run time and prompts the user to update that app ensuring 100% update compliance for a particular app version.
3. **Crash Log Reporting** – Captures crash logs and returns them to the mobile administrator without requiring user intervention

4. **App Usage** – Tracks app usage data and provides statistics in an analytic dashboard for mobile administrators

In conclusion, security was and will continue to be an evolving effort. Today's security polices and mechanisms must be updated tomorrow. As new holes, breaches and threats are found then the development of new polices must continue.

Realtime

Mobile apps can be split into two categories: “static” or “real-time.”

Static apps: are solitary, single-user experiences where content changes only when the user clicks a button, requests a new page, or does a “reload.” New information is presented only when the user asks for it.

Real-time apps are infinitely more engaging. These apps mimic behaviors we’re used to having in the real world: content is pushed to us “as it happens.” Real-time applications let you edit docs together, battle your online buddies. Real-time apps make sure citizens know about critical safety issues as they happen.

We developed Watt? with realtime usage in mind. You can get realtime power consumption, realtime control and realtime management of your home. While using Google Firebase technology, As an mobile application developer, I had to make sure that the application respond correctly and have good sense of realtime response. That affected the architecture, backend and the user interface of the application.

Agile from a developer

Agile methods represent a relatively new approach to software development, becoming wide-spread in the last decade.

Flexible software development method (or Agile application development method) is the approach to product development, in which the whole process is divided into a series of development cycles. In fact, when using Agile for mobile app development, the entire array of tasks is divided into multiple sub-tasks, each of which is a separate mini-project for the development team.

With the proper division of assignments required for the project, as well as the distribution of specialists into dedicated teams, development risks, including errors, are minimized. Each element of the product, developed by a separate unit, is, in fact, a separate independent project.

Agile practices

There are many well known practices for the Agile methodology:

- **The Planning Game** Over the years it has become very clear that there are many ways to shave this Yak. Some teams need more process around this than others. For some, a simple list of features will do. For others, a Kanban board will be sufficient. Still others will need the full suite of stories, and tasks, and releases, and story points, and... Well, you know. Choose wisely!

- **Customer Tests** Lots of customers don't want to be bothered with these tests. That's a shame, since they are demonstrably the best way to specify requirements. For those teams that have customers engaged enough to specify the requirements in terms of Cucumber tests, or FitNesse tests there is no better alternative. Teams that are not so fortunate are not likely to benefit from this practice. If the customers neither read nor write the tests, then high level unit tests written in code suffice.
- **Small Releases** It's hard to imagine a team that would not benefit from this practice. Keep the releases small. The more time you wait between exposing the customers to the system, the more can go wrong.
- **Whole Team** Again, it's hard to imagine a team that would not benefit from a close relationships between the business people, and the developers. Not all teams are so fortunate, of course.
- **Collective Ownership** As far as I'm concerned any team that has individual code ownership is deeply dysfunctional. If the owner of some part of the code decides to leave, the whole team is left in crisis mode. There are many ways to achieve collective ownership, but the bottom line is very simple. No single individual should be able to hold the team hostage. Every part of the code should be known by more than one person -- the more the better.
- **Coding Standard** This simply goes along with Collective Ownership. The code should look like the team wrote it, not like one of the individuals wrote it. The members of the team should agree on the way that their code will appear. This isn't rocket science.
- **Sustainable Pace** This is a real simple idea. Software projects are marathons, not sprints. You dare not run at a rate that you cannot sustain for the long term. Murphy tells us that any team that violates this practice is doomed to flame out at the worst possible moment.
- **Continuous Integration** Certainly there are teams who's projects are so small that setting up a CI server is redundant. However, for most teams this is such a positive win that neglecting it would be immoral, if not insane.

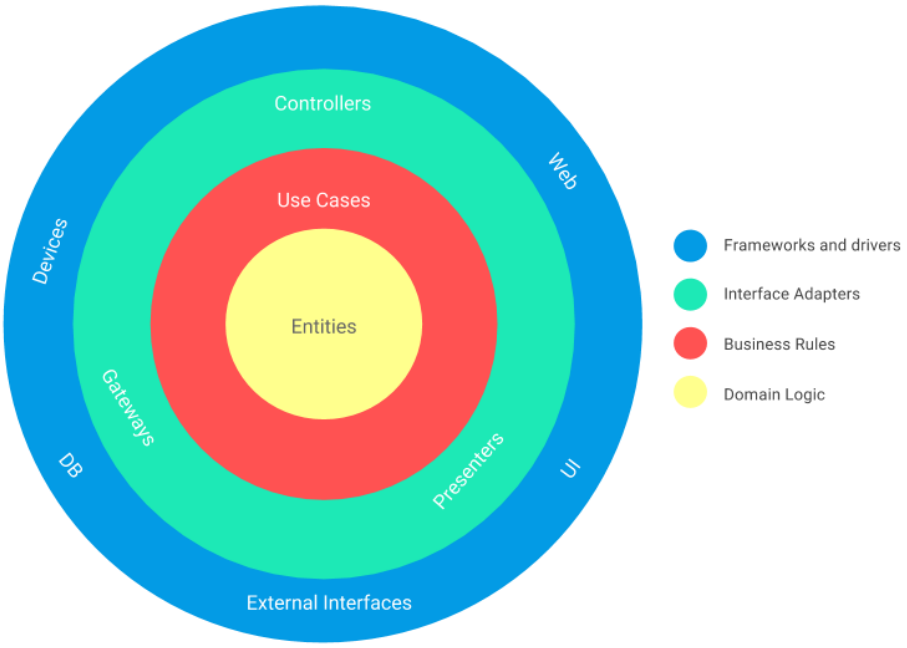
- **Pair Programming** Some teams benefit greatly by using this practice. Others do not. For the latter, some form of code review is likely necessary. In any case, it is a very good idea for every line of code to have been seen by more than one pair of eyes.
- **Simple Design** If we learned anything in the '90s it is that over-design is suicide. The level of design is team dependent, of course; but the simpler the better is simply a good rule of thumb.
- **Refactoring** Does anybody really want to argue that programmers should not keep their code as clean as possible? Does anyone want to argue that code should not be improved with time? Teams may choose different degrees of refactoring; but zero is probably not acceptable.
- **Test Driven Development** This is certainly the most controversial of all the Agile practices. But the controversy is not about the word Test. Virtually everyone agrees that writing unit tests is important. Some of us think that the order in which they are written is important too. Different teams will choose different strategies. But teams that ignore testing are not destined for rapid success.[4]

Development Challenges

Developing Watt? was not an easy task, it was full of challenges and sometimes problems. But there were no challenge that can't be overcame, and for each problem there is a solution. We just need to dig harder. Because Android is open source and support is everywhere these challenges has never been easier. But nothing is perfect, There were sometimes that I, as android developer, was stuck at it for days if not weeks, but surely everything passes.

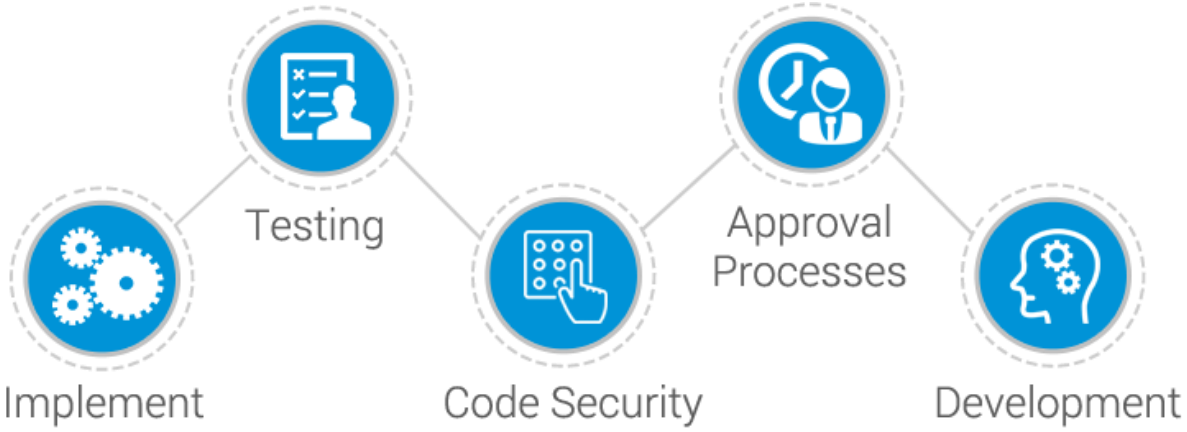
- **Custom views** The Android framework has a large set of View classes for interacting with the user and displaying various types of data. But sometimes your app has unique needs that aren’t covered by the built-in views. A well-designed custom view is much like any other well-designed class. It encapsulates a specific set of functionality with an easy to use interface, it uses CPU and memory efficiently, and so forth
- **Responsive layout** Android runs on a variety of devices that offer different screen sizes and densities. For applications, the Android system provides a consistent development environment across devices and handles most of the work to adjust each application's user interface to the screen on which it is displayed. At the same time, the system provides APIs that allow to control the application's UI for specific screen sizes and densities, in order to optimize the UI design for different screen configurations. For example, you might want a UI for tablets that's different from the UI for handsets. Although the system performs scaling and resizing to make your application work on different screens, you should make the effort to optimize your application for different screen sizes and densities. In doing so, you maximize the user experience for all devices and your users believe that your application was actually designed for their devices—rather than simply stretched to fit the screen on their devices.
- **Architecture** Unlike their traditional desktop counterparts which, in the majority of cases, have a single entry point from the launcher shortcut and run as a single monolithic process, Android apps have a much more complex structure. A typical Android app is constructed

out of multiple app components, including activities, fragments, services, content providers and broadcast receivers. Most of these app components are declared in the app manifest which is used by the Android OS to decide how to integrate your app into the overall user experience with their devices. The point of this is that your app components can be launched individually and out-of-order, and can be destroyed at anytime by the user or the system. Because app components are ephemeral and their lifecycle (when they are created and destroyed) are not under your control, you should not store any app data or state in your app components and your app components should not depend on each other.



- **Method Count limit** Android app (APK) files contain executable bytecode files in the form of Dalvik Executable (DEX) files, which contain the compiled code used to run your app. The Dalvik Executable specification limits the total number of methods that can be referenced within a single DEX file to 65,536—including Android framework methods, library methods, and methods in your own code. In the context of computer science, the term Kilo, K, denotes 1024 (or 2^10). Because 65,536 is equal to 64 X 1024, this limit is referred to as the '64K reference limit'.
- **Security** Android has built-in security features that significantly reduce the frequency and impact of application security issues. The system is designed so that you can typically build your apps with the default system and file permissions and avoid difficult decisions about security.

The following core security features help you build secure apps:



- The Android Application Sandbox, which isolates your app data and code execution from other apps.
- An application framework with robust implementations of common security functionality such as cryptography, permissions, and secure IPC.
- Technologies like ASLR, NX, ProPolice, safe_iop, OpenBSD dlmalloc, OpenBSD calloc, and Linux mmap_min_addr to mitigate risks associated with common memory management errors.
- An encrypted file system that can be enabled to protect data on lost or stolen devices.
- User-granted permissions to restrict access to system features and user data.
- Application-defined permissions to control application data on a per-app basis.
- It is important that you be familiar with the Android security best practices in this document. Following these practices as general coding habits reduces the likelihood of inadvertently introducing security issues that adversely affect your users.

- **Unit Testing** Unit tests are the fundamental tests in your app testing strategy. By creating and running unit tests against your code, you can easily verify that the logic of individual units is correct. Running unit tests after every build helps you to quickly catch and fix software regressions introduced by code changes to your app. A unit test generally exercises the functionality of the smallest possible unit of code (which could be a method, class, or component) in a repeatable way. You should build unit tests when you need to verify the logic of specific code in your app. For example, if you are unit testing a class, your test might check that the class is in the right state. Typically, the unit of code is tested in isolation; your test affects and monitors changes to that unit only. A mocking framework can be used to isolate your unit from its dependencies.

For testing Android apps, you typically create these types of automated unit tests:

- **Local tests**: Unit tests that run on your local machine only. These tests are compiled to run locally on the Java Virtual Machine (JVM) to minimize execution time. Use this approach to run unit tests that have no dependencies on the Android framework or have dependencies that can be filled by using mock objects.
- **Instrumented tests**: Unit tests that run on an Android device or emulator. These tests have access to instrumentation information, such as the Context for the app under test. Use this approach to run unit tests that have Android dependencies which cannot be easily filled by using mock objects.

Material Design

Google challenged themselves to create a visual language for users that synthesizes the classic principles of good design with the innovation and possibility of technology and science. This is material design. This spec is a living document that will be updated as we continue to develop the tenets and specifics of material design. Goals:

- Create a visual language that synthesizes classic principles of good design with the innovation and possibility of technology and science.
- Develop a single underlying system that allows for a unified experience across platforms and device sizes. Mobile precepts are fundamental, but touch, voice, mouse, and keyboard are all first-class input methods.

Principles

1- Material is the metaphor

A material metaphor is the unifying theory of a rationalized space and a system of motion. The material is grounded in tactile reality, inspired by the study of paper and ink, yet technologically advanced and open to imagination and magic.
Surfaces and edges of the material provide visual cues that are grounded in reality. The use of familiar tactile attributes helps users quickly understand affordances. Yet the flexibility of the material creates new affordances that supersede those in the physical world, without breaking the rules of physics.
The fundamentals of light, surface, and movement are key to conveying how objects move, interact, and exist in space and in relation to each other. Realistic lighting shows seams, divides space, and indicates moving parts.

2-Motion provides meaning

Motion respects and reinforces the user as the prime mover. Primary user actions are inflection points that initiate motion, transforming the whole design.
All action takes place in a single environment. Objects are presented to the user without breaking the continuity of experience even as they transform and reorganize.
Motion is meaningful and appropriate, serving to focus attention and maintain continuity.
Feedback is subtle yet clear. Transitions are efficient yet coherent.

3- Bold, graphic, intentional

The foundational elements of print-based design – typography, grids, space, scale, color, and use of imagery – guide visual treatments. These elements do far more than please the eye. They create hierarchy, meaning, and focus. Deliberate color choices, edge-to-edge imagery, large-scale typography, and intentional white space create a bold and graphic interface that immerse the user in the experience.
An emphasis on user actions makes core functionality immediately apparent and provides waypoints for the user.

Material characteristics

- Solid
- Occupies unique points in space
- Impenetrable
- Mutable shape
- Changes in size only along its plane
- Unbendable
- Can join to other material
- Can separate, split, and heal
- Can be created or destroyed
- Moves along any axis

Material Design Properties

Material Design has three main properties:

- **Physical properties**: Material Design uses material as a metaphor to create intuitive cues, consistency and visually pleasant experience. The visual cues were inspired by ink and paper. All the elements have a depth (x,y,z) and are 1 dp thick. Depth cues come from consistent shadows that come from the top to the bottom of the screen.
- **Transformational properties**: material can change shape, shrink and grow. Sheets of material can join together to become a single sheet, they can split and become whole again.

- **Movement properties:** motion helps users understand the flow between two states and provide them with meaning. Material can be moved anywhere along the plane but also can be raised vertically in the z direction which creates visual cues for interaction.

Material Design Limitations

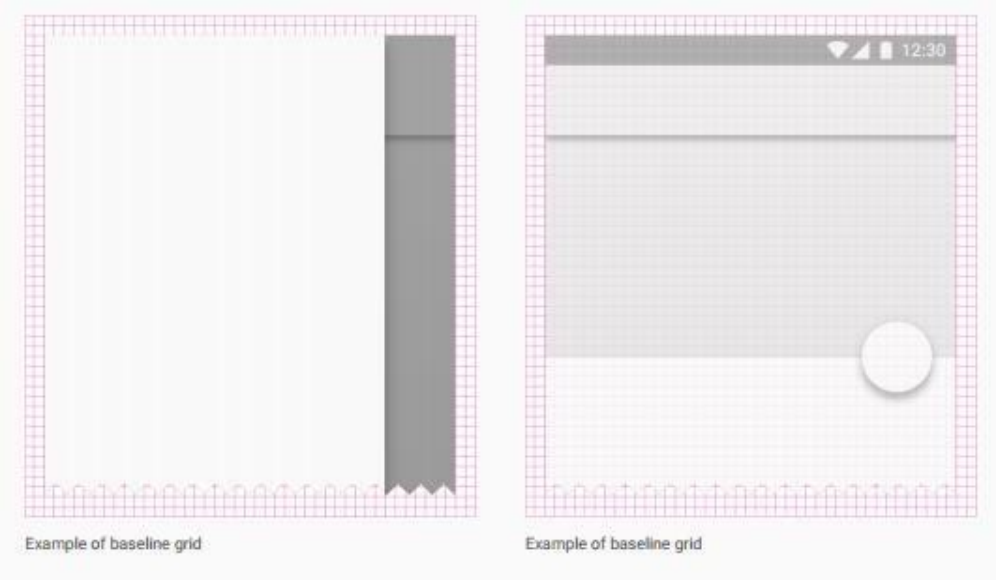
Material design makes affordances intuitive by relying on the physical world. However it has some limitations:

- Materials are solid and cannot pass through each other.
- Materials cannot bend.
- Materials cannot fold.

