

Module 5 : Web Applications For IoT

Faculty of Ocean Engineering Technologies & Informatics
Universiti Malaysia Terengganu
2023



MODULE OUTLINE



5.1

MODULE 5.1
Introduction to
Web Apps
Development
for IoT

MODULE 5.2
Designing web
pages using
HTML and CSS

5.2

5.3

MODULE 5.3
Interactivity of
web pages
using Javascript

MODULE 5.4
Data storage for IoT
data

5.4

5.5

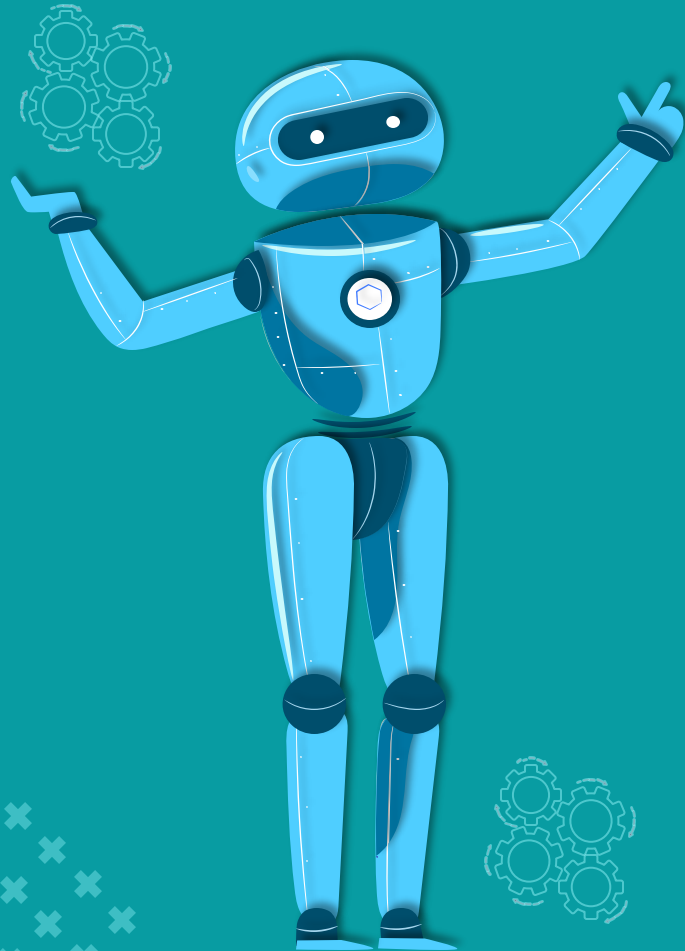
MODULE 5.5
Visualization of
IoT data

5.1



Introduction to Web Apps Development for IoT

- Overview of Internet Application
- Web Application Technology



5.1

Introduction to Web Apps Development for IoT

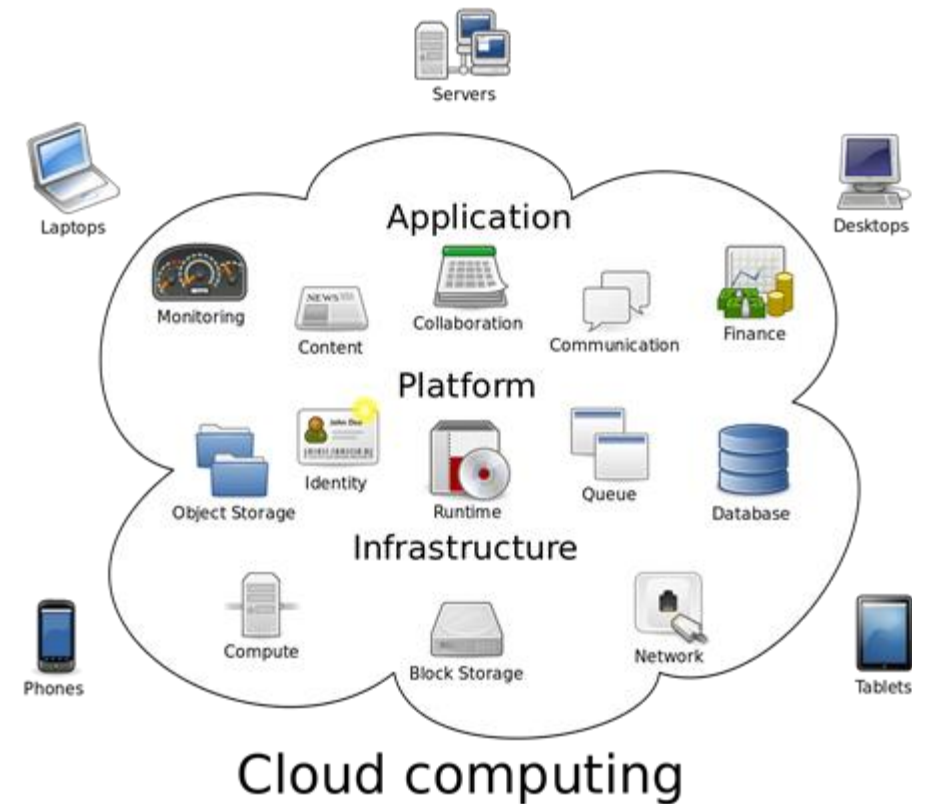
5.1.1 Overview of Internet Application

Keys:

1. To achieve seamless integration between business entities.
2. Machine to Machine
3. Human to Machine
4. Automated business process
5. Effective use of resources - human, machines, materials

Drawbacks:

1. Securities
2. Lack of expertise
3. Partner readiness / resistance to change
4. Need investment





5.1

Introduction to Web Apps Development for IoT

5.1.1 Overview of Internet Application

What are the internet and www?

- The Internet is the global system of interconnected computer networks that uses the Internet protocol suite (TCP/IP) to link devices worldwide.
- These networks are linked with various electronics devices and cables such as copper, fiber optics and radio waves.
- It can be used to transmit vast range of information resources such as text, images, emails, files as well as multimedia such as video streaming and voice or telephony.
- www is interchangeably used as internet, but there are not the same.
- World wide web is referred to as more to the application, while internet is the global interconnecting infrastructure of communication system.



5.1

Introduction to Web Apps Development for IoT

5.1.2 Web Application Technology - Web Server and Client

- Web Server is a software that can process the client request and send the response back to the client.
- For example, Apache is one of the most widely used web servers. Web Server runs on some physical machine and listens to client request on a specific port.
- A web client is a software that helps in communicating with the server.
- Some of the most widely used web clients are Firefox, Google Chrome, Safari, etc.
- When we request something from the server (through URL), the web client takes care of creating a request and sending it to the server and then parsing the server response and present it to the user.



5.1

Introduction to Web Apps Development for IoT

5.1.2 Web Application Technology - Web Server and Client

- Web Server and Web Client are two separate softwares, so there should be some common language for communication.
- HTML is the common language between server and client and stands for HyperText Markup Language.
- Web server and client needs a common communication protocol, HTTP (HyperText Transfer Protocol) is the communication protocol between server and client.
- HTTP runs on top of TCP/IP communication protocol.



5.1

Introduction to Web Apps Development for IoT

5.1.2 Web Application Technology - Web Server and Client

- Some of the important parts of the HTTP Request are:
 - **HTTP Method** – action to be performed, usually GET, POST, PUT etc.
 - **URL** – Page to access
 - **Form Parameters** – similar to arguments in a java method, for example user,password details from login page.



5.1

Introduction to Web Apps Development for IoT

5.1.2 Web Application Technology - HTTP

- HTTP means HyperText Transfer Protocol.
- HTTP is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.
- For example, when you enter a URL in your browser, this actually sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page. The other main standard that controls how the World Wide Web works is HTML, which covers how Web pages are formatted and displayed.



5.1

Introduction to Web Apps Development for IoT

5.1.2 Web Application Technology - HTTP

- HTTP is a Stateless Protocol
- HTTP is called a stateless protocol because each command is executed independently, without any knowledge of the commands that came before it.
- This is the main reason that it is difficult to implement Web sites that react intelligently to user input.
- This shortcoming of HTTP is being addressed in a number of new technologies, including ActiveX, Java, JavaScript and cookies.



5.1

Introduction to Web Apps Development for IoT

5.1.2 Web Application Technology - HTTP

- The primary or most-commonly-used HTTP verbs (or methods, as they are properly called) are **POST, GET, PUT, PATCH, and DELETE**.
- These correspond to create, read, update, and delete (or CRUD) operations, respectively.

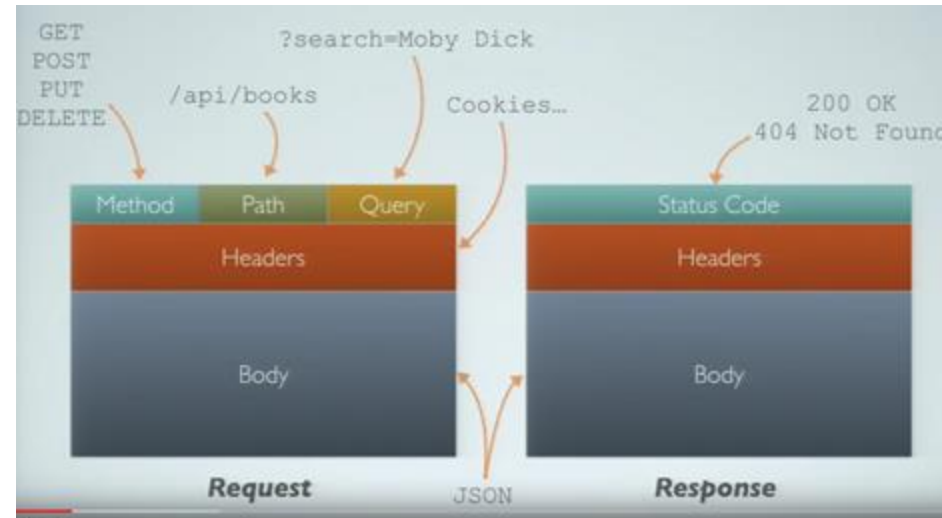
HTTP Verb	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

5.1

Introduction to Web Apps Development for IoT

5.1.2 Web Application Technology - HTTP

Example: accessing static web-pages and uploading files or images to a web server





5.1

Introduction to Web Apps Development for IoT

5.1.2 Web Application Technology - HTTP

Sample HTTP Request:

GET /FirstServletProject/jsps/hello.jsp HTTP/1.1

Host: localhost:8080

Cache-Control: no-cache



5.1

Introduction to Web Apps Development for IoT

5.1.2 Web Application Technology - HTTP

Some of the important parts of HTTP Response are:

Status Code – an integer to indicate whether the request was success or not. Some of the well known status codes are 200 for success, 404 for Not Found and 403 for Access Forbidden.

Content Type – json,text, html, image, pdf etc. Also known as MIME type

Content – actual data that is rendered by client and shown to user.



5.1

Introduction to Web Apps Development for IoT

5.1.2 Web Application Technology - HTTP

- Response Phase Form
 - Status line
 - Response header fields
 - blank line
 - Response body
- Status line format:
HTTP version status code explanation
Example: HTTP/1.1 200 OK
- Status code is a three-digit number; first digit specifies the general status
 - 1 => Informational
 - 2 => Success
 - 3 => Redirection
 - 4 => Client error
 - 5 => Server error
 - Status code 404 is for?????



5.1

Introduction to Web Apps Development for IoT

5.1.2 Web Application Technology - HTTP

Sample HTTP Response:

```
200 OK
Date: Wed, 07 Aug 2013 19:55:50 GMT
Server: Apache-Coyote/1.1
Content-Length: 309
Content-Type: text/html; charset=US-ASCII

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"https://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=US-
ASCII">
<title>Hello</title>
</head>
<body>
<h2>Hi There!</h2>
<br>
<h3>Date=Wed Aug 07 12:57:55 PDT 2013
</h3>
</body>
</html>
```

- MIME Type or Content Type: If you see above sample HTTP response header, it contains tag "Content-Type".
- It's also called MIME type and server sends it to the client to let them know the kind of data it's sending.
- It helps the client in rendering the data for the user.
- Some of the most used mime types are text/html, text/xml, application/xml etc.



5.1

Introduction to Web Apps Development for IoT

5.1.2 Web Application Technology - URL

A Uniform Resource Locator (URL), colloquially termed a web address

DOMAIN NAME

http://www.verisign.com/domain-names/online/index.shtml

URL



5.1

Introduction to Web Apps Development for IoT

5.1.2 Web Application Technology - URL

`https://localhost:8080/FirstServletProject/jsps/hello.jsp`

- **https://** – This is the first part of URL and provides the communication protocol to be used in server-client communication.
- **localhost** – The unique address of the server, most of the times it's the hostname of the server that maps to unique IP address. Sometimes multiple hostnames point to same IP addresses and web server virtual host takes care of sending a request to the particular server instance.
- **8080** – This is the port on which server is listening, it's optional and if we don't provide it in URL then request goes to the default port of the protocol. Port numbers 0 to 1023 are reserved ports for well-known services, for example, 80 for HTTP, 443 for HTTPS, 21 for FTP, etc.
- **FirstServletProject/jsps/hello.jsp** – Resource requested from server. It can be static html, pdf, JSP, servlets, PHP etc.

