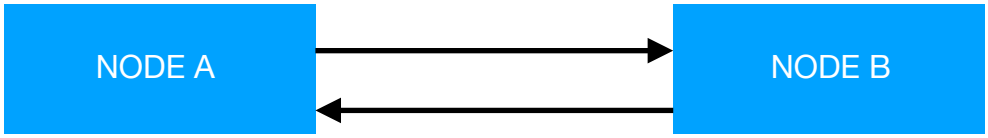
JSON AND JSONP

As we know, the network is composed by several nodes exchanging data. To get or return data we use the protocol HTTP, more in detail, from a browser we can use AJAX to issue HTTP requests to severs on the Web using XMLHttpRequest and getting a JSON object like in the picture below:

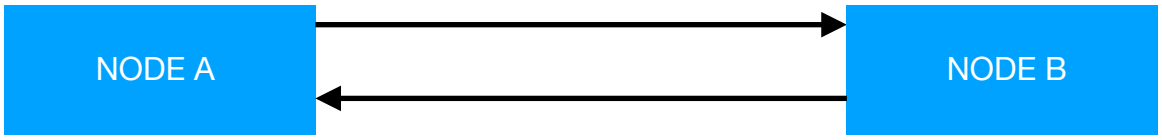
XMLHttpRequest



JSON Object

In this configuration there's a problem though: node A and node B have to be in the same domain according to the SAME ORIGIN POLICY for security reason. In fact consider the example in which you have two windows at your browser open, in the first window you are authenticated with your bank account and a malicious script is running in the second window, this latter can execute or steal some secret information.
By the way, it is possible to have an interaction between nodes in different domains using JSONP (JSON with padding) in which node A has locally defined the function X it wants to execute and using the script tag and the src attribute communicates with node B that replies back not returning a JSON object but the function X with fetched parameters.
Now since node A has X defined locally, it can execute X with the resulting parameters as shown in the picture below:

<script src="...."callback="myFunction">



myFunction(fetched parameters)

PROCESS/CPU/SECURITY MANAGEMENT

In general an application is composed by :

- Runtime supports (Java virtual machine, libraries, etc)
- And other components, everything is managed by the Activity Manager

Android operating system is mainly characterized by :

- Process management
- CPU management
- Security management

For what concerning process management, processes host applications and processes running applications and all processes are all created from the same process called Zygote, the peculiarity is that not fork and exec are executed as in Linux but only fork.
Zygote contains a pre-warmed execution environment, i.e. required to all app run (e.g JVM, libraries, etc.). Processes are ranked in the system according to the state of the contained application (foreground, visible, hidden, etc..).
Each process that runs an application belongs to a unique and different user so files created by an app can't be read from other apps. This is why Inter-Process communication via Intent is required: this communication occurs using a kind of message called INTENT:

- IMPLICIT INTENT: It's a message that has not the target, so the sender knows only the action and not the target. When and why it is used if the application doesn't know the target? The target is selected by the manager that has a kind of table in which are described, for each application, which Intent can response. EXAMPLE: If you want to use a browser, but you have two browser from the application you are using, then you can send an implicit intent to the system that shows a dialog to the user asking which installed browser he/she wants to use.
- EXPLICIT INTENT: It specifies which application will satisfy the Intent, by supplying either the target app's package name or a fully-qualified component class name. You will typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, you might start a new activity within your app in response to a user action, or start a service to download a file in the background.

Moreover the process management is responsible also of OOM state, Out Of Memory. In this state the system can't allocate other memory for the processes and the process management kill the processes(in determinate states) to recover memory.

For CPU management, it is possible to reduce battery drain reducing the CPU frequency (for example from 1GHZ to 50 MHZ) or deep sleeping setting a timer to wake up.

Finally, for what concerns Security Management only signed OS from known origin can be loaded (granting Integrity and Authenticity) and since each application is a sandbox, by default an app can't uses the components like sensors of the mobile phone without the granting of the user.

ACTIVITY

An application is composed by at leat one activity that is the controller of the GUI in the screen. It runs inside the main thread and it should be reactive to user's input. Activities are managed by the Activity Manager and via back stack: when new activity is created, it is placed on the top of the stack and becomes active. The Previous activity remains below in the stack until the running activity is destroyed, for example using the back button.