Component of Google Material guideline

1- Metrics & Keyline - Layout - Material design guidelines:

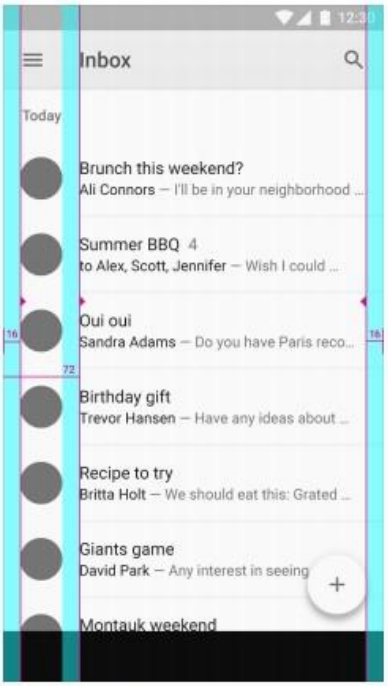
- **Baseline grids**
  all components align to an 8dp square baseline grid for mobile, tablet, and desktop. Iconography in toolbars align to a 4dp square baseline grid.



- **List Guideline:**

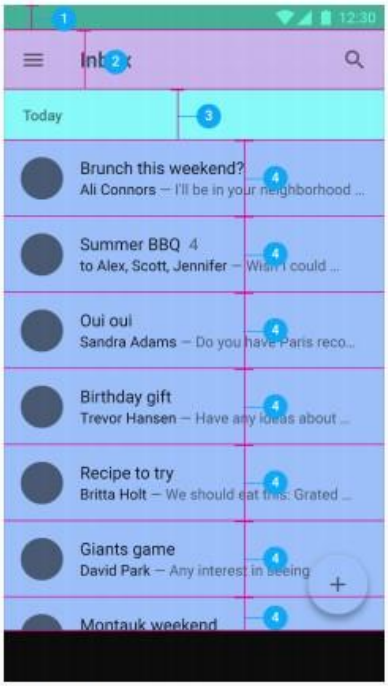
Vertical spacing

- Status bar: 24dp
- Toolbar: 56dp
- Title and list items: 72dp
- Subtitle: 48dp
- Space between content areas: 8dp



Keylines and margins

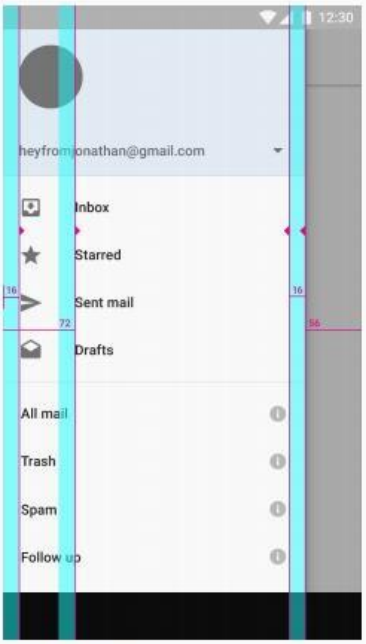
Screen edge left and right margins: 16dp
Content associated with an icon or avatar left margin: 72dp
Horizontal margins on mobile: 16dp



Vertical spacing

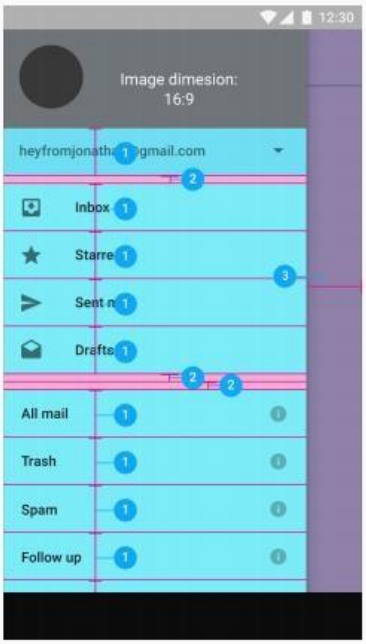
- Status bar: 24dp
- Toolbar: 56dp
- Subtitle: 48dp
- List item: 72dp

- **Navigation drawer:**
  A side navigation menu with icons, avatars, and text aligned on the left. Other icons align on the right.



Keylines and margins

Screen edge left and right margins: 16dp
Content associated with an icon or avatar left margin: 72dp
Side nav width: The screen width minus the height of the action bar. Here, the width is 56dp from the right screen edge.



Vertical spacing

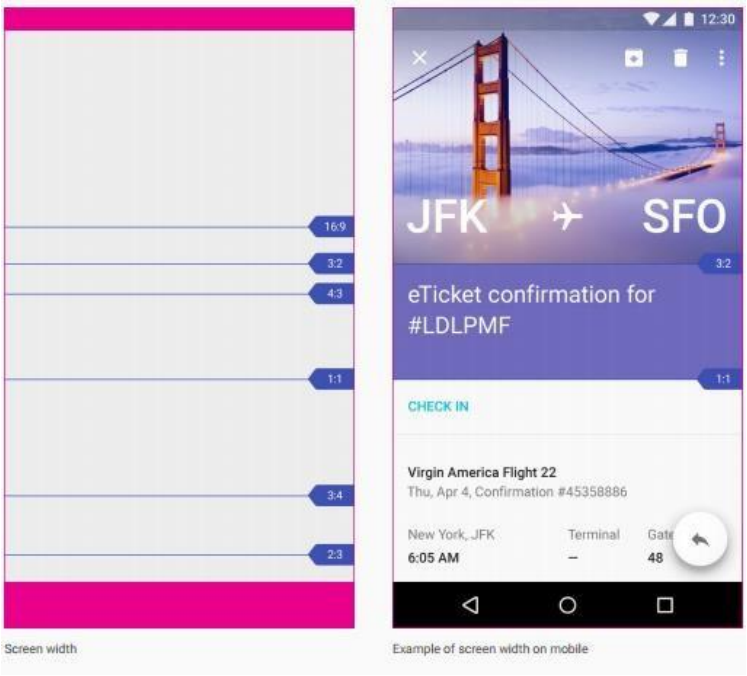
- Account menu and list items: 48dp
- Space between content areas: 8dp
- Navigation right margin: 56dp

- **Ratio key lines**

  The proportion of an element's width to its height (called the aspect ratio) applies to both UI elements and screen size. It is written as width: height.

  These aspect ratios are recommended:

  - 16:9
  - 3:2
  - 4:3
  - 1:1
  - 3:4
  - 2:3



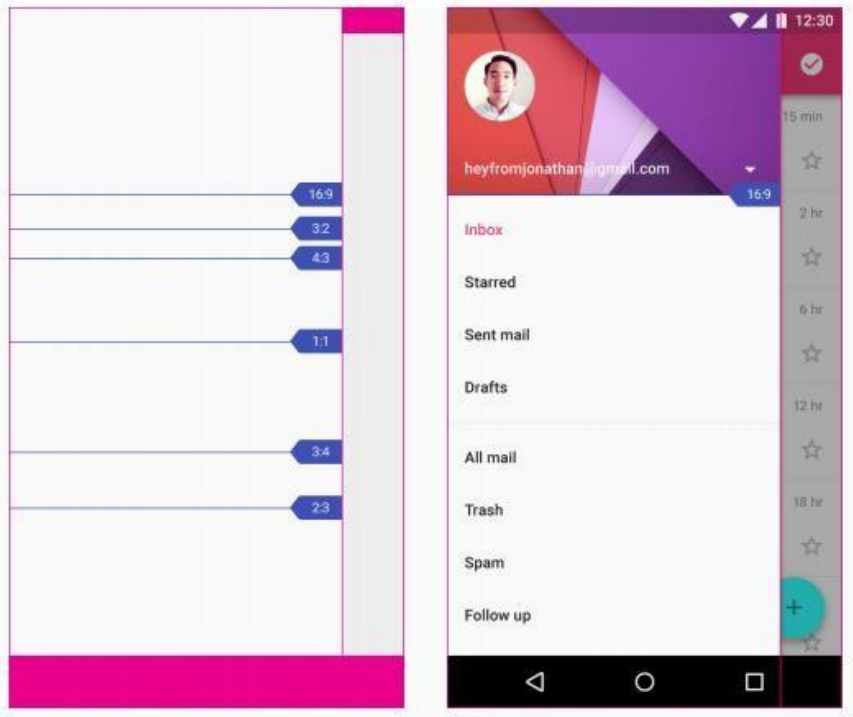
Screen width — Example of screen width on mobile

For example:

A 1:1 aspect ratio means an element has equal height and width. A 360dp wide image with a 2:3 aspect ratio has a height of 540. Determine the width or height of your element for a chosen aspect ratio using the below formulas. The aspect ratio is always expressed as a fraction. For example, 3:2 is treated as 3/2.

Width = Aspect ratio * Height

Height = Width/Aspect ratio



2- Color - Style - Material design guidelines:

- **Color**
  Color in material design is inspired by bold hues juxtaposed with muted environments, deep shadows, and bright highlights.

- **Color palette**
  Material takes cues from contemporary architecture, road signs, pavement marking tape, and athletic courts. Color should be unexpected and vibrant. This color palette comprises primary and accent colors that can be used for illustration or to develop your brand colors. They've been designed to work harmoniously with each other. The color palette starts with primary colors and fills in the spectrum to create a complete and usable palette for Android, Web, and iOS. Google suggests using the 500 colors as the primary colors in your app and the other colors as accents colors. Themes enable consistent app styling through surface shades, shadow depth, and ink opacity.

Example of color palette

- **Color schemes**

Choosing a color palette:
Your app's color palette may be defined by using a custom palette suited to your brand, such as monochromatic, black and white, full color, or neutral. Alternatively, you may use the material design color palette. All color palettes should include sufficient contrast between different UI elements.

Primary color : when using a primary color in your palette, this color should be the most widely used across all screens and components.
Secondary color : Palettes with a secondary color may use this color to indicate a related action or information. The secondary color may be a darker or lighter variation of the primary color.
Accent color : the accent should be used for the floating action button and interactive elements, Such as:

- Text fields and cursors
- Text selection
- Progress bars
- Selection controls, buttons, and sliders - Links

- **Text and background colors**

Text opacity: Text may be displayed with different degrees of opacity to convey how important certain information is relative to other information. The level of opacity used for text depends on whether your background is darker or lighter.

Dark text on light backgrounds: for dark text on light backgrounds, apply the following opacity levels:

- The most important text has an opacity of 87%. - Secondary text, which is lower in the visual hierarchy, has an opacity of 54%.
- Text hints (like those in text fields and labels) and disabled.
- Text have even lower visual prominence with an opacity of 38%.

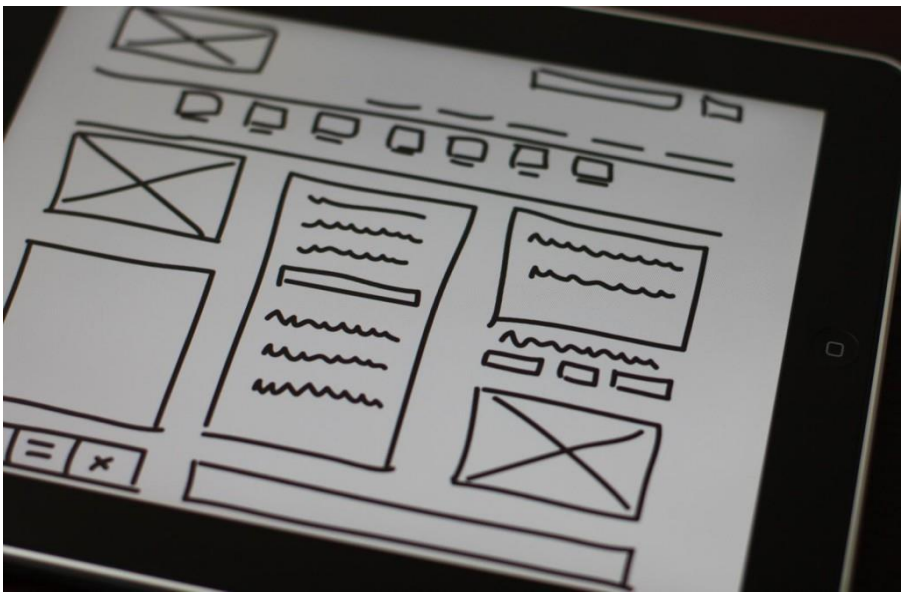
Dark text (#000000)	Opacity
Primary text	87%
Secondary text	54%
Disabled text, hint text, and icons	38%
Dividers	12%

White text on dark backgrounds: The table values relay relative levels of importance for white text on dark backgrounds. White text appearing on colored backgrounds should do so at an opacity of 100%.

Light text (#FFFFFF)	Opacity
Primary text	100%
Secondary text	70%
Disabled text, hint text, and icons	50%
Dividers	12%

WireFraming

A wireframe is a low-fidelity, simplified outline of your product. You can usually recognize them by their distinctive block layouts, use of lines to represent text, and "X" squares indicating placeholders for future images.



This barebones style makes wireframes a great tool early on, giving you time to cement your content architecture before diving into the details. Moreover, their simplicity is forgiving of mistakes and allows you to experiment, which takes some of the hassle out of designing the overall structure.

Wireframes Compared to Other Design Documents

The terminology used in design documentation often gets used interchangeably, so let's clarify the differences.

Think of **wireframes** as the skeleton. They loosely shape the final product, giving you a reliable idea of where everything will eventually go. The content is the muscle.

Next comes the **mockup**—the skin. Mockups are strictly visual. This is the documentation where you solidify your visual decisions, experiment with variations, and (optionally) create pixel-perfect drafts.

With the **prototype**, you breathe life into your creation. Prototypes test your interface ideas and generate the feedback necessary to keep the design headed in the right direction. Prototypes can (and should) be used during every stage of the design process, and can be in any fidelity. As we'll discuss below, you can even make a lo-fi prototype by adding interactivity to a wireframe.

Remember that **wireframes are only a means to a prototype**. In other words, prototypes are the most functional, useful documentation you can create.
Wireframes just help you focus the placement of content for your prototype.

Types of Wireframes

Most designers start by sketching wireframes on paper. Paper is faster and easier, but makes sharing with teams more difficult. And if you need to reference them often, the process inevitably takes longer.

Since wireframes are stepping stones to prototypes, you will eventually throw away your paper wireframes. One way to keep them useful is converting them into paper prototypes for usability testing: one person plays the role of the human computer, while the other person takes notes. This form of prototyping works well if you want to explore experimental concepts quickly and with minimal risk.

Some digital tools actually allow you to convert paper wireframes and prototypes into a collaborative digital tool.

The advantages of digital wireframes are clear: they can be shared instantaneously and convert into digital prototypes easier. As shown above, you can also add interactions directly by using simple and easy drag-and-drop, turning your wireframe into a lo-fi prototype in a matter of minutes. This lets you use (and test) the same document throughout the entire design process.

When Should You Wireframe?

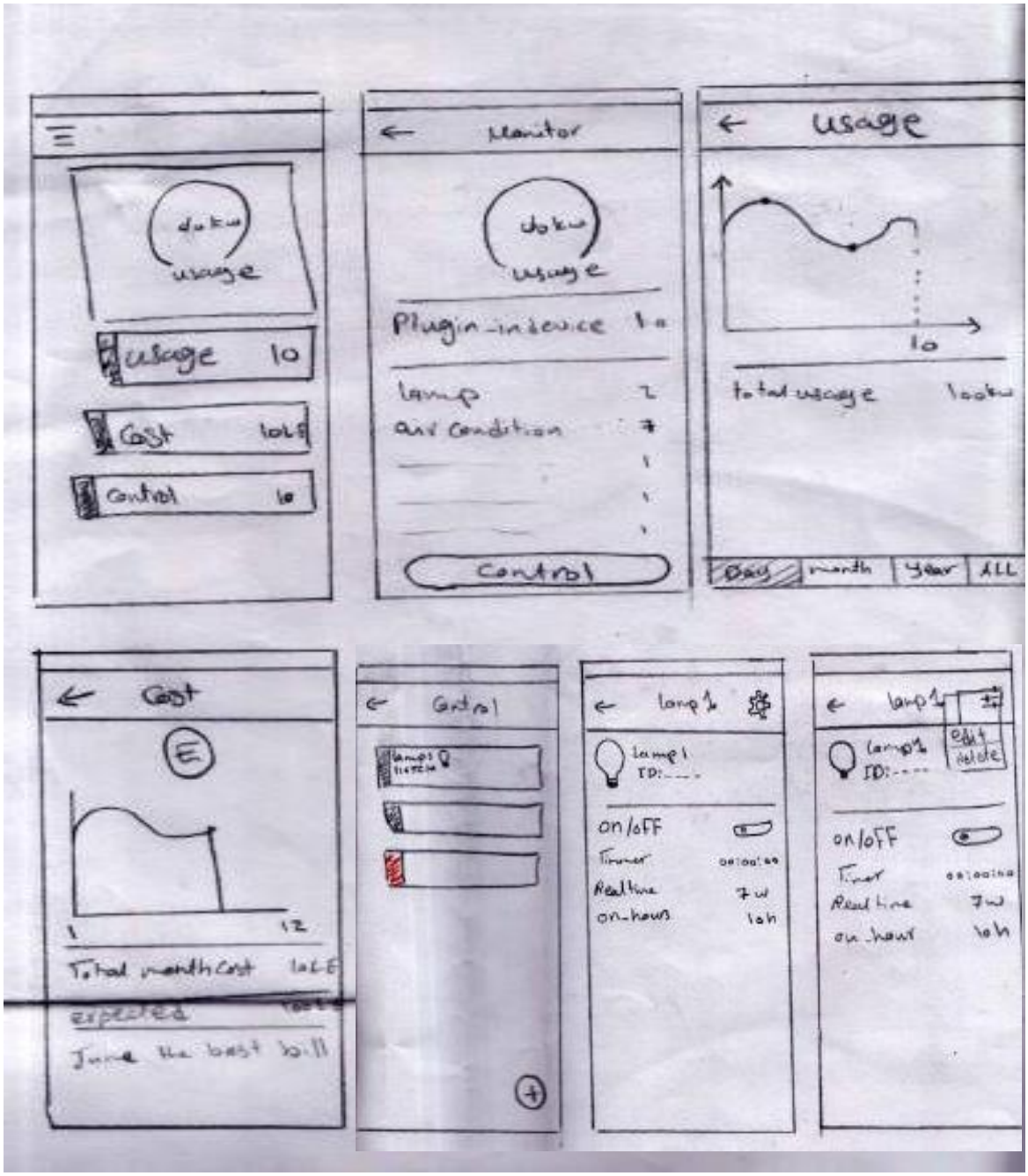
Wireframing is most useful at the beginning, when much of the product's structure is still up in the air.