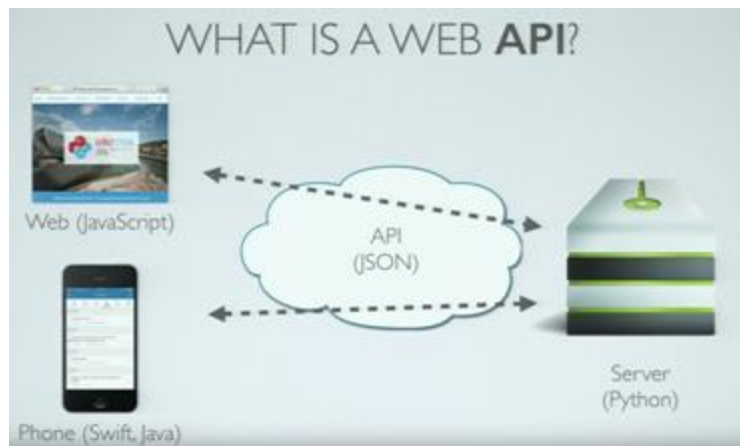
5.1

Introduction to Web Apps Development for IoT

5.1.2 Web Application Technology - RESTful

Representational state transfer (REST) or RESTful web services is a way of providing interoperability between computer systems on the Internet.

More advanced way of accessing and manipulating “web resource”.



REST CONVENTIONS				
	GET	PUT	POST	DELETE
Collection /books	List books		New book	
Item /books/123	Display book	Update book		Delete book
Controller /books/123/borrow			Borrow book	



5.1

Introduction to Web Apps Development for IoT

5.1.2 Web Application Technology - Endpoints or API

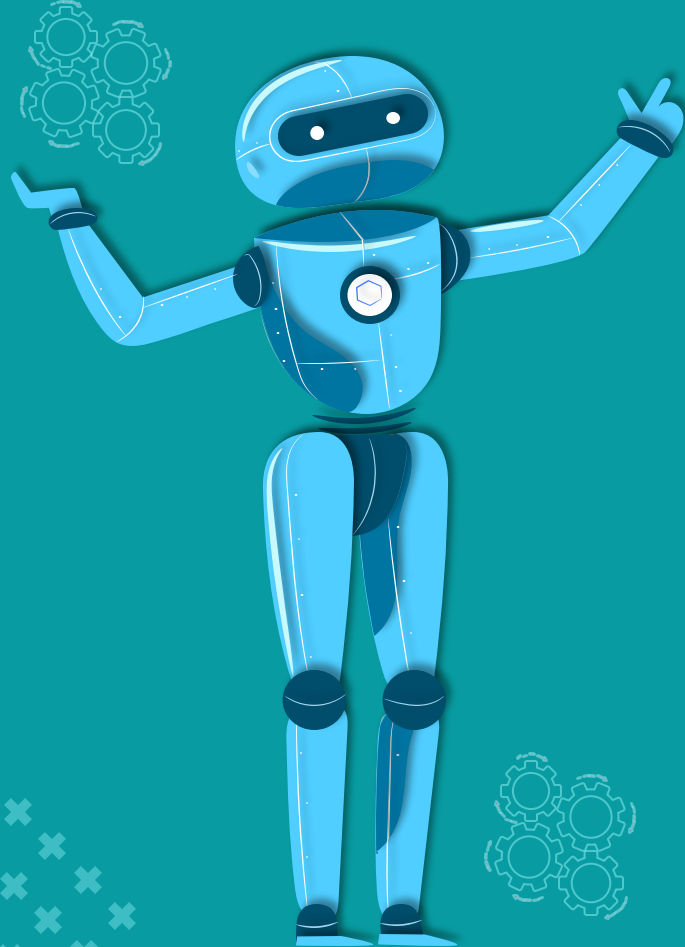
- API endpoints are the specific digital location where requests for information are sent by one program and where the resource lives. Endpoints specify where APIs can access resources and help guarantee the proper functioning of the incorporated software.
- An API essentially provides the language and contract for how two systems interact. Each API has documentation and specifications which determine how information can be transferred.
- Web application API uses URL format such as REST protocol, and speaks different HTTP 1.1 verbs (GET, POST, PUT, and DELETE) to perform tasks on the server.

5.2



Designing web pages using HTML and CSS

- Introduction to HTML
- Introduction to CSS
- Web Server/ Web-Framework





5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Origins and Evolution of HTML

- Derived from SGML
- Original intent: General layout of documents that could be displayed by a wide variety of computers
- Recent versions:
 - HTML 4.0 – 1997
 - Introduced many new features and deprecated many older features
 - HTML 4.01 - 1998 - A cleanup of 4.0
 - HTML5 was first released in public-facing form on 22 January 2008.



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Origins and Evolution of HTML

HTML VERSUS HTML5

HTML	HTML5
No standardized process to handle structurally incorrect HTML codes.	It supports persistent error handling via improvised error handling process.
It's not mobile friendly.	It's mobile friendly.
No audio and video support in HTML.	Audio/video elements can be integrated directly onto a web page.
It does not support all major web browsers.	It is supported by all major web browsers.
Vector Graphics is possible when used in conjunction with Flash, Silverlight, or similar third-party plugins.	Scalable Vector Graphics (SVG) is an integral part of the HTML5 language specification.
It does not allow JavaScript to run in browser.	It allows JavaScript to run in background.



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - HTML File

- HTML stands for Hyper Text Markup Language
- An HTML file is a text file containing small markup tags
- The markup tags tell the Web browser how to display the page
- An HTML file must have an 'htm' or 'html' file extension
- An HTML file can be created using a simple text editor , like....?
- HTM or HTML Extension?



5.2

Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Basic Syntax

- HTML elements are defined using HTML tags.
- HTML tags are surrounded by the two characters `<` and `>`, called angle brackets
- HTML tags normally come in pairs like `` and ``
- The first tag in a pair is the start tag, the second tag is the end tag
- The text between the start and end tags is the element content
- HTML tags are not case sensitive, `` means the same as ``



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Basic Syntax

- This is an HTML element:
`This text is bold`
- What is the start tag, content of the HTML element and the end tag?
- Tags can have attributes
 - Attributes can provide additional information about the HTML elements on your page.
 - Example, `<body bgcolor="red">`
 - Attributes always come in name/value pairs like this: name="value".
 - Quote Styles, "red" or 'red'?
- Comment form: `<!-- ... -->`
- Browsers ignore comments, unrecognizable tags, line breaks, multiple spaces, and tabs



5.2

Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Document Structure

The whole document must have `<html>` as its root

A document consists of a head and a body or frameset

The `<title>` tag is used to give the document a title, which is normally displayed in the browser's window title bar (at the top of the display)

```
<html>
```

```
<head>
```

```
<title>Title of page</title>
```

```
</head>
```

```
<body> This is my first homepage. <b>This text is
```

```
bold</b>
```

```
</body>
```

```
</html>
```



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Text Formatting

- Paragraph Elements
 - The <p> tag breaks the current line and inserts a blank line
 - The new line gets the beginning of the content of the paragraph
- W3C HTML Validation Service
 - <http://validator.w3.org/file-upload.html>
- Line breaks
 - The effect of the
 tag is the same as that of <p>, except for the blank line, No closing tag!
- Example of paragraphs and line breaks

On the plains of hesitation <p> bleach the bones of countless millions
 who, at the dawn of victory
 sat down to wait, and waiting, died.

- What is the typical display of this text??



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Text Formatting

- Headings
 - Six sizes, 1 - 6, specified with <h1> to <h6>
 - 1, 2, and 3 use font sizes that are larger than the default font size
 - 4 uses the default size
 - 5 and 6 use smaller font sizes

Aidan's Airplanes (h1)

The best in used airplanes (h2)

"We've got them by the hangarful" (h3)

We're the guys to see for a good used airplane (h4)

We offer great prices on great planes (h5)

No returns, no guarantees, no refunds, all sales are final! (h6)



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Text Formatting

- Font Styles and Sizes (can be nested)
 - Boldface - ``
 - Italics - `<i>`
 - Larger - `<big>`
 - Smaller - `<small>`
 - Monospace - `<tt>`
- Superscripts and subscripts
 - Subscripts with `<sub>`
 - Superscripts with `<sup>`
 - Example: `x₂³`
 - Display: `x23`
- All of this font size and font style stuff can be done with style sheets, but these tags are not yet deprecated



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Text Formatting

- Character Entities
 - There are some characters that HTML treats as special characters, so if you want one in a document, it must be coded
 - The Most Common Character Entities:

Result	Description	Entity Name	Entity Number
	non-breaking space	 	
<	less than	<	<
>	greater than	>	>
&	ampersand	&	&
"	quotation mark	"	"
'	apostrophe	'	'

- Horizontal rules
 - `<hr />` draws a line across the display, after a line break



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Links

- A link is specified with the href (hypertext reference) attribute of <a> (anchor) tag
- The content of <a> is the visual link in the document
- The syntax of creating an anchor:

```
<a href="url">Text to be displayed</a>
```

- This anchor defines a link to W3Schools:

```
<a href="http://www.w3schools.com/">Visit W3Schools!</a>
```

- How the line above will look like this in a browser??
- Links can have images:

```
<a href = "c210data.html" <img src = "smallplane.jpg" alt = "Small picture of an airplane " />>Info on C210 </a>
```




5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Links

- The Target Attribute
 - With the target attribute, you can define where the linked document will be opened.

```
<a href=http://www.test.com/ target="_blank">Hi</a>
```
- The Anchor Tag and the Name Attribute
 - The name attribute is used to create a named anchor. When using named anchors we can create links that can jump directly into a specific section on a page.
 - The line below defines a named anchor:

```
<a name="tips">Useful Tips Section</a>
```
 - To link directly to the "tips" section, add a # sign and the name of the anchor to the end of a URL, like this:

```
<a href="http://www.w3schools.com/links.htm#tips"> Jump to the Useful Tips Section</a>
```
 - A hyperlink to the Useful Tips Section from WITHIN the file "links.htm" will look like this:

```
<a href="#tips">Jump to the Useful Tips Section</a>
```



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Images

- Images are inserted into a document with the `` tag with the `src` attribute
- The `` tag is empty, which means that it contains attributes only and it has no closing tag.
- `src` stands for "source". The value of the `src` attribute is the URL of the image you want to display on your page.
- The syntax of defining an image: ``
- The `Alt` attribute is used to define an "alternate text" for an image. The value of the `alt` attribute is an author-defined text:

```

```

- In which cases, the `ALT` attribute will be useful??