The set of activities launched by a user is a Task and each application runs inside its own process. In general other threads besides the main one are created to perform long running operations (e.g. services)

Each activity has its own lifecycle characterised by several methods:

- Once and only one an activity is create, the onCreate method is executed.
- If the activity exits, the onDestroy method is executed.

In between, various methods are called, the most important are:

- onPause: the activity is partially visible but lost the focus (for example a dialog box is in foreground). A paused activity maintains all state and the information, but the system can kill it.
- onSaveInstaceState: using a bundle we can store small data that we can retrieve later on, for example when screen orientation changes.
- onStop: the activity is completely obscured by another one.

When resources are needed a stopped activity can be killed.

An activity can be created explicitly if it is the target of an Intent for example Activity A starts another activity B or implicitly: an activity declares to the system its ability to perform action through an intent-filter (declared in the manifest).

The calling activity creates an intent with the action and the activity manager presents a list of all activities that may perform the action(it more than one).

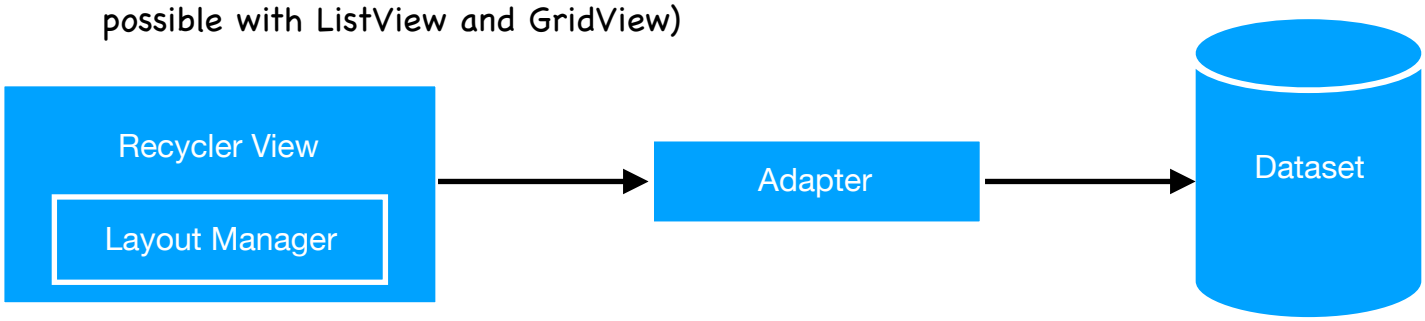
FRAGMENTS

Fragment is a portion of the screen and they are useful for large screen (like tablet) since you can divide the whole area in small areas.
The useful thing about fragments is that they can be reused. Fragments are hosted inside an Activity attaching it to a View that must be created(its View) from a xml file. A fragment have its own lifecycle. They can be added via XML file (static creation) or programmatically (dynamic creation).
The fragment needs to know its parent because the same fragment can be hosted by several activities. For this reason, a fragment can assume nothing about the kind of an activity.
The activity hosting more than one fragment needs to implement an interface with abstract methods in order to guarantee fragments to communicate.
Hence, the fragments have been created for tablet to handle a bigger view. The second reason is that they can be reused from different activities and they avoid to instantiate new activities so they are also important under a performance point of View.

LIST ACTIVITY/LIST VIEW / GRID VIEW / RECYCLER VIEW

The screen size of a device is limited, so many times to show all the content at once is infeasible. This is a common issue and android provides special classes and layout to deal with this problem.
There are different solutions:

- List Activity: it is a subclass of Activity and there is no layout to inflate. It allows to display an array of items, that are "clickable". An Array Adapter is required to transform an item(data) into a view, by default the adapter creates a view by calling toString() on each data object into the provided collection and places the result in a TextView. WARNING: ActionBar is not display.
- List View and Grid View: They are similar to List Activity, but they can extend AppCompatActivity, this means the possibility to display also the action bar.
- Recycler View: It is faster and more flexible, it automatically recycles views as they are no longer visible (more efficient than others). It requires a Layout Manager to manage views, moreover it requires an adapter to create a single view and update data of the view. It allows animations that are not possible with ListView and GridView)



STORAGE OPTIONS

When we develop an Android application we have to deal with dynamic/large/asynchronous data so we can use different storage options to deal with data.
The first option is to use the View Model: it provides data to the UI and survive to configuration changes. It is created the very first time the UI Controller is created.
Other storage options are:

