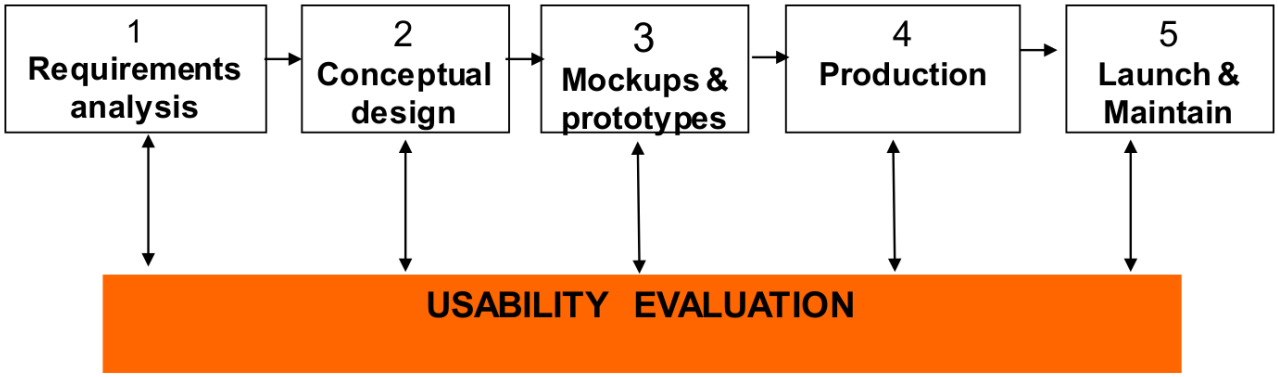
Hypermedia Applications

github.com/martinopiaggi/polimi-notes

2022-2023

Contents

1 Usability



The user is always right. It's always your fault, not user fault.

Usability is the measure of the effectiveness, efficiency and satisfaction when we specify users can achieve a specified goal in particular environment. In this course, we will focus on the functional view of usability, which emphasizes the simplicity and ease of use of any product.

1.1 Usability Evaluation

There are two ways to perform a usability evaluation:

- inspection: heuristic-based approach
- user testing: user behaviors are observed, recorded, and then analyzed by researchers (eye tracking and statistics on different parameters like time and tasks)

1.1.1 User testing

User profiles and user goals are used to segment the target audience and recruit users for usability testing.

User testing for usability evaluation is an example of **empirical research** (gaining knowledge through observed phenomena). The data gathered can be: - quantitative: task success rate and time on task - qualitative: user satisfaction and perceived difficulty

1.1.1.1 Generic workflow for user testing

1) Define test **goals**
2) Define qualitative and quantitative **variables** to measure and data to collect
3) Define **tasks** to assign to the users
4) Define **how to recruit** users
5) Build the **materials** for data collection (e.g., forms)
6) Create sw and hw setting
7) "Test the test" with a mini-sample: If needed, go-back and re-design the test

1.1.2 Inspection: heuristics-based

The most popular inspection method is heuristic evaluation, which is based on checklists and usability principles. Heuristics are a set of principles that guide in the discovery of usability flaws. Heuristics are useful for evaluating any interactive system, but they are not universally recognized.

1.1.2.1 Generic workflow for inspection

1) Identify typical user, his/her goal(s), expectations, background & experience
2) Test the user flows identified
3) Find problems with the eyes of the users who is trying to accomplish a task to get to an objective
4) For each problem find the severity, which heuristics it violates and possible solutions

Problem Description	Severity (1-5)	Violated Heuristics	Redesign suggestions
...	...	...	...

Ratings of severity:

0. Not a problem
1. Cosmetic problem; should be addressed if schedule permits.
2. Grammatical, spelling, or other errors; should be fixed as they can affect users' impressions of the interface and its creators, but won't impact usability.
3. Minor problem; should be fixed if possible as this will be an annoyance to users, but won't affect ability to achieve goals.
4. Major problem; important to fix, as this will impact users' ability to achieve goals.
5. Catastrophic problem; must be fixed before product is released.

2 Heuristics for User Interface Design

Usability heuristics are a set of guidelines or principles which serve as a benchmark for usability inspection. The three main types of usability heuristics are:

- navigation: how is easy to move use the interface and navigate
- content: actual quality of the content
- presentation: visual design and layout of the interface or system.

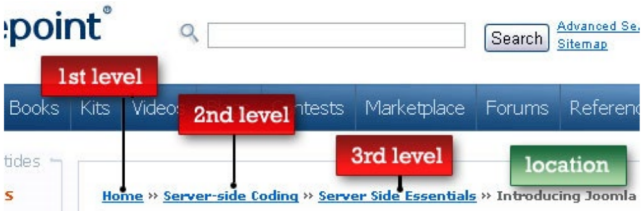
2.1 Nielsen Heuristics

Jakob Nielsen's 10 general principles are not only for website but for any interactive system. They are called "heuristics" because they are broad rules of thumb and not specific usability guidelines. Originally developed in 90s, the 10 heuristics themselves have remained relevant and unchanged.

Category	Heuristic
Navigation	Visibility of system status
Presentation	Match between system and the real world
Navigation	User control and freedom
Presentation	Consistency and standards
Presentation	Error prevention
Nav / Pres	Recognition rather than recall
Navigation	Flexibility and efficiency of use
Presentation	Aesthetic and minimalist design
Presentation	Help users recognize, diagnose and recover from errors
Content	Help and documentation

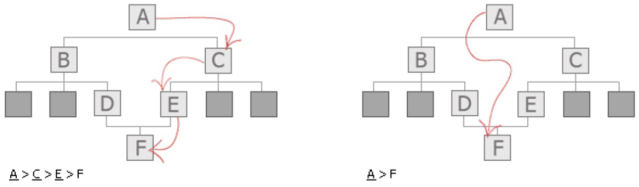
2.1.1 Visibility of system status

- The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
- If the system acts as a black box, it can lead to errors.
- Status bars show the status of the ongoing process by the system.
- Interactive labels show the current position of the user in the ongoing process.
- Breadcrumbs are another technique that shows the path the user has taken.
- Labels can also be used for attributes of a particular page, such as the semantic classification of content objects.
- It is a good practice to remove the “home” label as it is redundant.



Bad example:

- Interactive labels show users the path they have taken to arrive at a particular page.
- These labels are dynamically generated based on the pages the user has visited before arriving at the current page.
- However, there are risks of usability and visualization problems if the user has performed a high number of steps.
- When the number of steps is too high, the label may become too long, and it may be difficult for users to interpret and use effectively.



2.1.2 Match between system and the real world

- The system should use the language that is familiar to the user, with words, phrases, and concepts that the user understands.

- It is important to follow real-world conventions to make information appear in a natural and logical order

for the user.

2.1.3 User control and freedom

- Users may accidentally choose system functions and will need an emergency exit: undo and close functions to allow users to revert to a previous state.
- The emergency exit should be clearly marked and easily accessible
- The undo function should (instead of starting the selection process again from the beginning) just undo some actions.

2.1.4 Consistency and standards

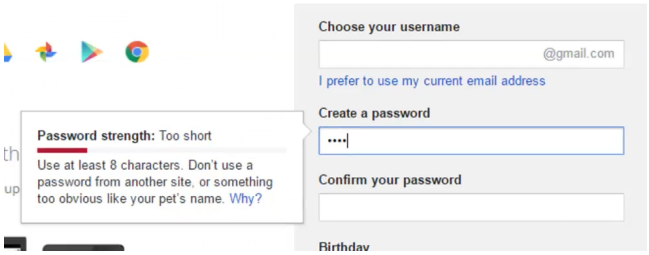
- Users should not have to wonder whether different words, situations, or actions mean the same thing in

the system.
- It is essential to follow conventions to ensure consistency across the system and make it easier for users to understand and use.

2.1.5 Error prevention

- Careful design » error messages: the design should aim to eliminate error-prone conditions.
- Example of careful design: It is common to end up on the wrong page on many websites. Instead of relying on the browser's back button, it is better to make the link's destination explicit and understandable through a clear link label.

Holy grail of error prevention:



Don´t make me insert all data and then throw me in an error page after I submitted the form. Just prevent it with an error popup.

2.1.6 Recognition rather than recall

- It is better to suggest a set of options to the user (**recognition**) than to expect them to remember the information (**recall**).
- The goal is to minimize "the RAM used of the user brain" aka cognitive load

2.1.7 Flexibility and efficiency of use

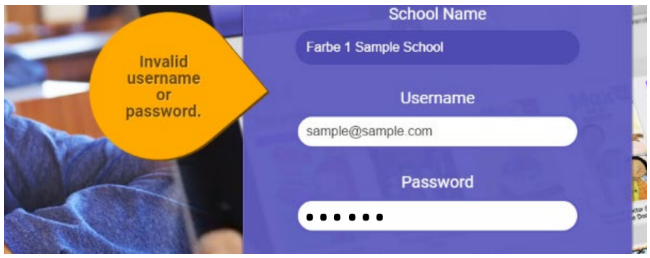
- Providing the user with the ability to personalize the system can make it more flexible and efficient to their needs.
- This allows the system to attract both inexperienced and experienced users.
- Examples of accelerators include system layout or keyboard shortcuts
- *Even in Webeep you can personalize your view.*

2.1.8 Aesthetic and minimalist design

- Everything you put in the view stimulates the user
- So every useless stuff decreases the relative visibility of the important stuff

2.1.9 Help users recognize, diagnose and recover from errors

- Error messages should be expressed in plain language (no error codes), precisely indicate the problem, and constructively suggest a solution.



- *Is this popup useful?*

2.1.10 Help and documentation

- Ideally, a system should be designed so that it can be used without the need for documentation.
- However, in some cases, it may be necessary to provide help and documentation to users.

