# **3 Web basic principles**

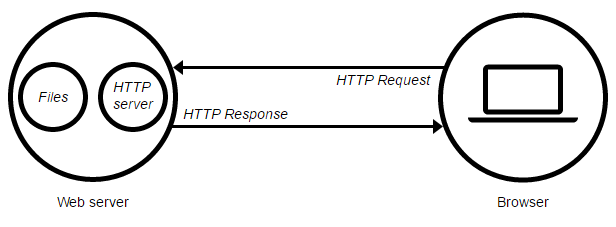
## **3.1 HTTP Protocol**

HTTP stands for Hypertext Transfer Protocol. It's a stateless, application-layer protocol for communicating between distributed systems, and is the foundation of the modern web. As a web developer, we all must have a strong understanding of this protocol.

### **3.1.1 HTTP Basics**

HTTP allows for communication between a variety of hosts and clients, and supports a mixture of network configurations.

This makes HTTP a stateless protocol. The communication usually takes place over TCP/IP (Transmission Control Protocol/Internet Protocol), but any reliable transport can be used. The default port for TCP/IP is 80, but other ports can also be used.



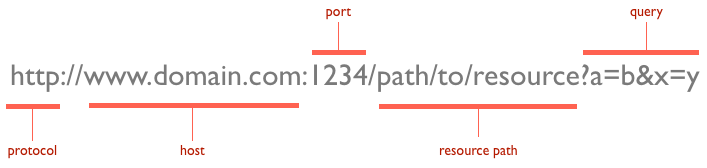
Custom headers can also be created and sent by the client.

Communication between a host and a client occurs, via a request/response pair. The client initiates an HTTP request message, which is serviced through a HTTP response message in return. We will look at this fundamental message-pair in the next section.

The current version of the protocol is HTTP/1.1, which adds a few extra features to the previous 1.0 version. The most important of these, in my opinion, includes persistent connections, chunked transfer-coding and fine-grained caching headers. We'll briefly touch upon these features in this article; in-depth coverage will be provided in part two.

### **3.1.2 URLs**

At the heart of web communications is the request message, which are sent via Uniform Resource Locators (URLs). URLs have a simple structure that consists of the following components:



The protocol is typically http, but it can also be https for secure communications. The default port is 80, but one can be set explicitly, as illustrated in the above image. The resource path is the local path to the resource on the server.

### **3.1.3 Verbs**

URLs reveal the identity of the particular host with which we want to communicate, but the action that should be performed on the host is specified via HTTP verbs. Of course, there are several actions that a client would like the host to perform. HTTP has formalized on a few that capture the essentials that are universally applicable for all kinds of applications.

These request verbs are:

* GET: fetch an existing resource. The URL contains all the necessary information the server needs to locate and return the resource.
* POST: create a new resource. POST requests usually carry a payload that specifies the data for the new resource.
* PUT: update an existing resource. The payload may contain the updated data for the resource.
* DELETE: delete an existing resource.

The above four verbs are the most popular, and most tools and frameworks explicitly expose these request verbs. PUT and DELETE are sometimes considered specialized versions of the POST verb, and they may be packaged as POST requests with the payload containing the exact action: create, update or delete.

## **3.2 Web Servers**

A web server is a computer system that processes requests via [HTTP](https://en.wikipedia.org/wiki/HTTP), The primary function of a web server is to store, process and deliver [web pages](https://en.wikipedia.org/wiki/Web_page) to [clients](https://en.wikipedia.org/wiki/Client_(computing)).

### **3.2.1 Web server’s definition**

A web server can refer to hardware or software, or both of them working together.

1. On the hardware side, a web server is a computer that stores a website's component files (e.g. HTML documents, images, CSS stylesheets, and JavaScript files, not covered in this report) and delivers them to the end-user's device. It is connected to the Internet and can be accessed through a domain name like mozilla.org.
2. On the software side, a web server includes several parts that control how web users access hosted files, at minimum an HTTP server. An HTTP server is a piece of software that understands [URLs](https://developer.mozilla.org/en-US/docs/Glossary/URL) and [HTTP](https://developer.mozilla.org/en-US/docs/Glossary/HTTP).

At the most basic level, whenever a browser needs a file hosted on a web server, the browser requests the file via HTTP. When the request reaches the correct web server (hardware), the HTTP server (software) sends the requested document back, also through HTTP.



### **3.2.2 Types of web servers:**

To publish a website, you need either a static or a dynamic web server.

A static web server, or stack, consists of a computer (hardware) with an HTTP server (software). We call it "static" because the server sends its hosted files "as-is" to your browser.

A dynamic web server consists of a static web server plus extra software, most commonly an application server and a database. We call it "dynamic" because the application server updates the hosted files before sending them to your browser via the HTTP server.

For example, to produce the final webpages you see in the browser, the application server might fill an HTML template with contents from a database. Sites like Wikipedia have many thousands of webpages, but they aren't real HTML documents, only a few HTML templates and a giant database. This setup makes it easier and quicker to maintain and deliver the content.