- File: In Linux a file supports three operations (READ,WRITE,EXECUTE). Also a directory is a special file containing other files. Operations are performed by a user, group or others.
  A user/group has a system-wide identifier (UID/GID).
  Each file is represented by the file class and it can be internal (internal flash memory in the device) or external (/sdcard)
- Content Provider: it manages access to a central repository, it is basically an API that exposes databases to other processes. It presents data to external applications as one or more tables.
  Permissions are required to use a content provider.
  Main content providers are:
  - AlarmClock (alarms to fire)
  - Contact Provider (stores all information about contacts)
  - Calendar Provider (Data stored in the calendar)

You can indirectly access a content provider using intents. They allow the user to access data in a provider even if your application doesn't have access permissions, either by getting a result intent back from an application that has permissions, or by activating an application that has permissions and letting the user do work in it.

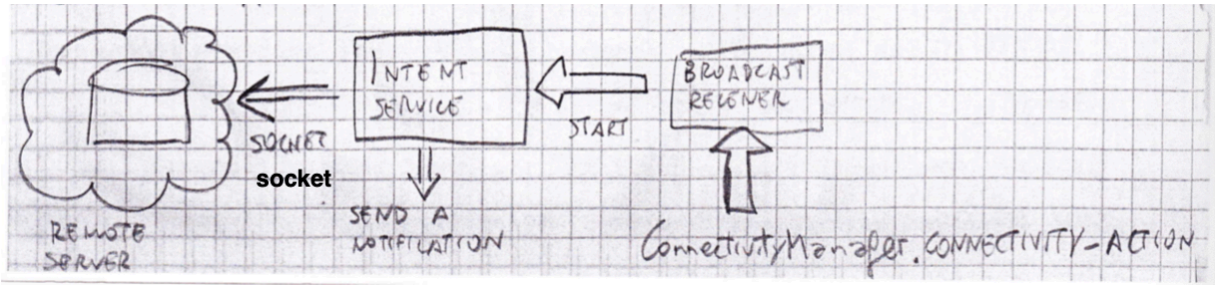
BUSINESS LAYER / THREADS / SERVICE / BROADCAST RECEIVER
Business layer is what application really makes.
Business logic takes data, process it and gives an output/result
There are severed solutions for the business layer:

- General solution : This solution provide to work with threads (as provided by Java). Java thread supports Thread class and Runnable interface. Here, a thread has a queue of messages associated to it. A Looper object is used to run a loop for receiving messages. The queue can store messages with node to be executed (Runnable). A Handler allows to send and process Message and Runnable objects to the queue.

- Pre-cooked frameworks (based on threads) mainly AsyncTask: It is an abstract class that simplifies the interaction with UI. This class allows to perform background operations and publishes the result on the UI thread, without having to manipulate threads and/or handlers. An asynchronous task is defined by a computation that runs on a background thread and whole result is published on the UI thread. The main method is:
  - DoInBackground: invoked on the background thread immediately after onPreExecute() finishes executing. This step is used to perform background computation that can take a long time. In this step can also use publishProgress method to publish one or more units of progress. These values are published on the UI thread in the onProgressUpdate step
- Software (OS managed) components that are Service and Broadcast receiver. A Service is an application component that runs in background, not interacting with the user, for an indefinite period of time. Services run in the main thread of their hosting process. CPU intensive operations or blocking operations should spawn its own thread in which to do that work. A service is a software component with its own lifecycle, it can run in background or foreground, it can private or system-wide.
  A service can be started, bound or both:
  - A started service is a service that an application component starts by calling startService(). Use started services for tasks that run in the background to perform long-running operations. Also use started services for tasks that perform work for remote processes.
  - A bound service is a service that an application component binds to itself by calling bindService(). Use bound services for tasks that another app component interacts with you to perform interprocess communication (IPC).