The benefit lies in visualizing the structure of a concept, but at a step above sketching. For example, you can sketch five different layouts for a product page, compare them side-by-side, and then narrow them down to three based on the visual flow for a wireframe.

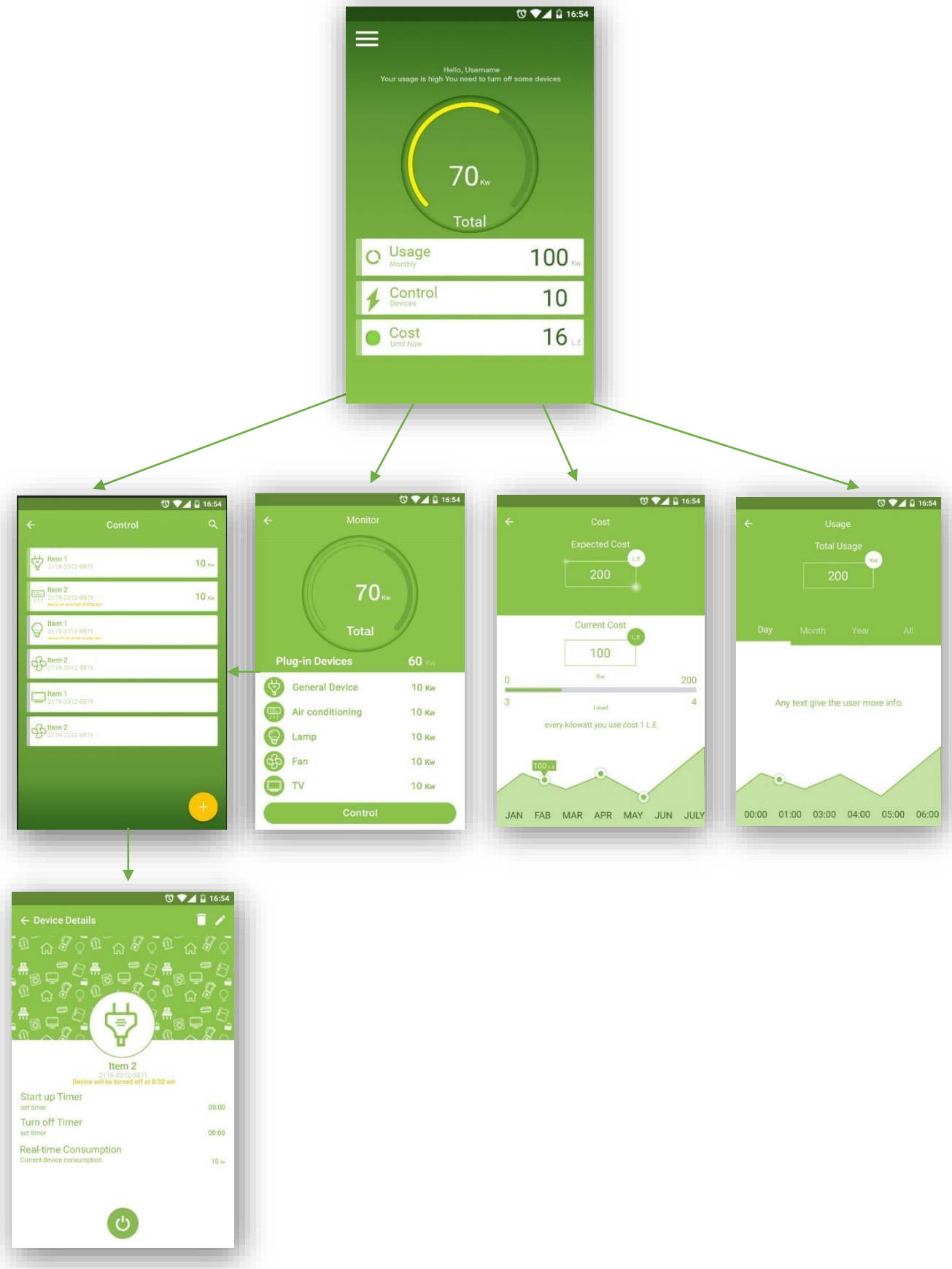
In later stages, the details have already been hammered out, so this kind of experimental structuring isn't as useful.

Watt? Mobile Application

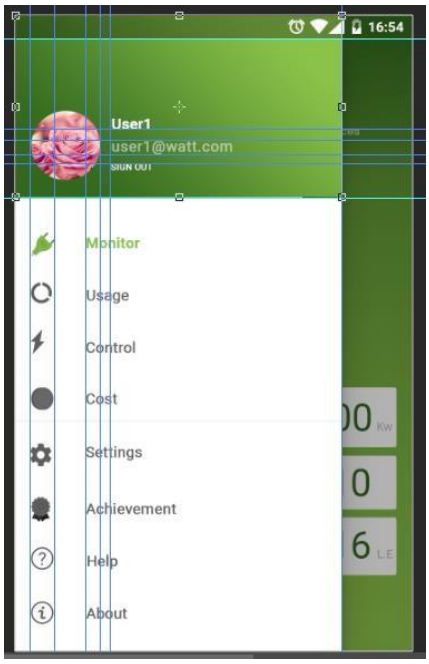
WireFraming

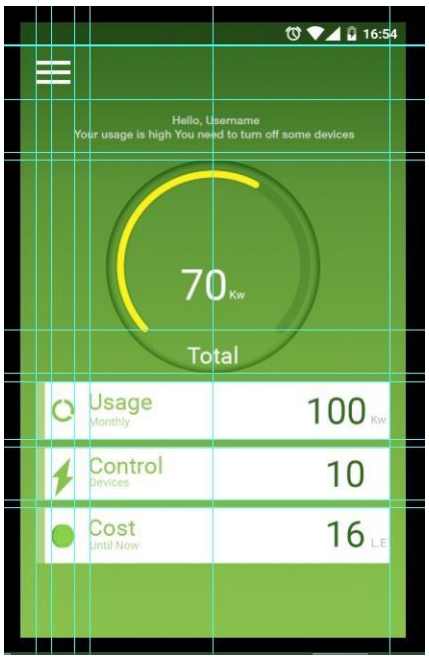


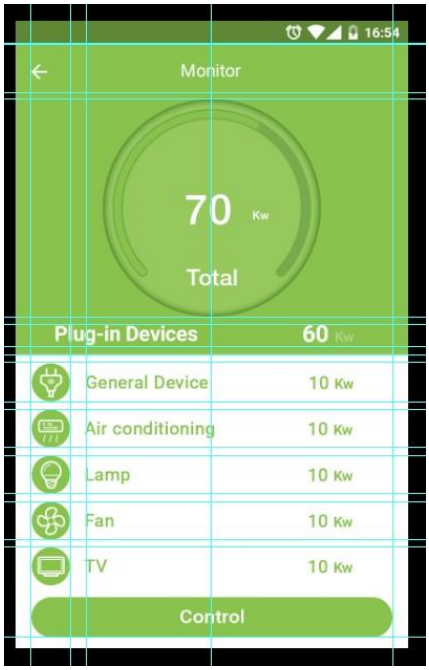
Structure of the application

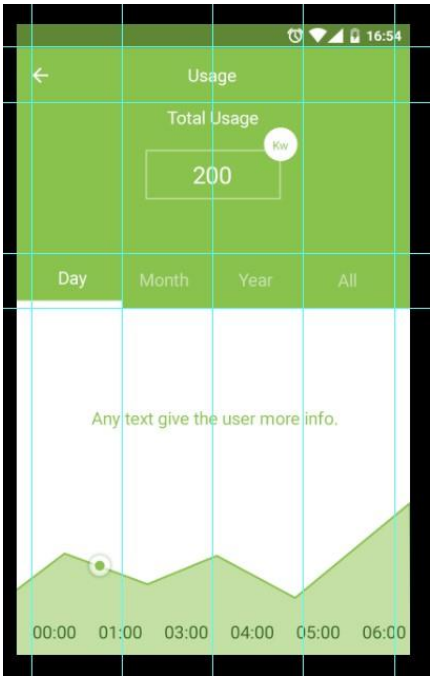


How we apply Metrics & Keyline - Layout - Material design guidelines in watt?









Screens with Grids

Watt? Color palette

Primary Color	Secondary color	Accent color
Light Green 500 #8BC34A	300 #AED581	A400 #FFC400

Green, the color of life, renewal, nature, and **energy**, is associated with meanings of growth, harmony, freshness, **safety**, fertility, and **environment**. Green is also traditionally associated with **money**, finances, banking, ambition, greed, jealousy, and wall street, green color is safe and means "go."

We choose accent color (yellow) to provide a contrast from the primary color (light green).

Watt? Icons

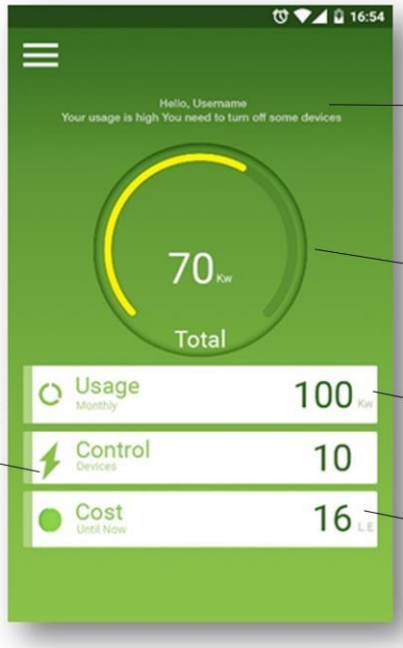
- **Devices Icons**

- **Other Icons**

Usage Icon Control Icon Cost Icon

Application Screens

- **Dashboard Screen**



State the usage to the user if it low, medium or high

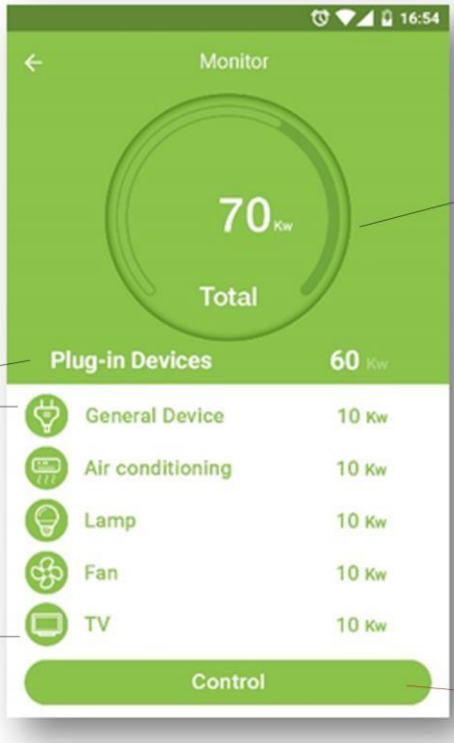
Show a brief note about user real time consumption

Show a brief note about user

Show a brief note about number of devices which turned on this momentactual total usage

the total cost according to the current usage.

- **Monitoring Screen**

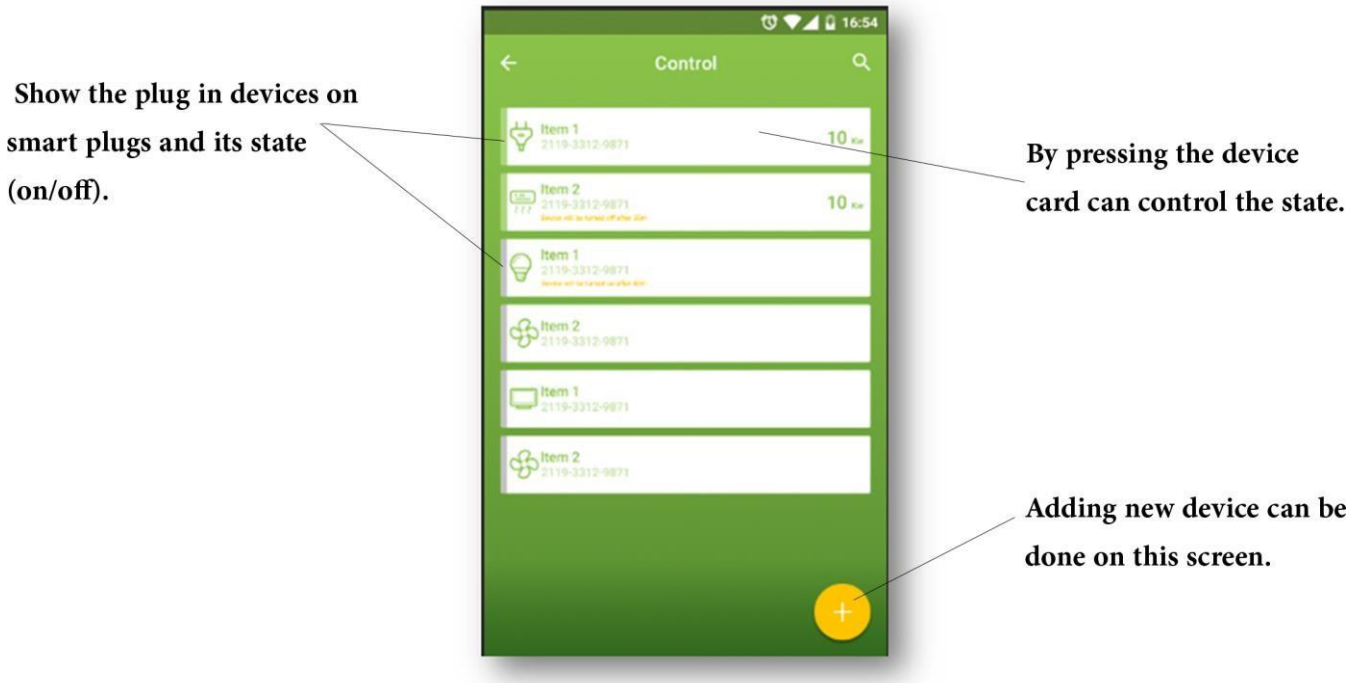


Show the real time consumption for place from smart meter

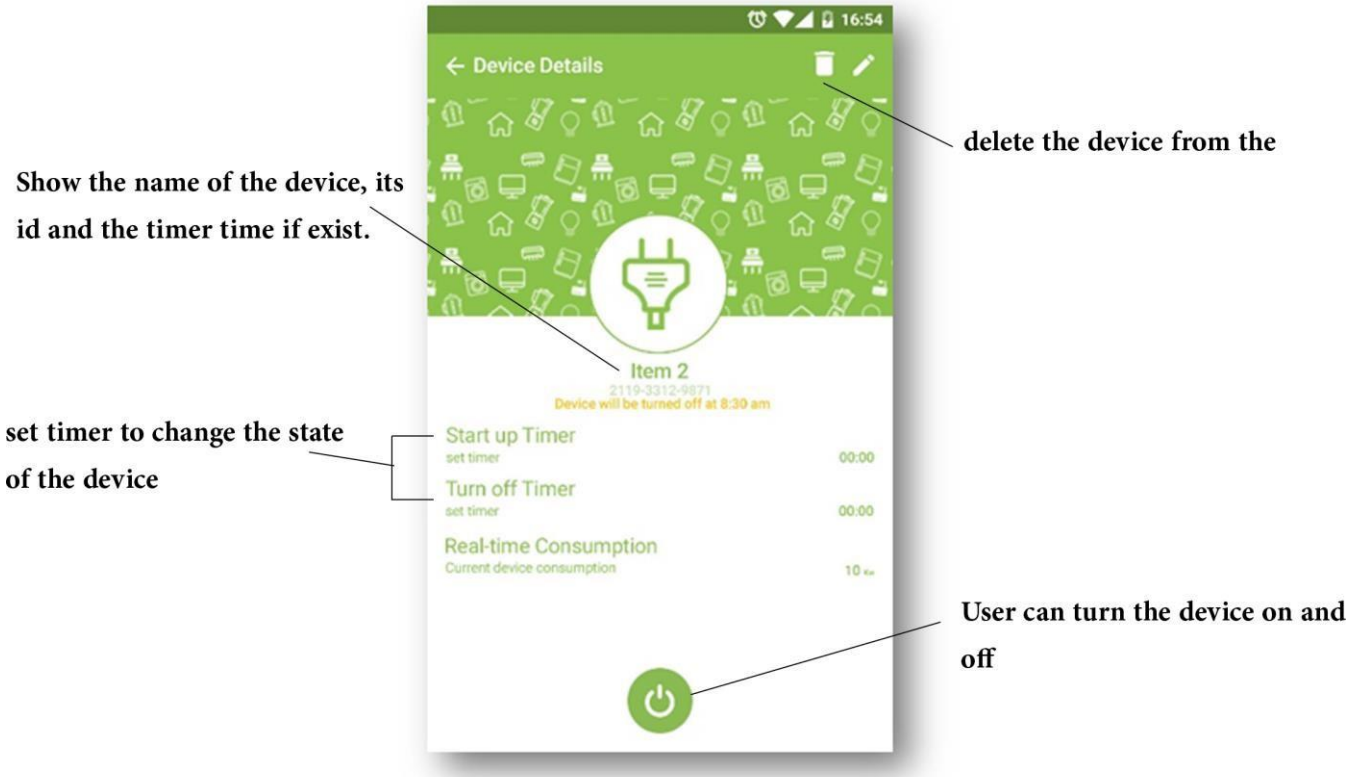
Show the real time consumption for the plug in devices on smart plugs.