2.2 ## MiLe Heuristics

MiLe (Milano Lugano) Usability Evaluation method is a comprehensive set of heuristics focused on organizing content in a clear and intuitive way, highlighting the importance of readability and consistency of visual elements.

Category	Heuristic
Navigation	Interaction consistency
Navigation	Group navigation
Navigation	Structural Navigation
Navigation	Semantic Navigation
Navigation	Landmarks
Content	Information overload
Content	Consistency of page content structure
Content	Contextualized information
Content	Content Organization (hierarchy)
Presentation	Text layout
Presentation	Interaction placeholders-semiotics
Presentation	Interaction placeholders-consistency
Presentation	Consistency visual elements
Presentation	Hierarchy
Presentation	Spatial allocation
Presentation	Consistency of Page Structure

2.3 Navigation/Interaction

- **Interaction consistency**: Do pages of the same type have consistent navigation links and interaction capabilities?
- **Group navigation-1**: Is it easy to navigate between and within groups of items, including from a list of items to their members and vice versa?
- **Group navigation-2**: Do menus create cognitive overload?
- **Structural navigation**: Is it easy to navigate between components of a topic?
- **Semantic navigation**: Is it easy to navigate between related topics in both directions?
- **Landmarks**: Are links available on all pages effective for users to reach key parts of the website?

2.4 Content

- **Information overload**: Is the amount of information on a page appropriate? There is too much content?
- **Consistency of page content structure**: Do pages presenting topics of the same category have consistent types of elements?
- **Contextualized information**: Does the page provide information to help users understand where they are?
- **Content organization (hierarchy)**: Is the hierarchical organization of topics appropriate for their relevance?

2.5 Presentation

- **Text layout**: Is the text readable and is font size appropriate?
- **Interaction placeholders-semiotics**: Do interactive elements have intuitive textual and visual labels/icons that convey their functional meaning?
- **Interaction placeholders-consistency**: Are textual or visual labels of interactive elements consistent in wording, shape, color, position, etc.?
- **Consistency of visual elements**: Do pages of the same type have consistent visual properties for visual elements?
- **Hierarchy-1**: Is the on-screen allocation of contents within a page appropriate for their relevance?
- **Hierarchy-2**: Is the on-screen allocation of visual elements appropriate for their relevance?
- **Spatial allocation-1**: Are semantically related elements close to each other?
- **Spatial allocation-2**: Are semantically distant elements placed distant from each other?
- **Consistency of page spatial structure**: Do pages of the same type have consistent spatial organization for visual elements?

3 Visual Design Principles

> "Any product (from software products such as websites and apps to hardware products such as toasters and hairdryers) can be broken down into fundamental elements of visual design."

These visual elements:

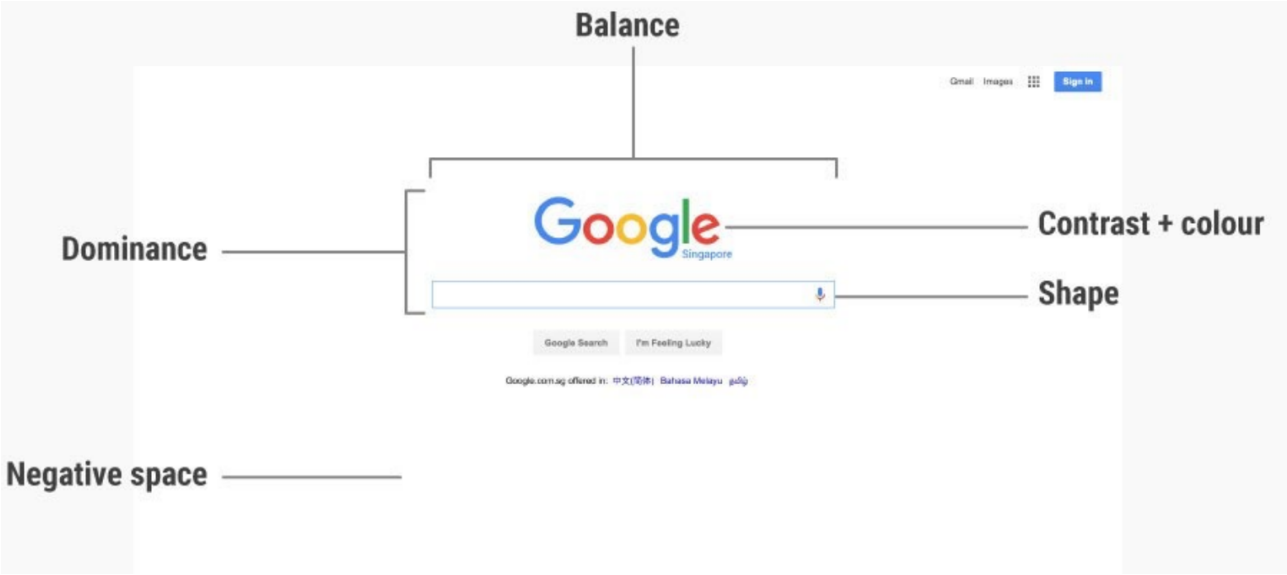
- Line
- Shape
- Value
- Volume
- Color
- Texture
- Typography

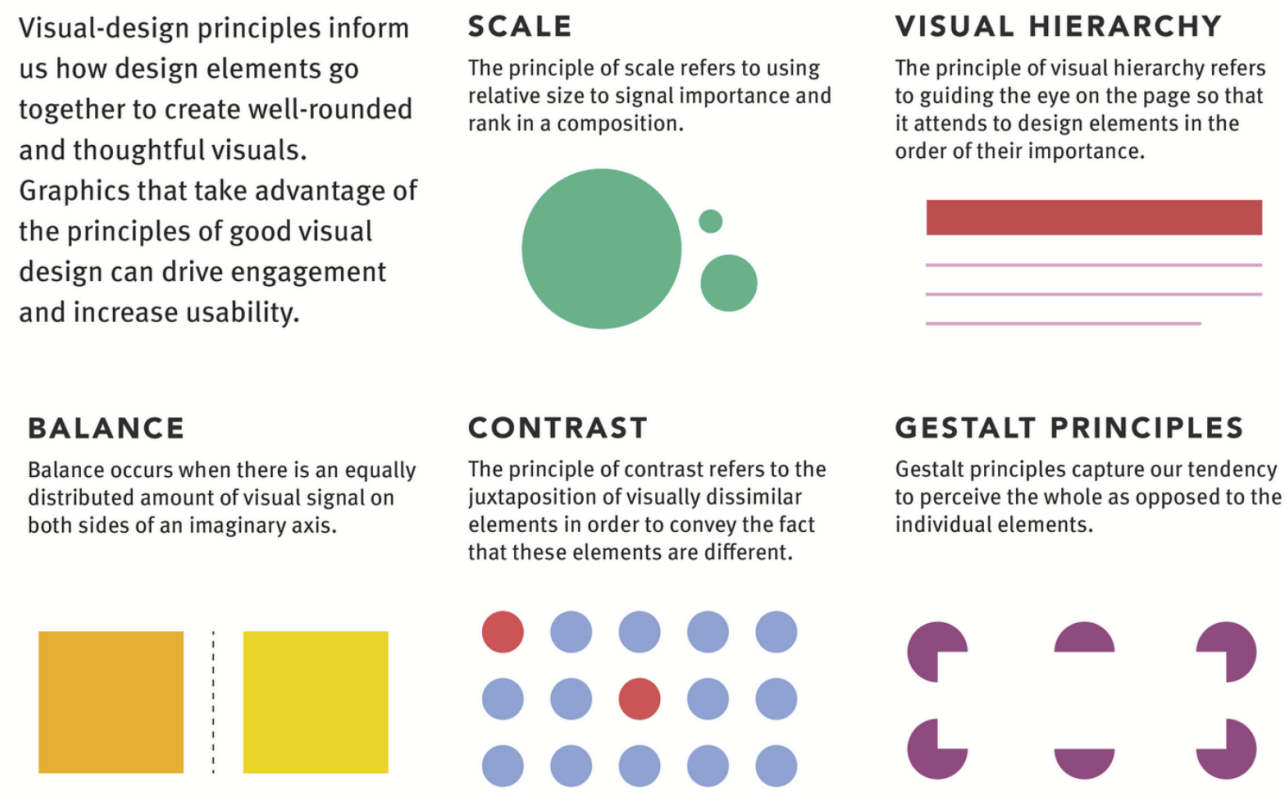
contribute to these main visual-design principles:

- **Scale**
- **Visual hierarchy**
- **Balance**:
  - Symmetrical: elements are symmetrically distributed relative to the central imaginary axis
  - Asymmetrical: elements are asymmetrically distributed relative to the central axis
  - Radial: elements radiate out from a central, common point in a circular direction.
- **Contrast**
  - Dominance
- **Gestalt**: Gestalt principles explain how humans simplify and organize complex images that consist of many elements, by subconsciously arranging the parts into an organized system that creates a whole, rather than interpreting them as a series of disparate elements. Proximity is probably one of the most

important Gestalt principles and is especially important for UX. It refers to the fact that items that are visually closer together are perceived as part of the same “group”. Also proximity, together with other features, can enforce “unity” the global sense of all elements being “a single thing”.