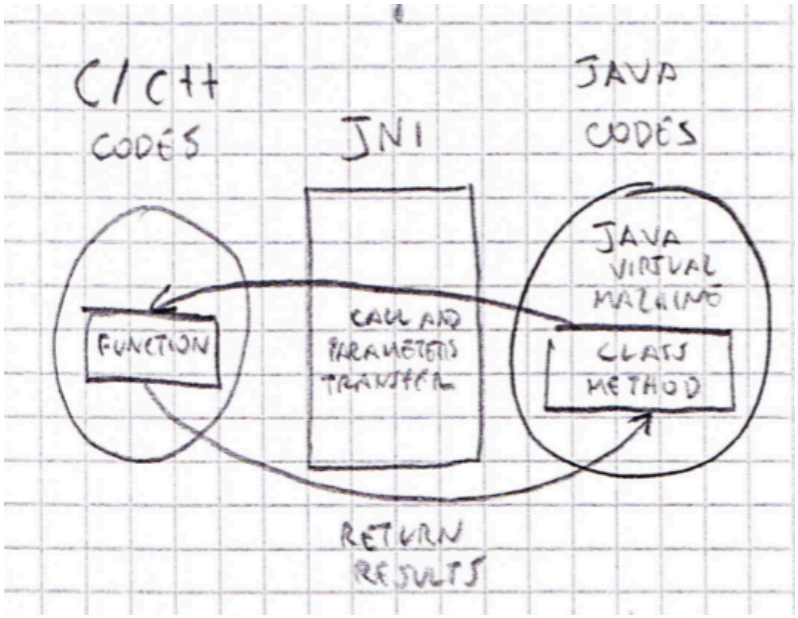
Sometimes a service may need to be started when some condition is met, for example connect to a server only when the wi-fi connection is available (not GSM).
These cases may be managed using Broadcast receivers. It may also useful to use a specialized service (Intent Service).
A Broadcast receiver is a receiver that wants to be notified when a broadcast event happens and when this latter happens the broadcast receiver starts to run.
The Intent Service class is a convenience class (subclass from the Service class) that sets up a worker thread for handling background.
Once the service has handled all queued requests, it simply exits.

General Approach intent service + broadcast



NATIVE APPS / HYBRID APPS

We use Native code because call C/C++ libraries from Java may be required for implementing critical code (performance reason, security reason etc.), using existing code, etc. The schema is the following:



The Java Native Interface completes the mapping between differences in function prototype definitions and variable types.
In general we have:

Java —> Native: JNI, native methods translated into executable code linked and available as a library.

Native —> Java: No executable code produced at compile time. Exploits Java reflection (any method can be called).

JS —> Android: Android code is made available to the JS interpreter (an android method is visibile in the JS namespace).

Android —> JS: Evaluate JS code at run-time (any string can be executed, similar to reflection).

Then

HYBRID APPS
Hybrid apps, like native apps, run on the device, and are written with web technologies (HTML5, CSS and JavaScript). Hybrid apps run inside a native container, and leverage the device's browser engine (but not the browser) to render the HTML and process the JavaScript locally. A web-to-native abstraction layer enables access to device capabilities that are not accessible in Mobile Web applications, such as the accelerometer, camera and local storage.

IOS BASICS

In iOS application follows a MVC pattern. The structure of the code strictly reflects this organization. Instead, in Android a controller can be an Activity that has its own layout, so in Android you don't have to follow strict rules, there is not always a clear distinction between components.
Apps are multi-threads with one main thread looping on UI related user events, and other threads used for executing long tasks, running in background. A fundamental concept is STORYBOARD.
The idea of StoryBoard was born at the beginning in iOS while in Android this concept has been introduced only in last versions.
A storyboard represents graphically your code, so all connections with, for example, arrows between activities, graphical views for activities, etc.
Another concept is about the architecture of iOS apps: there are a set of View controllers associated to the Views but they are part of Controller and not of the View.
For what concerns event handling, in iOS there is a first responder that is a handler receiving for the first time an event. It can decide to do nothing and the event is sent to the next handler that can manage it. There is a chain of event handlers.
The language used by iOS is Swift and it's object-oriented. Regarding the design, views in iOS are done basically dragging and dropping graphical elements into the view area, there isn't no XML file like in Android. In Android you can use an Intent to switch Activity, while in iOS, in order to switch to another View Controllers you would perform a Segue. In prepareForSegue method you can handle what you want to do in the next ViewController such as passing variables etc.
In really, the Broadcast Receiver is Notification in iOS and allows for messages to be sent out over all classes in a project and to be heard or received from anywhere or anyone listening.

