# ­­­­­­What I reviewed about HTML, also what I coded

HTML (HyperText Markup Language) is a markup language, not a programming language.

<!DOCTYPE html> mentions that a document is an HTML5 document. It is different from the <html></html> tag that is our root element in an HTML page.

HTML element: is defined between two tags, like headers, paragraphs, links, images, etc.

Attribute: attributes some characteristics to the HTML element, like href in <a>, src, width, height, alt in <img>, style in p, lang in html, title in p.

**Note**: alt is just for image elements, the title is just for text elements.

**Note**: <p> ignores spaces and line breaks, but <pre> keeps them all and shows them just in a fixed font like the Courier font.

Some style examples:

<tagname style="property:value;">

<body style="background-color:powderblue;">

<p style="color:red;">This is a paragraph.</p>

<p style="font-family:courier;">This is a paragraph.</p>

<p style="font-size:160%;">This is a paragraph.</p>

<p style="text-align:center;">Centered paragraph.</p>

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

Formatting tags which are used for formatting the text are listed below:

* <b> - Bold text
* <strong> - Important text
* <i> - Italic text
* <em> - Emphasized text
* <mark> - Marked text
* <small> - Smaller text
* <del> - Deleted text
* <ins> - Inserted text
* <sub> - Subscript text
* <sup> - Superscript text

Commenting a HTML tag is in this way: <!—some code here -->

For colors with can compose them with different quantities of 3 colors.

Bookmark is used for moving in a page. I took a note from this, it can be found on the folder of the note. Its main idea is to work with “id” and address it by #id.

**Image maps**: putting link on images

<img src="workplace.jpg" alt="Workplace" usemap="#workmap">  
  
<map name="workmap">  
  <area shape="rect" coords="34,44,270,350" alt="Computer" href="computer.htm">  
  <area shape="rect" coords="290,172,333,250" alt="Phone" href="phone.htm">  
  <area shape="circle" coords="337,300,44" alt="Coffee" href="coffee.htm">  
</map>

For complex shapes, we can poly as follows:

<area shape="poly" coords="140,121,181,116,204,160,204,222,191,270,140,329,85,355,58,352,37,322,40,259,103,161,128,147" href="croissant.htm">

Good usage of image maps: cooperation with JS

<map name="workmap">  
  <area shape="circle" coords="337,300,44" onclick="myFunction()">  
</map>  
  
<script>  
function myFunction() {  
  alert("You clicked the coffee cup!");  
}  
</script>

*<picture> and <img> difference*: (1) **Bandwidth**: small devices don’t need large files (2) **Format Support**: to put different formats of images

<picture>  
  <source srcset="img\_avatar.png">  
  <source srcset="img\_girl.jpg">  
  <img src="img\_beatles.gif" alt="Beatles" style="width:auto;">  
</picture>

How to create a horizontal list: <https://www.w3schools.com/html/tryit.asp?filename=tryhtml_lists_menu>

Inline and Block Elements: Inline ones just come after an element before them and it only takes up as much width as necessary, but block ones start a new line and take up the full width available, <div> and <span> is the most famous block and inline element.

Iframe is for showing a web page within a web page.

<iframe src="demo\_iframe.htm" style="height:200px;width:300px;border:none;" title="Iframe Example"></iframe>