Google is a champion in using contrast to create dominance:





Good website I found:

https://www.warhol.org/exhibitions/

https://www.awwwards.com/

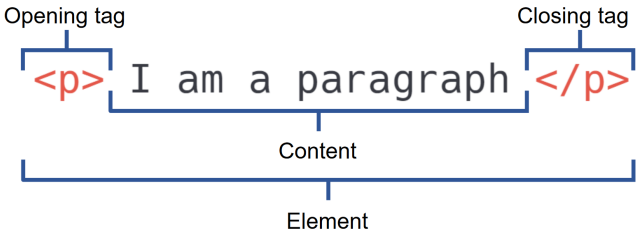
https://www.ideo.com/blog

https://medium.com/

https://www.ableton.com/en/

4 HTML

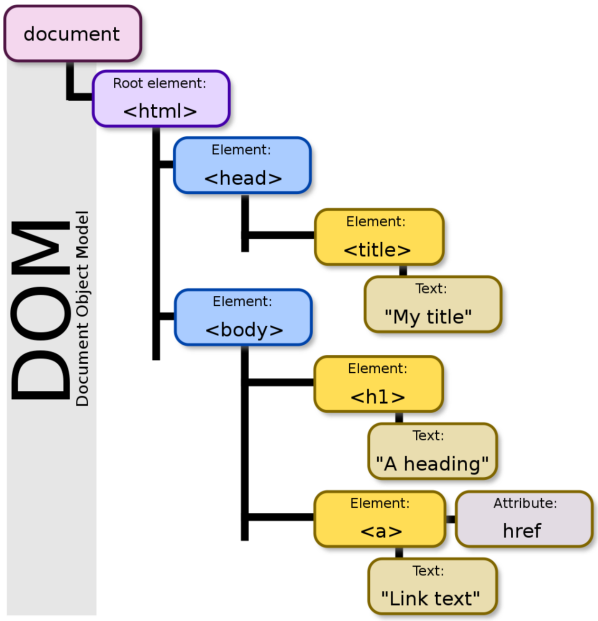
- HTML is a markup language that tells the browser how to structure a webpage:
  - immediate effect of changes
  - the browser won't tell you if there are any errors.
- any HTML page is composed of multiple HTML elements, which can be nested as long as the proper rules of opening and closing **tags** are followed.



Tag	What It Is
`<p>`	Paragraph
`<h1>`	Heading 1
`<code>`	code section
`<img>`	Image

4.1 DOM

The DOM is a programming interface for web documents that represents an HTML page as a logical tree. Each element defined in the HTML page becomes a DOM node in the tree structure, and developers can access and manipulate the content and structure of the web page through JavaScript or other programming languages. In addition to elements, the DOM also provides access to other important properties of a web page, such as the document title, metadata, and scripts. Understanding the structure and properties of the DOM is crucial for creating dynamic and responsive web pages.

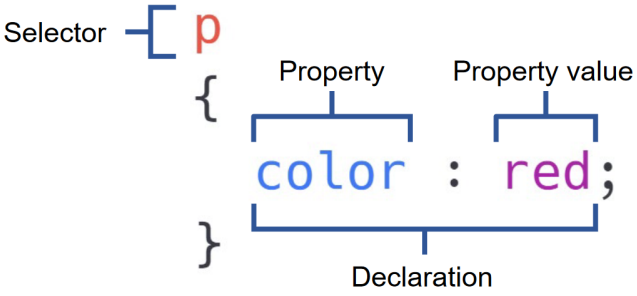


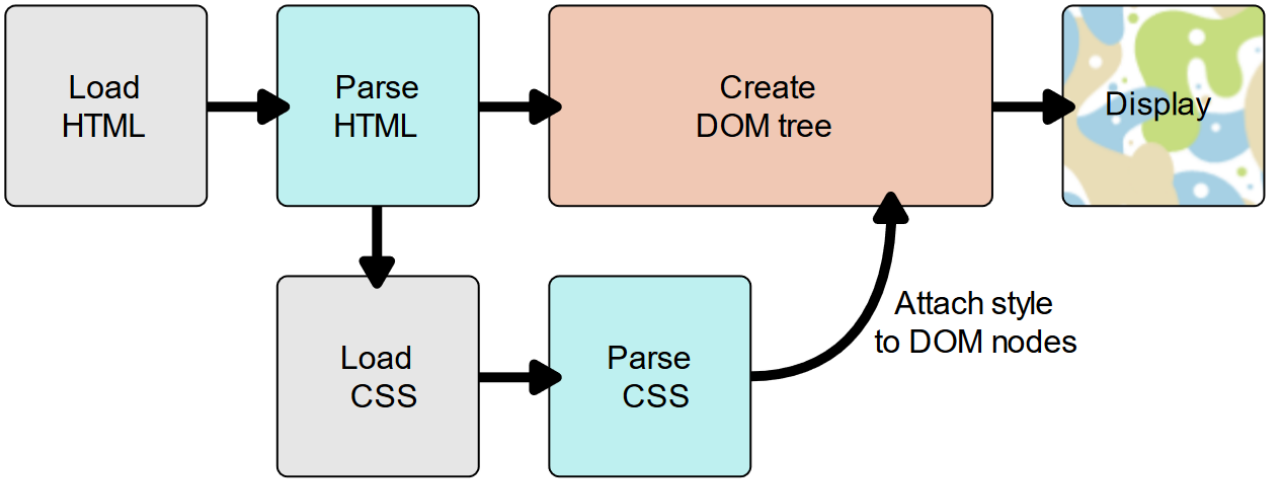
5 CSS

A website can have obviously multiple CSS associated with.

CSS (Cascading Style Sheets) is a language used to change the appearance of web pages. It is a rule-based language where each rule defines what changes need to be applied and the target element(s) in the web page. By

using CSS, developers can create visually appealing and consistent web pages by controlling the colors, fonts, layout, and other style properties. CSS allows for a separation of style and content, making it easier to maintain and update web pages over time. Overall, understanding CSS is an important aspect of web development and can greatly enhance the user experience of a website.





{width=50%}

When working with CSS, there are several types of selectors that can be used to target specific elements on a web page.

A type selector targets all instances of a specific HTML element type, such as `<p>` or `<h1>`. For example, the CSS rule "p { color: blue; }" would apply the color blue to all paragraphs on the web page.

A class selector targets all elements that have a specific class attribute. Classes are useful for applying the same style to multiple elements on a web page. For example, the CSS rule ".highlight { background-color: yellow; }" would apply a yellow background color to all elements with the class "highlight".

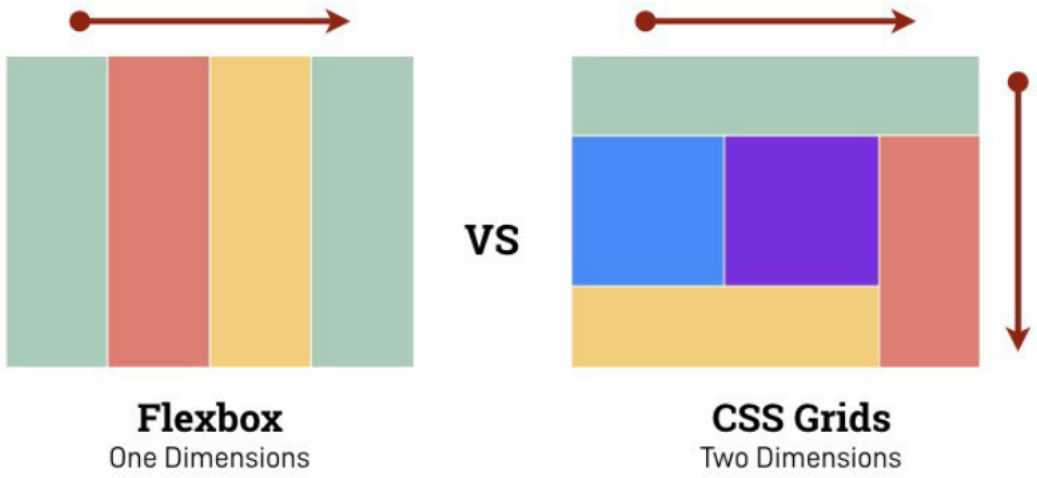
An ID selector targets a single element with a specific ID attribute. IDs should be unique on a web page, so this selector is useful for targeting a specific element that you want to apply a specific style to. For example, the CSS rule "#header { font-size: 24px; }" would apply a font size of 24 pixels to the element with the ID "header".

- Type selector: `elementName`
- Class selector `.className`
- ID selector `#idName`

There are situations where different rules can be applied to the same element.

CSS has two rules to resolve this situation:
- Cascade
- Specificity - !important keyword