User can go to control screen by pressing control button.

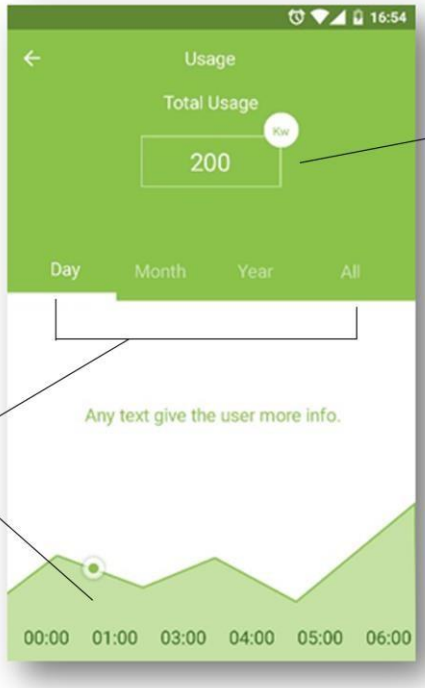
- **Control Screen**



- **Device Details Screen**



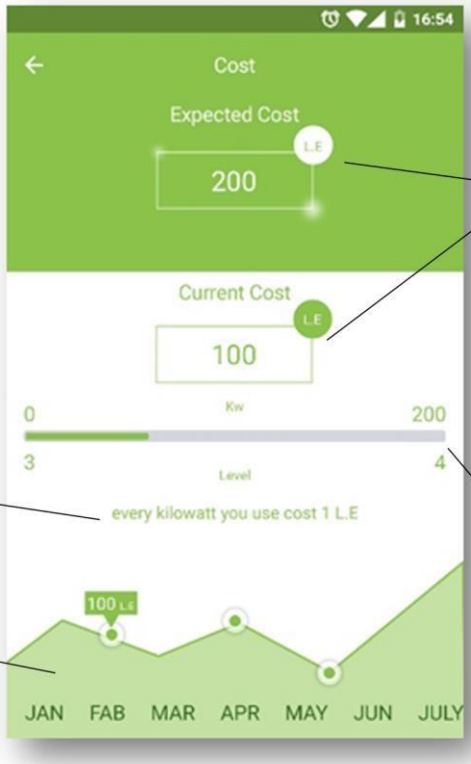
- **Usage Screen**



State the usage to the user from the beginning of the month until the moment

A graph illustrate the past usage during the day, the month, the year , and the beginning of the system

- **Cost Screen**



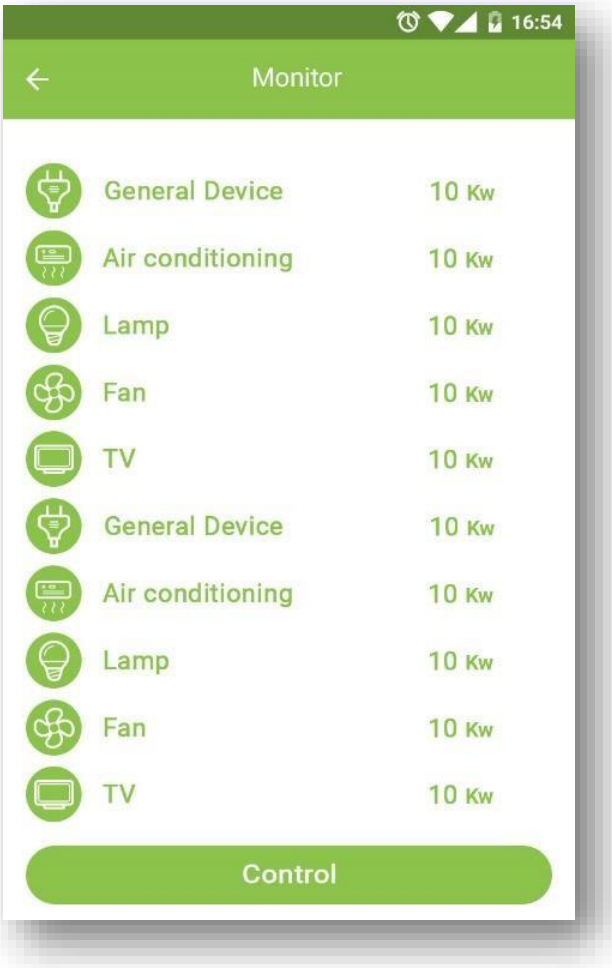
Show the Current Cost and the Expected cost according to the usage and the paying levels

Show the cost of the kw at this level.

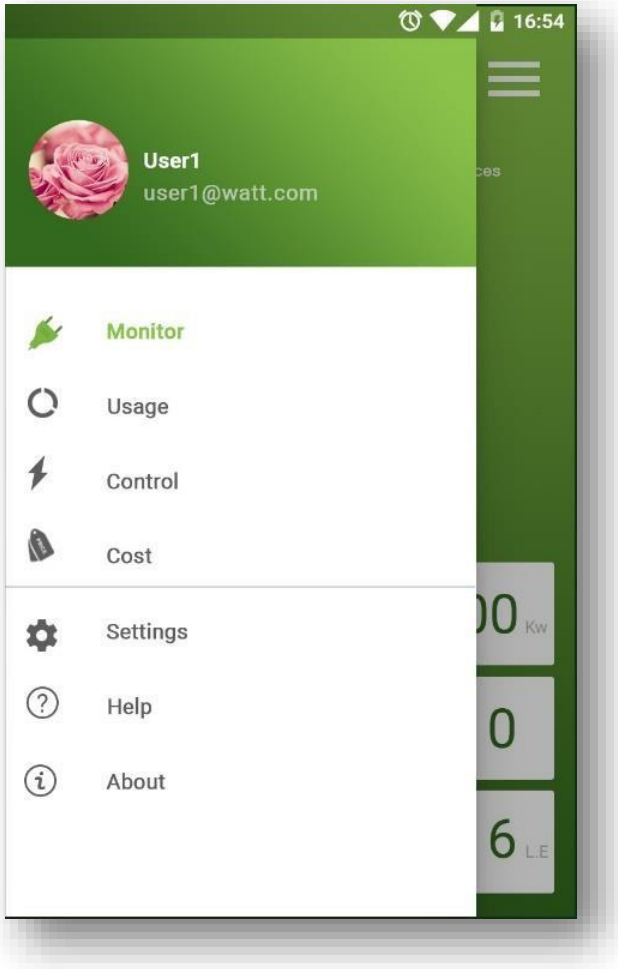
Show the state at the current paying level and the remaining.

A graph illustrate the past costs every month.

- **Other Screens**



Scrolled monitoring screen



Side Bar screen

Chapter 4

Networking

Introduction

An important aspect of the Internet of Things is that devices are networked in some way, and often connected to the Internet. Networking enables devices to communicate with other IoT devices and larger cloud-based servers. IoT devices can often be thought of as small parts of a much larger collective system which includes large servers based in the cloud. This chapter will introduce the networking and the Internet protocol in particular. Eventually, most IoT devices are connected to the Internet, so understanding the protocols associated with the Internet is important to the design of IoT devices.[13]

Communication

As the Internet of Things is growing very rapidly, there are a large number of heterogeneous smart devices connecting to the Internet. IoT devices are battery powered, with minimal compute and storage resources. Because of their constrained nature, there are various communication challenges involved, which are as follows :

- **Addressing and identification**: since millions of smart things will be connected to the Internet, they will have to be identified through a unique address, on the basis of which they communicate with each other. For this, we need a large addressing space, and a unique address for each smart object.
- **Low power communication**: communication of data between devices is a power consuming task, specially, wireless communication. Therefore, we need a solution that facilitates communication with low power consumption.
- Routing protocols with low memory requirement and efficient communication patterns.
- High speed and non-lossy communication.
- Mobility of smart things.

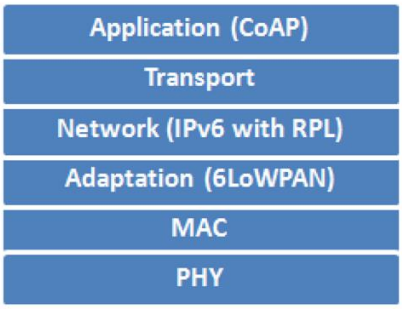
IoT devices typically connect to the Internet through the IP (Internet Protocol) stack. This stack is very complex and demands a large amount of power and memory from the connecting devices. The IoT devices can also connect locally through non-IP networks, which consume less power, and connect to the Internet via a smart gateway. Non-IP communication channels such as Bluetooth,

RFID, and NFC are fairly popular but are limited in their range (up to a few meters). Therefore, their applications are limited to small personal area networks. Personal area networks (PAN) are being widely used in IoT applications such as wearables connected to smartphones. For increasing the range of such local networks, there was a need to modify the IP stack so as to facilitate low power communication using the IP stack.

The leading communication technologies used in the IoT world are IEEE 802.15.4, low power WiFi, 6LoWPAN, RFID, NFC, Sigfox, LoraWAN, and other proprietary protocols for wireless networks.

IoT Network Protocol Stack

The Internet Engineering Task Force (IETF) has developed alternative protocols for communication between IoT devices using IP because IP is a flexible and reliable standard. The Internet Protocol for Smart Objects (IPSO) Alliance has published various white papers describing alternative protocols and standards for the layers of the IP stack and an additional adaptation layer, which is used for communication between smart objects.



1. **Physical and MAC Layer (IEEE 802.15.4)**. The IEEE 802.15.4 protocol is designed for enabling communication between compact and inexpensive low power embedded devices that need a long battery life. It defines standards and protocols for the physical and link (MAC) layer of the IP stack. It supports low power communication along with low cost and short range communication. In the case of such resource constrained environments, we need a small frame size, low bandwidth, and low transmit power.

   Transmission requires very little power (maximum one milliwatt), which is only one percent of that used in WiFi or cellular networks. This limits the range of communication. Because of the limited range, the devices have to operate cooperatively in order to enable multihop routing over longer distances. As a result, the packet size is limited to 127 bytes only, and the rate of communication is limited to 250 kbps. The coding scheme in IEEE 802.15.4 has built in redundancy, which makes the communication robust, allows us to detect losses, and enables the retransmission of lost packets. The protocol also supports short 16-bit link addresses to decrease the size of the header, communication overheads, and memory requirements.

2. **Adaptation Layer**. IPv6 is considered the best protocol for communication in the IoT domain because of its scalability and stability. Such bulky IP protocols were initially not thought to be suitable for communication in scenarios with low power wireless links such as IEEE 802.15.4.

   6LoWPAN, an acronym for IPv6 over low power wireless personal area networks, is a very popular standard for wireless communication. It enables communication using IPv6 over the IEEE 802.15.4 protocol. This standard defines an adaptation layer between the 802.15.4 link layer and the transport layer. 6LoWPAN devices can communicate with all other IP based devices on the Internet. The choice of IPv6 is because of the large addressing space available in IPv6. 6LoWPAN networks connect to the Internet via a gateway (WiFi or Ethernet), which also has protocol support for conversion between IPv4 and IPv6 as today's deployed Internet is mostly IPv4. IPv6 headers are not small enough to fit within the small 127 byte MTU of the 802.15.4 standard. Hence, squeezing and fragmenting the packets to carry only the essential information is an optimization that the adaptation layer performs.

3. **Network Layer**. The network layer is responsible for routing the packets received from the transport layer. The IETF Routing over Low Power and Lossy Networks (ROLL) working group has developed a routing protocol (RPL) for Low Power and Lossy Networks (LLNs).

   For such networks, RPL is an open routing protocol, based on distance vectors. It describes how a destination oriented directed acyclic graph (DODAG) is built with the nodes after they exchange distance vectors. A set of constraints and an objective function is used to build the graph with the best path. The objective function and constraints may differ with respect to their requirements. For example, constraints can be to avoid battery powered nodes or to prefer encrypted links. The objective function can aim to minimize the latency or the expected number of packets that need to be sent.

4. **Transport Layer**. TCP is not a good option for communication in low power environments as it has a large overhead owing to the fact that it is a connection oriented protocol. Therefore, UDP is preferred because it is a connectionless protocol and has low overhead.
5. **Application Layer**. The application layer is responsible for data formatting and presentation. The application layer in the Internet is typically based on HTTP. However, HTTP is not suitable in resource constrained environments because it is fairly verbose in nature and thus incurs a large parsing overhead. Many alternate protocols have been developed for IoT environments such as CoAP (Constrained Application Protocol) and MQTT (Message Queue Telemetry Transport).

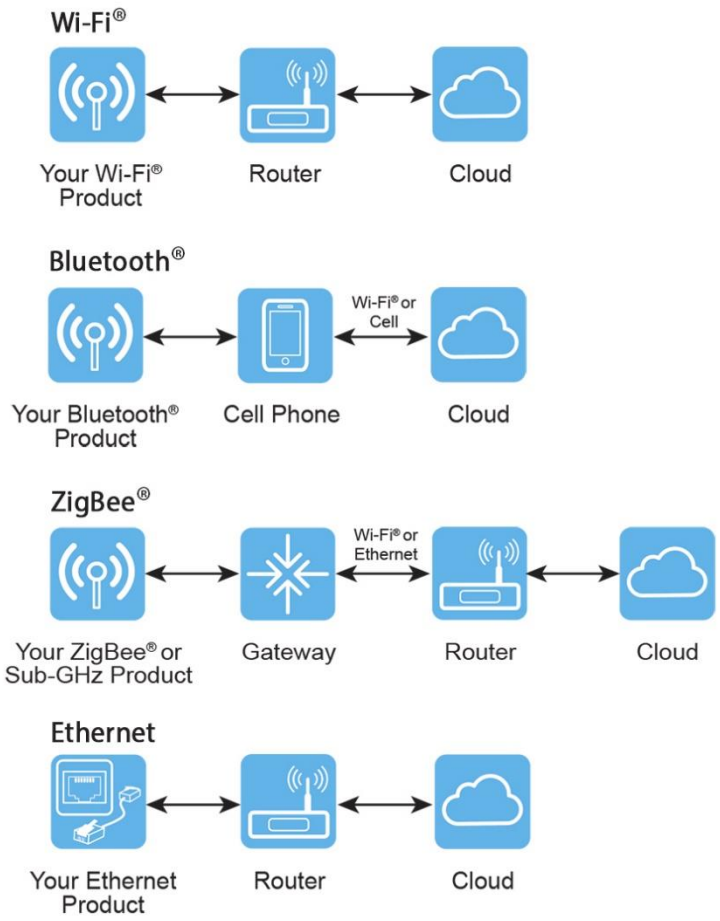
Low Power WiFi

The WiFi alliance has recently developed "WiFi HaLow," which is based on the IEEE 802.11ah standard. It consumes lower power than a traditional WiFi device and also has a longer range. This is why this protocol is suitable for Internet of Things applications. The range of WiFi HaLow is nearly twice that of traditional WiFi.

Like other WiFi devices, devices supporting WiFi HaLow also support IP connectivity, which is important for IoT applications. Let us look at the specifications of the IEEE 802.11ah standard. This standard was developed to deal with wireless sensor network scenarios, where devices are energy constrained and require relatively long range communication. IEEE 802.11ah operates in the sub-gigahertz band (900 MHz). Because of the relatively lower frequency, the range is longer since higher frequency waves suffer from higher attenuation. We can extend the range (currently 1 km) by lowering the frequency further; however, the data rate will also be lower and thus the tradeoff is not justified. IEEE 802.11ah is also designed to support large star shaped networks, where a lot of stations are connected to a single access point.

Watt? Networking

Due to the popularity of WiFi for all people, Watt? uses it to connect to the Internet. Every Watt? device has a built-in WiFi supported microcontroller. It needs to connect to a WiFi router to start working properly. Setting up a Home WiFi network to work with our product Watt? is very easy comparing to other networking protocols.



WiFi

WiFi is simply an alternative to network cables as a way to connect devices of a local area network (LAN). Prior to WiFi the only way to connect devices together was to run physical network cables between them, which can be inconvenient. WiFi allows devices to connect to one another the same way as when network cables are used, just without the actual cables. A WiFi network is basically a wireless local network.