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - List

- Unordered Lists
 - The list items are marked with bullets (typically small black circles).
 - An unordered list starts with the `` tag. Each list item starts with the `` tag
` Coffee Milk `
 - Here is how it looks in a browser:
 - Coffee
 - Milk
- Ordered Lists
 - The list items are marked with numbers.
 - An ordered list starts with the `` tag. Each list item starts with the `` tag.
` Coffee Milk `
 - Here is how it looks in a browser:
 1. Coffee
 2. Milk



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Tables

- A table is a matrix of cells, each possibly having content
 - The cells can include almost any element
- A table is specified as the content of a `<table>` tag
- A border attribute in the `<table>` tag specifies a border between the cells
- Tables are given titles with the `<caption>` tag, which can immediately follow `<table>`
- Each row of a table is specified as the content of a `<tr>` tag
- The row headings are specified as the content of a `<th>` tag
- The contents of a data cell is specified as the content of a `<td>` tag



5.2

Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Tables

```
<table border =  
"border"><caption> Fruit Juice  
Drinks </caption>  
  <tr>  
    <th> </th>  
    <th> Apple </th>  
    <th> Orange </th>  
    <th> Screwdriver </th>  
  </tr>  
  <tr>  
    <th> Breakfast </th>  
    <td> 0 </td>  
    <td> 1 </td>  
    <td> 0 </td>  
  </tr>
```

```
    <tr>  
      <th> Lunch </th>  
      <td> 1 </td>  
      <td> 0 </td>  
      <td> 0 </td>  
    </tr>  
    <tr>  
      <th> Dinner </th>  
      <td> 0 </td>  
      <td> 0 </td>  
      <td> 1 </td>  
    </tr>  
</table>
```

	Apple	Orange	Screwdriver
Breakfast	0	1	0
Lunch	1	0	0
Dinner	0	0	1



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Tables

- A table can have two levels of column labels
 - If so, the colspan attribute must be set in the <th> tag to specify that the label must span some number of columns

```
<tr>
```

```
    <th colspan = "3"> Fruit Juice Drinks </th>
```

```
</tr>
```

```
<tr>
```

```
    <th> Orange </th>
```

```
    <th> Apple </th>
```

```
    <th> Screwdriver </th>
```

```
</tr>
```

Fruit Juice Drinks		
Orange	Apple	Screwdriver



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Tables

- If the rows have labels and there is a spanning column label, the upper left corner must be made larger, using rowspan

```
<table border = "border">
```

```
<caption> Fruit Juice Drinks and Meals
```

```
</caption>
```

```
<tr>
```

```
<td rowspan = "2"> </td>
```

```
<th colspan = "3"> Fruit Juice Drinks
```

```
</th>
```

```
</tr>
```

```
<tr>
```

```
<th> Apple </th>
```

```
<th> Orange </th>
```

```
<th> Screwdriver </th>
```

```
</tr>
```

```
.....
```

```
</table>
```

	Fruit Juice Drinks		
	Apple	Orange	Screwdriver
Breakfast	0	1	0
Lunch	1	0	0
Dinner	0	0	1



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Forms

- A form is the usual way information is gotten from a browser to a server
- HTML has tags to create a collection of objects that implement this information gathering
 - The objects are called widgets (e.g., radio buttons and checkboxes)
 - When the Submit button of a form is clicked, the form's values are sent to the server
 - All of the widgets, or components of a form are defined in the content of a `<form>` tag
- The only required attribute of `<form>` is `action`, which specifies the URL of the application that is to be called when the Submit button is clicked

action = <http://www.cs.ucp.edu/cgi-bin/survey.pl>

- If the form has no action, the value of action is the empty string



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Forms

- The method attribute of <form> specifies one of the two possible techniques of transferring the form data to the server, get and post.
 - What are differences between these two methods?
- Widgets
 - Many are created with the <input> tag- The type attribute of <input> specifies the kind of widget being created
- Text
 - Creates a horizontal box for text input
 - Default size is 20; it can be changed with the size attribute
 - If more characters are entered than will fit, the box is scrolled (shifted) left
 - If you don't want to allow the user to type more characters than will fit, set maxlength, which causes excess input to be ignored

```
<input type = "text" name = "Phone" size = "12" >
```



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Forms

- Checkboxes - to collect multiple choice input
 - Every checkbox requires a value attribute, which is the widget's value in the form data when the checkbox is 'checked'
 - A checkbox that is not 'checked' contributes no value to the query string
 - By default, no checkbox is initially 'checked'
 - To initialize a checkbox to 'checked', the checked attribute must be set to "checked"



5.2

Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Forms

Grocery Checklist

```
<form action = "">
```

```
<p>
```

```
<input type = "checkbox" name = "groceries"  
  value = "milk" checked = "checked">
```

Milk

```
<br/>
```

```
<input type = "checkbox" name = "groceries"  
  value = "bread"> Bread
```

```
<br/>
```

```
<input type = "checkbox" name = "groceries"  
  value= "eggs"> Eggs
```

```
</p>
```

```
</form>
```

Grocery Checklist

☒ Milk ☐ Bread ☐ Eggs


5.2

Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Forms

- Radio Buttons - collections of checkboxes in which only one button can be 'checked' at a time
 - Every button in a radio button group MUST have the same name
 - If no button in a radio button group is 'pressed', the browser often 'presses' the first one

```
Age Category
<form action = "">
  <p>
    <input type = "radio" name = "age"
      value = "under20" checked = "checked"> 0-19
    <br/>
    <input type = "radio" name = "age"
      value = "20-35"> 20-35
    <br/>
    <input type = "radio" name = "age"
      value = "36-50"> 36-50
    <br/>
    <input type = "radio" name = "age"
      value = "over50"> Over 50
  </p>
</form>
```





5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Forms

- Menus - created with `<select>` tags
 - There are two kinds of menus, those that behave like checkboxes and those that behave like radio buttons (the default)
 - Menus that behave like checkboxes are specified by including the `multiple` attribute, which must be set to "multiple"
 - The `name` attribute of `<select>` is required
 - The `size` attribute of `<select>` can be included to specify the number of menu items to be displayed (the default is 1)
 - If `size` is set to `> 1` or if `multiple` is specified, the menu is displayed as a pop-up menu
 - Each item of a menu is specified with an `<option>` tag, whose pure text content (no tags) is the value of the item
 - An `<option>` tag can include the `selected` attribute, which when assigned "selected" specifies that the item is pre-selected



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Forms

Grocery Menu - milk, bread, eggs, cheese

```
<form action = "">
```

```
<p>
```

With size = 1 (the default)

```
<select name = "groceries">
```

```
<option> milk </option>
```

```
<option> bread </option>
```

```
<option> eggs </option>
```

```
<option> cheese </option>
```

```
</select>
```

```
</p>
```

```
</form>
```

A screenshot of a web browser window displaying the rendered HTML form. The text 'Grocery Menu - milk, bread, eggs, cheese' is shown at the top. Below it, the text 'With size = 1 (the default)' is followed by a dropdown menu that currently shows 'milk' and has a downward arrow on the right side.



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Forms

- After clicking the menu

A screenshot of a web browser window displaying a form titled "Grocery Menu - milk, bread, eggs, cheese". Below the title, there is a text input field containing "milk" and a dropdown menu. The dropdown menu is open, showing a list of items: "milk", "bread", "eggs", and "cheese". The "milk" option is currently selected and highlighted in blue. The text "With size = 1 (the default)" is visible to the left of the dropdown menu.



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Forms

- Text areas - created with `<textarea>`
 - Usually include the `rows` and `cols` attributes to specify the size of the text area
 - Default text can be included as the content of `<textarea>`
 - Scrolling is implicit if the area is overfilled

Please provide your employment aspirations

```
<form action = "">  
  <p>  
    <textarea name = "aspirations" rows = "3"  
      cols = "40">  
      (Be brief and concise)  
    </textarea>  
  </p>  
</form>
```

A screenshot of a web form. It contains a label "Please provide your employment aspirations" above a text area. The text area has a light blue border and contains the text "(Be brief and concise)". A vertical scrollbar is visible on the right side of the text area.



5.2 Designing web pages using HTML and CSS

5.2.1 Introduction to HTML - Forms

- Reset and Submit buttons
 - Both are created with `<input>`
`<input type = "reset" value = "Reset Form">`
`<input type = "submit" value = "Submit Form">`
 - Submit has two actions:
 - Encode the data of the form
 - Request that the server execute the server-resident program specified as the value of the action attribute of `<form>`
 - A Submit button is required in every form

--> SHOW a browser display of a form



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS

- Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in a markup language.
- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media
- CSS saves a lot of work. It can control the layout of multiple web pages all at once
- External stylesheets are stored in CSS files



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - HTML Problem

- HTML was never intended to contain tags for formatting a document.
- HTML was intended to define the content of a document, like: `<h1>This is a heading</h1>`
`<p>This is a paragraph.</p>`
- When tags like ``, and color attributes were added to the HTML 3.2 specification, it started a nightmare for web developers. Development of large web sites, where fonts and color information were added to every single page, became a long and expensive process.
- To solve this problem, the World Wide Web Consortium (W3C) created CSS. In HTML 4.0, all formatting could be removed from the HTML document, and stored in a separate CSS file.
- All browsers support CSS today.



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - Benefits

Benefits of CSS include:

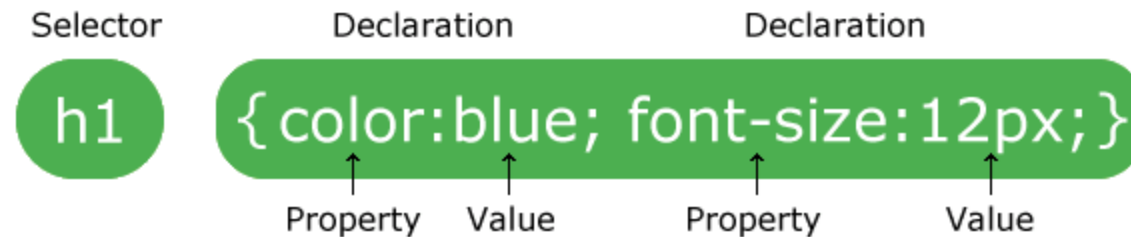
- control layout of many documents from one single style sheet;
- more precise control of layout;
- apply different layout to different media-types (screen, print, etc.);
- numerous advanced and sophisticated techniques.



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - Syntax

- A CSS rule has two main parts: a selector, and one or more declarations:



- h1 is a selector in CSS (it points to the HTML element you want to style: <h1>).
- color is a property, and blue is the property value
- font-size is a property, and 12px is the property value



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - Selector

- CSS selectors are used to "find" (or select) the HTML elements you want to style.
- We can divide CSS selectors into five categories:
 - Simple selectors (select elements based on name, id, class)
 - Combinator selectors (select elements based on a specific relationship between them)
 - Pseudo-class selectors (select elements based on a certain state)
 - Pseudo-elements selectors (select and style a part of an element)
 - Attribute selectors (select elements based on an attribute or attribute value)



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - Selector

The CSS element Selector

- The element selector selects HTML elements based on the element name.

Here, all <p> elements on the page will be center-aligned, with a red text color:

```
p {  
    text-align: center;  
    color: red;  
}
```



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - Selector

The CSS id Selector

- The id selector uses the id attribute of an HTML element to select a specific element.
- The id of an element is unique within a page, so the id selector is used to select one unique element!
- To select an element with a specific id, write a hash (#) character, followed by the id of the element.
- The CSS rule here will be applied to the HTML element with id="para1":

```
#para1 {  
    text-align: center;  
    color: red;  
}
```




5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - Selector

The CSS class Selector

- The class selector selects HTML elements with a specific class attribute.
- To select elements with a specific class, write a period (.) character, followed by the class name.
- In this example all HTML elements with class="center" will be red and center-aligned:

```
.center {  
    text-align: center;  
    color: red;  
}
```



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - Selector

The CSS class Selector

- You can also specify that only specific HTML elements should be affected by a class.
- In this example only <p> elements with class="center" will be center-aligned:

```
p.center {  
    text-align: center;  
    color: red;  
}
```



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - Selector

The CSS class Selector

- HTML elements can also refer to more than one class.
- HTML elements can also refer to more than one class.

```
<p class="center large">This paragraph refers to two classes.</p>
```



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - Selector

The CSS Grouping Selector

- The grouping selector selects all the HTML elements with the same style definitions.
- Look at the following CSS code (the h1, h2, and p elements have the same style definitions):

```
h1, h2, p {  
    text-align: center;  
    color: red;  
}
```



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - How to add CSS?

- When a browser reads a style sheet, it will format the HTML document according to the information in the style sheet.
- There are three ways of inserting a style sheet:
 - External CSS
 - Internal CSS
 - Inline CSS



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - How to add CSS?

External CSS

- With an external style sheet, you can change the look of an entire website by changing just one file!
- Each HTML page must include a reference to the external style sheet file inside the <link> element, inside the head section.
- External styles are defined within the <link> element, inside the <head> section of an HTML page:

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - How to add CSS?

External CSS

- An external style sheet can be written in any text editor, and must be saved with a .css extension.
- The external .css file should not contain any HTML tags.
- Here is how the "mystyle.css" file looks like:

"mystyle.css"

```
body {  
    background-color: lightblue;  
}  
  
h1 {  
    color: navy;  
    margin-left: 20px;  
}
```



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - How to add CSS?

Internal CSS

- An internal style sheet may be used if one single HTML page has a unique style.
- The internal style is defined inside the `<style>` element, inside the head section.
- Internal styles are defined within the `<style>` element, inside the `<head>` section of an HTML page:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
    background-color: linen;
}

h1 {
    color: maroon;
    margin-left: 40px;
}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```




5.2

Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - How to add CSS?

Inline CSS

- An inline style may be used to apply a unique style for a single element.
- To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.
- Inline styles are defined within the "style" attribute of the relevant element:

```
<!DOCTYPE html>
<html>
<body>

<h1 style="color:blue;text-align:center;">This is a heading</h1>
<p style="color:red;">This is a paragraph.</p>

</body>
</html>
```



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - How to add CSS?

Cascading Order

- What style will be used when there is more than one style specified for an HTML element?
- All the styles in a page will "cascade" into a new "virtual" style sheet by the following rules, where number one has the highest priority:
 - Inline style (inside an HTML element)
 - External and internal style sheets (in the head section)
 - Browser default
- So, an inline style has the highest priority, and will override external and internal styles and browser defaults.