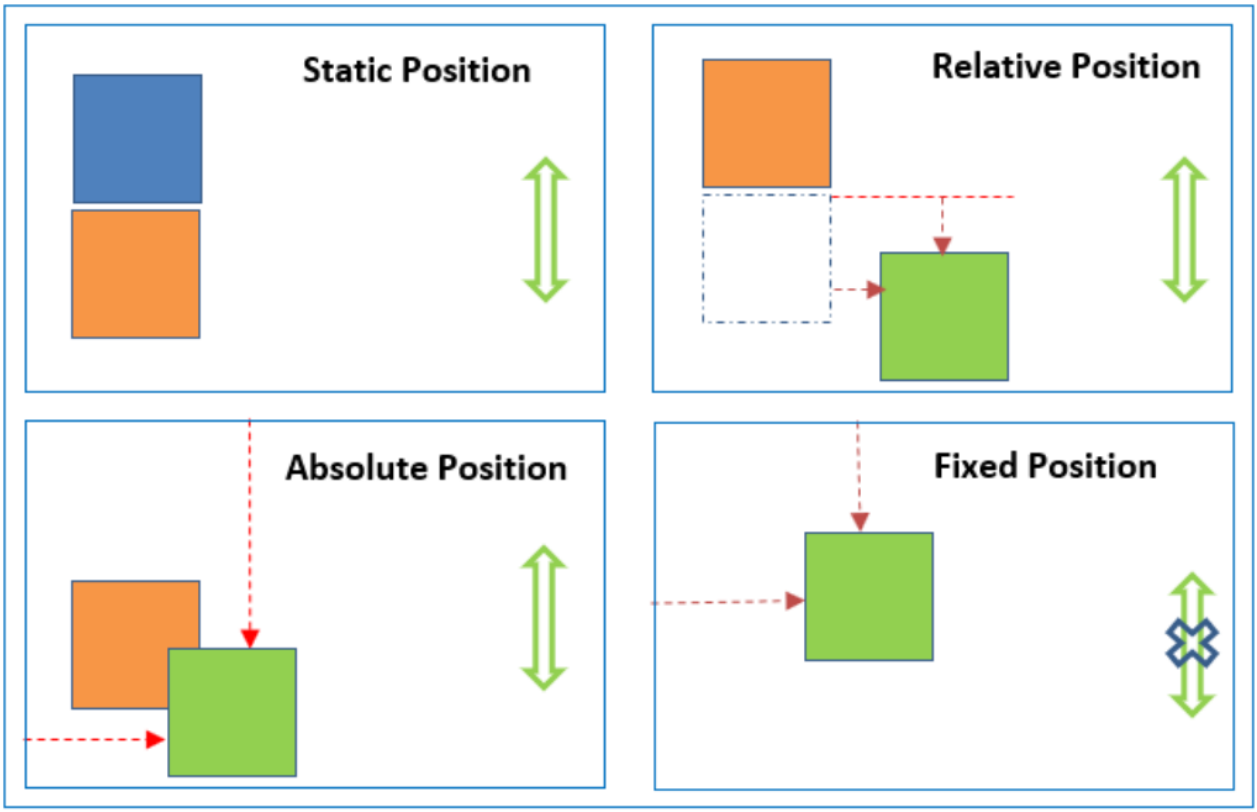
```
main  {
    width: 100%;
    display: flex;
    flex-direction: ro
    align-items: center;
}
```



ABSOLUTE		RELATIVE	
Pixels	px	Percentage	%
Centimetres	cm	Font-sizes	em - rem
Millimetres	mm	Character-sizes	ex - ch
Inches	in	Viewport Dimensions	vw-vh
Points	pt	Viewport Max	vmax
Picas	pc	Viewport Min	vmin

5.1 Responsive Design

Responsive web design is essential and involves designing web pages that can react and respond to different device screen sizes. In CSS, viewport sizing enables using device viewport dimensions to position and size elements on the page. Grid/flex layout is perfect for creating responsive. Also using percentages allow for proportional sizing of elements based on available screen space.



6 Javascript

JavaScript is a popular **scripting language** (scripting language is a programming language that is interpreted, which means that there isn't any compiler) used in web development to make web pages interactive and dynamic. It is a cross-platform, object-oriented language and it can be used to manipulate the DOM of a web page: show or hide elements on a page, create animations, update the content of a web page without requiring a full page reload.

```
<html>
    <head>
        <title>Test page </title>
        <script>
            // inline or saying: src = "script.js"
        </script>
    </head>
    <body>
    </body>
</html>
```

JavaScript has three kinds of variable declarations:

- `var` : declares a variable.
  - `Boolean`: true and false.
  - `Number`: an integer or floating-point number.
  - `BigInt`: an integer with arbitrary precision.

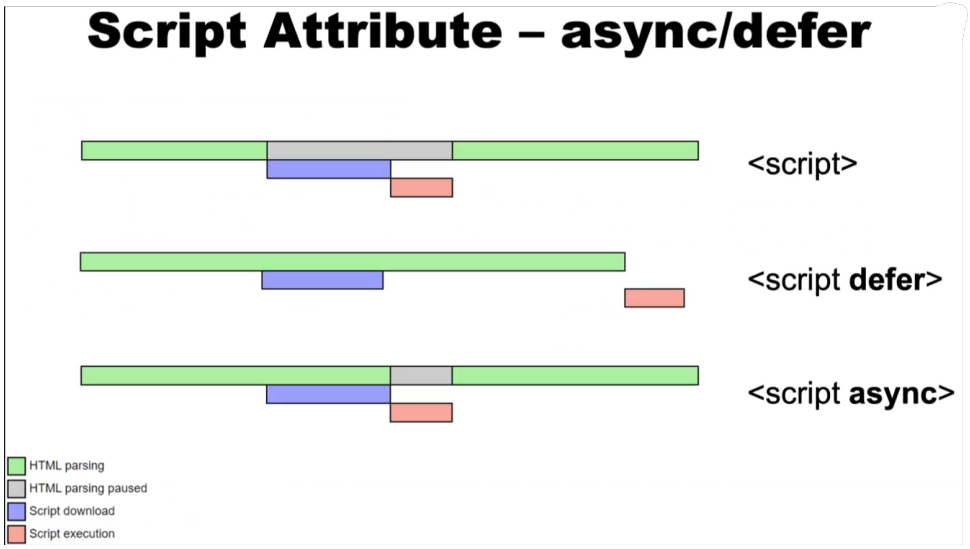
  - `String`: a sequence of characters that represent a text value.
  - `Symbol`: a data type whose instances are unique and immutable.
  - `undefined`: a top-level property whose value is not defined.
  - `null`: a special keyword denoting a null value. (Because JavaScript is case-sensitive, `null` is not the same as `Null`, `NULL`, or any other variant.)
  - `Object`: a complex data type that allows you to store collections of data.
- `let` : declares a block-scoped, local variable.
- `const` : declares a block-scoped, read-only named constant that must be initialized.

```
let myString = "string using quotation mark";
myString = 'string using apostrophe' ;
myString = "mixing the 'two"';
myString = 'now the "opposite"' ;

let stringOne = "first half, ";
let stringTwo = "second half";


let newString  = stringOne + secondHalf;
newstring = "Sentence:" + newstring;


let valueVar = 5
let test = 'The value is ${valueVar}'

let arrayOfArray = [1,2,3,[1,2,3],"42"];
let sequence = [1,2,3];
console.log(sequence[0]) //1
sequence[0] = 42; // [2,3]

// String methods
const myString = "Hello, World!";
const myStringArray = myString.split(" "); // Split string into an array by delimiter
console.log(myStringArray); // Output: ["Hello,", "World!"]
console.log(myString.toLowerCase()); // Output: "hello, world!"
console.log(myString.toUpperCase()); // Output: "HELLO, WORLD!"
console.log(myString.length); // Output: 13

// Array methods
const myArray = [1, 2, 3];
console.log(myArray.length); // Output: 3
myArray.push(4); // Add element to the end of the array
console.log(myArray); // Output: [1, 2, 3, 4]
myArray.pop(); // Remove last element from the array
console.log(myArray); // Output: [1, 2, 3]
const myArrayString = myArray.join(" - "); // Join array elements into a string with separator
console.log(myArrayString); // Output: "1 - 2 - 3"

// do-while statement
let i = 1;
do {
  console.log(i);
```

```
  i++;
} while (i <= 5);

// while statement
let j = 1;
while (j <= 5) {
  console.log(j);
  j++;
}

// if statement
const x = 10;
if (x > 5) {
  console.log("x is greater than 5");
} else {
  console.log("x is less than or equal to 5");
}

// for statement
for (let k = 0; k < 5; k++) {
  console.log(k);
}
```

OBJECTS != CLASSES

```
const person = {
    firstName = "Martino",
    lastName = "Piaggi",
    age: 22,

    fullName: function(){
        return this.firstName + " " + lastName;
    }

    increseAge: function(){
        this.age++;
    }
}
```



Scripts loaded using the defer attribute will run in the order they appear in the page and execute them as soon as the script **and content** are downloaded. (order matters)

Scripts loaded using the **async** attribute will download the script **without blocking rendering** the page and will execute it as soon as the script finishes downloading.

6.1 Event handlers

```
const btn = document.getElementById('my-button');
btn.addEventListener('click', myFunction);
function myFunction() {
  console.log('Button clicked!');
}
```

```
<div class="myClass">This is my div.</div>
```

```
const myClass = document.querySelector('.myClass');

// Add a mouseover event listener to the element
myClass.addEventListener('mouseover', function() {
  console.log('Mouse over!');
});

// Add a click event listener to the element
myClass.addEventListener('click', function() {
  console.log('Clicked!');
});
```

6.2 Classes

```
class Rectangle{
    constructor(height,width)
    {
        this.height = height;
        this.width = width;
```

```
    }
}
const r = new Rectangle(10,4);
```

6.3 Promise

Promise -> asynchronous task

In JavaScript, a Promise is an object that represents the eventual completion or failure of an asynchronous operation and allows you to attach callbacks to handle the result when it is available.

During its execution, a Promise can be in one of these states:
- pending: initial state, neither fulfilled nor rejected
- fulfilled: the operation was completed successfully
- rejected: the operation failed successfully

Promises are often used in conjunction with async/await to write asynchronous code in a more synchronous style.