The owner of the WiFi network is in total control. She can change the name of the network, the password, the number of connected clients, allowing them to exchange data with one another or not, and so on. Even the WiFi router or access point itself can be changed or turned on or off any time.

A home WiFi network, which is almost always hosted by a router, is independent from the Internet. Meaning that any devices on the network can always work with one another to share and back up data, print, stream local media and so on. A connection to the Internet, however, enables them to also access Internet-based services, such as Skype, Netflix, news, Facebook, Twitter and other services.

To connect a home WiFi network to the Internet, the router needs to be connected to an Internet source, such as a broadband modem, via its WAN port. When this link is complete, the WiFi signal of the local network will also provide a connection to the Internet for any device connected to the network. So WiFi is just one way to bring the Internet to a device. And this also explains why sometimes our WiFi signal is at full strength, yet we still can't access the Internet.

Internet

Generally known as the wide area network (WAN), the Internet connects computers from around the world. In reality, the Internet actually connects many local networks together, via a ton of routers. With the Internet, our home local network is no longer secluded but becomes part of one giant worldwide network.

The Internet connection is generally beyond the control of the users. Other than turning it on or off, the only other thing we can do is pay for the desired connection speed and hope we get what we pay for. Internet speed has progressively increased in the last decade. Ten years ago, a fast residential broadband connection generally capped somewhere between 1.5Mbps to 20Mbps; now it's between about 50Mbps to 150Mbps and even faster.

That said, most of the time, the speed of the Internet is still slower than that of a wired local network, which is either 100Mbps or 1,000Mbps. For a WiFi network, the speed of the local network depends on the standards used by the WiFi router (or access point) and the connected clients, and can sometimes be slower than a fast broadband wired Internet connection.

ESP8266 WiFi Module

As mentioned before, Watt? uses the *ESP8266 WiFi Module* as the core control and communication device.

The ESP8266 WiFi Module is a self contained SOC with integrated TCP/IP protocol stack that can give any microcontroller access to our WiFi network. The ESP8266 is capable of either hosting an application or offloading all WiFi networking functions from another application processor. Each ESP8266 module comes pre-programmed with an AT command set firmware, meaning, we can simply hook this up to an Arduino device and get a WiFi-ability very easily. The ESP8266 module is an extremely cost effective board with a huge, and ever growing, community.

This module has a powerful enough on-board processing and storage capability that allows it to be integrated with the sensors and other application specific devices through its GPIOs with minimal development up-front and minimal loading during runtime. Its high degree of on-chip integration allows for minimal external circuitry, including the front-end module, is designed to occupy minimal PCB area. The ESP8266 supports APSD for VoIP applications and Bluetooth co-existance interfaces, it contains a self-calibrated RF allowing it to work under all operating conditions, and requires no external RF parts. There is an almost limitless fountain of information available for the ESP8266, all of which has been provided by amazing community support.

Chapter 5

Hardware Layer

Introduction

Internet of Things (IoT) is a sprawling set of technologies and use cases that has no clear, single definition. One workable view frames IoT as the use of network-connected devices, embedded in the physical environment, to improve some existing process or to enable a new scenario not previously possible.

These devices, or *things*, connect to the network to provide information they gather from the environment through sensors, or to allow other systems to reach out and act on the world through actuators. They could be connected versions of common objects we might already be familiar with, or new and purpose-built devices for functions not yet realized. They could be devices that we own personally and have on our persons or in our homes, or they could be embedded in factory equipment, or part of the fabric of the city we live in. Each of them is able to convert valuable information from the real world into digital data that provides increased visibility into how our users interact with our products, services, or applications.

The specific use cases and opportunities across different industries are numerous, and in many ways the world of IoT is just getting started. What emerges from these scenarios is a set of common challenges and patterns. IoT projects have additional dimensions that increase their complexity when compared to other cloud-centric technology applications, including:

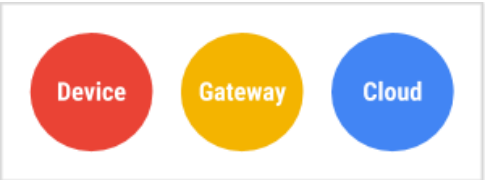
- Diverse hardware.
- Diverse operating systems and software on the devices.
- Different network gateway requirements.

This chapter explains the elements we can combine with Firebase Cloud Platform to build a robust, maintainable and end-to-end IoT solution on Cloud Platform.

Overview of the top level components

Here we divide the system into three basic components, the device, gateway, and cloud:

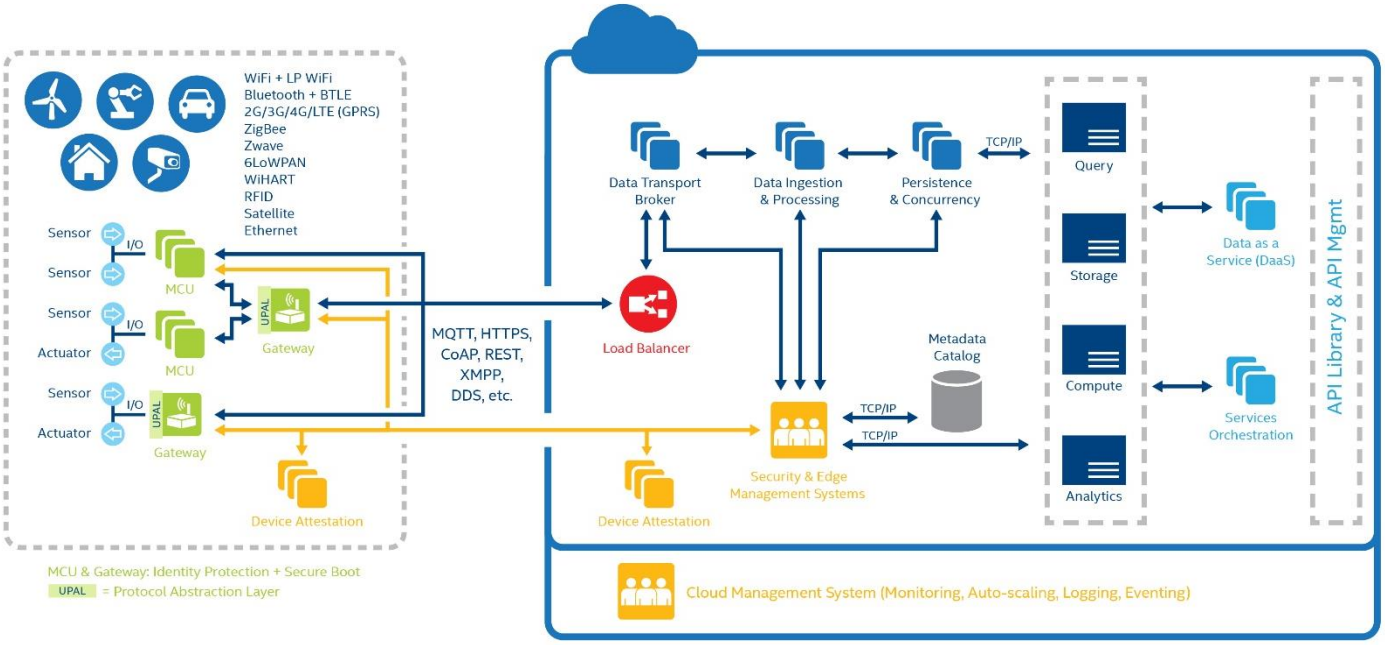
A *device* includes hardware and software that directly interacts with the world. Devices connect to a network to communicate with each other, or to centralized applications. Devices might be directly or indirectly connected to the Internet.



A *gateway* enables devices that are not directly connected to the Internet to reach cloud services. Although the term *gateway* has a specific function in networking, it is also used to describe a class of device that processes data on behalf of a group or cluster of devices. The data from each device is sent to Cloud Platform, where it is processed and combined with data from other devices, and potentially with other business-transactional data.[8]

Internet Of Things Architecture

Here is the detailed architecture design of the cloud-based Internet Of Things according to the top level components.[9]



We have discussed in details the the *Cloud* and the *Gateway* components in the previous chapters. Here we will talk about the *Device* component at the left side of the previous figure. This chapter explains in details the construction of the *Device* component in the cloud-based Internet Of Things project for Watt?.

Devices

It's not always clear what constitutes a device. Many physical things are modular, which means it can be hard to decide whether the whole machine is the device, or each module is a discrete device. There's no single, right answer to this question. As we design our IoT project, we'll need to think about the various levels of abstraction in our design and make decisions about how to represent the physical things and their relationships to each other.[8]

Embedded Systems

We have introduced the concept of the Internet of Things at a high level, defining the term and outlining its implications. Here we explore some of the details involved in the design and implementation of IoT devices. Unlike traditional computer-based systems, IoT devices are "embedded" within other devices in order to provide enhanced functionality without exposing the user to the complexities of a computer. The users interact with the device in a natural way, similar to their interactions with any other objects in the world. In this way, an embedded system has an interface that conforms to the expectations and needs of the users. Establishing a natural interface requires that the embedded system interface with the physical world directly through sensors, which read the state of the world, and actuators, which change the state of the world. Now we will discuss the structure of embedded systems and describe these interactions with the physical world.

As its name suggests, Embedded means something that is attached to another thing. An embedded system can be thought of as a computer hardware system having software embedded in it. An embedded system can be an independent system or it can be a part of a large system. An embedded system is a microcontroller or microprocessor based system which is designed to perform a specific task. For example, a fire alarm is an embedded system; it will sense only smoke.

Characteristics of an Embedded System

- **Single-functioned** An embedded system usually performs a specialized operation and does the same repeatedly. For example: A pager always functions as a pager.
- **Tightly constrained** All computing systems have constraints on design metrics, but those on an embedded system can be especially tight. Design metrics is a measure of an implementation's features such as its cost, size, power, and performance. It must be of a size to fit on a single chip, must perform fast enough to process data in real time and consume minimum power to extend battery life.
- **Reactive and Real time** Many embedded systems must continually react to changes in the system's environment and must compute certain results in real time without any delay.
- **Microprocessors based** It must be microprocessor or microcontroller based.
- **Memory** It must have a memory, as its software usually embeds in ROM. It does not need any secondary memories in the computer.
- **Connected** It must have connected peripherals to connect input and output devices.
- **Hardware/Software systems** Software is used for more features and flexibility. Hardware is used for performance and security.[10]

Basic Structure of an Embedded System

Most of the Embedded systems found are composed of a **Hardware** and a **Software** embedded on the that hardware. So we can define an embedded system as a computer based, software driven, reliable and real-time control system.

Embedded Hardware

Embedded Hardware Design and Development is an integral part of product development. These services are intended to complement the embedded systems design and software design and development services offered. This hardware design is for microcontrollers and microprocessors or smaller FPGA systems. The hardware design, both mechanical and electronics, is done in parallel with software development.

Embedded System Hardware Design Overview

The basic process starts with conceptual design during the embedded systems design and requirements specification phase. In this activity the requirements and conceptual design is explored further to ensure suitability for the final implementation.[11]

General considerations when choosing hardware

When choosing hardware, consider the following factors, which are affected by how the hardware is deployed:

- **Cost** Given the value of the data provided, think about what cost can be supported for each device.
- **I/O roles** The device might be primarily a sensor, an actuator, or some combination of the two roles.
- **Power budget** The device might have access to electricity, or power might be scarce. Think about whether the device will require battery or solar power.
- **Networking environment** Consider whether the device can be wired directly to the Internet as TCP/IP routable. Some types of connections, such as cellular, can be expensive with high traffic. Think about the reliability of the network, and the impact of that reliability on latency and throughput. If it is wireless, consider the range the transmission power achieves and the added energy costs.

Functional inputs and outputs

The devices used to interact with the physical world contain components, or are connected to peripherals, that enable sensor input or actuator output. The specific hardware we choose for these hardware I/O components should be based on the functional requirements. For example, the sensitivity or Intensity of the current we need to measure will determine what kind of current sensor we choose, or whether we need a current transformer instead. When using a device to produce output, we must consider requirements such as how loud a buzzer needs to sound, how fast a motor needs to turn, or how many amps a relay needs to carry.

In addition to the requirements determined by the environmental performance, the choice of these I/O components or peripherals might also be related to the type of information they are associated with. For example, a stepper motor can be set to a specific direction that might be represented in device state data, while a microphone might be steadily sampling data in terms of

frequencies, which is best transmitted as telemetry. These components are connected to the logic systems of the device through a hardware interface.[12]

Device platforms

There is an incredible amount of diversity in the specific hardware available to us for building IoT applications. This diversity starts with the options for hardware platforms. Common examples of platforms include single-board-computers such as the Beaglebone, Raspberry Pi, and Intel Edison as well as microcontroller platforms such as the Arduino series, ESP Chips, boards from Particle, and the Adafruit Feather.

Each of these platforms lets us connect multiple types of sensor and actuator modules through a hardware interface. These platforms interface with the modules using a layered approach similar to those used in general-purpose computing. If we think about the common, everyday computer mouse, we can consider the layers of peripheral, interface, driver and application. On a typical operating system, such as Linux or Windows, the hardware input is interpreted by a driver, which in turn relies on OS services, and might be part of the kernel.

Watt? Hardware components

To create product concepts that will stand out from the competition, a variety of components are implemented. The world of IoT sensors and hardware is vast and incorporates a variety of components with varying qualities, sizes and functions. Here we categorize some of the major components commonly used to create Watt? products.

Infrastructure components

- **Microcontroller** The processor of the electronic board. This component manages communication and transmits the commands in the circuit.

  Watt? Project uses the ESP8266 WiFi module. It has a built-in microcontroller, its features are :

  - 32-bit RISC CPU: Tensilica Xtensa L106 running at 80 MHz
  - 64 KiB of instruction RAM, 96 KiB of data RAM
  - External QSPI flash: 512 KiB to 4 MiB (up to 16 MiB is supported)
  - IEEE 802.11 b/g/n WiFi
    - Integrated TR switch, balun, LNA, power amplifier and matching network
    - WEP or WPA/WPA2 authentication, or open networks
  - 16 GPIO pins
  - SPI
  - I^2C
  - I^2S interfaces with DMA (sharing pins with GPIO)
  - UART on dedicated pins, plus a transmit-only UART can be enabled on GPIO2
  - 10-bit ADC (this is a Successive Approximation ADC)

- **Memory** The device that stores information applied by the microcontroller to operate the system and can contain information used by different components in the system. In some cases, the memory forms part of the microcontroller chip; and in cases where more memory is required, it can form a separate component. The ESP8266 WiFi Modules of Watt? have an external QSPI 4 MiB flash.
- **Communication device** The component that manages the connection to a network – making the product 'smart'. The communication could take the form of one of a few protocols. for example, through a Bluetooth or WiFi connection. The communication module could form part of the microcontroller as in our case for the ESP8266 microcontroller chip.

Power supply components

- **Electricity network** Usually a viable option if the product is stationary and requires higher voltage and current. This is typically the case when applied to heating elements, heavy motors or other mechanical components and products that are in constant operation mode. In the case of Watt?, its devices are already wired to the home electricity network to measure the power consumption. So it would be better to use this the network as a power source.

To provide Watt? devices with the 5 DC voltage they needs, We have added a power supply module to the circuit.

The circuit is operated from the AC mains from 110V to 220V 50/60 Hz which is then converted to DC voltage by the AC-to-DC module with an output of 5 VDC and output current up to 600mA. It is the HLK-PM01 from Hi-Link. One Advantage of the HLK-PM01 is the small footprint 34x20x15 cm.

- **Voltage Regulator** The 5VDC is required for the circuit operation, although the WiFi module is working at 3.3VDC so a small 3.3V voltage regulator is required.

So, 3.3V LM1117 LDO with a TO-220 package which has also very small footprint is used in the circuit.

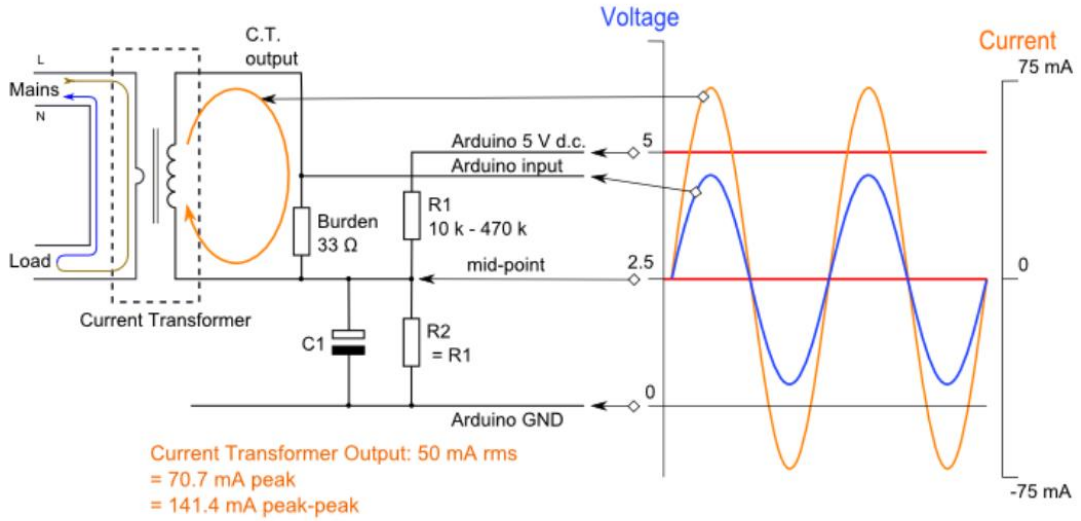
Current Sensors

Two types of current sensor are used in Watt? Project. One for the smart meter and the other for the smart plug.