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - CSS Colors

CSS Color Names

- In CSS, a color can be specified by using a color name.
- CSS/HTML support 140 standard color names.

https://www.w3schools.com/colors/colors_names.asp



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - CSS Colors

CSS Background Color

- You can set the background color for HTML elements.

```
<h1 style="background-color:DodgerBlue;">Hello World</h1>
```

```
<p style="background-color:Tomato;">Lorem ipsum...</p>
```

Hello World

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - CSS Colors

CSS Text Color

- You can set the color of text.

```
<h1 style="color:Tomato;">Hello World</h1>
```

```
<p style="color:DodgerBlue;">Lorem ipsum...</p>
```

```
<p style="color:MediumSeaGreen;">Ut wisi enim...</p>
```

Hello World

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.



5.2 Designing web pages using HTML and CSS

5.2.2 Introduction to CSS - CSS Colors

CSS Border Color

- You can set the color of borders.

```
<h1 style="border:2px solid Tomato;">Hello World</h1>
```

```
<h1 style="border:2px solid DodgerBlue;">Hello World</h1>
```

```
<h1 style="border:2px solid Violet;">Hello World</h1>
```

Hello World

Hello World

Hello World



5.2

Designing web pages using HTML and CSS

5.2.3 Web Server/ Web-Framework

Serve static web pages

1. Apache
2. IIS
3. Nginx

Serve dynamic web pages

1. Laravel (Apache + PHP)
2. IIS (ASP + DotNet)
3. Flask, Django (Werkzeug + Python)
4. GO
5. Tomcat + JAVA





5.2 Designing web pages using HTML and CSS

5.2.3 Web Server/ Web-Framework -Flask

What is a Framework ?

A framework "is a code library that makes a developer's life easier when building reliable, scalable, and maintainable web applications" by providing reusable code or extensions for common operations.

Flask is written in programming language python.

There are a number of frameworks for Python, including Flask, Tornado, Pyramid, and Django





5.2 Designing web pages using HTML and CSS

5.2.3 Web Server/ Web-Framework -Flask

What is a Flask framework ?

Flask is a web application framework written in Python. It is developed by Armin Ronacher, who leads an international group of Python enthusiasts named Pocco.

Flask is based on the Werkzeug WSGI toolkit and Jinja2 template engine. Both are Pocco projects.



Flask





5.2 Designing web pages using HTML and CSS

5.2.3 Web Server/ Web-Framework -Flask

WSGI

Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development.

WSGI is a specification for a universal interface between the web server and the web applications.



Flask





5.2 Designing web pages using HTML and CSS

5.2.3 Web Server/ Web-Framework -Flask

Werkzeug

It is a WSGI toolkit, which implements requests, response objects, and other utility functions. This enables building a web framework on top of it. The Flask framework uses Werkzeug as one of its bases.



Flask





5.2 Designing web pages using HTML and CSS

5.2.3 Web Server/ Web-Framework -Flask

Jinja2

Jinja2 is a popular templating engine for Python. A web templating system combines a template with a certain data source to render dynamic web pages.

Flask is often referred to as a micro framework. It aims to keep the core of an application simple yet extensible.

Flask does not have built-in abstraction layer for database handling, nor does it have form a validation support. Instead, Flask supports the extensions to add such functionality to the application. Some of the popular Flask extensions are discussed later in the tutorial.



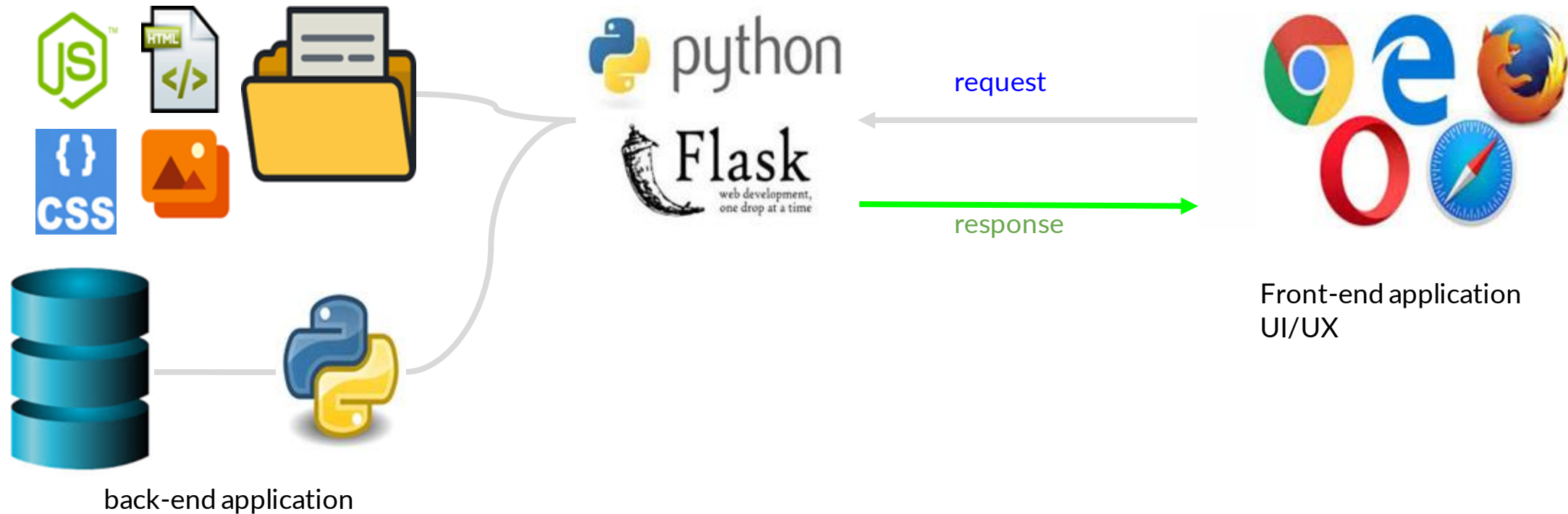
Flask



5.2

Designing web pages using HTML and CSS

5.2.3 Web Server/ Web-Framework -Flask





5.2

Designing web pages using HTML and CSS

5.2.3 Web Server/ Web-Framework - Build Flask App

1. Install Flask library to your development server.

```
$ pip install Flask
```

1. Write flask code
from flask import Flask
app = Flask(__name__)

```
@app.route("/")  
def hello():  
    return "Hello World!"
```

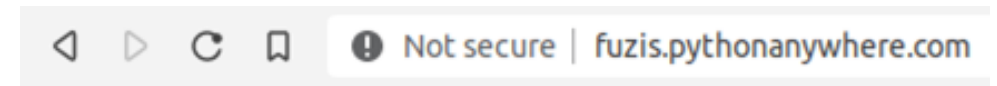
```
if __name__ == "__main__":  
    app.run()
```

3. Run the code from a terminal

```
$python app.py
```

4. Access to your web-application from a browser !

```
http://localhost:5000
```



Hello World !

5.2.3 Web Server/ Web-Framework- Response

It is a proper development to place your codes in a proper directory structure so that different kinds of files can be grouped together and easy to manage. The screen capture below shows a simple directory structure for flask app.

```
(base) fuzis@fsvivo:~/flaskapp$ tree
.
├── src
│   └── app
│       ├── static
│       └── templates
4 directories, 0 files
```

/flaskapp	The main directory of your flask project
/src	main directory of your application codes
/app	Your particular flask application.
/static	Location for static files or material such as pictures, JavaScripts, CSS files
/templates	Location for template html files



5.2

Designing web pages using HTML and CSS

5.2.3 Web Server/ Web-Framework- Response

1. Modify flask code from flask import Flask
app = Flask(__name__)

```
@app.route("/hello/<visitor>")  
def hello():  
    return  
    render_template("hello.html",  
    visitor=visitor)
```

```
if __name__ == "__main__":
```

```
    app.run(host='0.0.0.0',port=8000,debug=True)
```

2. Restart the server

```
$python app.py
```

3. Access to your web-application from a browser !

```
http://localhost:5000/hello/yourname
```



Hello Fuzi Shariff!



5.2 Designing web pages using HTML and CSS

5.2.3 Web Server/ Web-Framework - Response

Head:
css , JS
links

```
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width,initial-scale=1">
  <title>fuzisapp</title>
  <link href="{{ url_for('static', filename='css/bootstrap.min.css')}}" rel="stylesheet">
  <link href="{{ url_for('static', filename='css/main.css')}}" rel="stylesheet">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>

</head>
<body>
{% block content %}
{% endblock %}
</body>
<!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
<script src="{{ url_for('static', filename='js/bootstrap.min.js')}}"></script>
</html>
```

base.html

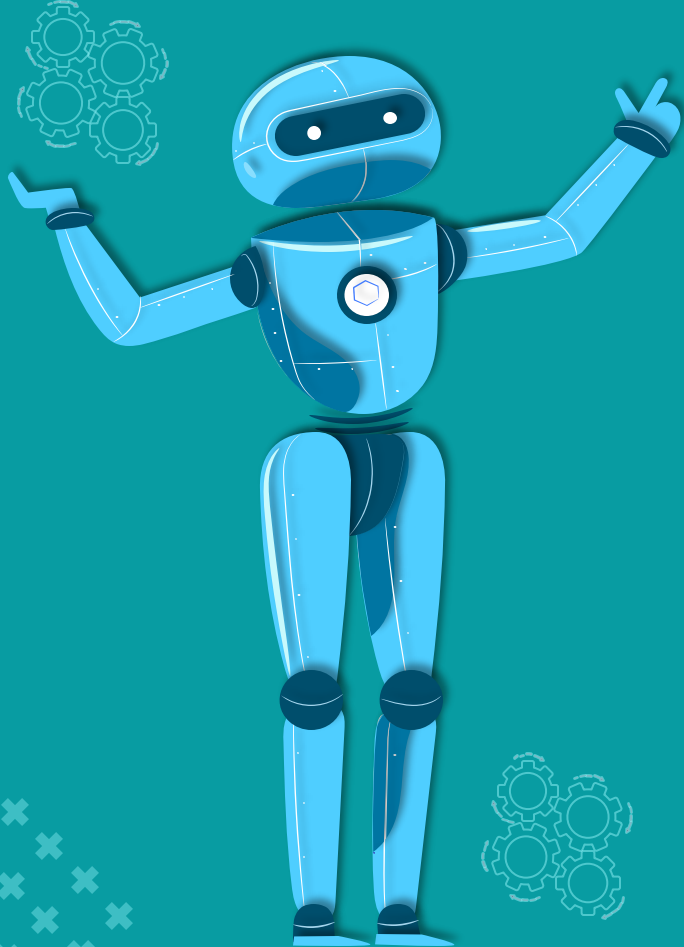
hello.html

```
{% extends "base.html" %}
{% block content %}
<div class="container">
<hr>
<h2>
  Hello {{visitor}}!
</h2>
{% endblock %}

</div>
```

Parameter

5.3



Interactivity of web pages using Javascript

- What is Javascript?
- Example of Javascript
- How does JavaScript works?
- Javascript Syntax