### **3.2.3 Famous web servers:**

There are 4 primary web servers:

* [Apache](https://httpd.apache.org/) (provided by Apache)
* [IIS](http://www.iis.net/) (provided by Microsoft and short for Internet Information Server)
* [nginx](https://www.nginx.com/) (provided by NGINX, Inc. and pronounced like “Engine X”)
* and [GWS](https://en.wikipedia.org/wiki/Google_platform#Software) (provided by Google and short for Google Web Server)

Currently, Apache is the most popular with IIS gaining in popularity soon becoming the most popular web server. nginx is an extremely popular alternative as it is very fast and very lightweight, while GWS is the least used with a small percentage of use.

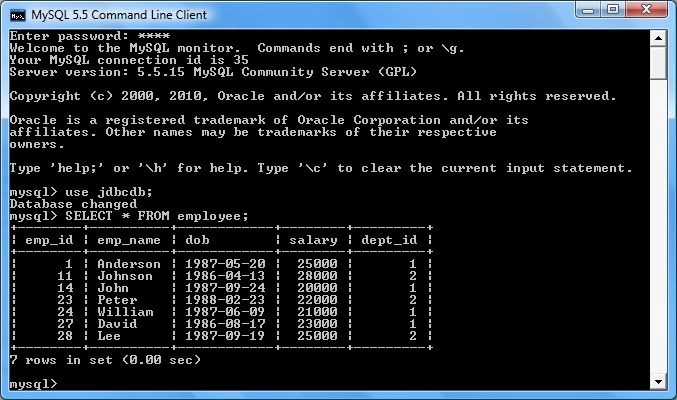
## **3.3 Databases**

Organized collection of data is called database. It is really important to organize data and different database model uses unique processes to store large data sets using processes requiring information. For example, the availability of a table in a hotel can be measured by the vacancies.

**3.3.1 Database management systems**

Database management systems (DBMSs) are computer software applications that interact with the user define programs, applications, and metadata to capture and analyze data. A general management purpose DBMS is designed to allow structure and design to any data by using some administration protocols of databases.

Some of the well-known DBMSs include MySQL, PostgreSQL, Microsoft SQL Server, Oracle, SAP and DB2. So, a database is a place to store and retrieve organized data. In practical usage, web hosting, databases store your personal blog data, the date they were live, the opinions people posted with them, and your administrative information. It’s all stored in the database classified according to the database model that they support.

using program. According to its characteristics SQLite is a popular choice as an embedded database for local/client storage in application software such as different web browsers. It is perhaps the most widely deployed database server engine used today not only by all leading browsers, operating systems but also by embedded systems, among others.

**3.4** **LAMP Stack**

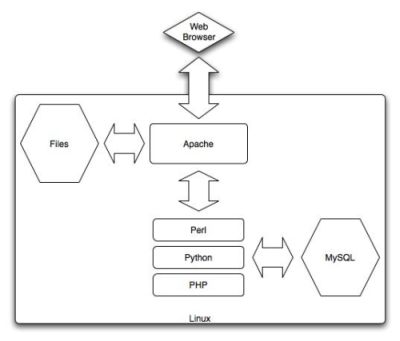
LAMP is an archetypal model of web service [stacks](https://en.wikipedia.org/wiki/Solution_stack), named as an [acronym](https://en.wikipedia.org/wiki/Acronym) of the names of its original four [open-source](https://en.wikipedia.org/wiki/Open-source) components: the [Linux](https://en.wikipedia.org/wiki/Linux) [operating system](https://en.wikipedia.org/wiki/Operating_system), the [Apache HTTP Server](https://en.wikipedia.org/wiki/Apache_HTTP_Server), the [MySQL](https://en.wikipedia.org/wiki/MySQL) [relational database management system](https://en.wikipedia.org/wiki/Relational_database_management_system) (RDBMS), and the [PHP](https://en.wikipedia.org/wiki/PHP) [programming language](https://en.wikipedia.org/wiki/Programming_language). The LAMP components are largely interchangeable and not limited to the original selection. As a solution stack, LAMP is suitable for building [dynamic web sites](https://en.wikipedia.org/wiki/Dynamic_web_site) and [web applications](https://en.wikipedia.org/wiki/Web_application).

**3.4.1 Exploding the Acronym**

Simply exploding the acronym on a letter by letter basis gives us the following elements:

* Linux
* Apache Web server
* MySQL database
* Perl, Python, or PHP

Individually, each of these items is a powerful component in its own right. The key to the idea behind LAMP, a term originally coined by Michael Kunze in the German magazine c't in 1998, is the use of these items together. Although not actually designed to work together, these open source software alternatives are readily and freely available. This has lead to them often being used together. In the past few years, their compatibility and use together has grown and been extended. Certain extensions have even been created specifically to improve the cooperation between different components.



Each of the components in the LAMP stack is an example of Free or Open Source Software (FOSS). The benefit of the FOSS approach is three-fold. First, the nature of FOSS software means applications are available for free download, making them readily available to a wide range of people without payment. That makes the software incredibly attractive to a wide range of users who would otherwise have to pay for "professional" commercial tools, which is often an expensive step in producing a Web site.

Second, FOSS licenses are open and thus have few restrictions on their use and deployment of applications based on the FOSS technology. It is possible to develop and deploy LAMP-based projects without paying any license fees for distributing the software, and this, again, makes it popular for both hobbyists and professionals alike.

Third, and a major reason for the growth and use of FOSS technology (including LAMP), is that because users have access to the source it is much easier to fix faults and improve the applications. In combination with the open license, this simplifies the development process for many enterprises and gives them flexibility that simply isn't available within the confines of a proprietary or commercial-based product.