CLOUD COMPUTING /SAAS /PAAS

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

The central concept of cloud computing is service. Two main features of cloud computing:

- Possibility to use the service (for example a remote repository) from machines with several operating systems.
- Synchronization of all clients (if you update the repository in the cloud with a computer, you will see your update in your smartphone as well).

In general cloud computing refers to one of these models:

- SaaS (Software as a Service) an example is Algolia
- PaaS (Platform as a Service) an example is Heroku
- IaaS (Infrastructure as a Service) an example is Amazon web services, in particular Amazon EC2

SaaS means the capability provided to the consumer to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g. web-based email) or a program interface. The consumer doesn't manage or control the underlying cloud infrastructure including network, servers, operating systems, storage or even individual application capabilities with the possible exception of limited user-specific application configuration settings.
Another concept is BaaS (Backend as a service) that are a set of features provided to you by the provider, for example crash reporting: it sends you why your application has crashed(e.g firebase)

PaaS is the capability provided to the consumer to deploy onto cloud infrastructure acquired applications using programming languages, libraries, services and tools supported by the provider. The consumer doesn't manage or control the underlying cloud infrastructure including network, servers, OS or storage, but has control over the deployed applications and possibly configuration settings for application-hosting environment.

An important concept in PaaS is that about containers: they include the application and all of its dependencies, but share the kernel with other containers. They run as an isolated process in user space on the host operating system.

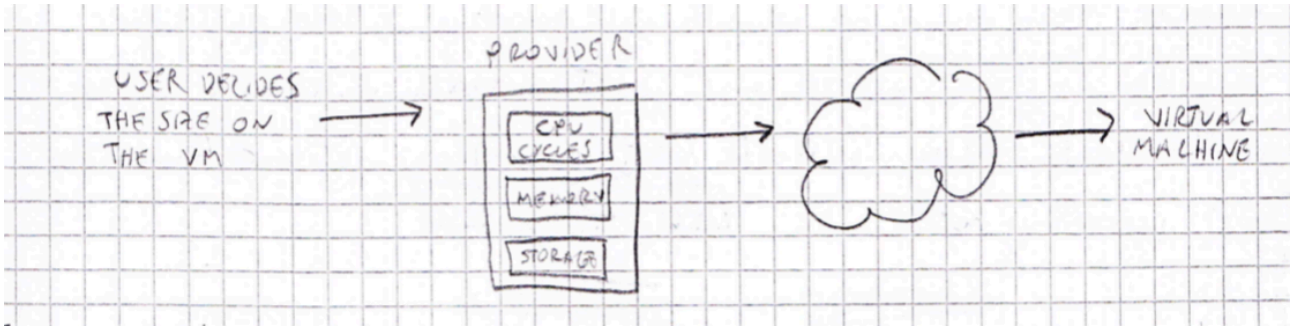
One example of PaaS is Heroku, where containers are called dynos. We can distinguish three dynos:

- Front-end (web dyno): they receive HTTP traffic from the routers.
- Backend (worker dyno): they are typically used for background jobs
- On Shot (One-off dyno): they are used for administrative tasks like database migrations, console sessione etc.

Dynos are used for scalability a more predictable and scalable architecture is to background the high-latency or long-running work in a process separate from the web layer and immediately respond to the user's request with some indicator of work progress.
To scale horizontally, add more dynos. For example, adding more web dynos allows you to handle more concurrent HTTP requests and therefore higher volumes of traffic.
To scale vertically, use bigger dynos. The maximum amount of RAM available to your application depends on the dyno type you use.

INFRASTRUCTURE AS A SERVICE / VIRTUALIZATION

IaaS is a form of cloud computing that provides virtualized computing resources over the Internet. IaaS are online services that provide high-level APIs used to dereference various low-level details of underlying network infrastructure like physical computing resources, location, data partitioning.
A hypervisor such as Oracle Virtual Box runs the virtual machines as guests. Pools of hypervisors within the cloud operational system can support large numbers of virtual machines and the ability to scale services up and down according to customer's varying requirements.
Typically IaaS involve the use of technology like Open Stack, which manage the creation of a virtual machine and decides on which hypervisor to start it, enables VM migration features between hosts, allocates storage volumes and attaches them to VMs.