5.3 Interactivity of web pages using Javascript

5.3.1 What is javascript?

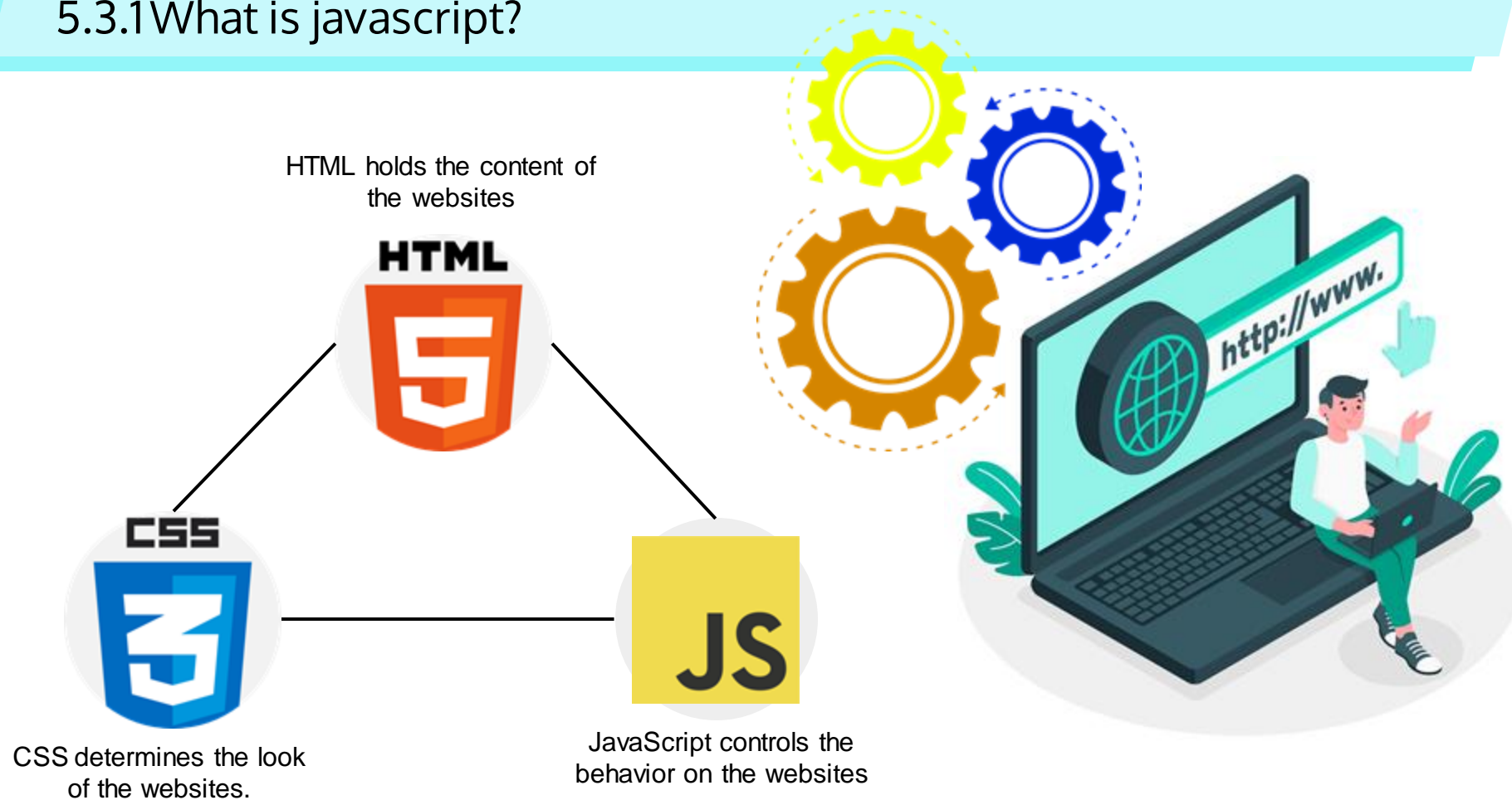


<https://github.com/voodootikigod/logo.js>

- JavaScript is a scripting language that enables web developers/designers to build more functional and interactive websites.
- JavaScript is not the same as Java.
 - Java is a compiled language
 - JavaScript is a scripting language

5.3 Interactivity of web pages using Javascript

5.3.1 What is javascript?





5.3 Interactivity of web pages using Javascript

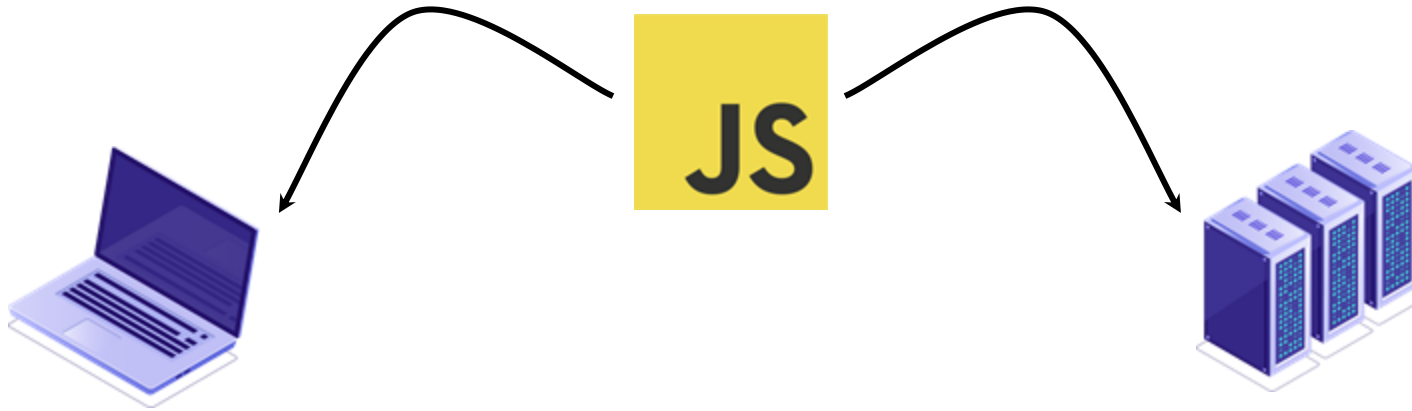
5.3.2 Example of Javascript?

Show me the Content!

<https://developers.elementor.com/building-a-simple-custom-widget-with-javascript/>

5.3 Interactivity of web pages using Javascript

5.3.3 How does Javascript works?



Client-side Javascript runs on the client browser. The script is included or embedded within the HTML documents and the script will be interpreted by the browser.

Server-side javascript runs on the server side. The server side script enables back-end access to the database, files systems and server

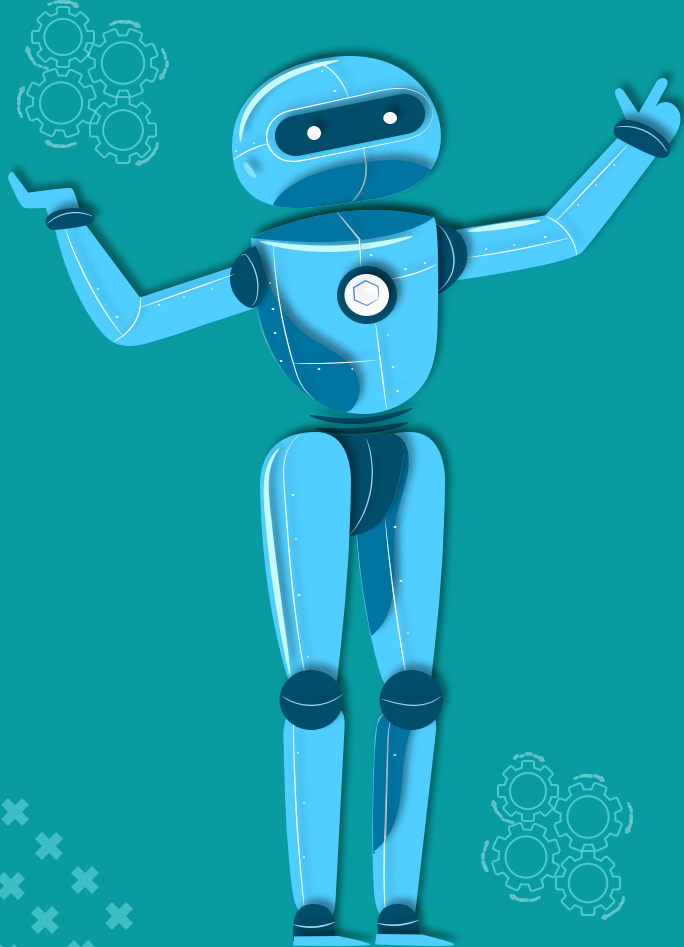


5.3 Interactivity of web pages using Javascript

5.3.4 Javascript syntax

```
index.html
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Sample Javascript</title>
5 </head>
6 <body>
7   <h3>Introduction to Javascript</h3>
8   <p id="azleen">These sentences will be hide once
9     you've clicked on the button HIDE. And will be unhide
10    you've clicked on the button UNHIDE</p>
11   <button type="button" onclick="document.getElementById
12     ('azleen').style.display='none'">HIDE</button>
13   <button type="button" onclick="document.getElementById
14     ('azleen').style.display='block'">UNHIDE</button>
15 </body>
16 </html>
```





Data Storage for IoT Data

- Data VS Information
- What is a Database
- How to manage a Database?
- Database Architecture
- Advantage of using DBMS



5.4 Data Storage for IoT Data

5.4.1 Data vs. Information

- Data are raw facts.
- Information is the process of revealing meaning from the given data.
- Information requires context to reveal meaning
- Raw data must be formatted for storage, processing, and presentation.

5.4 Data Storage for IoT Data

5.4.1 Data vs. Information

- Accurate, relevant, timely information is the key to good decision making.
- Picture on the right shows the National Speed limit in Malaysia.
- Referring back to the meaning of data and information, can you differentiate whether this signage is showing us information or just data?





5.4 Data Storage for IoT Data

5.4.2 What is a Database

- Database is a shared, integrated computer structure that stores a collection of data.
- Database can be located in our organization or distributed over the cloud.



Database



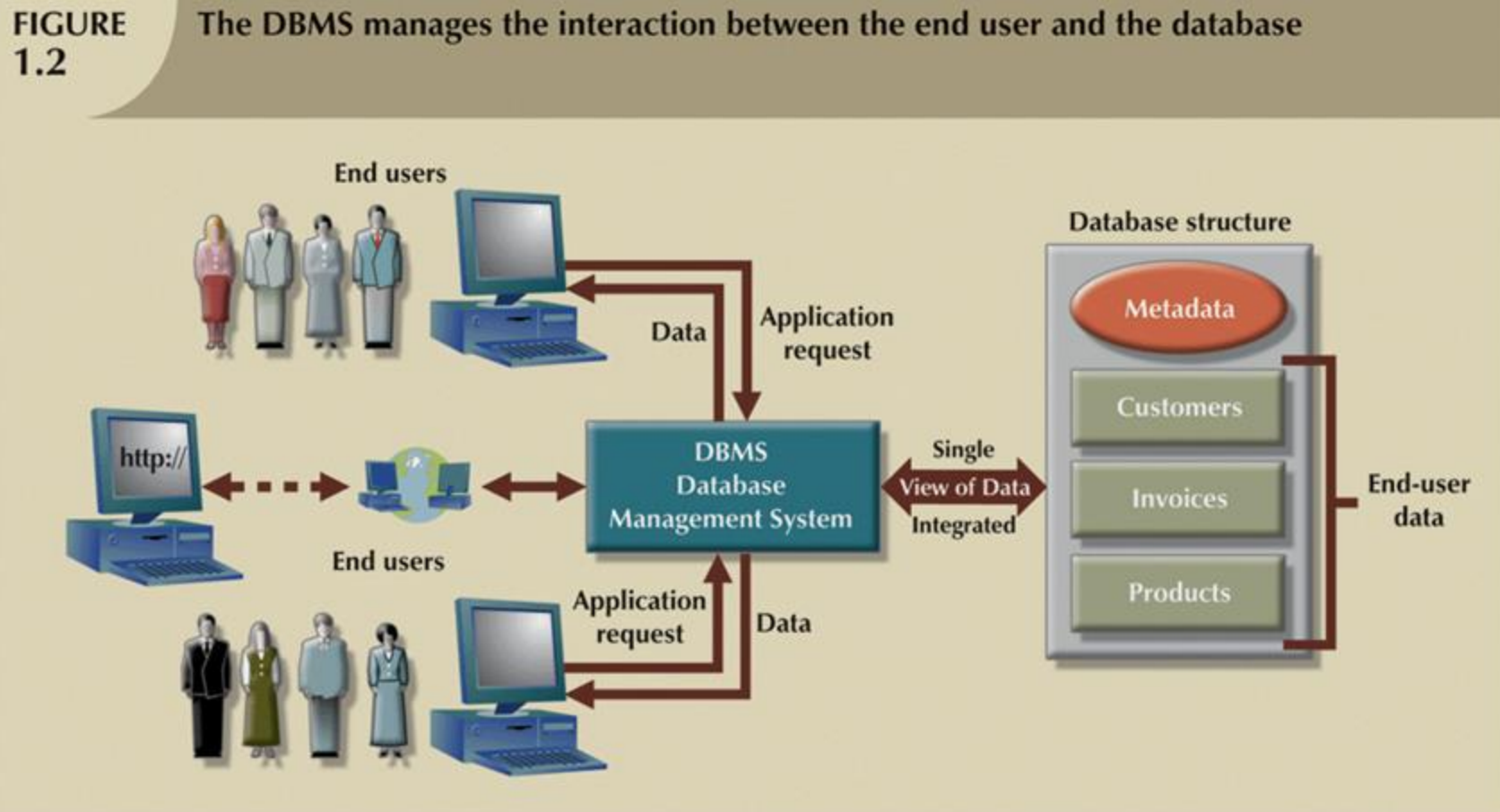
5.4 Data Storage for IoT Data

5.4.3 How to manage a Database?

- Databases are being managed using a software called DBMS.
- Database management system (DBMS) is a collection of programs that:
 - Manage structure and control access to data
 - Act as an intermediary between the user and the database
 - Provide structure to stored as file collection
 - Give access database
 - Enables data to be shared
 - Integrates many users' views of the data