**3.4.2 Using the LAMP Stack**

Choosing to use LAMP in your business is not about cost — although many enterprises will be attracted to the low-cost required for both development and deployment. Instead, choosing LAMP for your organization is about the benefits it provides, as summarized below.

* Flexibility: There are no limits to what you can do with the LAMP stack, either technically or because of licensing restrictions. This allows you the flexibility to build and deploy applications in a method that suits you, not the supplier of the technology you are using.
* Customization: Because LAMP components are open source, they have built up a huge array of additional components and modules that provide additional functionality. The open source approach enables you to do the same, customizing components and functionality to suit your needs.
* Ease of Development: You can write powerful applications using LAMP technology in relatively few lines of code. Often the code is straightforward enough that even nonprogrammers can modify or extend the application.
* Ease of Deployment: With neither licensing issues nor the need to compile applications, deployment is often as easy as copying an application to a new host. Most hosting services provide LAMP-based environments as standard, or they can be deployed using a Linux distribution, such as Fedora or Debian.
* Security: With many eyes developing the software and years of use by a wide range of users and community groups, LAMP technology is secure and stable. Problems are normally fixed very quickly, and without the need for a costly support contract.
* Community and Support: A wide and experienced group of people is willing to provide help and support during the development and deployment of LAMP-based applications.

Many successful business have already leveraged the use of LAMP technology. Many heavily trafficked Web sites use LAMP, or components of it, to support their applications.

# **4 IoT Robot Interface**

## **4.1 Stack Infrastructure Components**

A robot that is controlled manually by a human needs some kind of interface in order to control it. The type of robot, its functions, technologies used and purpose all decide what kind of interface to use. For a robot that will be used form huge distances with continuous stream of data, connecting the robot to an internet gateway will provide access to the robot from anywhere in the world giving it the power to be used in really remote locations as long as it’s connected to internet. The technology used to connect to internet (e.g. wifi, satellite, etc.) is not a concern in the point of view of user interface, but the characteristics of the web application are the main concern here.

**4.1.1 Goals to achieve**

In order to define the web app characteristics, actions needed to be done by the interface have to be specified first. There are multiple actions need to be done with the interface that can be listed as:

* Receiving the robot state which can be divided as:
  + Getting the current moving state
  + Getting the camera stream
  + Getting the current object recognition results for a specific frame
* Sending commands to the robot, which is essentially asking to receive one of the robot states at a given time, divided as:
  + Sending move instructions to robot
  + Request robot camera stream
  + Request recognition of objects in a specific frame

**4.1.2 Characteristics of the web interface**

We can now define the web app characteristics as:

1. **Single user:** The robot can only be controlled by 1 person at a time.
2. **Real-time System:** The app only deals with real-time data and states and doesn’t store any previous or old data or states
3. **Direct access to hardware drivers:** The interface has to request binary stream data from a camera and give movement instructions to motors

**4.1.3 Infrastructure components Decisions**

Infrastructure for web applications is the same as the web stack used including the hosting machine containing the stack. Follows are each of the components and the candidates that can be used and which candidate is the best to be used.

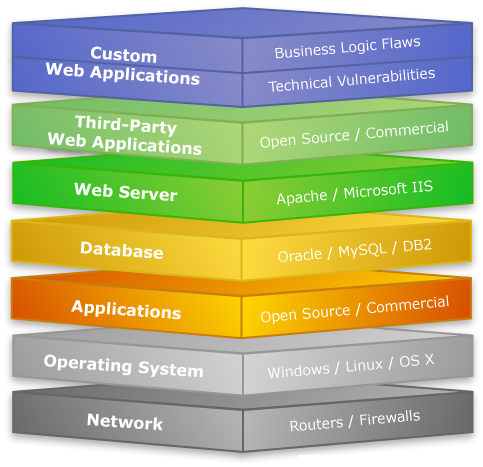


Figure 1: Web Stack components example

**4.1.3.1 Hosting machine**

The hosting machine is the hardware that serves the web app and runs its stack. There are two options for the hosting machine of the app:

* Use a hosting machine from a datacenter
* Use the raspberry pi as a the hosting machine

The datacenter basically has more CPU power, more memory, more disk space and a huge bandwidth capacity. The raspberry pi in the other hand has limited resources in all these aspects and also the robot applications obligates it to connect to internet wirelessly which increases the chance of packet loss making the bandwidth smaller than what is intended to be.



Figure 2: servers inside a datacenter

Now we show a comparison between each selection in respect to the web app characteristics:

* Being a single user system:

That means that the app won’t use many resources from the machine. As only a very limited number of requests are issued at a time. Datacenters are designed to be able to handle hunders of thousands of requests which is a huge wasted resources for such a simple app, which makes the raspberry pi resources enough for this kind of application.

* Being a Real-time System & having access to hardware drivers

Real-time systems need to have minimum delay as possible. Since the datacenters has huge bandwidths we can initially say that the datacenters would be more efficient, but then the datacenter will have to route requests again to the pi in order to access the hardware drivers making it a 2 rounds trip which adds an overhead of intermediate server communication to the initial cost of communicating with the pi directly. Which in overall add delay to the whole process, which is not wanted.