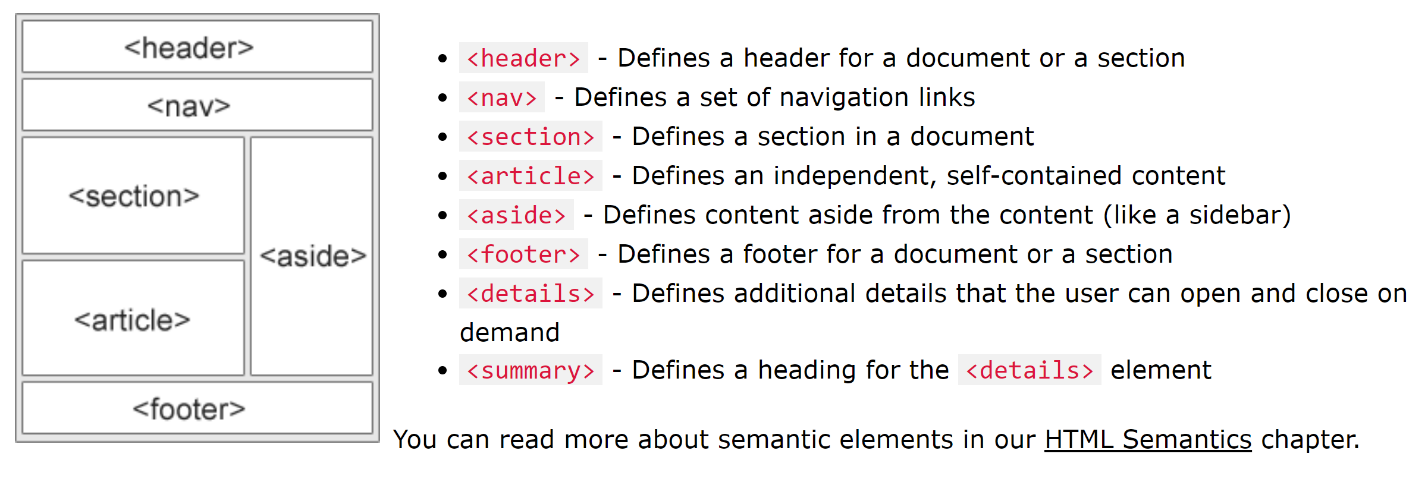
We can use JS codes in out HTML file by writing our script between two <script> and </script> tags. The example can be viewed in the following link:

<https://www.w3schools.com/html/tryit.asp?filename=tryhtml_script_html>

<head> is for metadata, like <title> which is for SEO, <style> for CSS, <link> for linking to CSS stylesheets, <meta> for different purposes from language, character set to keywords and viewpoint.

<meta name="viewport" content="width=device-width, initial-scale=1.0">

**Layout in HTML**:



These can be formatted easily by the help of CSS frameworks like Bootstrap and W3.CSS.

HTML Forms: For receiving data from the user.

**Notes on GET:**

* Appends the form data to the URL, in name/value pairs
* NEVER use GET to send sensitive data! (the submitted form data is visible in the URL!)
* The length of a URL is limited (2048 characters)
* Useful for form submissions where a user wants to bookmark the result
* GET is good for non-secure data, like query strings in Google

**Notes on POST:**

* Appends the form data inside the body of the HTTP request (the submitted form data is not shown in the URL)
* POST has no size limitations, and can be used to send large amounts of data.
* Form submissions with POST cannot be bookmarked

There many input types, which make the validation very easy.

I also learned how to embed a video from YouTube using Iframe.

# What I reviewed about CSS, also what I coded

CSS stands for Cascading Style Sheets

External CSS:

<head>  
  <link rel="stylesheet" href="styles.css">

<link rel="stylesheet" href="https://www.w3schools.com/html/styles.css">  
</head>

Inside of a CSS file:

body {  
  background-color: powderblue;  
}  
h1 {  
 color: blue;  
}  
p {  
  color: red;

border: 2px solid powderblue;

  padding: 30px;

  margin: 50px;

  font-family: courier;  
  font-size: 160%;  
}

Link attributes:

<style>  
a:link {  
  color: green;  
  background-color: transparent;  
  text-decoration: none;  
}  
  
a:visited {  
  color: pink;  
  background-color: transparent;  
  text-decoration: none;  
}  
  
a:hover { // it is a pseudo-class  
  color: red;  
  background-color: transparent;  
  text-decoration: underline;  
}  
  
a:active { // it is also a pseudo-class  
  color: yellow;  
  background-color: transparent;  
  text-decoration: underline;  
}  
</style>

The “**float**” style of an image makes it out of a block.