5.4 Data Storage for IoT Data

5.4.4 Database Architecture



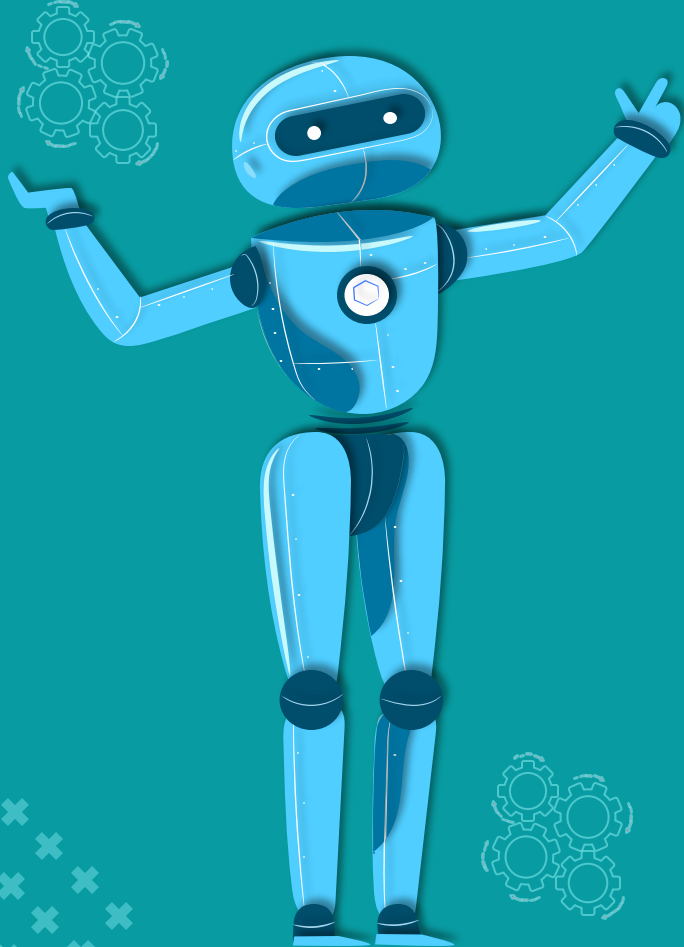


5.4 Data Storage for IoT Data

5.4.5 Advantage of using DBMS

- Improved data sharing
- Improved data security
- Better data integration
- Minimized data inconsistency
- Improved data access
- Improved decision making
- Increased end-user productivity

5.5



Visualization of IoT data

- Dashboard Restaurant Table Management Application



5.5

IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Use case:

You are given a requirement for a restaurant to develop an ordering system using an electronic menu for its customer. At the dining table, customers will find an electronic menu controlled by a microcontroller with a menu linked to its touchpads.

Your job is to develop an embedded application such that, whenever a customer touches the selected touchpad designated with a menu set, the microcontroller automatically places the order onto the restaurant dashboard. The customer can make repeated orders by touching different touchpads designated with other sets of menus, or repeatedly touches the same touchpad to make multiple orders.

The microcontroller relies on WiFi connection available in the restaurants to connect to the dashboard server that manages the order. Cooks in the kitchen will use the information to cook the meals based on the orders displayed on the dashboard.





5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

16:41:07

Restaurant Table Management

7-3-2020

Tables Arrangement

Table A	Table B	Table C
Food Drinks	Food Drinks	Food Drinks
Table D	Table E	Information
Food Drinks	Food Drinks	

copy-right by fuzishariff@gmail.com

5.5

IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

- The Dashboard application need to display table arrangement as in rows and columns
- There are two rows and three columns
- A table is represented as one of the cell in the table
- Each cell has information regarding:
 - Table ID
 - List of food ordered
 - List of drinks ordered
 - One cell contain information to be shared with restaurant servers.

16:41:07

Restaurant Table Management

7-3-2020

Tables Arrangement

Table A		Table B		Table C	
Food	Drinks	Food	Drinks	Food	Drinks

Table D		Table E		Information	
Food	Drinks	Food	Drinks		

copy-right by fuzisharif@gmail.com

5.5

IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

- Embedded IoT application will communicate with the web application to send order information from customer over the WiFi.
- The information consists the required data about table ID, time of order, food list and drinks list
- Table order status also displayed but it not handled by IoT application but, restaurant staff will interact with the web application interface.

14:00:11

Restaurant Table Management

8-3-2020

The order data sent by IoT device to the restaurant dashbord server will atumatically displayed on the dashboard for cooks in the kitchen

Table A

Serving

Sun Mar 8 07:04:47 2020

FOOD

• beef burger and cheese

DRINKS

• iced pepsi

Table B

Serving

Sun Mar 8 07:06:49 2020

FOOD

• lamb chop in barbeque sauce

DRINKS

• hot coffee with cream

Table C

Booked by WSkill

Sun Mar 8 07:10:25 2020

FOOD

DRINKS

Table D

Complete

Sun Mar 8 07:08:39 2020

FOOD

• lamb chop in black-pepper sauce
 • beef burger and cheese

DRINKS

• hot chocolate with cream
 • iced pepsi

Table E

BOOKED by WSkills

Sun Mar 8 05:52:10 2020

FOOD

DRINKS

Information



5.5

IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Web Application Server code:

```
from flask import Flask, jsonify, render_template, request
from data import data
app = Flask(__name__)
```

```
@app.route("/")
def hello():
    return "Hello Python World!"
```

```
@app.route("/tables")
def tables():
    return jsonify(data)
```

```
@app.route("/table/<tid>", methods=["POST"])
def table(tid):
    if tid in data.keys():
        order = request.json
        data[tid]['food'].extend(order['food'])
        data[tid]['drinks'].extend(order['drinks'])
        return jsonify(data[tid])
```

```
@app.route("/status/<tid>", methods=["PATCH"])
def tablestatus(tid):
    if tid in data.keys():
        tblstatus = request.json
        data[tid].update(tblstatus)
        return jsonify(data[tid])
```

```
@app.route("/mainpage")
def mainpage():
    return render_template("tables.html")
```

```
@app.route("/information", methods=["GET", "POST"])
def info():
    if request.method == "POST":
        if "information" in request.json.keys():
            data["information"] = request.json['information']
            return f"recieved info: {request.json['information']}"
    else:
        return jsonify(data["information"])
if __name__ == "__main__":
    app.run(host='0.0.0.0', debug=True)
```



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Web Application Server code:

```
from flask import Flask, jsonify, render_template, request
from data import data
app = Flask(__name__)
```

The above code is python directive to import several libraries that we are going to need to build the web application.

1. flask
 - a. flask refers to python module that contains other libraries needed in Flask application.
1. Flask
 - a. Python micro web framework itself.
 - b. It will be responsible to manage all other python libraries that supports the web applicaiton such as REST protocol and HTML template engine.



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Web Application Server code:

```
from flask import Flask, jsonify, render_template, request
from data import data
app = Flask(__name__)
```

The above code is python directive to import several libraries that we are going to need to build the web application.

3. jsonify
 - a. Python library to ensure our serve will render JSON response properly to the client.
 - b. JSON format is required in order for server to communicate with client using REST protocol.
3. render_template
 - a. render_template is the library responsible to process any user specified HTML file to be sent together with the response.
 - b. The library will utilise Jinja2 library ship within flask to handle the dynamic rendering of html files on the fly.
 - c. Jinja2 capable of processing variables from python to be embedded into an html file and format the variable values with html tags preferred, such as `<h1>{{myvalue}}</h1>`



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Web Application Server code:

```
from flask import Flask, jsonify, render_template, request
from data import data
app = Flask(__name__)
```

The above code is python directive to import several libraries that we are going to need to build the web application.

5. request

- request is a python library shipped together with flask to handle http request object sent by user http client.
- we can use request object extract query values from GET method use by client
- we can extract key-value pair data that user sent using html form
- We can obtain JSON data from request if the user client uses REST method to send data.



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Notes on flask request:

- The data from a client's web page is sent to the server as a global request object. In order to process the request data, it should be imported from the Flask module.
- Important attributes of request object are listed below –
- Form – It is a dictionary object containing key and value pairs of form parameters and their values.
- args – parsed contents of query string which is part of URL after question mark (?).
- Cookies – dictionary object holding Cookie names and values.
- files – data pertaining to uploaded file.
- method – current request method.



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Web Application Server code:

```
from data import data  
app = Flask(__name__)
```

6. from data import data
 - a. This is a directive to include external code from python file name "data.py" and use the object data in it.
 - b. Object data here refer to a dictionary object that hold the restaurant table data.
 - c. Server will load the dictionary object into memory and use it as database.
 - d. Note that, dictionary object has same structure as JSON data.
6. app = Flask(__name__)
 - a. This python statement instruct python to create an object "app" by instantiating "Flask" class that is imported in the earlier directive.
 - b. "app" object is now inherit all functionalities of Flask web application.
 - c. __name__ is passed as an argument to set app as the "main" module running after being instantiated. It is an attribute of a program, in this case set to Flask.



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Web Application Server code:

```
@app.route("/")  
def hello():  
    return "Hello Python World!"
```

8. This block of code is a function in python. It consists of two portions instructions, decorator and the actual functions:
 - a. `@app.route("/")`, is called a decorator function in python. It begins with "@" and followed by the name of the decorator function.
 - b. The function invoked by the decorator is `app.route()`. "route" is a built in Flask function to register an endpoint, in this case "/" passed as an argument, i.e. to be the default endpoint of the flask application. This endpoint is associated with another function python suppose to run, and it is the function right below the decorator statement name "hello()"
 - c. When ever flask application receives a request from a client arriving on "/" endpoint, flask app will execute the "hello()" function.
 - d. In this code, "hello()" function just returns "Hello Python World ! "
 - e. User will see the response "Hello Python World ! " in his browser.



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Web Application Server code:

```
@app.route("/tables")
def tables():
    return jsonify(data)
```

9. This block of code is another decorated function that python runs when it receives a request from client on its endpoint `"/tables"`. So this is the way how we can add functionalities or API endpoints to our web application.
 - a. `"tables()"` function simply returns the object named `"data"` that is imported earlier to the client.
 - b. `"data"` object is of type dictionary, similar to JSON structure, but browser will not understand how to render it since it is python object.
 - c. `jsonify(data)` function is called to transform `"data"` type into proper JSON data type understood by the browser, and the data consists on all information pertaining to restaurant tables
 - d. `jsonify()` also ensures that the flask app will response to the client as REST application (`application/json`).



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

10. This block of code is for endpoint to update customer order information to a table record.
- app.route registers the endpoint `"/table/<id>"` differently from other simple endpoint such as `"/hello"` because it will accept a dynamic query parameters in addition to `"/table/"`, another one will be any String value to be concatenated with the `"/table/"`.
 - app.route also ensures that the REST endpoint only accept request with method of `"POST"`, since we provide another argument to it which is `methods=["POST"]`
 - The function assign to this endpoint is named `table(tid)`, and it must receive an argument `tid` which is set to equal to `<tid>` query parameter in app.route decorator function.
 - "Tid" suppose to be tableID value and IF statement is used to check if the ID provided is available in server `"data"` object ----> if `tid` in `data.keys()`:
 - `order = request.json` ---> this code will extract JSON data arrives with POST request, and it is supposed to contain order information.
 - Each data in JSON object related to food and drinks is then updated to table record using dictionary data operation.

Web Application Server code:

```
@app.route("/table/<tid>",  
methods=["POST"])  
def table(tid):  
    if tid in data.keys():  
        order = request.json  
  
        data[tid]['food'].extend(order['food'])  
  
        data[tid]['drinks'].extend(order['drinks'])  
        return jsonify(data[tid])
```



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Web Application Server code:

```
@app.route("/status/<tid>", methods=["PATCH"])
def tablestatus(tid):
    if tid in data.keys():
        tblstatus = request.json
        data[tid].update(tblstatus)
        return jsonify(data[tid])
```

11. This block of code is for endpoint to update order information about table status.
 - a. Argument `"/status/<tid>"` for `app.route()` decorator registers an endpoint `"status/<tid>"`
 - b. `methods=["PATCH"]` ---> The decorator only allow REST PATCH method.
 - c. `tblstatus = request.json` ---> this statement will extract json data about table status and pass the value to variable `tblstatus`.
 - d. `data[tid].update(tblstatus)` ---> this statement will update the table dictionary data pertaining to data related to dictionary key = `"status"`. Values for keys `food` and `drink` are left unchanged.



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

12. This block of code is similar to earlier endpoint code but the function being decorated is slightly different.
- a. It is because we call flask built-in function call called `render_template()`.
 - b. If you notice this function was imported in the first line of code.
 - c. `render_template()` function needs arguments
 - d. Name of the html to be rendered
 - e. Parameter that will be used together with the html template file during rendering, but it is not used here.
 - f. The template file used here is "tables.html"
 - g. "Table.html" is an html file that provide layout for restaurant tables and also contains CSS/Bootstrap library and JavaScript functions to provide better UI/UX experience to users such as grid layout, card and AJAX REST queries.