In general, even if a hosting machine within a datacenter is used, the needed for another server on raspberry pi is still needed to control the robot. And as the shortest route between 2 points is a straight line, having the **raspberry pi as the hosting machine directly is or best option** here.

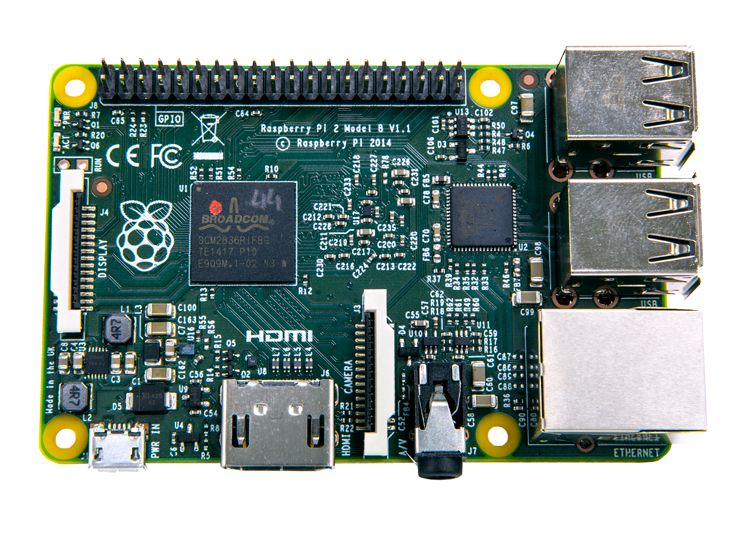


Figure 3: Raspberry pi board

**4.1.3.2 Operating system**

Choosing the raspberry pi as out hosting machine limits the options we have on choosing the operating system used to hold the stack to light-weight operation systems. As the there is many solutions to use any preferred stack in any operating system, this choice is not really a problem here. The only characteristic that concerns us here is the accessibility of the hardware in real-time. The raspberry pi supports using Raspbian OS which is a linux-debian based operating system and has great support especially with pi-specific hardware like the raspberry pi camera. That’s why **Raspbian OS is used** for this project.

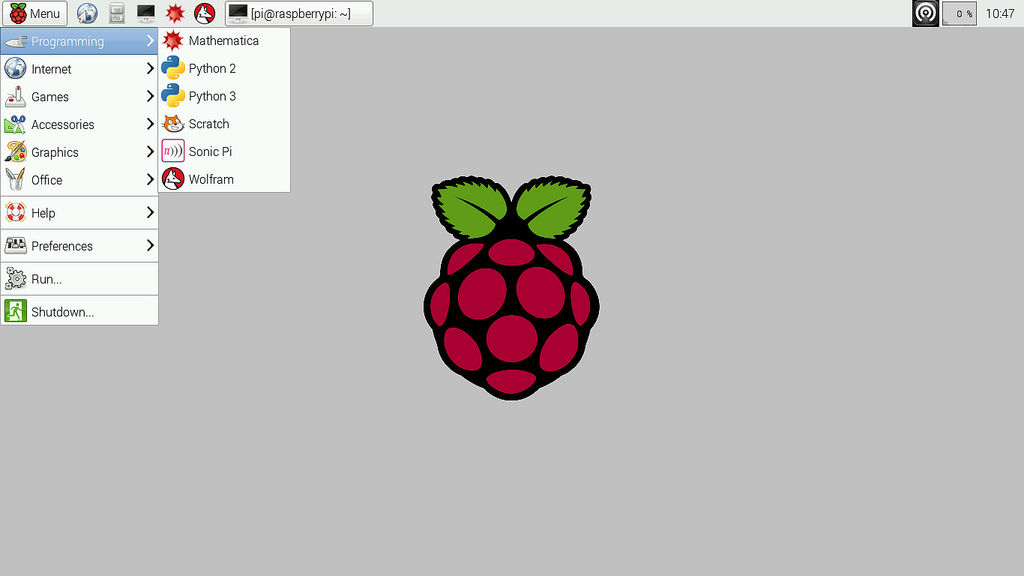


Figure 4: Raspbian OS graphical interface

**4.1.3.3 Server-side scripting language**

The server-side scripting language is the language used to write the server logic on the hosting machine, or the ‘backend’ of our web app. It doesn’t include any scripting or markup languages used to build the user interface which is run on the user’s browser, or as known as the ‘frontend’.

There are many options here like PHP, Ruby, Python, NodeJS, etc. All of these would have no problem with handling a single user or be used in a real-time application.



Figure 4: Example of server-side scripting languages

Raspberry pi offers great support for python in accessing its own hardware easily and efficiently. This makes python one of the best candidates that can combine between processing web requests and controlling system hardware like motors or camera without any need of external scripts which would add an overhead to every request.

Even if Python doesn’t have native support for web applications like PHP for example and may not have the speed of other scripting languages like Ruby but having direct access to raspberry pi GPIO, I2C lines and serial camera makes **Python the best scripting language to be used in such an application** without much competition.

**4.1.3.4 Web Server**

There are different types of web servers available like Apache, IIS, Nginx and LiteSpeed. Mainly the web server listens to HTTP requests sent to a port and runs a script based on the URL specified in the HTTP request and returns the output of the script in an HTTP response. The importance of web servers can be abstracted in two main functions: HTTP handling and threading. Threading is so important in web apps as it allows the web server to handle many requests at the same time allowing many users to use the applications in parallel.