I watched several videos and coded simultaneously with them on how to use class, id.

I learned how to change the color, and how to use gradient. How to put background image, style it, and give HTML elements borders.

Also, I learned how to use the inspect feature of the browser to change the style of the elements in a fast way.

Combinators in CSS are something that specify the parent child order like <ul li> or <ul > li>.

Grouping in CSS is a help for reducing code redundancy. It is something like as follows:

h1, h2, h3, .class, #id {

font-color: red;

}

And about text formatting: I learned how to change the font of the text which comes in the paragraphs. We can change the size of a font with a font-size attribute using pixels or rems. Their difference is that px is not scalable but em and rem are scalable. Change in the root of the em and rem is going to change the interpreted font size.

/\* style \*/

html {

font-size: 18px;

}

section {

font-size: 14px;

padding: 3em;

}

/\* computed style \*/

html {

font-size: 18px;

}

section {

font-size: 14px;

padding-bottom: 42px;

padding-left: 42px;

padding-right: 42px;

padding-top: 42px;

}

On rem:

/\* style \*/

html {

font-size: 18px;

margin: 2rem;

}

/\* computed style \*/

html {

font-size: 18px;

margin-bottom: 36px;

margin-left: 36px;

margin-right: 36px;

margin-top: 36px;

}

Text-align: center, right, left 🡪 changing the alignment of the text inside an element.

Changing the style of links 🡪 text-style: none;

Text-transform: uppercase;

Word-spacing: 2em;

Line-height: 2em;

Then I learned how to use Google Fonts using the prepared API by Google.

<link rel="preconnect" href="https://fonts.gstatic.com">  
<link href="https://fonts.googleapis.com/css2?family=**Source+Sans+Pro**&display=swap" rel="stylesheet">

font-family: 'Source Sans Pro', sans-serif;

**Note**: using too many fonts on a website can be distractive, so we have to try to keep it simple. For this purpose, we can use the fonts in a standard or import way by URL.

**Responsive CSS Images**: we can use margin and padding in auto mode inside a div in which our image elements are placed there. Also, we have to use a percentage to specify the images’ width and height. For example, we can give max-width a number in percentage and height an auto.

**Rows and columns in CSS**:

I learned how to do it and the code can be found in the subfolder of the CSS folder on my GitHub page.

**Building the Navbar**:

I built it using CCS and JS.

**Note**: Using class is for styling, but id is useful when we want to interact with the JS.

I finished by building a simple responsive website.

# What I learned about Bootstrap 4, also what I coded

Bootstrap is one the popular frameworks for building mobile first website or applications. The most interesting thing about it is that it is pre-built with loads of classes and stylings, so it virtually provides us what we are going to need in frontend design.

On the first step, I learned how to download and use it from its website. We can use the document of this framework to put what we want in our website.

I learned how to use documentation of the Bootstrap to use different component or arrange components in **row, column** responsive structures.

I enjoyed using **cards** for sign-in design, and **carousels** for image slide shows. Also, **modals** are fascinating. Finally, I ended this technology by writing a sample website. Additionally, I learned about “**awesome font**”.

# What I learned about Sass, also what I coded

**SASS** stands for **S**yntactically **A**wesome **S**tyle **S**heet. This is from the sass-lang.com:

“Sass is the most mature, stable, and powerful, professional grade CSS extension language in the world.”

Some of Sass advantages:

* CSS compatible
* Feature rich
* Mature – more than 9 years old
* Industry approved
* Tons of frameworks that support Sass
* Easy and fast to write code with

Sass in fact is a CSS preprocessor -> A program that processes its input data to produce output that is used as input to another program.

Why to use Sass? CSS can get messy, unorganized, cluttered, etc. on the other hand, SASS is clean, simple to organize, easy to read. Also, writing code with SASS save time because with SASS, we are not going to care about curly braces and semicolons. Besides, SASS syntax is easy to learn. Additionally, there are lots of features like variables that make our lives as programmer easy.