Web Application Server code:

```
@app.route("/mainpage")
def mainpage():
    return render_template("tables.html")
```





5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

12. This block of code is similar to earlier endpoint however it REST endpoint is set accept GET and POST methods.
- a. Since REST methods or verbs allows GET and POST, an IF statement must be used to check what method is used in each request, and this is handled by the function "info()"
 - b. If POST method is used, flask expects a JSON data with key contains "information". If it does, the JSON data is processed and python object named "data" with "information" key is updated.
 - c. If GET method is used, then flask will return the existing information available in "data" object belong to record with key = "information". The response information is then updated in the page template that triggers the request.

Web Application Server code:

```
@app.route("/information", methods=["GET", "POST"])
def info():
    if request.method == "POST":
        if "information" in request.json.keys():
            data["information"] = request.json['information']
            return f"received info: {request.json['information']}"
        else:
            return jsonify(data["information"])
```



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

```
if __name__ == "__main__":  
    app.run(host='0.0.0.0',debug=True)
```

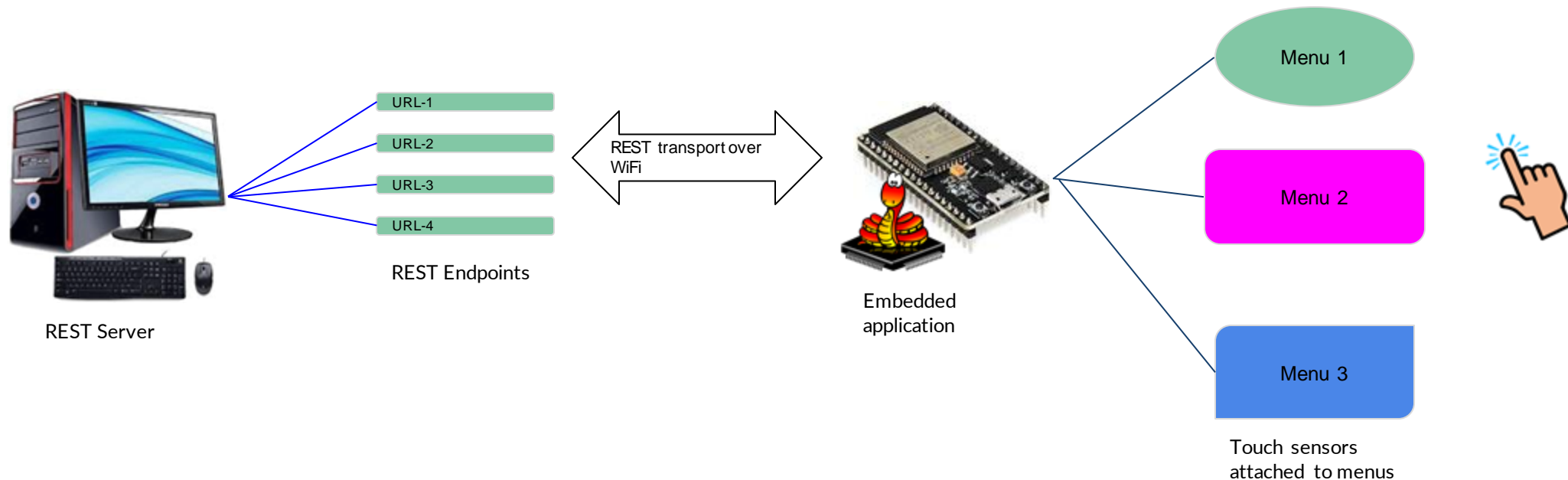
12. This final block of code is the one most important code to launch flask application
- `if __name__ == "__main__":` ---> This statement is doing the checking that the flask code that we write is executed by python directly or not. And if it is, then the code should have it's identifier as `"__main__"`.
 - Indeed, if we check the third line of code, it's statement is `"app = Flask(__name__)"`.
 - If the checking of code unique identifier is `"__main__"`, then `app.run(host='0.0.0.0',debug=True)` is to be executed by python.
 - `app.run()` contains an indefinite loop (i.e. it runs forever until we quit it), this is how a server should be running.
 - `app.run()` requires arguments of type "keyword-arguments":
 - `host = '0.0.0.0'` means, make the flask app accessible from any network interfaces available on the host. Example, 127.0.0.1 is localhost, then on the NIC's can be any assigned IP addresses such as 10.1.1.1. Or 192.168.1.10 etc.
 - `debug = True` means, it is run under development mode, which prints out error message in detail to the client browser. It must be set to `"False"` when run in production mode to avoid security risks.
 - `port=8000`, if you want to expose the flask app on port 8000, by default it is run on port 5000 and we do not need to add to the code for this choice.
 - To run the app , we simply invoke python and pass the app.py code to be executed from command console
`#>python3 app.py`
 - That is all regarding python code and flask library and REST implementation, the REST codes are completed.

5.5

IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Communicating embedded application with our REST web application server.



- The illustration is showing a schematic diagram of how a restaurant table management application will be built using embedded system and a web application supporting REST protocols.
- Python programming language will be used at both ends, i.e. server and client sides.



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Use case:

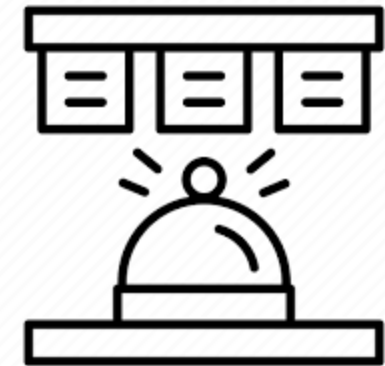
Let's use the microcontroller touch sensor capability to capture orders from customers at a dining table.

Consider each table will be equipped with a microcontroller that will be linked several of its touch sensors to touchpads that represents choices of menu, that consists of variety sets of food and drink.

The customer will touch the selected touchpad, and microcontroller will evaluate the choice of the order set. Upon successful order, an LED will be lit up to acknowledge the order has been sent to the web application server.

Developer must follow the API endpoints given by web application developer in order to send order data in correct JSON format.

The choice of communication protocol will be HTTP-REST or MQTT

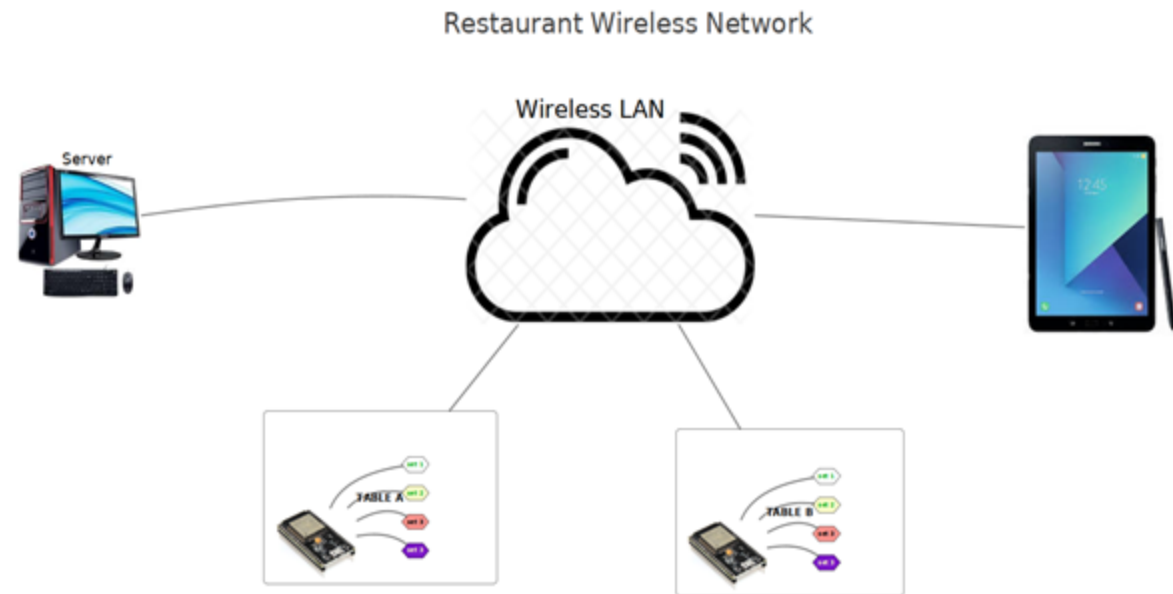


5.5

IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

1. Program your IoT device to connect to WiFi network
2. Import network and data communication library into your code, use MQTT or REST
3. Configure network client and mqtt client.
4. Program the touch sensors to select order data base on GPIO id.
5. Send the data base on touch sensor being touch to server REST endpoint or MQTT topic.





5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Write an embedded program to accomplish this task using REST protocol provide by "urequests.py" micropython REST library.

Assuming the server ip address is 192.168.1.23

Developers inform that an order data can be POST(ed) to the endpoint `"/table/<tableid>"`

The data structure must be a in JSON format... like

```
{  
  "food":["lamb chop in barbeque sauce"],  
  "drinks":["hot coffee with cream"]  
}
```

16:41:07	Restaurant Table Management	7-3-2020
Tables Arrangment		
Table A	Table B	Table C
Food Drinks	Food Drinks	Food Drinks
Table D	Table E	Information
Food Drinks	Food Drinks	

copy-right by fuzishan1@gmail.com

5.5

IoT Web Application and Code Walkthrough

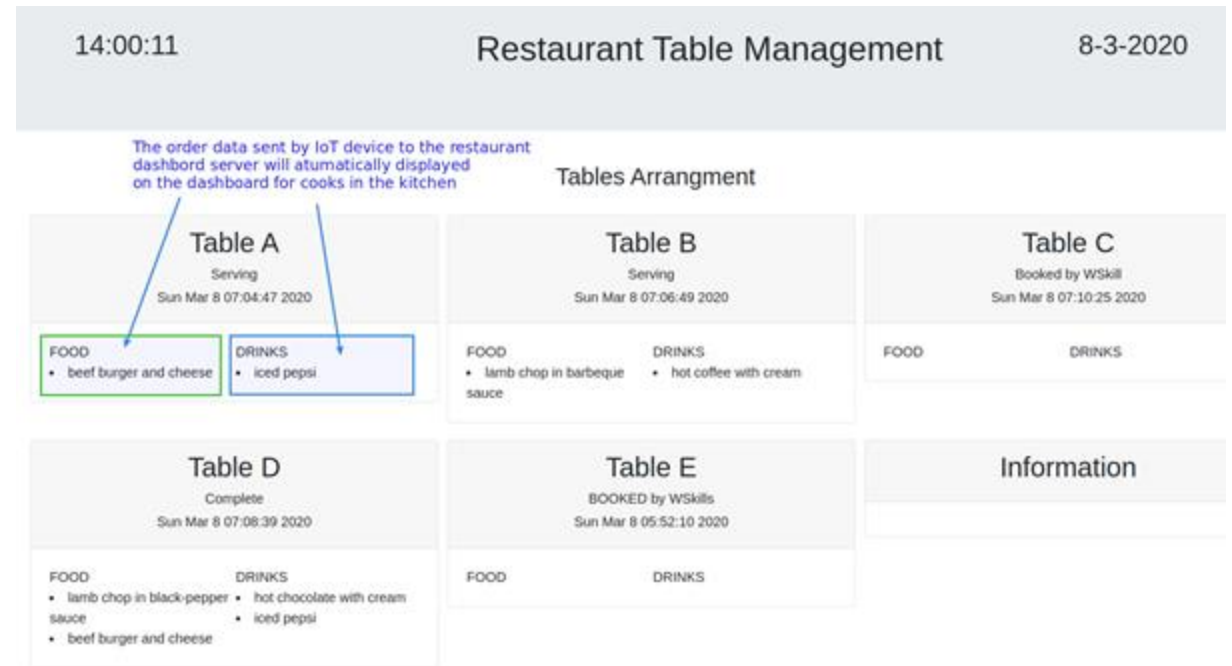
5.5.1 Dashboard Restaurant Table Management Application

This is the expected updated restaurant web application will look like when users place orders using the embedded ordering menu using touch sensors linked to the microcontroller.

The dashboard developer has given the endpoints details and methods to use in order for the IoT to be able to send data successfully to the back-end system.

It used REST technology to accomplish this data communication process.