6.4 Async

The keyword async transforms any function from synchronous to asynchronous. When an async function is called, a Promise object is returned to which it is possible to do chaining

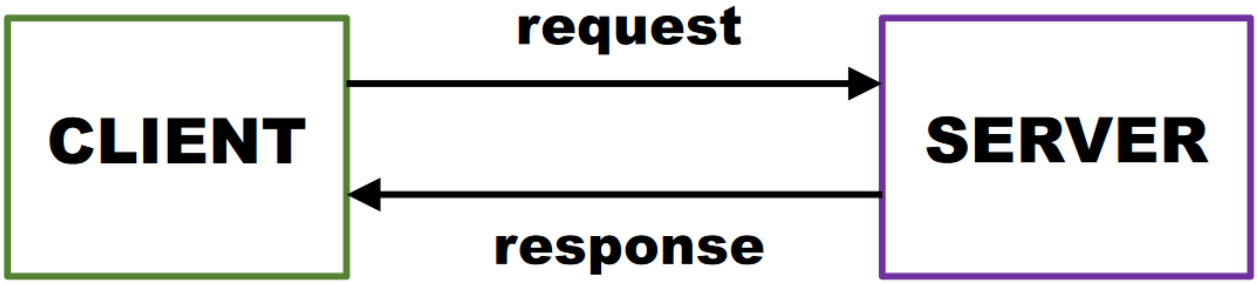
```
async function doAsync(){
//stuff
}

doAsync().then(onResolve,onReject);
```

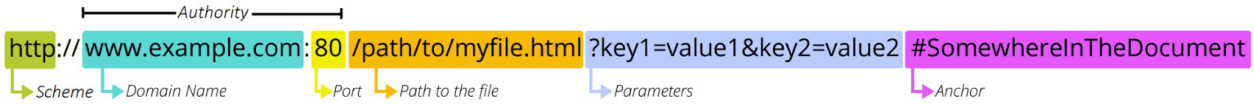
The await operator is used before calling an async function or creating a new Promise object. This allows to wait for the return value of a Promise instead of receiving the Promise object.

`resolve()` produces a value after an asynchronous (aka, async) operation completes successfully, or an error if it does not complete successfully due to time out, network error, and so on.

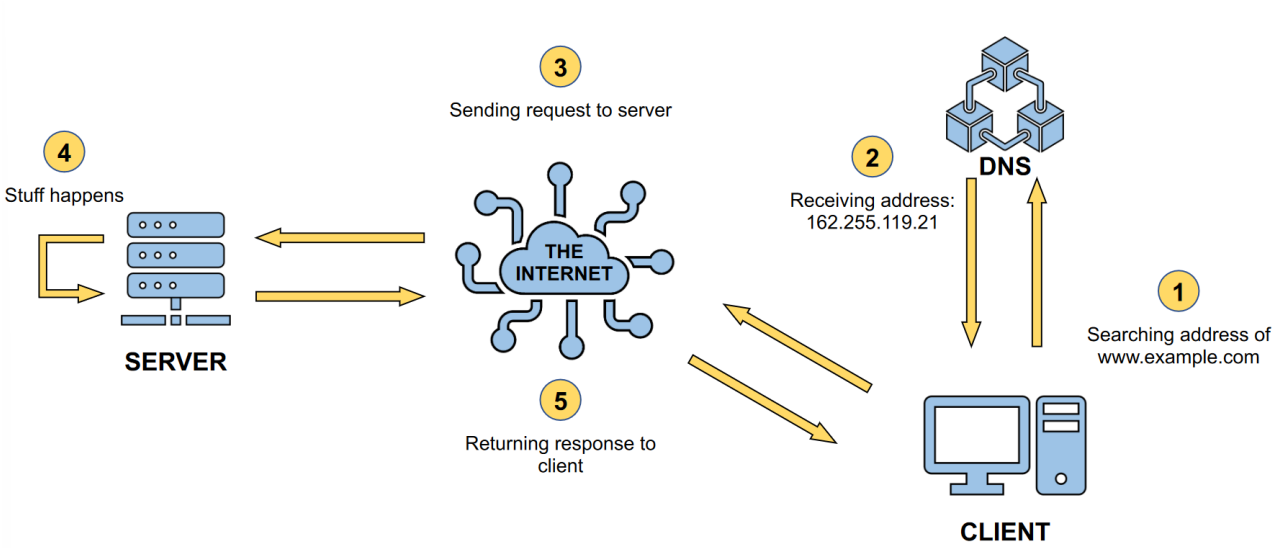
```
// Create a function that returns a Promise that resolves after a delay
function delay(time) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve();
    }, time);
  });
}

// Use async/await to delay for 1 second and log a message to the console
async function main() {
  console.log('Starting...');
  await delay(1000);
  console.log('1 second has passed.');
}

// Call the main function
main();
```

Basic stuff:





| Field | Sample | Description | | :— | :— | :— | | Scheme | http | The protocol used. | | Domain name | https://martino.im | The location of the resource. | | Port | :80 | The “gate” used to access the resource. | | Path | //path/to/myfile.html | The path to the resource we are looking for. | | Parameters | ?key1=value1&key2=value2 | Parameters used to manage the operations for the resource. | | Anchor | #SomewhereInTheDocument | A bookmark to a location in the webpage. |



Possible answers:

Code	Description	Actual Meaning (Server's perspective)
2xx	The action was successfully received, understood and accepted.	Everything is okay!
3xx	The client must perform additional actions to fulfil the request.	I know what you want, but it is not there, anymore. Let me tell you the new location...
4xx	The request is syntactically incorrect or cannot be fulfilled	I have actually no idea of what you want.
5xx	The server has failed to fulfil an apparently valid request.	I just decided to blow up and stop, working.

The most commonly utilized web servers are:

- Apache
- NGINX
- Microsoft IIS
- LiteSpeed Web Server

6.5 Static files and dynamic files

Static files are content that is already generated and complete and, when served, doesn't require any additional information/data. HTML, JS, CSS, images are example of static files.

There are several reasons why **dynamic** files are crucial in web development:

- **Efficiency and Scalability:** Scale web applications as demand grows is easier since the server can generate pages and content as needed, rather than relying on static files that must be manually updated.
- **Customization and Interactivity:** generating pages on-the-fly, dynamic files make it possible to personalize web pages based on the individual user's preferences and behaviors. This is probably the most important feature of a dynamic website.

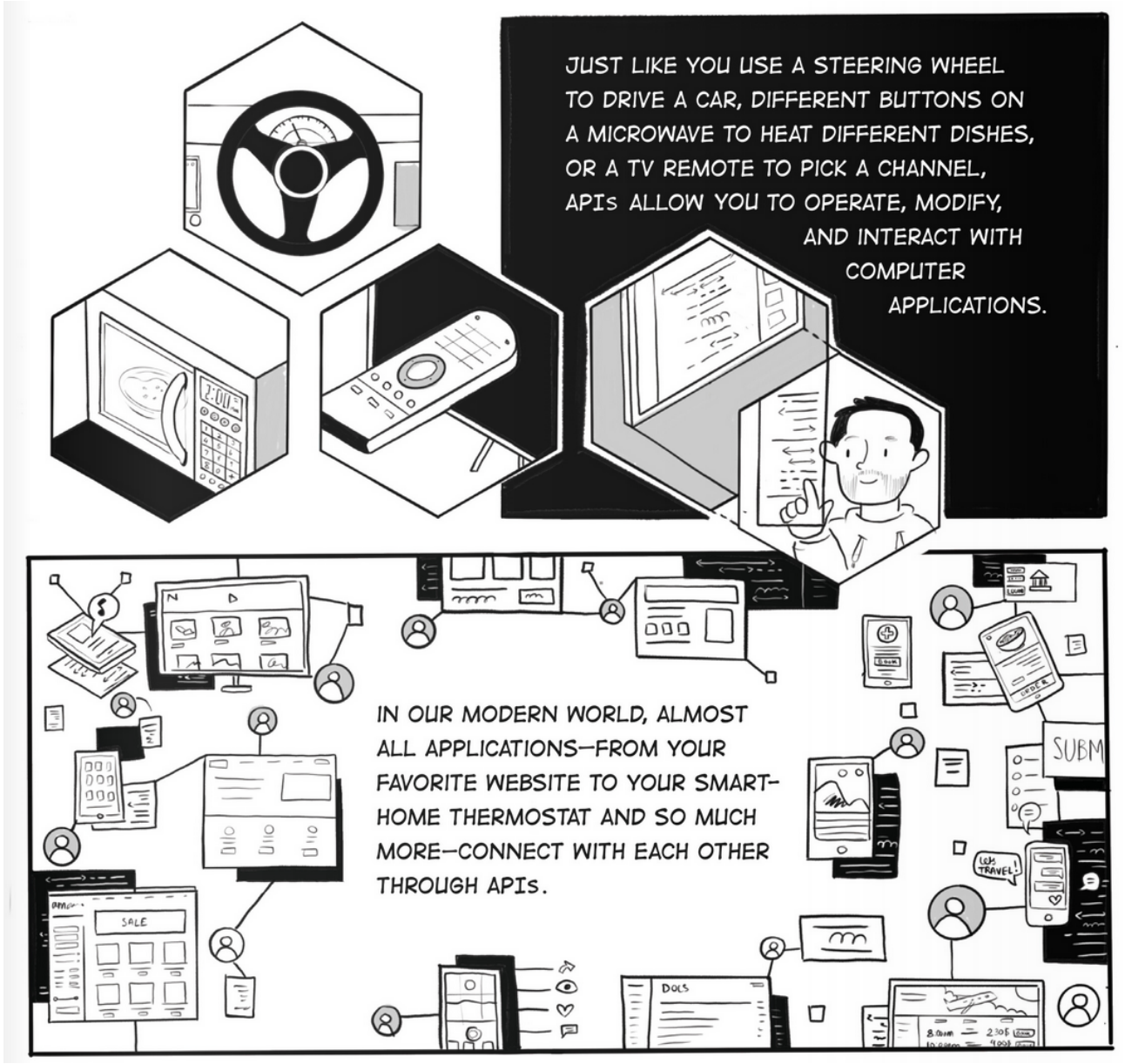
6.6 JSON

- JSON stands for JavaScript Object Notation.
- It is used as a data transfer convention over the internet.
- JSON is written in the format of a JavaScript Object/Array.