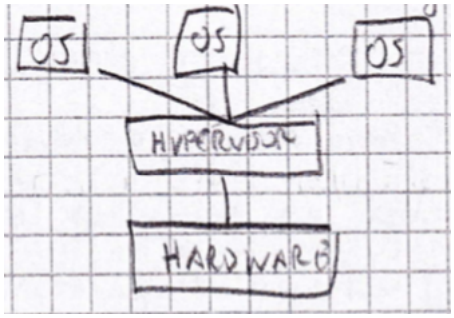
The main idea of virtualization is that virtual memory is higher than physical memory and several processes concurrently on the same HW, see the same ISA. HW is shared thanks to an OS that manages critical instructions. A virtual

machine is a logic machine whose ISA is implemented exploiting software running on a physical machine.
Two main types of VM:

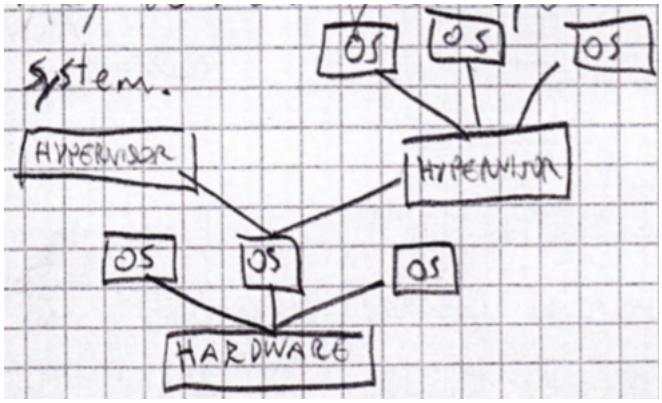
- Native ML = MF: same ISA. Indeed instructions of the MF are in large part executed on the real CPU. Sensitive instruction are trapped
- Emulation ML ≠ MF (different ISA), in this case we have HW emulation and language level emulation

In general, for what concerns the Classical virtualization, a virtual machine monitor (VMM) executes guest operating system directly, but at a reduced privileged level. The VMM intercepts traps from the de-privileged guest and emulates the trapping instruction against a virtual machine state. Remember that a computer on which a hypervisor runs one or more virtual machines is called a host machine and each virtual machine si called a guest machine.
We have two types of hypervisor:

- native or bare metal: These hypervisors run directly on the host's hardware to control the hardware and the manage guest operating systems



- Hosted: These hypervisors run on a conventional operating system just as other computer programs do. A guest operating system runs as a process on the host. They abstract guest operating systems from the host operating system.



Through virtualization, workload isolation is achieved since all program instruction are fully contained inside a VM, which leads to improvements in security.
Virtualization makes HW consolidation possible, meaning to consolidate individual workloads onto a single physical platform, reducing the total cost of ownership.

DISPLAY AND ASPECT RATIO

A digital image is a matrix W x H of pixels (Width, Height), a pixel is the smaller visible unit that can be controlled.
The resolution in pixel of an image is the number of pixels, while the Aspect Ratio is the ratio between W / H (for example 4:3 or 16:9).

The images are visualized on a screen. Each screen has its own characteristics, similar to the images. In particular, we have :

- Physical dimension: diagonal (in inch)
- Spatial resolution : pixel per inch

The touch screen of the smartphone is a surface X x Y, and the diagonal is D = $\sqrt{X^2 + Y^2}$.
L'aspect ratio is AR = Y / X where the first is bigger than the second.
The dimension of the screen is given by the length of the diagonal in inch.

$$AR = Y : X \rightarrow Y = AR * X$$

$$D = \sqrt{X^2 + (X * AR)^2} = X\sqrt{(1 + AR^2)} \rightarrow X = \frac{D}{\sqrt{(1 + AR^2)}}$$

The resolution of the screen indicates how many pixels are visualized per inch, o pixel density.

Consideriamo uno schermo da 4.7" con AR = 16:9 e PPI = 326. La risoluzione Pixel è :
W = Y * PPI = 4,1 * 326 = 1336
H = X * PPI = 2,3 * 326 = 750.