For installing SASS, ruby is a requirement. Then, SASS can be installed with “**gem install sass**” command. Then, we create files with “. sass” extension, which in them we write our sass code.

To work with sass, first we make a file with sass extension, then we have to link a file with css file to out html. Then, we have to watch the sass file by a terminal command.

**sass –-watch styles.sass: styles.css**

Or we can do it manually too.

**sass styles.sass styles.css**

The most important feature of SASS is to organize our sass file in different folders and different file with regard to the Scalable and Modular Architecture for CSS paradigm. In this way we respect the determined rule by Jonathan Snook. Then, we just use the following command to change what there are in our SASS folder to a CSS folder.

**sass –-watch SASS:CSS**

If we want to avoid creating additional files like map and partial CSS files, we can do it just by adding underscore to the first of the name of the SASS files in our structure.

Finally, I started to do a project with SASS snippets that make any web developer’s life much easier.

The most important thing to add is that every web developer must respect the following rules if he wants to keep his code neat and manageable.

<http://smacss.com/>

Finally, I practiced a non-responsive page with some videos for devslopes on the YouTube. But, the most important thing I grasped is that I have to review the material and also try to learn CSS well first because SASS, Bootstrap etc. on the frontend zone are based on the CSS and if a person does not know CSS well, he is going to make tons of mistakes.

# What I learned about JS, also what I coded

I watched the tutorial videos and everything was ok about how to use this language. It is like the Python language. I have to learn these concepts:

Also, developed a percent calculator listening to a form, then finally disabling its default behavior not to refresh the form after submit by using event.preventDefaults();.

I have to work on this language and learn its concepts at in Advanced Level.

# What I learned about C#, also what I coded

# What I reviewed about SQL, also what I coded

First of all, I reviewed the basic concepts of data bases like table, primary key, alternate key (keys other than primary key), candidate key (column that its values are unique per row), foreign key (creates a relationship between two tables), compound key (has two or more attributes that allow you to uniquely recognize a specific record. It is possible that each column may not be unique by itself within the database.), compound key (COMPOSITE KEY is a combination of two or more columns that uniquely identify rows in a table. The combination of columns guarantees uniqueness, though individually uniqueness is not guaranteed. Hence, they are combined to uniquely identify records in a table.), surrogate key (an artificial key, like an auto number).

The difference between compound and the composite key is that any part of the compound key can be a foreign key, but the composite key may or maybe not a part of the foreign key.

**SQL**: **S**tructured **Q**uery **L**anguage: for interaction with Relational **D**ata **B**ase **M**anagement **S**ystems (DBMSs).

What SQL can do for us?

1. **CRUD** operations on data
2. Creating and managing databases
3. Designing and managing database tables
4. Performing administration tasks (security, user management, import/ export, etc.)

SQL = DQL + DDL + DCL + DML

**Query**: a set of instructions given to the RDBMS that tell the RDBMS what information you want it to retrieve for you.

SELECT employee.name, employee.age

FROM employees

WHERE employee.salary > 30000

Then I installed MySQL to test around with Relational Databases and do some exercise in designing data bases and writing queries.

We can command to MySQL in its shell to do something for us like:

create database db\_name;

popsql is a software, which makes the learning phase of SQL really enjoyable by providing visual features instead of just working with black and white command line.

Basic data types that are used in SQL databases:

INT, DECIMAL(M, N), VARCHAR(N), BLOB –Binary Large Object-,DATE, TIMESTAMP

An example of creating a table:

CREATE TABLE student(

    student\_id      INT     PRIMARY KEY,

    name            VARCHAR(35),

    major           VARCHAR(35)

);

Other commands can describe a table or a feature of a table or delete a table with the following instructions:

DESCRIBE student;

DROP TABLE student;

A table can be edited for example a column to be added to it or removed from it.