Refer to earlier web application walk through

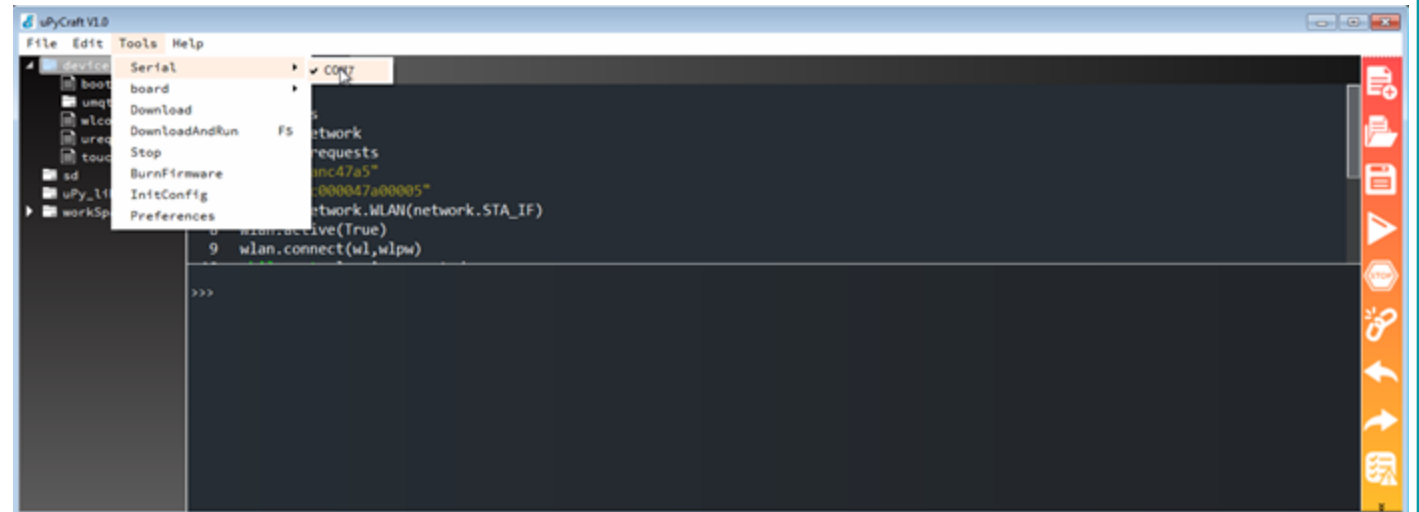


5.5

IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

1. Connect your microcontroller to the USB port and allow it to boot properly.
2. Make sure the uPyCraft IDE is connecting to the correct com port...In this case COM7
3. Click on the device, list of older programs may exist...
4. Create a new python file and give it a name wlconnect.py.
5. Ensure that you have the necessary libraries copied to the device directory as well...
6. In this case we need "urequests.py".





5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

1. Notice in the debug window at the bottom of IDE...
2. The default response b'Hello Python World' received from your flask REST server and printed.

Congratulations!

You have successfully programmed an embedded application using an IoT device and communicate via network to a REST application server...

Please note, this communication test is only accessing "/" default endpoint, an it is just a GET request, so there will be no update on the web application dashboard.

The screenshot shows an IDE with a file explorer on the left containing files like boot.py, umqtt, wlconnect.py, urequests, touchapp.py, sd, uPyLib, and workspace. The main editor displays a Python script named wlconnect.py with the following code:

```
1 import network
2 import urequests
3
4 wl = "wlan0"
5 wlpw = "c000047a000005"
6 wlan = network.WLAN(network.STA_IF)
7 wlan.active(True)
8 wlan.connect(wl,wlpw)
9 while not wlan.isconnected():
10     pass
11 print(wlan.ifconfig())
12
13 websvr = "http://192.168.1.23:8000"
14
15 r = urequests.get(websvr)
16 print(r.content)
```

Below the code, the execution output is shown:

```
>>>
>>>
>>>
Ready to download this file,please wait!
...
download ok
exec(open('./wlconnect.py').read(),globals())
['192.168.1.20', '255.255.255.0', '192.168.1.1', '1.1.1.1']
b'Hello Python World!'
>>>
```

A yellow arrow points to the output line `b'Hello Python World!'` with the text "Notice the serve response".



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Time to test the flask web application...

Now let us test the REST endpoint related to restaurant application....

Let's try to access the data from the server using HTTP / GET requests from one of the end-points.

Use this endpoint:

"http://<servername_ip>:8000/tables"

```
>>> uri = websvr + "tables"
>>> uri
'http://192.168.1.23:8000tables'
>>> uri = websvr + "/tables"
>>> uri
'http://192.168.1.23:8000/tables'
>>> rsps = urequests.get(uri)
>>> print(rsps.content)
b'{\n  "A": {\n    "drinks": [\n      "iced pepsi"\n    ],\n    "food": [\n      "beef burger and cheese"\n    ],\n    "status": "Serving",\n    "table": "A",\n    "time": "Sun Mar 8 07:04:47 2020"\n  },\n  "B": {\n    "drinks": [\n      "hot coffee with cream"\n    ],\n    "food": [\n      "lamb chop in barbeque sauce"\n    ],\n    "status": "Serving",\n    "table": "B",\n    "time": "Sun Mar 8 07:06:49 2020"\n  },\n  "C": {\n    "Status": "idle",\n    "drinks": [\n      "hot chocolate with cream"\n    ],\n    "food": [\n      "lamb chop in black-pepper sauce"\n    ],\n    "status": "BOOKED by WSkills",\n    "table": "C",\n    "time": "Sun Mar 8 14:17:37 2020"\n  },\n  "D": {\n    "drinks": [\n      "hot chocolate with cream",\n      "iced pepsi"\n    ],\n    "food": [\n      "lamb chop in black-pepper sauce",\n      "beef burger and cheese"\n    ],\n    "status": "Complete",\n    "table": "D",\n    "time": "Sun Mar 8 07:08:39 2020"\n  },\n  "E": {\n    "drinks": [\n      "iced pepsi"\n    ],\n    "food": [\n      "beef burger and cheese"\n    ],\n    "status": "BOOKED by WSkills",\n    "table": "E",\n    "time": "Sun Mar 8 05:52:10 2020"\n  },\n  "information": ""\n}\n'
```




5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

From the example above, we use micropython to build the endpoint based on its existing variable values in memory...

```
uri = websvr + "tables"
```

Here uri will be our endpoint variable, we concatenate the value server address in variable websvr and test the value:

```
'http://192.168.1.23:8000tables'
```

This URI value seems to be correct, but there is a slight mistake.... It is missing one "/" before the parameter "tables", It is easily fixed in the code follows:

```
uri = websvr + "/tables"
```

Use the code below to re-execute REST request to the server:

```
rsps = urequests.get(uri)
```



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

After execution, REPL prompted with no error, this means that the REST request has been successful.

The "rsps" variable is the variable corresponding to the request object, now it holds the data being sent by the server response.

```
print(rsps.content)
```

This command prints the content or data available in rsps object as shown below...

```
b'{"A":{"drinks":["iced pepsi"], "food":["beef burger and cheese"], "status": "Serving", "table": "A", "time": "Sun Mar 8 07:04:47 2020"}, "B":{"drinks":["hot coffee with cream"], "food":["lamb chop in barbeque sauce"], "status": "Serving", "table": "B", "time": "Sun Mar 8 07:06:49 2020"}, "C":{"Status": "idle", "drinks":["hot chocolate with cream"], ...
>>>
```

The data is a large string object that can be processed as a JSON object.



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Testing POST request to send data to REST Server

So far our embedded client has been able to connect to web application server and request some JSON data.

How about sending some data to be updated on the server side and get it displayed on the server dashboard.

If you remember from the previous web app development, we need to talk to REST using POST verb or command and combine it with JSON or FORM data.

Since we are using embedded application, there will be no user interface to display the form to user. So we are left with JSON data option only to use.

Now let's try to post some data according to REST API endpoint and JSON data structure format.

Let say, the customer wants to order as set for:
Food: "Fish and Chips with Salads"
Drink: "hot coffee latte"

The the JSON data format looks like as follows:

```
{  
  "food":["fish and chips with salads"],  
  "drinks":["hot coffee latte"]  
}
```



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Testing POST request to send data to REST Server

Now we need to create a python object to hold this data (JSON structure).

```
menu = {"food":["fish and chips with salads"],  
        "drinks":["hot coffee latte"]}
```

To do this we can use python built in function "dict", the function that builds a dictionary object which has a similar structure as JSON data.

```
>>> menu = dict(food = ["fish and chips with salads"], drinks = ["hot coffee latte"])  
>>>  
>>> menu  
{ 'drinks': ['hot coffee latte'], 'food': ['fish and chips with salads'] }  
>>> |
```

Please take note the difference between JSON data format and python dictionary data format.

Also note that it looks like "food" and "drinks" switch positions, but luckily it doesn't matter to python.



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Testing POST request to send data to REST Server

Now the data is ready to be posted to the server. What we need now is the correct server address and end-point that will be able to process the data..

Checking the REST endpoints of the web application we can see that the end point we are interested in is:
`http://<servername_ip>:8000/table/<table_ID>`

To use this in python console, we need to build the endpoint using String data type and store it in a variable..

So let's build the URI for this end-point, and assign it to a variable.

We choose table A, since the table is available...

Table A	
idle	
FOOD	DRINKS



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Testing POST request to send data to REST Server

```
>>> menu = dict(food = ["fish and chips with salads"], drinks = ["hot coffee latte"])
>>>
>>> menu
{'drinks': ['hot coffee latte'], 'food': ['fish and chips with salads']}
>>> uri
'http://192.168.1.23:8000/tables'
>>> websvr
'http://192.168.1.23:8000'
>>> uri = websvr + "/table" + "/A"
>>> uri
'http://192.168.1.23:8000/table/A'
>>> |
```

With the python code shown above , now the "uri" object holds the correct end-point value that we need.

'http://192.168.1.23:8000/table/A'



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Building POST request in Micro Python to send data to REST Server

Next we will attempt to post the menu data to the endpoint using micro python "urequests" REST client library.

Another piece of information is required to inform the web application server that we want the data to be processed as a JSON application. Or else, it assumes XML/HTML by default, which is designed for human visualization.

The information is needed in the header parameter of urequests object like shown below:

```
headers = {'content-type': 'application/json'}
```

Finally, the complete instruction to use for micropython will be as follows:

```
>>> rps = urequests.post(uri, json=menu, headers = headers)
```

"rps" is a variable to hold the response object from the REST server.



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Building POST request in Micro Python to send data to REST Server

```
>>> rps = urequests.post(uri, json=menu, headers = headers)
```

Lets execute the command.... You should receive confirmation from the web application server the response like shown in the screenshot below.

```
uri = websvr + "/table" + "/" + A
>>> uri
'http://192.168.1.23:8080/table/A'
>>> menu
{'drinks': ['hot coffee latte'], 'food': ['fish and chips with salads']}
>>> headers
{'content-type': 'application/json'}
>>> rps = urequests.post(uri, json=menu, headers=headers)
>>> rps.content
b'{"drinks": ["hot coffee latte"], "food": ["fish and chips with salads"], "status": "idle", "table": "A", "time": ""}'
>>> |
```

Table A	
idle	
FOOD	DRINKS
• fish and chips with salads	• hot coffee latte

Shown above is the response from successful POST requests to end-point for ordering a dinner set of the restaurant web application. We can now check how it appears on the dashboard.

Note that list of food and drinks should be updated for Table A



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Using POST request of different endpoint to update the Table status.

We also need to send another piece of JSON data to set the table status as being served...

The endpoint should be used is:

`http://<servername_ip>:8000/status/<table_ID>`

Let's set the status as value as "Serving now..."

Our JSON data should be:

Data:

```
{  
  "status": "Serving now..."  
}
```

```
uri = websvr + "/status" + "/" + A  
>>> uri  
'http://192.168.1.23:8000/status/A'  
>>> tblsts = dict(status = "Serving now..."  
... )  
>>> tblsts  
{'status': 'Serving now...'}  
>>> rps = urequests.post(uri, json = tblsts , headers=headers)  
>>> rps.content  
b'Updated: table A, Serving now...'  
>>> |
```

Note that we use dict() function to create the JSON data structure for table status `tblsts = dict(status = "Serving now...")`



5.5 IoT Web Application and Code Walkthrough

5.5.1 Dashboard Restaurant Table Management Application

Using POST request of different endpoint to update the Table status.

As you can see, the information about the menu being ordered and the status of the table being served are displayed correctly for human visualization.

Congratulations! Your IoT is capable of using REST data communication protocol and is now ready to work as required by the web application API.

What we need to do now is to write a complete micropython program and load it to the IoT so that it can response to user's input and carry out the menu order automatically.

Table A	
Serving now...	
Sun Mar 8 22:18:32 2020	
FOOD	DRINKS
• fish and chips with salads	• hot coffee latte