It’s obvious that one of the main functions of web servers is not really needed because one of the main characteristics of the application is that it’s only used by a single user at a time. That allows us to use a simpler web server that can save more resources than used by its competitors, especially on a limited resources device as raspberry pi. As python is used as the server-side scripting language, Python offers many solutions that can be used to handle light, small requests at a time. So the **flask development server was used** to easily and efficiently run our web app on the raspberry pi.

**4.1.3.5 Database**

One of the main characteristics of the application is that it’s a real-time application that doesn’t store any previous or old data or states. That means there is **no need to add a database to our stack at all**. This is more suitable to the limited space presented by the raspberry pi which is most used anyway for the recognition feature which is discussed in a different section.

## **4.2 Robot control interface**

The control interface is the graphical interface that the user used to control the robot over the internet. Let’s recall the main functions the interface has to be capable of doing:

* Receiving the robot state:
  + Getting the current moving state
  + Getting the camera stream
  + Getting the current object recognition results for a specific frame
* Sending commands to the robot:
  + Sending move instructions to robot
  + Request robot camera stream
  + Request recognition of objects in a specific frame

**4.2.1 Interface components**

Putting the required actions in mind, the interface can be divided into 4 parts:

1. The video stream: which show what’s in front of the robot to be able to control it remotely
2. The robot controls: which are divided into:
   1. Movement controls: which control the robot movement
   2. Recognition button: which orders the robot to recognize what is in front of it
3. State display: which shows the current states of all the robot functions

Below is the control interface built, showing all of its components used to achieve the desired functions.

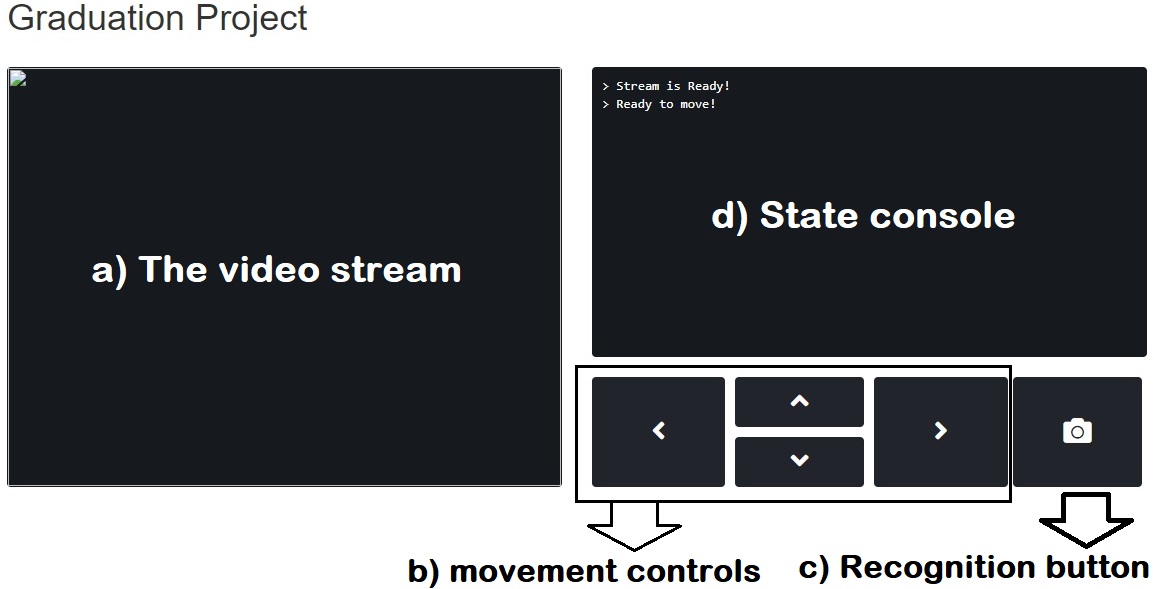


Figure 1: control interface design  
a) Video stream, b) movement controls,  
c) recognition button, d) state console

**4.2.2 Interface layers**

The control interface is constructed as a web page and can be accessed though a browser. The interface can be divided into 3 main layers: The structure layer, the presentation layer and the behavior layers:

1. The structure or content layer
2. The Presentation layer
3. The Behavior layer

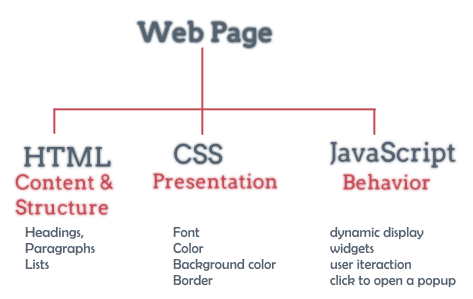


Figure 2: Web page layers

**4.2.2.1 Structure layer**

The structure or content layer of a web page is the underlying HTML code of that page. HTML consists of a series of short codes typed into a text file, these are the tags. A browser reads the file and translates the text into a visible form. The structure layer is just as a house's frame creates a strong foundation upon which the rest of the house is built, a solid foundation of HTML creates a platform upon which a website can be created.