ALTER TABLE student ADD gpa DECIMAL(4, 2);

ALTER TABLE student DROP COLUMN gpa;

We can insert data into tables with the help of following commands:

INSERT INTO student VALUES(2020213, "JACK", "CS", 14.32);

INSERT INTO student(student\_id, name) VALUES(789654, "JACKIE");

Columns can be configured in a way not to be NULL and be UNIQUE:

CREATE TABLE student(

    student\_id      INT     PRIMARY KEY,

    name            VARCHAR(35) NOT NULL,

    major           VARCHAR(35) UNIQUE

);

In fact, primary key is a column that is not null and also unique.

We can specify a default value for each column as follows:

CREATE TABLE teacher(

    teacher\_id      INT  PRIMARY KEY,

    name            VARCHAR(35) NOT NULL,

    major           VARCHAR(35) DEFAULT 'not-specified'

);

We can update a row of data:

UPDATE student

SET major = 'computer science’

WHERE major = 'CS';

We can remove a row of data too.

DELETE FROM student

WHERE major = 'computer science';

But, the most important skill is being able to ask properly from the DB to get the data we need.

SELECT name, major

FROM student

ORDER BY student\_id ASC (or DESC);

We can limit the number of rows we want to get from the table by using LIMIT keyword.

SELECT name, major

FROM student

ORDER BY student.name

LIMIT 2;

Not-equal sign in SQL is <>.

SELECT name, major

FROM student

WHERE name <> "JACKIE";

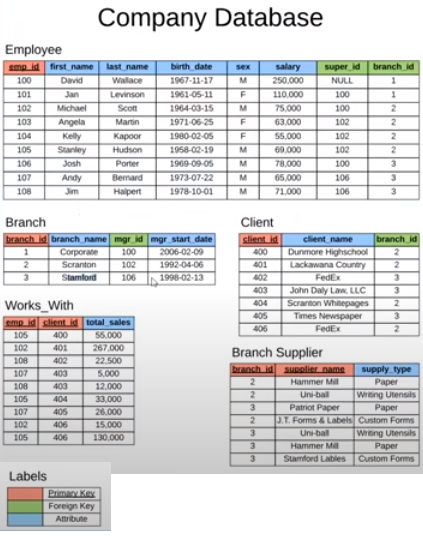
We can check if a column’s values are within a list to be shown:

SELECT name, major

FROM student

WHERE name IN ("JACKIE", "JOE", "Ali");

Then the schema of a database as shown in the following images was created.



Because other tables are not created yet so we cannot define them yet then we have to just alter this table when we create them.

CREATE TABLE employee (

    emp\_id  INT PRIMARY KEY,

    first\_name  VARCHAR(40),

    last\_name   VARCHAR(40),

    birth\_day   DATE,

    sex         VARCHAR(1),

    salary INT,

    super\_id INT,

    branch\_id INT

);

Now it is branch table’s turn.

 CREATE TABLE branch (

     branch\_id INT PRIMARY KEY,

     branch\_name VARCHAR(40),

     mgr\_id INT,

     mgr\_start\_date DATE,

     FOREIGN KEY(mgr\_id) REFERENCES employee(emp\_id) ON DELETE SET NULL

 );

Then, we have to change the employee table to make foreign keys to establish connection between the created tables.

ALTER TABLE employee

ADD FOREIGN KEY(branch\_id) REFERENCES branch(branch\_id)

ON DELETE SET NULL;

ALTER TABLE employee

ADD FOREIGN KEY (super\_id) REFERENCES employee(emp\_id)

ON DELETE SET NULL;

Then it is the client table’s turn:

CREATE TABLE client (

    client\_id INT PRIMARY KEY,

    client\_name VARCHAR(40),

    branch\_id   INT,

    FOREIGN KEY(branch\_id) REFERENCES branch(branch\_id) ON DELETE SET NULL

);

The next step is creating works\_with table:

CREATE TABLE works\_with (

    emp\_id INT,

    client\_id INT,

    total\_sales INT,

    FOREIGN KEY(emp\_id) REFERENCES employee(emp\_id) ON DELETE CASCADE,

    FOREIGN KEY(client\_id) REFERENCES client(client\_id) ON DELETE CASCADE

);

Finally, it is branch supplier table’s turn to be created:

CREATE TABLE branch\_supplier (

    branch\_id   INT,

    supplier\_name   VARCHAR(40),

    supply\_type     VARCHAR(40),

    PRIMARY KEY(branch\_id, supplier\_name),

    FOREIGN KEY (branch\_id) REFERENCES branch(branch\_id) ON DELETE CASCADE

);

Now everything is created we just need to fill the tables with data.

INSERT INTO employee VALUES(100, 'DAVID', 'Wallance', '1967-11-17', 'M', 25000, NULL, NULL);

INSERT INTO branch VALUES(1, 'Corporate', 100, '2006-02-09');

UPDATE employee

SET branch\_id = 1

WHERE emp\_id = 100;

INSERT INTO employee VALUES(101, 'Jan', 'Levinson', '1961-05-11', 'F', 110000, 100, 1);

INSERT INTO branch\_supplier VALUES(2, 'Hammer Mill', 'Paper');

INSERT INTO branch\_supplier VALUES(2, 'Uni-ball', 'Writing Utensils');

INSERT INTO branch\_supplier VALUES(3, 'Patriot Paper', 'Paper');

INSERT INTO branch\_supplier VALUES(2, 'J.T. Forms & Labels', 'Custom Forms');

INSERT INTO branch\_supplier VALUES(3, 'Uni-ball', 'Writing Utensils');

INSERT INTO branch\_supplier VALUES(3, 'Hammer Mill', 'Paper');

INSERT INTO branch\_supplier VALUES(3, 'Stamford Lables', 'Custom Forms');

INSERT INTO client VALUES(400, 'Dunmore Highschool', 2);

INSERT INTO client VALUES(401, 'Lackawana Country', 2);

INSERT INTO client VALUES(402, 'FedEx', 3);

INSERT INTO client VALUES(403, 'John Dally Law, LLC', 3);

INSERT INTO client VALUES(404, 'Scranton Whitepages', 2);

INSERT INTO client VALUES(405, 'Times Newspaper', 3);

INSERT INTO client VALUES(406, 'FedEx', 2);

INSERT INTO works\_with VALUES(105, 400, 55000);

INSERT INTO works\_with VALUES(102, 401, 267000);

INSERT INTO works\_with VALUES(108, 402, 22500);

INSERT INTO works\_with VALUES(107, 403, 5000);

INSERT INTO works\_with VALUES(108, 403, 12000);

INSERT INTO works\_with VALUES(105, 404, 33000);

INSERT INTO works\_with VALUES(107, 405, 26000);

INSERT INTO works\_with VALUES(102, 406, 15000);

INSERT INTO works\_with VALUES(105, 406, 130000);

Ok, finally populating our DB is done. Now we can go and test and write our queries. Below some basic queries are listed:

SELECT \*

FROM employee;

SELECT \*

FROM client;

SELECT \*

FROM employee

ORDER BY salary DESC;

SELECT \*

FROM employee

ORDER BY salary ASC;

SELECT \*

FROM employee

ORDER BY sex, first\_name, last\_name;

SELECT first\_name AS "FN", last\_name AS "LN"

FROM employee;

SELECT DISTINCT sex

FROM employee;

Functions in SQL!