6.7 API and REST

APIs enable abstraction of the internal workings of a system, providing only the relevant parts to the programmer. An API is categorized as REST API if it adheres to the following principles:

- Client-Server paradigm
- Stateless
- Uniform interface
- Cacheability
- Layered system



NGINX and Apache are good for static file serving and load balancing. In 2009, JavaScript was recognized as an actual programming language and not just a browser tool.

6.8 NodeJS and modules

Node.js is an environment for running JavaScript code outside a web browser, using the V8 engine. Modules are officially used to package and import code inside other modules. All modules used are saved in the `node_modules` folder within the project. **Endpoints** are the main part of the web server, and each one corresponds to a functionality offered by the server.

It's very important to not upload `node_modules` folder since using `npm install` inside the folder any module is automatically fetched with `npm`.

```
import express from 'express';
const app = express();

//Each endpoint corresponds to a functionality that we offer

// Simple endpoint that send a message when called
    app.get('/', (req, res) => {
        res.send("Everything's OK!").status(200);
    })

// Endpoint that saves a value in the database
    app.post('/db', async (req, res) => {
        // The data is stored in the body of the request
        let data = req.body;

        // try/catch is a construct that allows to try to do something.
        // If it fails, the operation inside the catch are executed
        try {
            // Trying to save data in the DB
            await model.DB.create(data);
            res.status(200).send()
        }
        catch {
            // Send back error in case something went wrong (for example, empty fields)
            res.status(400).send()
        }
    })

// The server listen port 3000
app.listen(3000);
```

Method	Function	Description
GET	`app.get(...)`	Requests that usually want to retrieve data.
POST	`app.post (...)`	Requests that want to create new data.
PUT	`app.put(...)`	Requests that want to modify existing data.
DELETE	`app.delete (...)`	Requests that want to delete existing data

Postman is a website that can be used for testing servers. It is basically used to simplify the creation of APIs.

6.9 Random notes

6.9.1 Nodejs Express

6.9.2 NodeJS project

When you land into a project where there is `package.json` you can install everything ... using `npm install` .

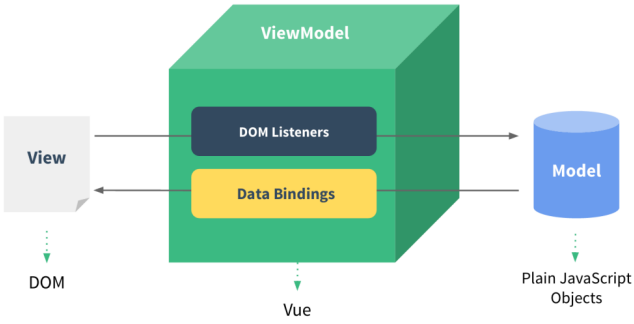
Remember to not commit `node_modules`.

`npm run build`

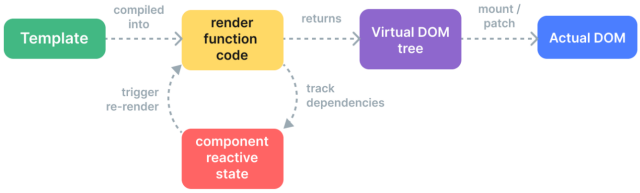
7 Vue.js

Vue.js is a JavaScript framework for creating interactive and reactive web frontends. It simplifies the process of rendering data to the DOM using a straightforward template syntax. Some features:

- **Declarative Rendering**: Vue extends HTML to describe output based on JS state using templates.
- **Reactivity**: Vue tracks state changes and updates the DOM efficiently.
- Based on **MVVM** pattern. While the controller is the entry point to the application in MVC, in MVVM the view is the entry point. MVVC code is event-driven in MVVM.



- Vue.js has a **virtual DOM** which solves bottleneck issue with searching and updating the "traditional" DOM in web apps. Virtual because Vue rapresents the DOM with JavaScript objects. This allows for programmatically creating, inspecting, and composing structures and leaves the direct DOM manipulation to the renderer.



- It can be used in a **widget approach** mode where Vue can control parts or entire HTML pages but also as a **Single page application (SPA)** where Vue controls the entire frontend (the server sends one file with all Vue code, and Vue takes over the UI).

7.1 Vue.js features

7.1.1 Text interpolation

The most basic form of data binding is text interpolation using the "Mustache" syntax (double curly braces):

```
<span>Message: {{ msg }}</span>
```

You can specify arbitrary data inside Javascript and then you can to refer the data inside html code . The data will fetch the value stored inside Javascript. If the data changes the view automatically displays the new value.

For example if in JS I write this:

```
const HelloVueApp = {
  data() {
    return {
      message: 'Hi'
    }
  }
}

Vue.createApp(HelloVueApp).mount('#hello-motherfuckers')
```

in html I can do this:

```
<div id="hello-motherfuckers" class="demo">
  {{ message }}
</div>
```

to see the `Hi` . Any changes to `message`will trigger reactivity and update the view accordingly.

7.1.2 Attribute interpolation

You can dinamically change some value coming from the backend, like for example an image.

7.1.3 Directives

VueJS has built-in directives such as `v-if`, `v-else`, `v-show`, `v-on`, `v-bind`, and `v-model`, which are used to reactively apply side effects to the DOM when the value of its expression changes. With the `v-if`, `v-else-if` and `v-else` directives it's possible to render a block based on the condition in the directive's expression (conditional rendering) directly inside the html page.

Using `v-for` directive

`v-model`

`v-if`

7.1.4 Event Listener

To listen to DOM events and perform certain actions upon triggering, we can use the `v-on` directive, which can be shortened to `@`. For instance, we can use `v-on:click` or `@click` to trigger an action when a click event occurs.

7.1.5 Computed vs Method properties

For any complex logic that includes reactive data, it is better to use a computed property or a method. Methods and computed properties achieve the same result, but computed properties are cached based on their reactive dependencies: it only re-evaluates when some of its reactive dependencies have changed. This means that multiple access to a computed property with no changed reactive dependencies, will immediately return the previously computed result without having to run the getter function again.

7.1.6 Vue component

Vue components are reusable instances that can be easily called by writing the name assigned to them. Components are elements that can be used in different places and pages inside the project. To manage the increasing size of JS code while defining new components, we can split components **over multiple files**. This makes it easier to locate things and manage the code.

7.1.6.1 Layout Since multiple pages/components can share the same layout, the **layout** can be seen as a **component wrapper** for all pages. This is an easy way to **centralize** all the **styles** that are commonly used by all pages but also to create easily different layouts for different screen resolutions.

```
<!--
    Default layout used by all the page
-->
<template>
    <head>
        <title>Celesta Capital</title>
```

```
    </head>
    <div>
        <TheHeader />
        <div class = 'page'>
            <slot />
        </div>
        <TheFooter />
    </div>
</template>
```

7.1.6.2 Passing data between components

Communication between a component and its parent is achieved via two tools:

• `props`, which involves passing data to the component. • `emit`, which calls an event to trigger a function in the parent component.

7.1.6.2.1 Props

If in my js script section of the component I write this:

```
export default{
    props: {
        text: String,
        counter: Number
    }
}
```

I cant later **pass data** to the component in this way:

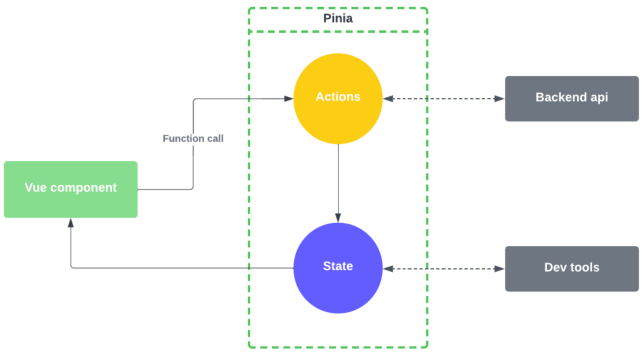
```
<html>
...
<MyComponentName :text="Passed String" />
...
</html>
```

7.1.6.2.2 Emit

TODO

7.2 Pinia

Pinia is a state management pattern + library for Vue to handle a centralized store for all the components in an application. It models a global entity managing a global state.



7.3 Nuxt

Nuxt is a framework over a framework which provides a lot of features. One of them is the file-system routing which basically it's an automatic routing for every page:

- `/pages/index.vue` → `localhost:3000`
- `/pages/contacts.vue` → `localhost:3000/contacts`
- `/pages/folder/index.vue` → `localhost:3000/folder`
- `/pages/folder/page.vue` → `localhost:3000/folder/page`

Nuxt provides `<NuxtLink>` component which is a drop-in **replacement** for both Vue Router's `<RouterLink>` component and HTML's `<a>` tag. It intelligently determines whether the link is *internal* or *external* and renders it accordingly with available optimizations (prefetching, default attributes, etc.).