The structure layer is where all the components of the interface are stored. HTML structure can consist of text and images, and can include hyperlinks to navigate through the interface.  This is coded in standards-compliant HTML5 and can include text, images, and multimedia (video, audio, etc.).

Every aspect of a site's content should be represented in the structure layer, including other layers resources as well. This allows users who have JavaScript turned off or who can't view CSS access to the entire website, if not all of its functionality.

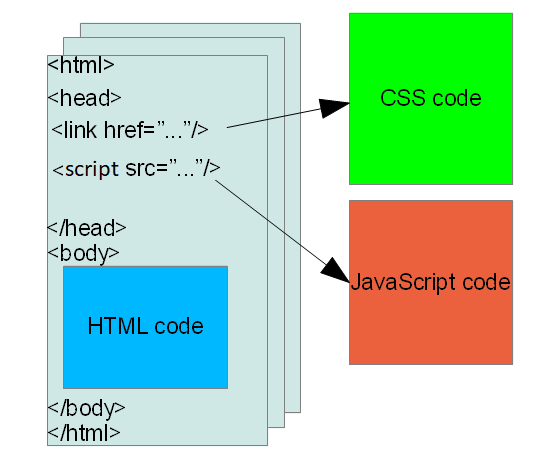


Figure 3: example of HTML document

Below is a screenshot of the control interface structure without using any styles at all

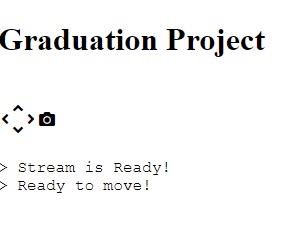


Figure 4: Control interface without styles

All the interface components are presented like the movement arrows and the capture button and state console. But there are no visible containers to present the components clearly. Also the stream section is not visible at all yet.

**4.2.2.2 Presentation layer**

This layer dictates how a structured HTML document will look to a user and is defined by CSS (Cascading Style Sheets). These files contain stylistic instructions for how the document should be displayed in a web browser. The style layer usually includes [media queries](https://www.lifewire.com/how-do-you-write-css-media-queries-3469990) that change a site's display based on [screen size and device](https://www.lifewire.com/multi-device-web-design-3470008).

All visual styles for a website should reside in an external style sheet. Multiple style sheets can be used but are not needed for a 1 page interface.

Below is a screenshot of the control interface after applying styles to it

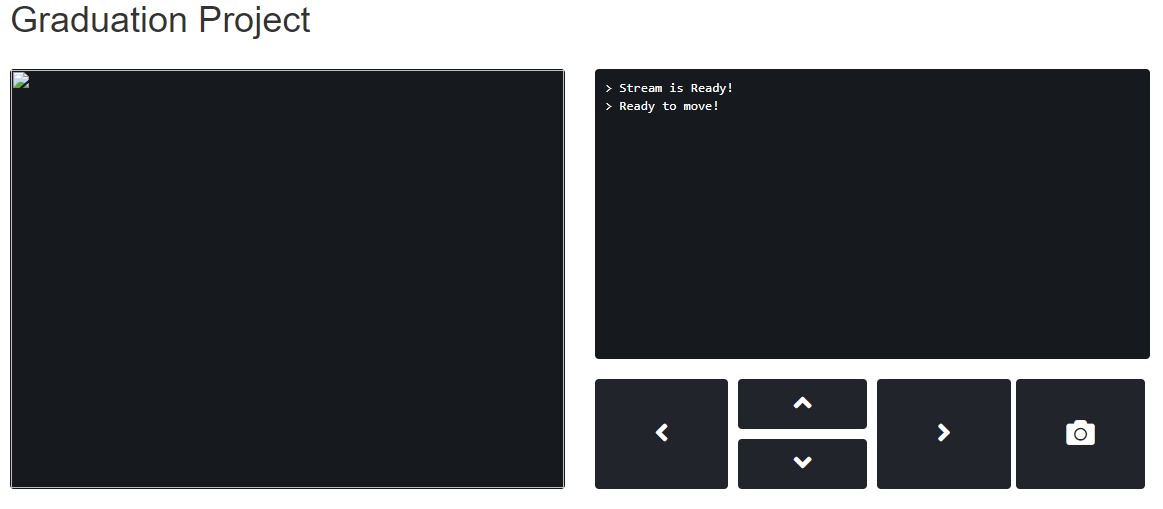
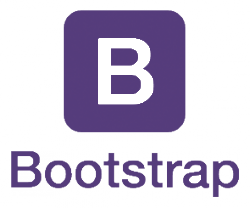
****

Figure 5: Control interface with styles on desktop

Now all the components of the interface are clear and can be used more easily even for a person who uses this interface for the first time.

The interface looks great and very usable on a desktop machine with a large screen, but it’s not the same for tablets and mobile phones; which are mostly used to control the robot. That’s why we have to add ‘responsiveness’ to the control interface.

Responsiveness in the presentation layer means the ability to use the interface in any given device or screen size. That’s why ‘bootstrap’ is used in the interface. Bootstrap is a front-end web framework that was created by Twitter for faster creation of device responsive web applications.