SELECT COUNT(emp\_id)

FROM employee;

SELECT COUNT(emp\_id)

FROM employee

WHERE sex = 'F' AND birth\_day > '1970-01-01';

SELECT AVG(salary)

FROM employee;

SELECT AVG(salary)

FROM employee

WHERE sex = 'M';

SELECT SUM(salary)

FROM employee;

SELECT sex, COUNT(sex)

FROM employee

GROUP BY sex;

SELECT SUM(total\_sales), client\_id

FROM works\_with

GROUP BY client\_id;

Defining pattern in SQL (wild card!):

-- % = any # characters

-- \_ = one character

SELECT \*

FROM client

WHERE client\_name LIKE '%LLC';

SELECT \*

FROM branch\_supplier

WHERE supplier\_name LIKE '% Lable%';

SELECT \*

FROM employee

WHERE birth\_day LIKE '\_\_\_\_-10-\_\_';

SELECT \*

FROM employee

WHERE birth\_day LIKE '%-02-%';

SELECT \*

FROM client

WHERE client\_name LIKE '%school%';

Union operator in SQL:

SELECT first\_name

FROM employee

UNION

SELECT branch\_name

FROM branch;

SELECT client\_name, client.branch\_id

FROM client

UNION

SELECT supplier\_name, branch\_supplier.branch\_id

FROM branch\_supplier;

SELECT salary

FROM employee

UNION

SELECT total\_sales

FROM works\_with;

JOIN in SQL – for combining information from different table that there is a relationship between them – inner join is the default action that takes place when left or right is not specified explicitly. In inner join only rows are outputted that the specified criteria matches:

SELECT employee.emp\_id, employee.first\_name, employee.last\_name, branch.branch\_name

FROM employee

JOIN branch

ON employee.emp\_id = branch.mgr\_id;

Nested Queries:

SELECT employee.first\_name, employee.last\_name

FROM employee

WHERE employee.emp\_id IN (

SELECT emp\_id

FROM works\_with

WHERE total\_sales > 30000);

SELECT client.client\_name

    FROM client

    WHERE client.branch\_id = (

        SELECT branch.branch\_id

        FROM branch

        WHERE branch.mgr\_id = 102

        LIMIT 1

);

**Trigger in SQL**: in fact, triggers are a block of SQL code that can define a certain action that should happen when a certain operation gets performed on the database. A trigger is something that it has to be commanded on the terminal. In the following example, first a table created.

CREATE TABLE trigger\_test(

    message VARCHAR(100)

);

Then the following code is given to the MySQL command line in 3 steps as showed:

DELIMITER $$ (1)

CREATE

    TRIGGER my\_trigger BEFORE (or AFTER) INSERT (or DELETE)

    ON employee

    FOR EACH ROW BEGIN

        INSERT INTO trigger\_test VALUES('added new employee');

    END $$ (2)

DELIMITER ; (3)

We can remove trigger with DROP command:

DROP TRIGGER trigger\_test;

# I also reviewed the Git basics and you can find all I did on my GitHub page.

Always we have to try to use a control version system like Git.

The problems that these systems try to solve:

* Reverting to old files
* Maintaining code within a team.

Then, I reviewed some of Unix shell commands like cd, dir, mkdir, ls, touch, cp, mv, rm, rmdir, etc.

After that I moved to review what I knew about Git. I started add an SSH key to my GitHub page. Then, I created two projects in which in them I pushed what I did in several days. I list below the commands:

**git init**

**git add .**

**git commit -m “the message”**

**git status**

**git log**

**git checkup #number-of-a-commit**

**git branch**

**git checkout master**

**git remote -v**

git remote add <origin🡪 we can name it everything we like> <link-of-the-repository-which-we-are-going-to-push-our-project>

git pull origin master

if there are no conflicts, the merge is going to be done on the project. However, if any conflict occurs, we have to solve it first by forcing it or talking with the teammates and solving it.

GitHub is a server for pushing for our codes on it, but Git is a CVS which can act locally. We can use any other server to synchronize our project’s codes with our teammates. Bitbucket is a competitor of GitHub. It is cheaper than GitHub. However, GitHub is more well-known for open source project rather than other services.

# What I learned about Kendo UI framework

# What I learned about hosting a website, Node, MongoDB

DigitalOcean can be a good choice because its cost is low and provides a scalable environment that can be manipulated.