- **Current Transformer** A current transformer (CT) is a type of transformer that is used to measure AC Current. It produces an alternating current (AC) in its secondary which is proportional to the AC current in its primary. A big advantage of CT is that the metering circuit is insulated from the primary high current system.

The CT used has a turns ratio of 1:0.0005 with a maximum current of 100 A which is sufficiently enough for household applications.

Theo output voltage of the current transformer is then fed to an Analog to Digital Converter (ADC) with 10-bit accuracy embedded inside the ESP8266 WiFi module. As the output of the CT is a sin wave which varies between positive and negative value, and the ADC has a Vref of 0 volts, so an interface circuit is a necessity. The interface circuit is composed of 2 series resistors which acts as voltage divider with an output of 2.5 VDC which is required to shift the CT AC output voltage.

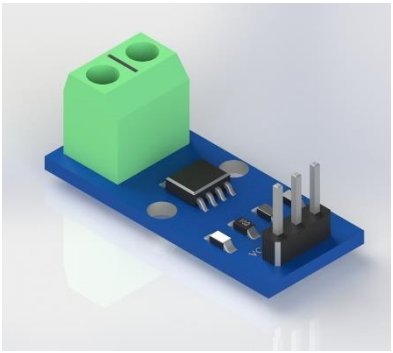


Then the calculated consumed power is sent to a cloud server to be accessed any where.

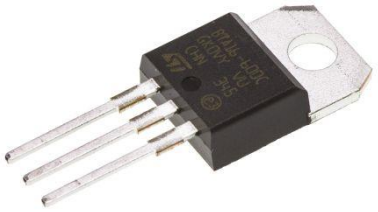
- **ACS712 Current Sensor** A Hall Effect based current sensor is used to measure the consumed current by the devices attached to the smart plug with a maximum current measurement of 20 A.

Allegro's ACS712-20A provides economical and precise solutions for AC or DC current sensing in industrial, commercial, and communications systems. The device package allows for easy implementation. It is provided in a small, surface mount SOIC8 package.

A **current sensor module** based on the ACS712 IC could not be found in Egypt rather than the absence of soldering instruments required to solder such an small SMD IC to a printed circuit board PCB. Which if used, will assist in reducing the total size of the printed circuit board of Watt? Smart Plug.



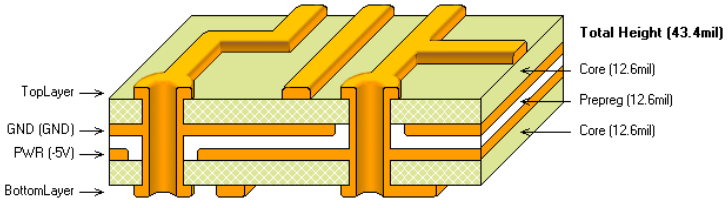
- **Control Circuit** Watt? Smart Plug is capable to control any AC device up to 3.5 KW, a medium current snubberless Triac is used for the purpose of AC controlled switch with up to 16 A on-state RMS current which fits ideally with household applications.



ST Microelectronics's BTA16 also features a very high Repetitive peak off-state voltage of 600 V and small Triggering gate current of 50 mA. A microcontroller GPIO pin is used in corporation with a PNP transistor to supply enough triggering current to switch the Triac ON or OFF. An Indication green LED also is used to indicate the state of the device to the user.

Printed Circuit Board Design and Layout

A printed circuit board (PCB) mechanically supports and electrically connects electronic components using conductive tracks, pads and other features etched from copper sheets laminated onto a non-conductive substrate. Components (e.g. capacitors, resistors or active devices) are generally soldered on the PCB. Advanced PCBs may contain components embedded in the substrate.



PCBs can be single sided (one copper layer), double sided (two copper layers) or multi-layer (outer and inner layers). We used double sided board for smaller overall footprint. Conductors on different layers are connected with vias. Multi-layer PCBs allow for much higher component density.

FR-4 glass epoxy is the primary insulating substrate. A basic building block of the PCB is an FR-4 panel with a thin layer of copper foil laminated to one or both sides. In multi-layer boards multiple layers of material are laminated together.

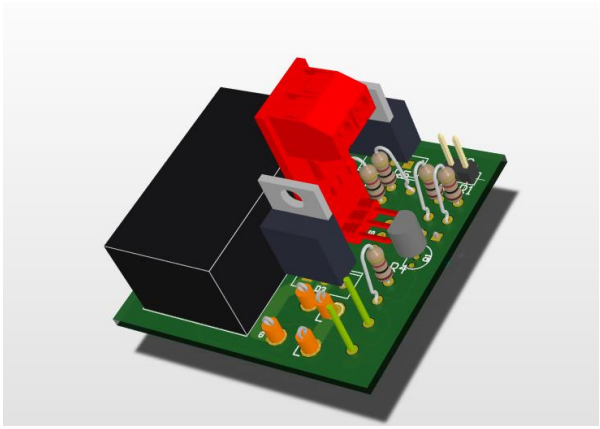
PCB Design using EDA software

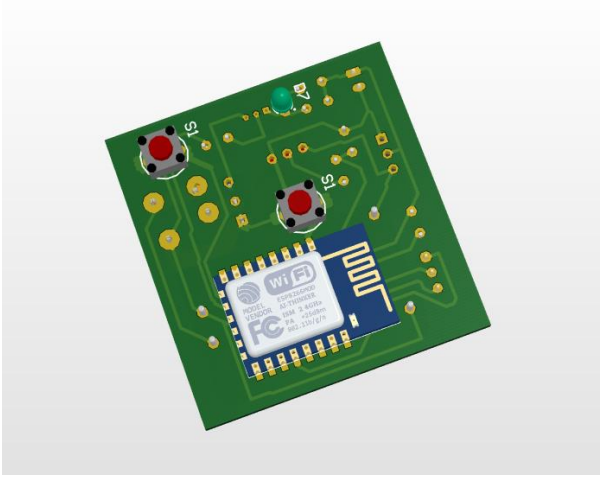
We opted for using high technology, free and community based EDA called CircuitMaker.

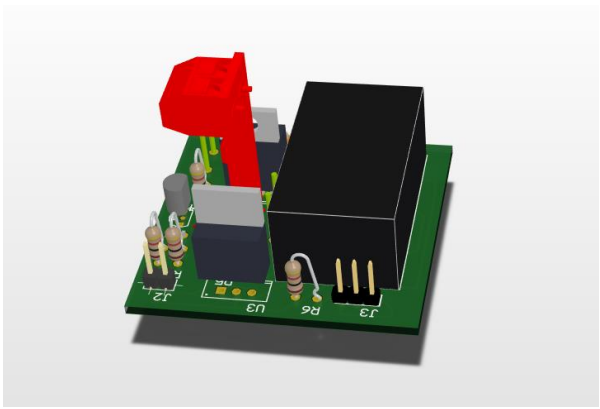
CircuitMaker is the best free-to-use schematic and PCB design tool for the Open Source Hardware community built by Altium, so we can have complete confidence in the technology that drives most of the world's PCB design activity. That's why they have included powerful routing, hierarchical schematic entry, auto-routing, and Native 3D™ technology.

PCB layout and component placing

Accurate component placement is made to shrink the PCB size to the minimum. Also components' size chosen accurately to pick the smallest size components, so we moved to SMD components rather than through hole components as shown in figures bellow:







PCB production Capabilities

Items	Manufacturing Capabilities
Number of Layers	1-10 layers
Material	FR-4 (King Board)
Maximum PCB Size(Dimension)	500x1100 mm
Board Size Tolerance(Outline)	±0.2mm/±0.5mm
Board Thickness	0.4-2.4mm
Board Thickness Tolerance(t≥1.0mm)	±10%
Board Thickness Tolerance(t<1.0mm)	±0.1mm
Min Trace	0.1mm/4mil
Min Spacing	Min manufacturable spacing is 4mil (0.1mm), strongly suggest to design spacing above 6mil (0.15mm) to save cost.
Outer Layer Copper Thickness	1oz/2oz/3oz(35µm/70µm/105µm)
Inner Layer Copper Thickness	1oz/1.5oz(35µm/50µm)
Drill Sizes (CNC)	0.2-6.3mm
Min Width of Annular Ring	0.15mm(6mil)
Finished Hole Diameter (CNC)	0.2mm-6.2mm
Finished Hole Size Tolerance(CNC)	±0.08mm
Solder Mask	LPI
Minimum Character Width(Legend)	0.15mm
Minimum Character Height (Legend)	0.8mm
Character Width to Height Ratio (Legend)	1:5
Minimum Diameter of Plated Half Holes	0.6mm
Surface Finishing	HASL with leadHASL lead freeImmersion gold
Solder Mask	Green ,Red, Yellow, Blue, White ,Black
Silkscreen	White, Black, None

PCB Manufacturing Challenges

There are a lot of challenges have faces us in manufacturing the printed circuit board:

- High Cost by the local manufacturers in Egypt.
- No high quality manufacturers found in Egypt.
- High customs if manufactured abroad in china.
- Preventing the manufactured products to arrive, from the national security in Egypt.
- Take long time to arrive if manufactured abroad.
- Bad quality of hand-made PCBs.
- Bigger size when manufactured in home.

Embedded Software

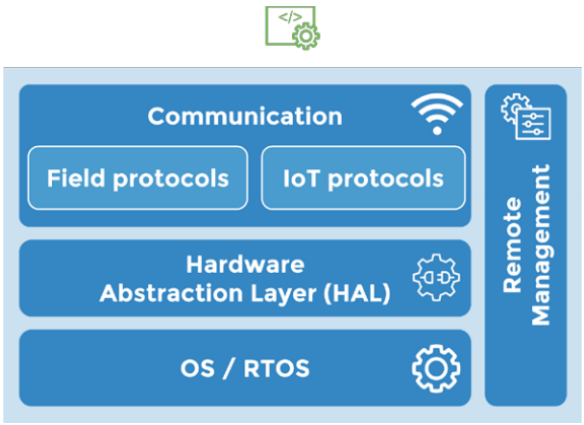
Internet of Things devices are implemented using both hardware and software components. Dedicated hardware components are used to implement the interface with the physical world, and to perform tasks which are more computationally complex. Microcontrollers are used to execute software that interprets inputs and controls the system. Now we will discusses the roles of the software components in the system. The functions of common hardware components are described and the interface between the software and hardware through the microcontroller will be explained. IoT devices often use an operating system to support the interaction between the software and the microcontroller.[13]

The "Thing" in the IoT is the starting point for an IoT solution. It is typically the originator of the data, and it interacts with the physical world. Things are often very constrained in terms of size or power supply; therefore, they are often programmed using microcontrollers (MCU) that have very limited capabilities. The microcontrollers powering IoT devices are specialized for a specific task and are designed for mass production and low cost.

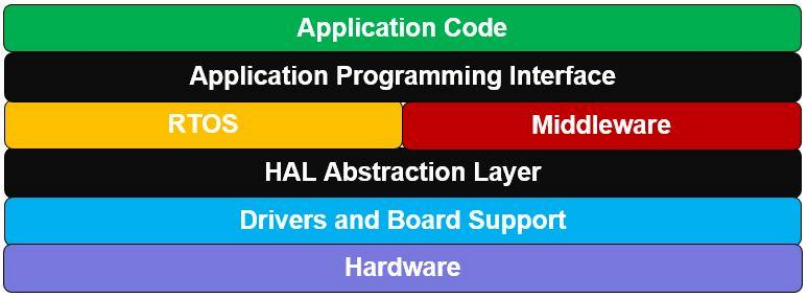
The Software Stack

The software running on MCU-based devices aims at supporting specific tasks. The key features of the software stack running on a device may include:

- **IoT Operating System** many devices like Watt? will run with 'bare metal' using on of the embedded software architectures below, but some will have embedded or real time operating systems that are particularly suited for small constrained devices, and that can provide IoT-specific capabilities.



- **Hardware Abstraction** Hardware Abstraction Layer (HAL) is a software layer that enables access to the hardware features of the MCU, such as flash memory, GPIOs, serial interfaces, etc. It provides function API-based service to the higher-level layers (ex: Application Framework, customer application, Et cetera) that allows them to perform hardware oriented operations independent of actual hardware details.[14]



An operating system abstracts common computing resources such as memory and file I/O. The OS also provides very low-level support for the different hardware interfaces. Generally these abstractions are not easy to use directly, and frequently the OS does not provide abstractions for the wide range of sensor and actuator modules you might encounter in building IoT solutions.

You can take advantage of libraries that abstract hardware interfaces across platforms. These libraries enable you to work with a device, such as a motion detector, in a more straightforward way. Using a library lets you focus on collecting the information the module provides to your application instead of on the low-level details of working directly with hardware.

Some libraries provide abstractions that represent peripherals in the form of lightweight drivers on top of the hardware interfaces. Examples of these libraries include the EMBD Go library, Arduino-wiring, Arduino-ESP8266 and Firmata.[12]

- **Communication Support** Connectivity, it is one of the main things to keep in mind while developing any Internet of Things project. Many communication technologies are well known such as WiFi, Bluetooth, ZigBee and 2G/3G/4G cellular. Depending on the application, factors such as range, data requirements, security and power demands and battery life will dictate the choice of one or some form of combination of technologies. These are some of the major communication technologies on offer to developers. We have introduced more details on IoT Networking in the previous chapter.
- **Remote Management** the ability to remotely control the device to upgrade its firmware or to monitor its battery level. [16]

Software Architectures for Embedded Systems

Software architecture, according to ANSI/IEEE Standard 1471-2000, is defined as the "fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution." Embedded software, as we have said, must interact with the environment through sensors and actuators, and often has hard, real-time constraints.

The organization of the software, or its architecture, must reflect these realities. Usually, the critical aspect of an embedded control system is its speed of response which is a function of (among other things) the processor speed and the number and complexity of the tasks to be

accomplished, as well as the software architecture. Clearly, embedded systems with not much to do, and plenty of time in which to do it, can employ a simple software organization (a vending machine, for example, or the power seat in your car). Systems that must respond rapidly to many different events with hard real-time deadlines generally require a more complex software architecture (the avionics systems in an aircraft, engine and transmission control, traction control and anti-lock brakes in your car). Most often, the various tasks managed by an embedded system have different priorities: Some things have to be done immediately (fire the spark plug precisely 20◦ before the piston reaches top-dead-center in the cylinder), while other tasks may have less severe time constraints (read and store the ambient temperature for use in a calculation to be done later).

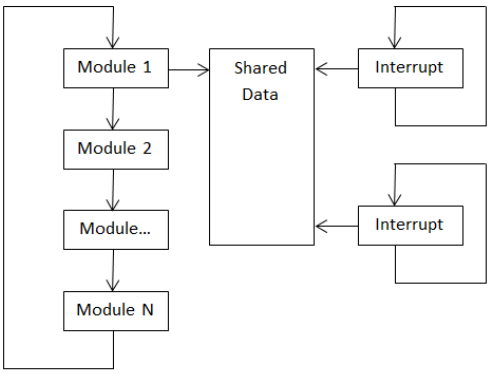
The four software architectures that will be discussed in the following sections:

- **Round robin**
- **Round robin with interrupts**
- **Function queue scheduling**
- **Realtime operating systems**

Watt? Software Architecture

Watt? is a simple embedded system that doesn't take care of any critical constraints. By looking to the requirements of Watt? System, We could see that the most suitable architecture with low development cost is the *Round robin with interrupts* architecture.

Round robin architecture is the simplest possible software architecture; the software organization consists of one main loop wherein the processor simply polls each attached device in turn, and provides service if any is required. After all devices have been serviced, start over from the top again. One step up on the performance scale is *Round robin with interrupts*. Here, urgent tasks get handled in an interrupt service routine, possibly with a flag set for follow-up processing in the main loop. If nothing urgent happens (emergency stop button pushed, or intruder detected), then the processor continues to operate round robin, managing more mundane tasks in order around the loop.



The obvious advantage to round *Round robin with interrupts* is that the response time to high-priority tasks is improved, since the ISR always has priority over the main loop (the main loop will always stop whatever it's doing to service the interrupt), and yet it remains fairly simple. The worst case response time for a low priority task is the sum of the execution times for all of the code in the main loop plus all of the interrupt service routines.[17]

Computing environment

The computing environment of our platform executes the software. These computing environments are the bridge between the logic of your application code and the physical hardware of the platform. Based on the hardware constraints of power and cost, the capabilities of the processor will vary. Some computing environments consist of a full system on a chip (SOC), which can support an embedded Linux operating system. Microcontroller-based devices might be more constrained, and your application code could run directly on the processor without the support of an operating system.[12]

Our computing environment is a full WiFi MCU SOC attached with an External 4MiB EEPROM. It has the Ability to connect to any WiFi network and also to run a WiFi Hotspot all beside executing the embedded software code at the same time. The ESP8266 WiFi MCU provides us the computing and the connectivity in a very small, low power, Very cheap, single chip.