Below is a screenshot of the control interface on a smart phone after using bootstrap to make it responsive

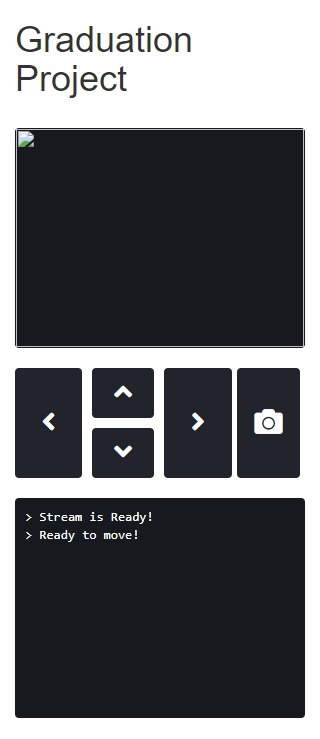


Figure 6: Control interface on a smart phone

**4.2.2.3 Behavior layer**

The behavior layer makes a website interactive, allowing the page to respond to user actions or to change based on a set of conditions. JavaScript is the most commonly used language for the behavior layer.

When developers refer to the behavior layer, most of them mean the layer that is activated directly in the web browser. This layer is used to interact directly with the DOM (Document Object Model). [Writing valid HTML](https://www.lifewire.com/best-free-html-editors-for-windows-3471313) in the content layer is important for DOM interactions in the behavior layer. An example of such an interaction is clicking on a direction arrow on the document, which is considered a DOM element; the behavior layer will interact with that action by sending a request to the robot to move at the same direction is the clicked direction.

The behavior layer has 3 responsibilities in the robot control interface:

1. Capture user interactions: it’s used to capture mouse clicks and keyboard pressed keys and bind those events to a specific behavior (e.g. send a request to the robot to move)
2. Highlight user actions: It manipulate DOM elements to indicate that a certain event has happened like changing the style of a pressed button or write message on the console
3. Check robot availability: The robot, as well as the user’s device, might experience connection losses. It would be inconvenient to reload the interface every time such a loss happens. That’s why the behavior layer is responsible of checking the robot state and attempt to reconnect when the connection is lost at any given time.

It’s clear that the behavior layer is responsible of all the communication between the frontend in the user’s device and the robot itself. Using asynchronous requests to the server in order to not freeze all functions in the interface till one request is finished (aka. AJAX).

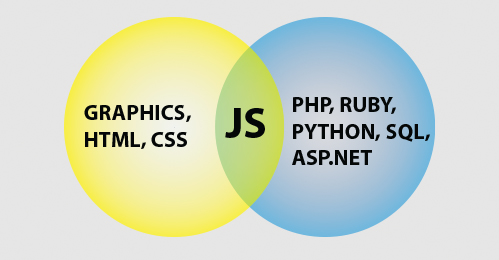


Figure 7: Frontend and backend interactions in web app

This layer is the layer that requests and displays all functions mentioned before and required for the interface and the robot to function correctly.

**4.3 Server-side Robot Logic**

The control interface of the robot is considered the frontend in web terminologies as it is what the user interacts with; but it doesn’t perform the actual functions of the robot. The interface captures the user interactions the movement controls and sends a request to the robot to move accordingly but it doesn’t move the robot itself. The same goes for video stream and object recognition; the interface only requests the function but doesn’t perform it itself.

The server-side script written is the one responsible of receiving and processing HTTP requests and interacting with the hardware drivers and does the actual function required; which is considered the backend of the application.

**4.3.1 Backend functionalities**

The backend consists of a flask python app. That script is responsible of doing all the functionalities of the system; these functionalities are detailed as:

* Serving the control interface upon request
* Capture camera frames and send it back to requests as stream
* Send movement instructions to the underlying microcontroller via a common communication protocol
* Recognize objects in a frame and send the results back upon request

It’s clear that the app only do a function only if it received a request to do it, that’s why it’s basically called a ‘server’. All of these functions required direct access to hardware drivers; which Python presents especially on raspberry pi devices.

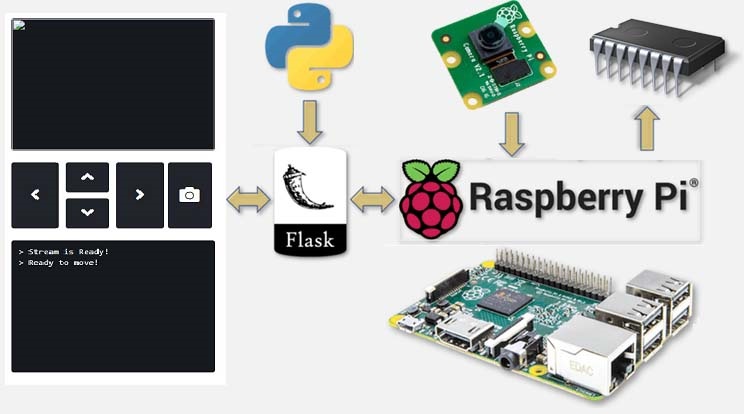


Figure 8: Raspberry pi interactions

**4.3.1 Backend structure**

The structure of the backend can be divided into 2 parts.

1. Files structures: Which represents the separation of concerns in the application
2. Routes structure: Which is represented by the HTTP URLs used to access different robot functionalities

The files structure is listed below; but it’s not our concern for now as it might change without affecting the way the robot behaves at all.