7.3.1 Single Page Application (SPA) and SSR

- SPA (Single-page application):
  - A web app that loads only one web document and updates its content via JavaScript APIs
  - Saves time and bandwidth as only necessary content is loaded
- Server-Side applications
- Client-side applications

8 study random

https://vuejs.org/style-guide/

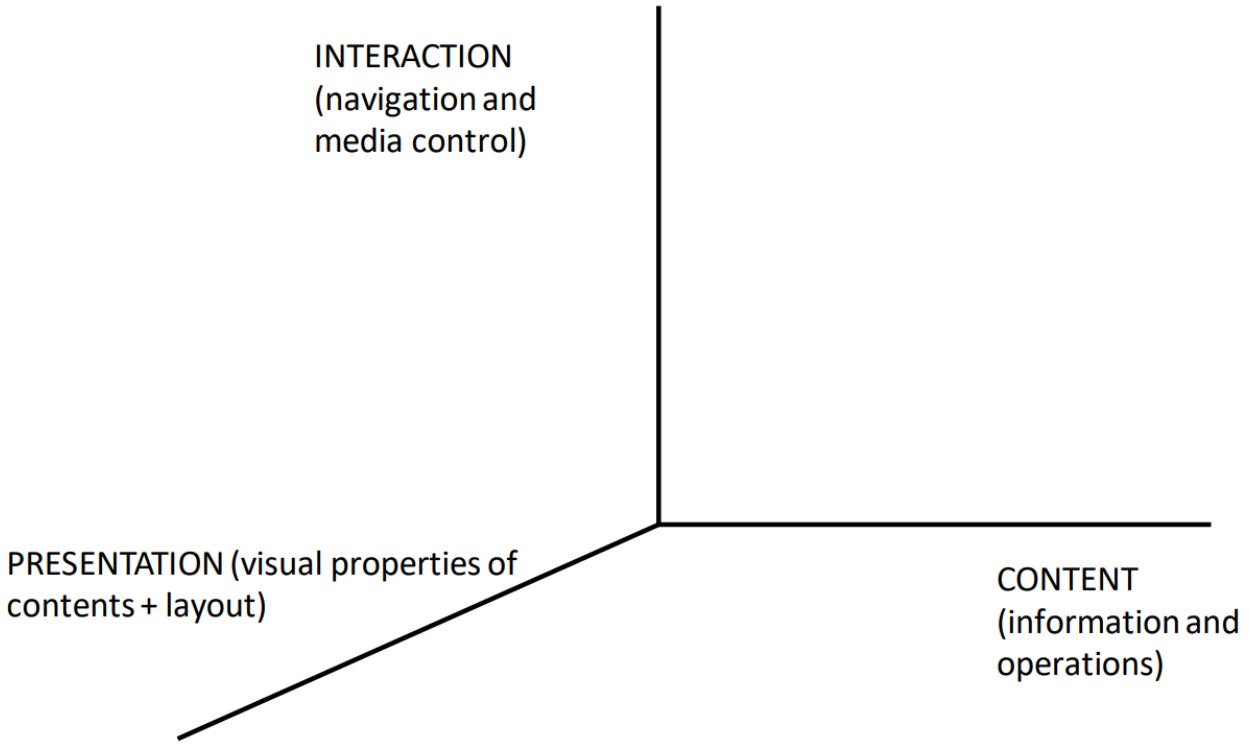
inline fold extension to minimize

9 Web Design

9.1 Prologue

As in many different Computer Science problems, a good way to approach the complexity of web designing is using a "divide et impera" approach: separate the different design concerns.

The 3 dimensions are always the same:



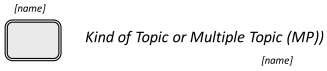
The final pages integrate content, interaction, and presentation elements BUT the design PROCESS must be kept separate. "you design once for many things" : you have to design the general structure obv, not every single page.

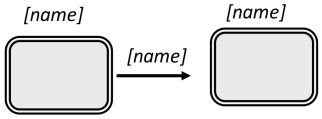
9.2 IDM

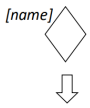
We also need a model to describe to support a systematic mapping to navigation and presentation design. A possible model is **IDM: Interactive Dialogue Model**. According to IDM, designing a web application involves designing the dialogue between humans and the application. IDM focuses on the in-the-large features only and is composed of sub-models for the various design dimensions. It doesn't include modeling how to interact with animations, videos, and other multimedia content, nor does it describe the detailed layout properties of pages.

9.2.1 Content IDM

- **Topic**: something unique that can be the subject of conversation between the user and the interactive application 

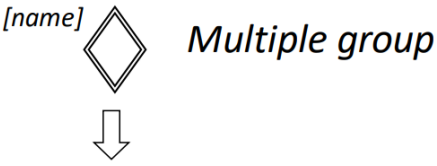
- **Multiple Topic** or **Kind of Topics**: a category (a "class") of possible subjects of conversation 

- **Relevant Relation**: It models how topics and instances of kind of topics are related. It determines how the dialogue can switch from a topic to a different one. 

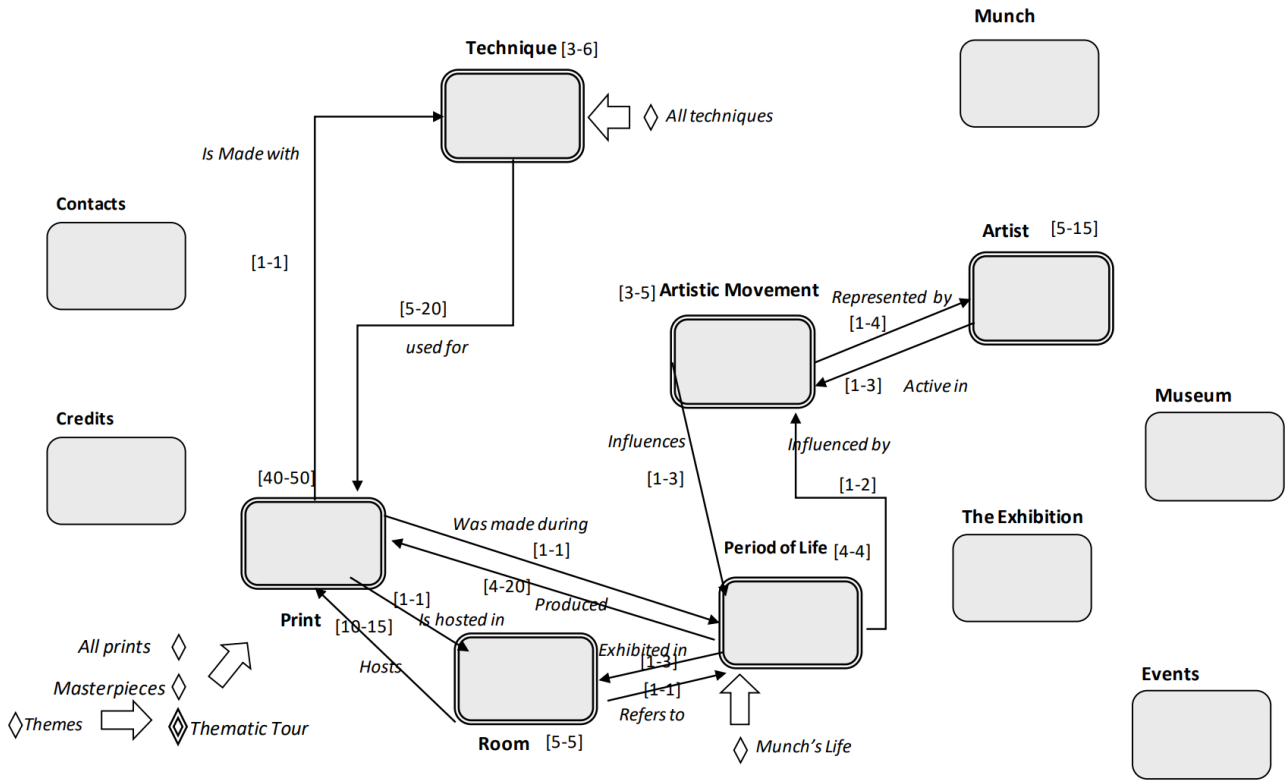


Group of Topics (GOT): the arrow points to the kind topics that are grouped

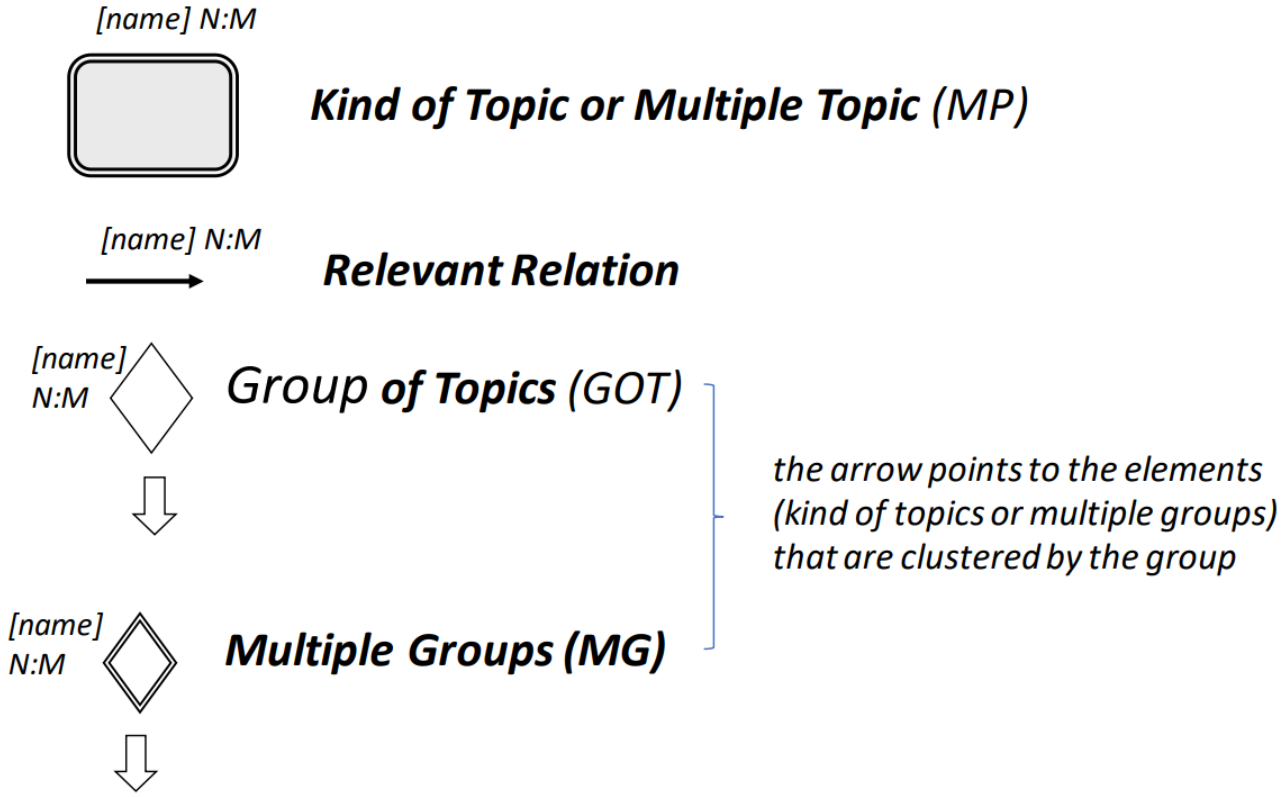
- **Group of topic**: a set of possible topics (subjects) of conversation



- **Multiple Groups**: It models a set of groups having common characteristics



9.2.1.1 Cardinality Cardinality refers to the expected minimum and maximum number of instances and should be associated with multiple topics, relevant relationships, and multiple groups. It helps in planning the overall size of the application and estimating editorial effort.