Nodejs is what is required on a server to be able to host a basic static website or an angular or react project.

After buying an account with some resources like CPU, storage on a server, an SSH key is added to the panel to be able to communicate with the server to put our website’s file on it to be shown on the web. Then we can connect to our server with the following command in the terminal:

$ ssh -i id\_rsa root@ip\_address

**Note**: the ip\_address can be found in our account on the server.

**Nginx** is one of the most popular web servers in the world and is responsible for hosting some of the largest and highest-traffic sites on the internet. It is more resource-friendly than Apache in most cases and can be used as a web server or reverse proxy. After installing the Nginx on our server account “Welcome to Nginx!” will be shown. Then, we are ready to get our project uploaded on the server.

Then we make SSH key on our server then we add it to our GitHub or other version control website – like GitLab- that we work with to be able to pull the project from there.

After purchasing a domain, we add it to the server. Then, we have to add the nameservers to the panel where we bought our domain. However, we have to go to nginx folder to available website, then build a file and configure the settings to show our website as it is taught in the DigitalOcean tutorials.

**NodeJS**: It is really useful tool helping us interpreting our JavaScript codes. Also, it makes web services up and going easily, talking to mobile apps. It is a very powerful tool. Additionally, Node.js’s package ecosystem, npm, is the largest ecosystem of open source libraries in the world. Before node, JS codes just were able to run on browsers, but with it now we can run JS codes on different operating systems like Windows. After installing node, we can have source codes in .js extension, then be able to run with them with “node filename.js” command in the terminal.

**Basics of Node**: JSON (JavaScript Object Notation) stole its popularity from XML. These days it is the most popular one used for mobile app development for sending data to mobile applications. JSON makes changing JS objects to received and sent data on the internet easy. Parse the data with JSON.parse(), and the data becomes a JavaScript object.

In a project folder, we can make the node package manager to control the dependencies by “npm init” command. Then, if we install any package with the “--save" argument, the package will be added to the package.json file. When we send our project somewhere else for example when we push it on a GitHub repository, we don’t need to upload the installed packages. The installer just has to enter “npm install” to install all dependencies of the project.

Then learned how to start a server with some lines of JS and also using Postman to interact with that server to receive and sent data to the server. In the example, I kept the data received from the clients on the RAM of the server on an array, so when the server is killed and revived again, the data will be lost. So, we need Data Base.

Get requests are for showing, Post for storing sth, and put is for updating.

“nodemon” tool helps us not to kill the server whenever we make some changes to our code.

Npm install -g nodemon

Then we can run our server with nodemon command.

**MongoDB**: mongoDB is based on a document model – different from traditional relational data bases. MongoDB’s aim is to solve the inefficiency of traditional relational database – when we want to avoid parse tables, anyway we have to traverse several tables (harder to understand the project, harder to add features, inefficient data fetch). In mongoDB, data is stored in records called documents and it is just like physical documents. Working with mongoDB is completely simple. With the following commands, I could add a record to a data base then see what happened by terminal. However, working with the terminal directly can make typos that end in mistakes to the structure of data – data integrity.

show dbs

use learning-mongo

db.products.insert({“productName”:”Car”, “price”:2500.00});

show collections

db.products.find()

db.products.find().pretty()

db.products.update(“productName”:”Car”, $set:{“price”:3000.59})

db.products.remove({“productName”:”Car”})

With reading the document provided by mongoDB’s website everything can be implemented easily and smoothly.

Mongoose package lets our project to interact easily with the mongoDB.

killall mongod

Then a simple API project was developed which through it I could get a vision on what happens totally.

# What I learned about React

This is a JS library for building user interfaces. We can install it globally with the following command in the terminal.

**npm install –g create-react-app**

Then, we can create a project with the following command.

**npx create-react-app swag-shop-web**

We can launch it with the following command – in the project’s main directory.

**npm start**