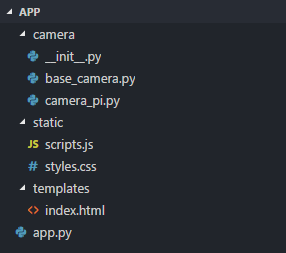


Figure 9: Backend files structure

Routes structure is the logical structure of paths used by the control interface to access specific function. Essentially each functionality the app does has its own HTTP route that browsers can send requests to and a for each route there is a function that process the HTTP request and serves the desired functionality back in a server-client way of communication.

**4.3.1.2 How routes work**

Each route constructs a URL. The server accepts specific HTTP verbs for each URL providing a standardized API (Application programming interface) to make communications clear and easy.

For example: If the raspberry pi has IP address of *192.168.0.1* and the web server listens to port number *5000* for HTTP requests, and the server serves the camera stream through the route *`/video\_feed`* if it gets a GET HTTP request; then a user, or in our case the control interface, can access video stream by sending a GET request to:

*http://192.168.0.1:5000/video\_feed*

Requests can be sent asynchronously to the backend using the control interface. Routes that accept GET requests like the root route and the video stream route can be accessed directly through a browser by visiting their URLs as described in the previous example.

**4.3.1.3 Routes mapping**

Follows are the routes structure for the backend app, showing the HTTP verb used to access them and the required query data to be sent in order for the route to response successfully. How each route does its function and how the response of each route is calculated is described with more detail in later sections.

1. *`/`* The root route  
   The root route is the route specified by only the domain name or IP address of the server with the port number only.
   1. HTTP verb: GET
   2. Required query data: None
   3. Response: returns the control interface to the browser
2. *`/video\_feed`* The stream route
   1. HTTP verb: GET
   2. Required query data: None
   3. Response: returns *multipart/x-mixed-replace* HTTP response with an MJPEGimages stream creating the video stream
3. *`/move`* The movement instructions gateway
   1. HTTP verb: POST
   2. Required query data: data specifying the required movements to be done
      1. ‘left’ (Boolean)
      2. ‘up’ (Boolean)
      3. ‘right’ (boolean)
      4. ‘down’ (boolean)
   3. Response: JSON formatted response containing the current robot moving state
4. *`/recognize`* The stream route
   1. HTTP verb: POST
   2. Required query data: None
   3. Response: JSON formatted response containing the results of the recognition operation

**4.3.2 Functions implementation**

This is the deepest point of the web application. The next layer is a hardware layer and a microcontroller layer. Follows are the detailed way of how the backend communicates with hardware to make a certain function happen.

**4.3.2.1 Serving the control interface**

The first thing the application does is to serve the robots control interface to the user in order to interact with the rest of the robot. This is done simply by sending a response containing the HTML document of the control interface.

Once the browser receives the HTML document, it will render it requesting any resources specified in the HTML document; which are basically:

* CSS style sheets to handle the document presentation
* JS scripts to handle events and interactions with the interface
* The video streaming specified as a link of an image that should be fetched

The CSS and JS files are static files that the server serves to the user the same way it serves the HTML document of the control interface.

**4.3.2.2 Video Streaming**

Streaming was mentioned for awhile now without defining streaming in a technical way. Streaming is a technique in which the server provides the response to a request in chunks. Streams are useful in these cases:

* **Very large responses:** Having to assemble a response in memory only to return it to the client can be inefficient for very large responses. An alternative would be to write the response to disk and then return the file but that adds I/O to the mix. Providing the response in small portions is a much better solution, assuming the data can be generated in chunks.
* **Real time data:** For some applications a request may need to return data that comes from a real time source. An example of this is a real time video or audio feed. A lot of security cameras use this technique to stream video to web browsers.

An interesting use of streaming is to have each chunk replace the previous one in the page, as this enables streams to animate in the browser. With this technique you can have each chunk in the stream be an image, and that gives a video feed that runs in the browser.

The secret is to use a multipart HTTP response, which consist of a header that includes one of the multipart content types, followed by the parts, separated by a boundary marker and each having its own part specific content type.

There are several multipart content types for different needs. For the purpose of having a stream where each part replaces the previous part the *`multipart/x-mixed-replace`* content type must be used. This is the structure of a multipart video stream:

HTTP/1.1 200 OK

Content-Type: multipart/x-mixed-replace; boundary=frame

--frame

Content-Type: image/jpeg

<jpeg data here>

--frame

Content-Type: image/jpeg

<jpeg data here>

As seen above, the structure is pretty simple. The main *Content-Type* header is set to *multipart/x-mixed-replace* and a boundary string is defined. Then each part is included, prefixed by two dashes and the part boundary string in their own line. The parts have their own *Content-Type* header, and each part can optionally include a *Content-Length* header with the length in bytes of the part payload, but at least for images browsers are able to deal with the stream without the length.

There are many ways to stream video to browsers, and each method has its benefits and disadvantages. The method that works well with the streaming feature of Flask and the application nature that needs to extract frames out of the stream to recognize objects in it is to stream a sequence of independent JPEG pictures. This is called [Motion JPEG](http://en.wikipedia.org/wiki/Motion_JPEG), and is used by many IP security cameras. This method has low latency, but quality is not the best, since JPEG compression is not very efficient for motion video.

Once the server receives a request for the video feed, it starts to create a *multipart/x-mixed-replace* HTTP response appending a camera frame as JPEG image to it followed by the boundary line and keeps updating and sending that response till the stream page is closed.