ESEMPIO
Dobbiamo trovare la dimensione fisica data una diagonale di D=4", 640x1136 pixels —> AR=1136/640

$$X = \frac{D}{\sqrt{(1+AR^2)}} \qquad PPI = \frac{Pixels\,along\,X}{X} = \frac{640}{1{,}94}$$

I pixel che appartengono alla diagonale sono pari alla somma a:

Pixels along the diagonal = $\sqrt{1920^2 + 1080^2}$ –> Nel Full HD

RUBY ON RAILS

Ruby is a programming language like Java, C, Python, etc.
On top of Ruby there is an architecture called Rails based on the Model View Controller pattern (MVC).
The Rails architecture is used to implement a web server, the main components are:

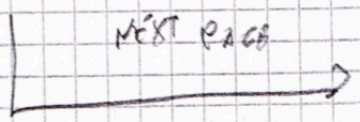
- Web browser: interacts with Web server through HTTP.
- Web server: takes the HTTP request and look up a kind of routing table.

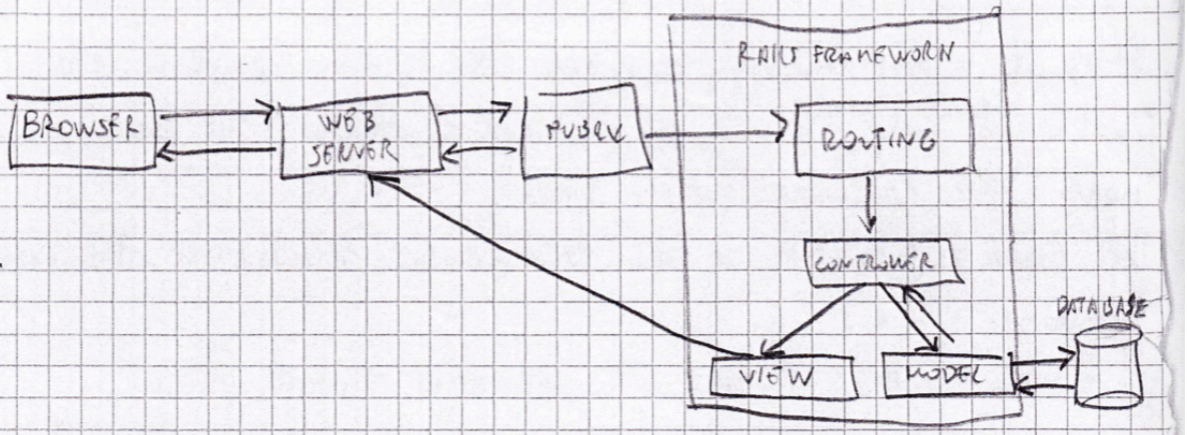
For example:

GET ⟹ is mapped to Controller_#nameOfTheMethod

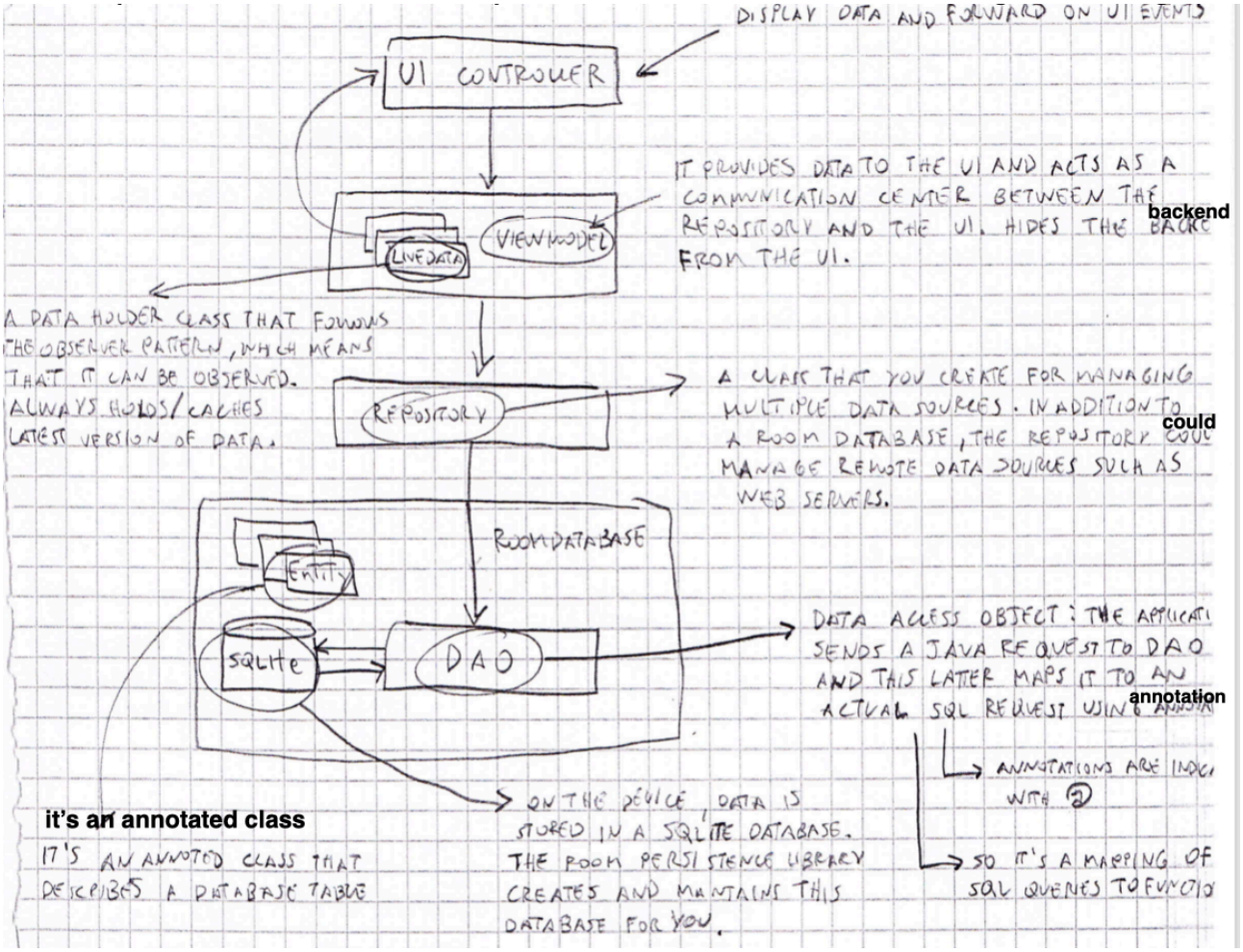
So there is a mapping between URI and controller methods.
The routing information are build using rake.
For each REST resource is possible to specify the CRUD operations.
More in detail, the controller takes the data coming from the request and decide what to do. For example update the model, you don't see directly the database but you see the object that is mapped into the database.
After the reply of the model (like an ACK "the database has been updated"), the controller sends the updated data to the View that sends it to the Web server that replies to the Web browser.

It is important to notice that View and Model don't interact directly between each other, there is the Controller in the middle.
The following scheme summarize the architecture:





In config folder there is a file called routes.rb in which you map the HTTP verb with the controller action.
Finally, Ruby on Rails is characterized by gems and the gem file contains those regarding the application.


DISPLAY DATA AND FORWARD ON UI EVENTS
UI CONTROLLER
IT PROVIDES DATA TO THE UI AND ACTS AS A COMMUNICATION CENTER BETWEEN THE REPOSITORY AND THE UI. HIDES THE BACKE FROM THE UI.
backend
VIEWMODEL
LIVEDATA
A DATA HOLDER CLASS THAT FOLLOWS THE OBSERVER PATTERN, WHICH MEANS THAT IT CAN BE OBSERVED. ALWAYS HOLDS/CACHES LATEST VERSION OF DATA.
REPOSITORY
A CLASS THAT YOU CREATE FOR MANAGING MULTIPLE DATA SOURCES. IN ADDITION TO A ROOM DATABASE, THE REPOSITORY COULD MANAGE REMOTE DATA SOURCES SUCH AS WEB SERVERS.
could
ROOMDATABASE
ENTITY
SQLite
DAO
DATA ACCESS OBJECT: THE APPLICATI SENDS A JAVA REQUEST TO DAO AND THIS LATTER MAPS IT TO AN ACTUAL SQL REQUEST USING
annotation
ANNOTATIONS ARE INDICATED WITH @
SO IT'S A MAPPING OF SQL QUERIES TO FUNCTIO
it's an annotated class
IT'S AN ANNOTED CLASS THAT DESCRIBES A DATABASE TABLE
ON THE DEVICE, DATA IS STORED IN A SQLITE DATABASE. THE ROOM PERSISTENCE LIBRARY CREATES AND MAINTAINS THIS DATABASE FOR YOU.