Hardware Interfaces

IoT hardware platforms use a number of common interfaces. Most hardware interfaces are serial interfaces. Serial interfaces generally use multiple wires to control the flow and timing of binary information along the primary data wire. Each type of hardware interface defines a method of communicating between a peripheral and the central processor. Sensor and actuator modules can support one or more of these interfaces. The utilized interfaces in Watt? Project are:

- **GPIO**. General-purpose input/output pins are connected directly to the processor. As their name implies, these pins are provided by the manufacturer to enable custom usage scenarios that the manufacturer didn't design for. GPIO pins can be designed to carry digital or analog signals.
    - *Digital GPIO* are used in Watt? Smart Plug to control the the plugged appliance that is wired to a Triac circuit through an on-board push button. The button has a second hidden function when it is pushed for five seconds to clear the saved WiFi configurations (Network SSID and Password). This second function is implemented also in Watt? Smart Meter.
    - *Analog pins* have access to an on-board analog-to-digital conversion (ADC) circuit. An ADC periodically samples a continuous, analog waveform, such as an analog audio signal, giving each sample a digital value between zero and one, relative to the system voltage. We use the analog pin to connect the current sensor on it to get and calculate the current value that the sensor has read.

  When you read the value of a digital I/O pin in code, the value can must be either HIGH or LOW, where an analog input pin at any given moment could be any value in a range. The range depends on the resolution of the ADC. For example a 10-bit ADC can yield a wide range of values, from 0 to 1024. More values means higher resolution and thus a more faithful digital representation of any given analog signal.

  The ADC *sampling rate* determines the frequency range that an ADC can reproduce. A higher sampling rate results in a higher maximum frequency in the digital data. For example, an audio signal sampled at 44,100 Hz produces a digital audio file with a

frequency response up to 22.5 kHz, ignoring typical filtering and other processing. The *bit precision* dictates the resolution of the amplitude of the signal.

- **SPI**. Serial Peripheral Interface Bus devices employ a master-slave architecture, with a single master and full-duplex communication. SPI specifies four logic signals:
  - SCLK: Serial Clock, which is output from the master
  - MOSI: Master Output Slave Input, which is output from the master
  - MISO: Master Input Slave Output, which is output from a slave
  - SS: Slave Select, which is an active-low signal output from master

  In Watt? Devices, SPI is connected to the external EEPROM that they use so save permanent data.

- **UART**. Universal Asynchronous Receiver/Transmitter devices translate data between serial and parallel forms at the point where the data is acted on by the processor. UART is required when serial data must be laid out in memory in a parallel fashion. UART is used for the debugging purposes, where some messages are printed while the code is running. We also need the UART to flash the core MCU with the embedded software.

On-device processing

After data is collected from a sensor, the device can provide data processing functionality before sending the data to the cloud. Processing includes things like:

- Converting data from ADC output format to a typical current value
- Calculating the realtime power consumption
- Validating data to ensure it meets a set of rules
- Handling the external interrupts
- Reading the Internet time
- Calculating the cumulative energy consumptions
- Storing data in the EEPROM
- Combining data into aggregate values

Appendix

- The full source code for Watt? Smart Meter can be found on Github at https://github.com/elzoughby/Watt-SmartMeter

- The full source code for Watt? Smart Plug can be found on Github at https://github.com/elzoughby/Watt-SmartPlug

- All the book resource files and images can be found on Github at https://github.com/elzoughby/Watt-Book

Chapter 6

Development Management

Introduction

Project Management is the discipline of organizing and managing resources in such a way that the project is completed within defined scope, quality, time and cost constraints. This also applies to Software Projects.
According to the 10th edition of the annual CHAOS report from The Standish Group, only 34% of projects are completed successfully, and the main reason for this failure is poor project management.

Student projects within universities, such as graduation projects, lack good management of the development processes, especially projects that rely on software engineering. This is due to the students' lack of awareness of the need to manage the software projects as well as the weakness of their management experience resulting from lack of practice. This is what we have tried to avoid in Watt? Project.

Software Project Management
The goal of software project management is to understand, plan, measure and control the project such that it is delivered on time and on budget. This involves gathering requirements, managing risk, monitoring and controlling progress, and following a software development process.

Why does Software Fail?
Usually for these reasons:

- Unrealistic or unarticulated project goals
- Inaccurate estimates of needed resources
- Badly defined system requirements
- Poor reporting of the project's status

- Unmanaged risks
- Poor communication among supervisor, developers, and manager
- Use of immature technology
- Inability to handle the project's complexity
- Sloppy development practices
- Poor project management
- Stakeholder politics
- Commercial pressures

Therefore, the software engineer and the development team must choose the most appropriate way to work on the project and the best way to manage and coordinate the work among them.

Agile Software Development

"Agile Development" is an umbrella term for several iterative and incremental software development methodologies. The most popular agile methodologies include Extreme Programming (XP), Scrum, Crystal, Dynamic Systems Development Method (DSDM), Lean Development, and Feature-Driven Development (FDD).

Agile software development describes a set of principles for software development under which requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams. It advocates adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change. These principles support the definition and continuing evolution of many software development methods.

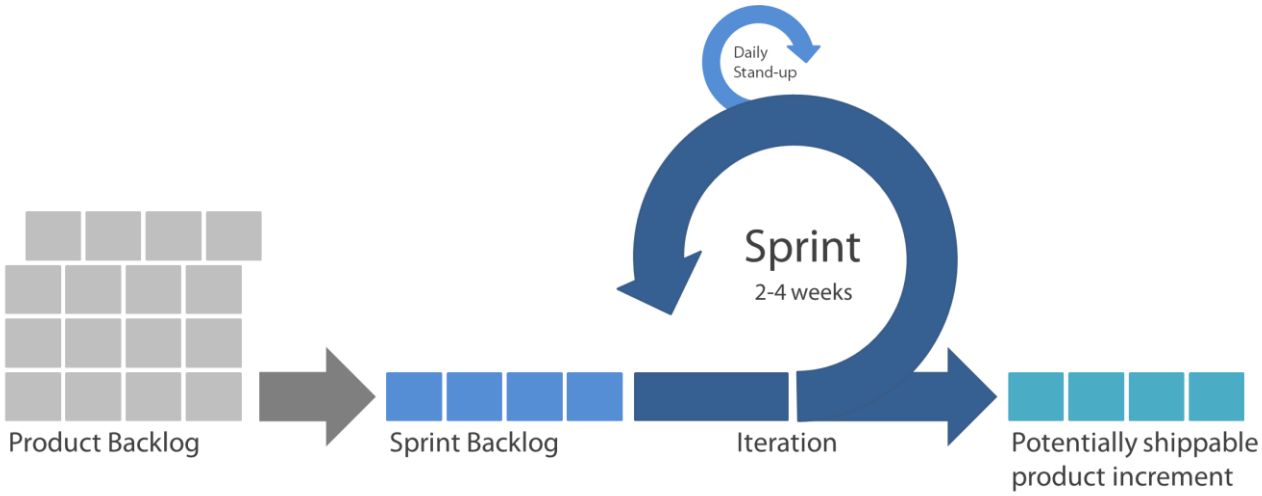
What is Scrum?

Scrum is an Agile framework for completing complex projects. Scrum originally was formalized for software development projects, but it works well for any complex, innovative scope of work. The possibilities are endless. The Scrum framework is deceptively simple.
Scrum is a process framework that has been used to manage complex product development since the early 1990s. Scrum is not a process or a technique for building products; rather, it is a framework within which you can employ various processes and techniques. Scrum makes clear the relative efficacy of your product management and development practices so that you can improve.

Scrum is:

- Lightweight
- Simple to understand
- Difficult to master

The Scrum framework consists of Scrum Teams and their associated roles, events, artifacts, and rules. Each component within the framework serves a specific purpose and is essential to Scrum's success and usage.



Why We Use Scrum?

Scrum has the power to transform project management across every industry, every business, and even across life in general; and

- Scrum is not unproven hype. It's a solid and successful Agile framework that's been applied to a variety of projects and teams.
- Team-based approach.
- Improve communication and maximize cooperation.
- Focus on the product more than documents.
- It has the flexibility and can remedy default.
- The Scrum team can adjust scrum with the working environment.

Scrum is not a Methodology

Scrum has no exhaustive and formal prescriptions on how to design and plan the behavior of all software development actors against time, let alone how these designs and plans must be documented and stored. Scrum has no rules for upfront predictions of document types and deliverables to be produced or the time of their production. Instead of installing and thriving on hand-overs, toll gates and control meetings like software development methodologies typically do, Scrum removes them as a major source of delays and waste.
Methodologies are composed of stringent and mandatory sequences of processes and procedures, implementing predefined algorithms. As such, methodologies tend to replace the creativity, autonomy and thinking of people with components like phases, tasks, must-do practices, techniques and tools. As long as the methodology is being followed everyone feels safe, because they are formally covered, even in the absence of working results. Methodologies depend on high degrees of predictability, otherwise the preset algorithms fail.

Scrum is the opposite of a big collection of interwoven mandatory components. Scrum is not a methodology. Scrum implements the scientific method of empiricism. Scrum replaces a programmed algorithmic approach with a heuristic one, with respect for people and self-organization to deal with unpredictability and solving complex problems.

Why is it Called Scrum?

When Jeff Sutherland created the scrum process in 1993, he borrowed the term "scrum" from an analogy put forth in a 1986 study by Takeuchi and Nonaka, published in the Harvard Business Review. In that study, Takeuchi and Nonaka compare high-performing, cross-functional teams to the scrum formation used by Rugby teams. Scrum is the leading agile development methodology, used by Fortune 500 companies around the world. The Scrum Alliance exists to transform the way we tackle complex projects, bringing the Scrum framework and agile principles beyond software development to the broader world of work.

Scrum Theory

Scrum is founded on empirical process control theory, or empiricism. Empiricism asserts that knowledge comes from experience and making decisions based on what is known. Scrum employs an iterative, incremental approach to optimize predictability and control risk.Three pillars uphold every implementation of empirical process control : transparency, inspection, and adaptation.

- **Transparency :** Significant aspects of the process must be visible to those responsible for the outcome.Transparency requires those aspects be defined by a common standard so observers share a common understanding of what is being seen.For example, Those performing the work and those accepting the work product must share a common definition of “Done”.
- **Inspection :** Scrum users must frequently inspect Scrum artifacts and progress toward a Sprint Goal to detect undesirable variances. Their inspection should not be so frequent that inspection gets in the way of the work. Inspections are most beneficial when diligently performed by skilled inspectors at the point of work.
- **Adaptation :** If an inspector determines that one or more aspects of a process deviate outside acceptable limits, and that the resulting product will be unacceptable, the process or the material being processed must be adjusted. An adjustment must be made as soon as possible to minimize further deviation.

Scrum Values

All work performed in Scrum needs a set of values as the foundation for the team's processes and interactions. And by embracing these five values, the team makes them even more instrumental to its health and success.

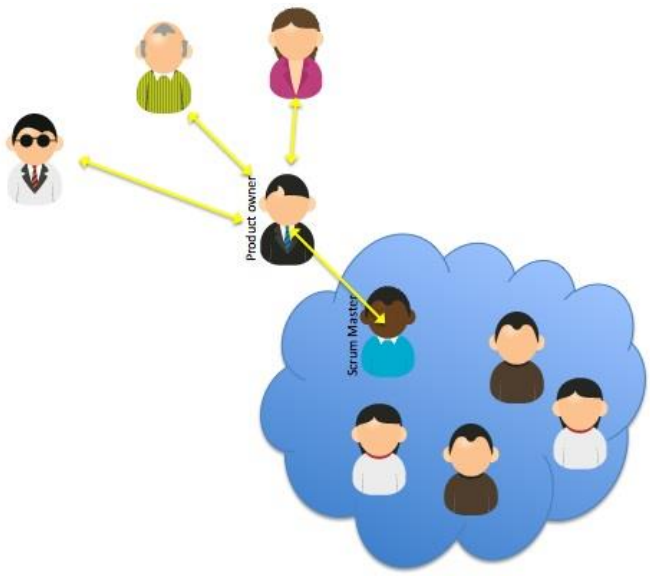
- **Focus :** Because we focus on only a few things at a time, we work well together and produce excellent work. We deliver valuable items sooner.
- **Courage :** Because we work as a team, we feel supported and have more resources at our disposal. This gives us the courage to undertake greater challenges.

- **Openness :** As we work together, we express how we're doing, what's in our way, and our concerns so they can be addressed.
- **Commitment :** Because we have great control over our own destiny, we are more committed to success.
- **Respect :** As we work together, sharing successes and failures, we come to respect each other and to help each other become worthy of respect.

Scrum Team

The Scrum Team consists of a Product Owner, the Development Team, and a Scrum Master. Scrum Teams are self-organizing and cross-functional. Self-organizing teams choose how best to accomplish their work, rather than being directed by others outside the team. Cross-functional teams have all competencies needed to accomplish the work without depending on others not part of the team. The team model in Scrum is designed to optimize flexibility, creativity, and productivity.

Scrum Teams deliver products iteratively and incrementally, maximizing opportunities for feedback. Incremental deliveries of "Done" product ensure a potentially useful version of working product is always available.



Within the Scrum Framework three roles are defined:

- Scrum Product Owner
- Development Team
- Scrum Master

Each of these roles has a defined set of responsibilities and only if they fulfill these responsibilities, closely interact and work together they can finish a project successfully.

Scrum Product Owner

The Scrum Product Owner is a central role within the Scrum Framework. Most of the responsibilities of the classical product manager and the project manager are combined within this single role.

He represents the end customer and/or other stakeholders and is responsible for maximizing the value of the product by ensuring that the right work is done at the right time.
As a consequence this means of course that the Scrum Product Owner has to work very closely with the Scrum Team and coordinates their activities over the whole lifetime of the project. No one else is allowed to tell the development team to work from a different set of priorities.

The Scrum Product Owner has a number of responsibilities:

- Managing the Scrum Product Backlog
- Release Management
- Stakeholder Management
- Work closely with the Scrum Team

The Product Owner may do the above work, or have the Development Team do it. However, the Product Owner remains accountable.
The Product Owner is one person, not a committee. The Product Owner may represent the desires of a committee in the Product Backlog, but those wanting to change a Product Backlog item's priority must address the Product Owner.

For the Product Owner to succeed, the entire organization must respect his or her decisions. The Product Owner's decisions are visible in the content and ordering of the Product Backlog. No one is allowed to tell the Development Team to work from a different set of requirements, and the Development Team isn't allowed to act on what anyone else says.

Managing the Scrum Product Backlog

The Scrum Product Owner is the only person allowed to manage the contents of the Scrum Product Backlog. This means he has to:

- Create, maintain and clearly describe the Scrum Product Backlog items
- Prioritize the items to best achieve goals and mission
- Ensuring that the Scrum Team understands the items in the Scrum Product Backlog.

Stakeholder Management
External stakeholder should not communicate directly with the Scrum Team. Instead the Scrum Product Owner should collect and discuss required functionalities with the different Stakeholders (e.g. customer, marketing, service etc). These requirements are then combined and filtered before giving it to the team in the form of prioritized Scrum Product Backlog Items.

In Watt? project, the Watt? team is the stakeholder, so we created the Scrum Product Backlog items and managed it.

Development Team

Within the Scrum Framework all work delivered to the customer is done by dedicated Scrum Teams. A Development Team is a collection of individuals working together to deliver the requested and committed product increments.

To work effectively it is important for a Development Team that everyone within the team

- follows a common goal
- adheres the same norms and rules
- shows respect to each other

When setting up a new Development Team one always has to keep in mind that no new team will deliver with the highest possible performance right from the beginning. After setting up the team it has to go through certain phases : Forming, Storming, Norming, Performing.

Development Team Size
Optimal Development Team size is small enough to remain nimble and large enough to complete significant work within a Sprint.
Fewer than **three** Development Team members decrease interaction and results in smaller productivity gains. Smaller Development Teams may encounter skill constraints during the Sprint, causing the Development Team to be unable to deliver a potentially releasable Increment.
Having more than **nine** members requires too much coordination. Large Development Teams generate too much complexity for an empirical process to manage.

Scrum Master

The Scrum Master is responsible for ensuring Scrum is understood and enacted. Scrum Masters do this by ensuring that the Scrum Team adheres to Scrum theory, practices, and rules.
The Scrum Master is a servant-leader for the Scrum Team. The Scrum Master helps those outside the Scrum Team understand which of their interactions with the Scrum Team are helpful and which aren't. The Scrum Master helps everyone change these interactions to maximize the value created by the Scrum Team.

Traditional Management
Traditionally an individual is declared a 'manager' when having hierarchical control over other individuals. A traditional manager exerts power. A traditional manager commands people through the assignment of to-be-done work; expressed as tasks or work packages, given on a daily base or via to-be-followed plans. Subsequently a traditional manager follows up on the execution of the assigned work. The traditional manager does not wait for the results of the work but wants to see how the work is being carried out, who is doing it, when the tasks are being performed and how much time it is taking. The time it takes is in general compared to the time that was instructed the task should take.

Scrum Master Management
Contrary to the traditional idea of a 'manager', a Scrum Master has no formal power over the people in the Development Team, not deciding over their careers, incentives, etc. A Scrum Master does not manage the people or their tasks. But a Scrum Master does manage (via) the Scrum process. Within an organization a Scrum Master is accountable for the maximization of Scrum, for ensuring that people, teams, departments and the organization realize the highest benefits possible from using Scrum. This requires management skills, traits and insights.

A Scrum Master is explicitly responsible for removing Impediments. Impediments are elements that limit the efficiency and progress of a Development Team in areas that are beyond the reach of self-organization of a Development Team. Impediments are most often found in the wider organization, in company processes, procedures, and structures.

Scrum Master Service to the Development Team

The Scrum Master serves the Development Team in several ways, including:

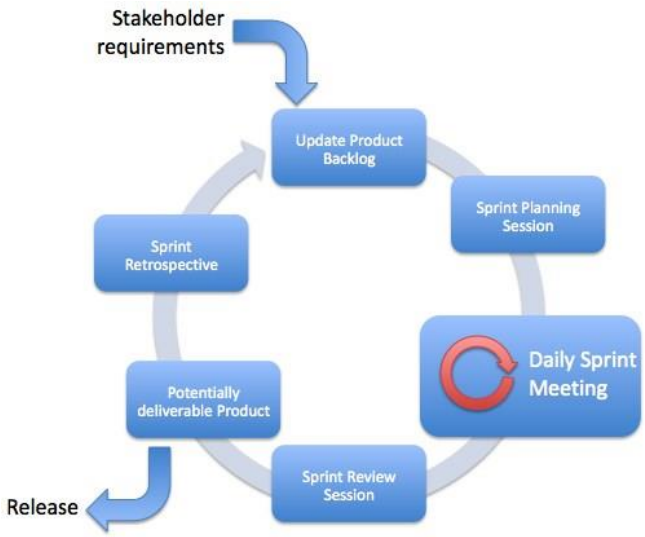
- Coaching the Development Team in self-organization and cross-functionality;
- Helping the Development Team to create high-value products;
- Removing impediments to the Development Team's progress;
- Facilitating Scrum events as requested or needed; and,
- Coaching the Development Team in organizational environments in which Scrum is not yet fully adopted and understood.

Scrum Events

Prescribed events are used in Scrum to create regularity and to minimize the need for meetings not defined in Scrum. All events are time-boxed events, such that every event has a maximum duration. Once a Sprint begins, its duration is fixed and cannot be shortened or lengthened.

The remaining events may end whenever the purpose of the event is achieved, ensuring an appropriate amount of time is spent without allowing waste in the process.

Other than the Sprint itself, which is a container for all other events, each event in Scrum is a formal opportunity to inspect and adapt something. These events are specifically designed to enable critical transparency and inspection. Failure to include any of these events results in reduced transparency and is a lost opportunity to inspect and adapt.



The Sprint

The heart of Scrum is a Sprint, a time-box of one month or less during which a "Done", useable, and potentially releasable product Increment is created. Sprints best have consistent durations throughout a development effort. A new Sprint starts immediately after the conclusion of the previous Sprint.

Each Sprint start with two planning sessions to define the content of the Sprint : the WHAT-Meeting and the HOW-Meeting. The combination of these two meeting are also defined as **Sprint Planning Meeting**. In the WHAT-Meeting the Scrum Team commits to the User Stories from the Scrum Product Backlog and it uses a HOW-Meeting to break the committed User Stories into smaller and concrete tasks. Then implementation begins.

Sprints are always short : normally about 2- 4 weeks. When a Sprint's horizon is too long the definition of what is being built may change, complexity may rise, and risk may increase.

Sprints contain and consist of :

- The Sprint Planning
- Daily Scrums
- The development work
- The Sprint Review
- Sprint Retrospective

Sprint backlog and Product Backlog

What is the Product Backlog?

The Product Backlog is an ordered list of everything that might be needed in the product and is the single source of requirements for any changes to be made to the product. The Product Owner is responsible for the Product Backlog, including its content, availability, and ordering.

The Product Backlog lists all features, functions, requirements, enhancements, and fixes that constitute the changes to be made to the product in future releases. Product Backlog items have the attributes of a description, order, estimate and value.

Product Backlog refinement is the act of adding detail, estimates, and order to items in the Product Backlog. This is an ongoing process in which the Product Owner and the Development Team collaborate on the details of Product Backlog items. During Product Backlog refinement, items are reviewed and revised. The Scrum Team decides how and when refinement is done.

Sprint Backlog

The sprint backlog is the list of work the development team must address during the next sprint. The list is derived by the scrum team progressively selecting product backlog items in priority order from the top of the product backlog until they feel they have enough work to fill the sprint. The development team should keep in mind its past performance assessing its capacity for the new sprint, and use this as a guide line of how much 'effort' they can complete.

The product backlog items may be broken down into tasks by the development team.

Tasks on the sprint backlog are never assigned; rather, tasks are signed up for by the team members as needed according to the set priority and the skills of the team. This promotes self-organization of the development team, and developer buy-in.

Sprint Planning Meeting

The plan of sprint is created by the collaborative work of the entire Scrum Team in the Sprint Planning Meeting .
Sprint Planning is time-boxed to a maximum of eight hours for a one-month Sprint. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that attendants understand its purpose. The Scrum Master teaches the Scrum Team to keep it within the time-box.
Sprint Planning answers the following:

- What can be delivered in the Increment resulting from the upcoming Sprint?
- How will the work needed to deliver the Increment be achieved?

The input to this meeting is the Product Backlog, the latest product Increment, projected capacity of the Development Team during the Sprint, and past performance of the Development Team.
The number of items selected from the Product Backlog for the Sprint is solely up to the Development Team. Only the Development Team can assess what it can accomplish over the upcoming Sprint.

After the Development Team forecasts the Product Backlog items it will deliver in the Sprint, the Scrum Team crafts a Sprint Goal. The Sprint Goal is an objective that will be met within the Sprint through the implementation of the Product Backlog, and it provides guidance to the Development Team on why it is building the Increment.

Sprint Goal
The Sprint Goal is an objective set for the Sprint that can be met through the implementation of Product Backlog. It provides guidance to the Development Team on why it is building the Increment. It is created during the Sprint Planning meeting.
The Sprint Goal gives the Development Team some flexibility regarding the functionality implemented within the Sprint.
The selected Product Backlog items deliver one coherent function, which can be the Sprint Goal.
The Sprint Goal can be any other coherence that causes the Development Team to work together rather than on separate initiatives.

Daily Scrum Meeting

The Daily Scrum is a 15-minute time-boxed event for the Development Team to synchronize activities and create a plan for the next 24 hours. This is done by inspecting the work since the last Daily Scrum and forecasting the work that could be done before the next one. The Daily Scrum is held at the same time and place each day to reduce complexity. During the meeting, the Development Team members explain:

- What did I do yesterday that helped the Development Team meet the Sprint Goal?
- What will I do today to help the Development Team meet the Sprint Goal?
- Do I see any impediment that prevents me or the Development Team from meeting the Sprint Goal?

The Development Team uses the Daily Scrum to inspect progress toward the Sprint Goal and to inspect how progress is trending toward completing the work in the Sprint Backlog.
The Daily Scrum optimizes the probability that the Development Team will meet the Sprint Goal. Every day, the Development Team should understand how it intends to work together as a self organizing team to accomplish the Sprint Goal and create the anticipated Increment by the end of the Sprint. The Development Team or team members often meet immediately after the Daily Scrum for detailed discussions, or to adapt, or replan, the rest of the Sprint's work.

The Scrum Master ensures that the Development Team has the meeting, but the Development Team is responsible for conducting the Daily Scrum. The Scrum Master teaches the Development Team to keep the Daily Scrum within the 15-minute time-box.

Sprint Review Meeting

A Sprint Review is held at the end of the Sprint to inspect the Increment and adapt the Product Backlog if needed. During the Sprint Review, the Scrum Team and stakeholders collaborate about what was done in the Sprint. Based on that and any changes to the Product Backlog during the Sprint, attendees collaborate on the next things that could be done to optimize value.

This is an informal meeting, not a status meeting, and the presentation of the Increment is intended to elicit feedback and foster collaboration.

This is a four-hour time-boxed meeting for one-month Sprints. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that attendants understand its purpose. The Scrum Master teaches all to keep it within the time-box.

The Sprint Review includes the following elements:

- Attendees include the Scrum Team and key stakeholders invited by the Product Owner.
- The Product Owner explains what Product Backlog items have been "Done" and what has not been "Done".
- The Development Team discusses what went well during the Sprint, what problems it ran into, and how those problems were solved.
- The Development Team demonstrates the work that it has "Done" and answers questions about the Increment.
- The Product Owner discusses the Product Backlog as it stands. He or she projects likely completion dates based on progress to date (if needed).
- The entire group collaborates on what to do next, so that the Sprint Review provides valuable input to subsequent Sprint Planning.
- Review of how the marketplace or potential use of the product might have changed what is the most valuable thing to do next; and,
- Review of the timeline, budget, potential capabilities, and marketplace for the next anticipated release of the product.

The result of the Sprint Review is a revised Product Backlog that defines the probable Product Backlog items for the next Sprint. The Product Backlog may also be adjusted overall to meet new opportunities.

Sprint Retrospective Meeting

The Sprint Retrospective is an opportunity for the Scrum Team to inspect itself and create a plan for improvements to be enacted during the next Sprint.
The Sprint Retrospective occurs after the Sprint Review and prior to the next Sprint Planning.

This is a three-hour time-boxed meeting for one-month Sprints. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that attendants understand its purpose. The Scrum Master teaches all to keep it within the time-box. The Scrum Master participates as a peer team member in the meeting from the accountability over the Scrum process.

The purpose of the Sprint Retrospective is to:

- Inspect how the last Sprint went with regards to people, relationships, process, and tools.
- Identify and order the major items that went well and potential improvements; and,
- Create a plan for implementing improvements to the way the Scrum Team does its work.

The Scrum Master encourages the Scrum Team to improve, within the Scrum process framework, its development process and practices to make it more effective and enjoyable for the next Sprint. During each Sprint Retrospective, the Scrum Team plans ways to increase product quality by adapting the definition of “Done” as appropriate.
By the end of the Sprint Retrospective, the Scrum Team should have identified improvements that it will implement in the next Sprint. Implementing these improvements in the next Sprint is the adaptation to the inspection of the Scrum Team itself. Although improvements may be implemented at any time, the Sprint Retrospective provides a formal opportunity to focus on inspection and adaptation.

Monitoring Progress Toward a Goal
At any point in time, the total work remaining to reach a goal can be summed. The Product Owner tracks this total work remaining at least every Sprint Review.
The Product Owner compares this amount with work remaining at previous Sprint Reviews to assess progress toward completing projected work by the desired time for the goal. This information is made transparent to all stakeholders.

Increment Concept

The Increment is the sum of all the Product Backlog items completed during a Sprint and the value of the increments of all previous Sprints.

At the end of a Sprint, the new Increment must be “Done,” which means it must be in useable condition and meet the Scrum Team’s definition of “Done.” It must be in useable condition regardless of whether the Product Owner decides to actually release it.

Definition of Done (DoD)
In order to be able to decide when an activity from the Sprint Backlog is completed, the Definition of Done (DoD) is used. It is a comprehensive checklist of necessary activities that ensure that only truly done features are delivered, not only in terms of functionality but in terms of quality as well. The DoD may vary from one Scrum Team to another, but must be consistent within one team.

There might be different DoD at various levels:

- DoD for a Scrum Product Backlog item (e.g. writing code, tests and all necessary documentation)
- DoD for a sprint (e.g. install demo system for review)
- DoD for a release (e.g. writing release notes)

ScrumBut

ScrumButs are reasons why teams can’t take full advantage of Scrum to solve their problems and realize the full benefits of product development using Scrum. Every Scrum role, rule, and timebox is designed to provide the desired benefits and address predictable recurring problems. ScrumButs mean that Scrum has exposed a dysfunction that is contributing to the problem, but is too hard to fix. A ScrumBut retains the problem while modifying Scrum to make it invisible so that the dysfunction is no longer a thorn in the side of the team.
A ScrumBut has a particular syntax: (ScrumBut)(Reason)(Workaround).

Sometimes organizations make short term changes to Scrum to give them time to correct deficiencies. For example, "done" may not initially include regression and performance testing because it will take several months to develop automated testing. For these months, transparency is compromised, but restored as quickly as possible.

Watt? Project Workflow

The Watt? Development Team works in various fields including Embedded Hardware and Software, Android Development and User Interface Design. These different fields are integrated to achieve the features offered by Watt?.
At the project Kick-off meeting, Product Backlog and User Stories were produced. Those to define the tasks required from each member and divide them into Sprints.

Watt? User Stories

- I want to know the power consumption of whole my house (Real time).
- I want to know the consumption from the beginning of the month till now.
- I want to know the consumption of the previous months for a year.
- I want to know the consumption for every device (Real time).
- I want to know the consumption for every device from the beginning of the month till now.
- I want to switch my devices on or off.
- I want to set a timer to control my devices.
- I want to know the price from the beginning of the month till now.
- I want to know the price expected from my live consumption to the end of the month.
- I want to know the price saved by the system.
- I need to have an account in the app.
- I need to add a user to use the system.
- I need to add a new system to my account.

Watt? team worked to achieve the project goal over six Sprints, each Sprint takes 2 weeks, starting with the planning meeting during which the next tasks are defined and ends with the review meeting. During the period of work, the project was designed and developed to ensure that the desired results and features are achieved.

We sought after taking most of the advantages of Scrum framework and to ensure that the values of Scrum were maintained effective.

Watt? ScrumButs

During working on Watt? project, we encountered some challenges in the management of development with Scrum, resulting from the nature of the project the team is working on, as well as the educational commitments that differ from the working life.

Full-time jobs : One of the basic characteristics of the development team at Scrum is that it works full-time for a specific duration daily, Which cannot be controlled while studying at a university with daily study obligations.

Multiple tracks : The Scrum team consists of members working in different tracks, causing some confusion in the distributing and executing tasks within Sprint.
For example, there is a dependence between the UI graphic design and the android development, so the designer must finish his task before the android developer begins work.
There is also an integration between the Embedded software and android development, which requires some care in the management of tasks.

Daily Meeting overhead : It is difficult for the Scrum team to achieve the daily meeting, due to the required academic commitments as well as the different working times of each member, They do not work for a fixed time.

First practice : Working on the Watt project is the first practice of the team using the Scrum framework, this led to some difficulties at first, Such as coordinating Scrum meetings and determining the Sprints goals.

Dealing with Watt? ScrumButs

With ScrumBut particular syntax,

- We use Scrum, but having a Daily Scrum every day is too much overhead, so we made it 3 times per week.
- We use Scrum, but The team does not work in one place, so Some tasks are performed via remote communication.
- We use Scrum, but The team does not work full-time, so Some tasks are deferred to the next sprint.
- We use Scrum, but The team does not have much experience in dealing with Scrum, so We use some experts to cover the default.

References

[1] “Pros & Cons of Internet Of Things (IOT)”. Bhaskara Reddy Sannapureddy. February 25, 2015. Retrieved July, 2017

[2] “A Simple Explanation Of The Internet Of Things”. Jacob Morgan. May 13, 2014. Retrieved July, 2017

[3] Luke Hohmann. “Beyond Software Architecture: Creating and Sustaining Winning Solutions”. January 30, 2003

[4] “Agile is not now, nor was it ever, Waterfall”. Robert Martin (uncle bob). 16 Oct 2015. Retrieved July, 2017

[5] “Why You Should Build Apps With An API Backend”. Jennifer Riggins. March 16, 2015. Retrieved July, 2017

[6] “NoSQL (Not Only SQL database)”. Margaret Rouse. March 2017. Retrieved July, 2017

[7] “Firebase Documentation”. Firebase platform2017. Retrieved July, 2017

[8] "Overview of Internet of Things". Google Cloud Platform. April 19, 2017. Retrieved July 7, 2017.

[9] "Intel steps into the Internet of platforms". Intel® IoT Platform. 2014. Retrieved July 7, 2017.

[10] "Embedded Systems - Overview". Tutorialspoint. Retrieved July 7, 2017.

[11] "Embedded Hardware Design and Development". rowebots.com. June 2015. Retrieved July 7, 2017.

[12] "Overview of Internet of Things". Google Cloud Platform. April 19, 2017. Retrieved July 7, 2017.

[13] Ian Harris. "Introduction to the Internet of Things and Embedded Systems". Coursera.com.

[14] Srikar Deshmukh. "Hardware Abstraction Layer". Microchip Technology Inc.. February 2017.

[15] allavi Sethi and Smruti R. Sarangi (2017). "Internet of Things: Architectures, Protocols, and Applications". Journal of Electrical and Computer Engineering. Article ID 9324035, 25 pages. Volume 2017.

[16] "The Three Software Stacks Required for IoT Architectures". Eclipse IoT. September 2016.

[17] J. A. Cook and J. S. Freudenberg. "Embedded Software Architecture". EECS UMich. Fall 2008.

[18] Pallavi Sethi and Smruti R. Sarangi (2017). "Internet of Things: Architectures, Protocols, and Applications". Journal of Electrical and Computer Engineering. Article ID 9324035, 25 pages. Volume 2017

[19] Dina Gamal Darwish (2015). "Improved Layered Architecture for Internet of Things". www.meacse.org/ijcar/. pp.214-223. ISSN 2305-9184 Volume 4

[20] An Agile Framework for Completing Projects”. Scrum Alliance. February 24, 2016. Retrieved July 7, 2016

[21] Gunther Verheyen (2014). “A Pocket Guide of Scrum”. guntherverheyen.com

[22] Ken Schwaber, Jeff Sutherland. "The Scrum Guide" . Scrum.org, July 22, 2016.

[23] “Scrum Roles”. International Scrum Institute 2017. Retrieved July 7, 2016

[24] “Introduction to Software Project Management”. Software Engineer Training. July 3, 2013. Retrieved July 8, 2016