9.2.1.2 C-IDM in the small In the small the C-IDM notation is based on content tables. It describes the sets of data associated to each topic, kind of topic, group, or multiple group. Content table includes:

- Data for describing topic/kind of topic
- Data defining a preview of their relationships (for kinds of topics only)
- Data describing the whole group
- Data providing a preview of group members (for groups)

9.2.2 Navigation IDM

At the end a website consists of a network of pages which are atomic units for visualization and interaction and containers for contents and links. We can eventually classify pages in these categories:

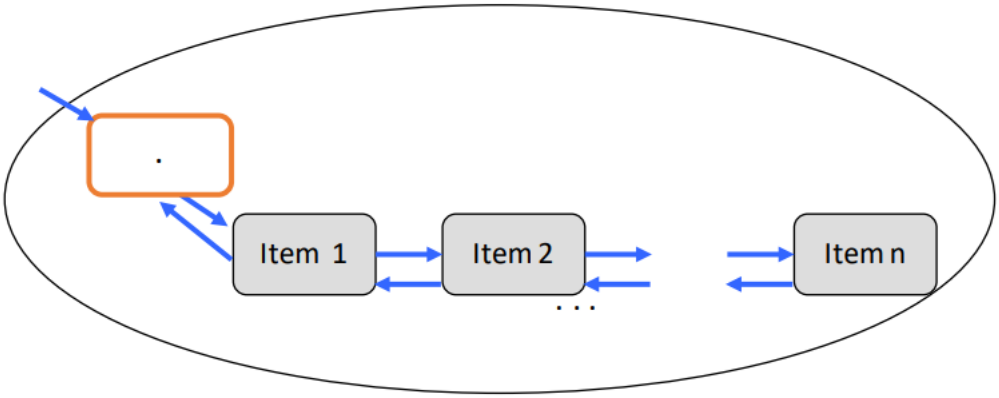
- **Single Topic Pages**: where users consume contents about a given single topic
- **Multiple Topic Pages**: e where users consume contents about topics of a given kind
- **Introductory pages**: where users understand what are a group is about, and what are its members.
- **Transition Pages**: where users see the list of topics related to a given page
- **Home page**: it eventually can have a totally different page and it's focused on brand values

We can classify links in these categories:

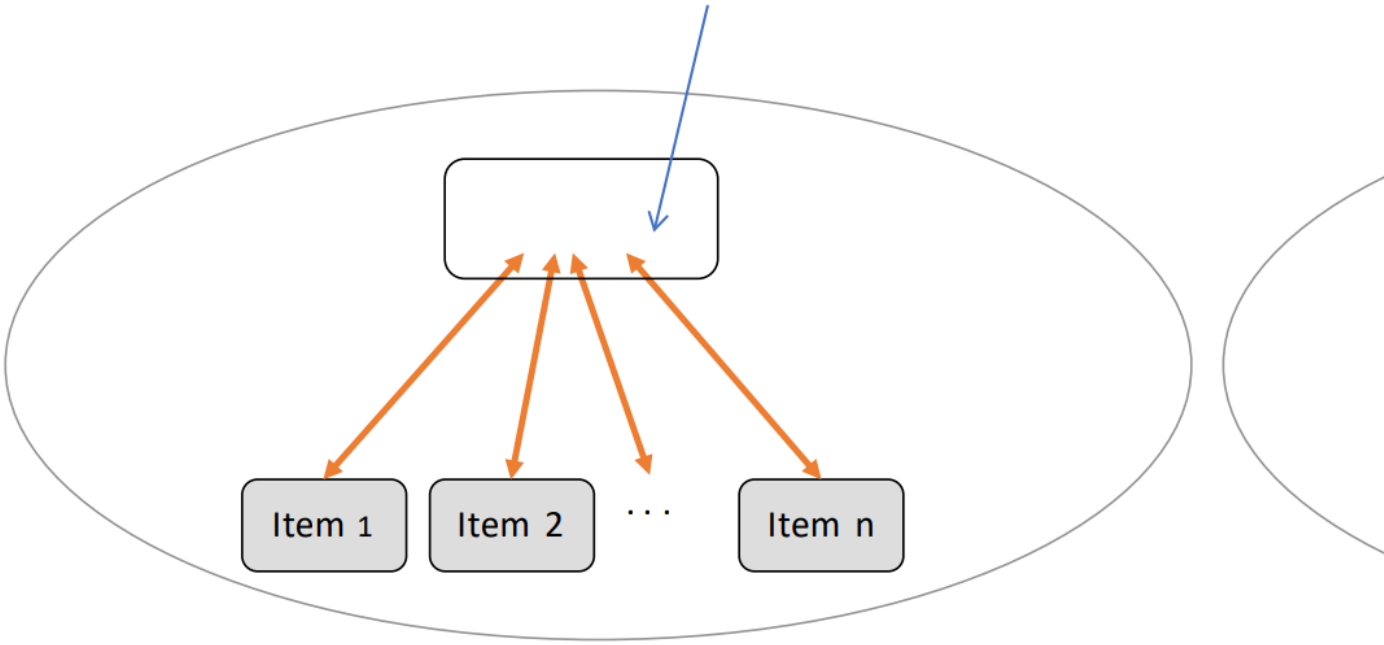
- **Structural Links**: to move across pages of the **same topic** and keep the user within the same "cognitive context". The user is exploring another piece of the same thing.
- **Transition Links**: to move the user from a topic to a different topic ("cognitive jump").
- **Group Links**: to support navigation across Introductory pages of a group to the pages of the group members. Can be static or eventually dynamic when the same page belongs to multiple groups and there is some logic behind the "destination of the link".
- **Landmarks**: links available in all pages.

9.2.2.1 Navigation pattern Navigation strategies that have been proved effective and usable for navigation in large networks of interconnected contents. The three main ones are:

- **All-to-all** links
- **Guided tour**: Identify an order among the group members, and creating sequential bidirectional links among them. A **circular** guided tour is possible too.



- **Index**: The index pattern consists of defining bidirectional links (from the member to the entry point of the group collection) from the entry point of the group collection (the introductory page in IDM) to each member.



9.2.3 Presentation IDM

Presentation design consists in:

- Decide where to place contents and links
- Choose the orientation of info
- Determine where to place links
- Decide on link labels and icons

- Set layout properties (size, color, etc.)
- Organize visual space

every step following the usability principles

9.3 Scenarios

Scenarios are used to illustrate design decisions through a "story about use" of an application by a typical user or persona. Basically is the same concept of use cases/ user stories or storyboards. Scenarios are useful throughout the whole product lifecycle, from requirements elicitation to evaluation of a prototype or implemented system. They can be described textually or visually, using images or videos. This is the structure of the scenario:

1. End User Profile: description of users who will use the system
2. Goal(s): Problems that the application solves
3. Context: situation of use of the application and user's environment
4. Task Textual and Visual Narrative: a narrative description of the interaction
    - Textual presentation of user's tasks
    - Screenshots, diagrams, and other graphical representations

10 Deployment

In this course we see how to **host** the website using Github pages and Vercel + Supabase.

10.1 Github pages

GH pages is a service offered by GitHub to host **static** website from a repository. The idea is to generate all the pages of the Nuxt project and then serve them by using GitHub Pages. GitHub Pages Docs

To add the gh-pages functionalities, you must follow these requirements:

- The repository needs to be public, unless you pay to keep it private.
- You must have a branch or folder in your current branch that contains the static HTML pages.

To generate static pages in Nuxt, use the "generate" command. This command pre-renders every route in your application and saves the result as plain HTML files. These files can be deployed on any static hosting service. The result can be found in the ".nuxt" folder, specifically in the "public" subfolder. Additionally a **dist** folder is created in the root folder. This is a shortcut to the public folder.

Install gh-pages as a developer dependencies of the project with `npm install gh-pages --save-dev`. Add the deploy script inside the package.json file of your project:

```
{
    // Other fields
    scripts: {
        // other scripts
        deploy: "gh-pages -d dist"
    }
    // Other fields
}
```

You can push the folder with all the static pages in the `gh-pages branch` of the repo with `npm run deploy`. We have to "fix" the correct path setting in nuxt config file (`nuxt.config.ts`):

```
export default defineNuxtConfig({
    //Other settings
    app: {
```

```
        baseURL: '/<name of the repository>/'
    }
})
```

To avoid that Jekyll see folder `_nuxt` as a special folder (in its syntax) we can add a `.nojekyll` file in the root of the project or chaning the name of the folder always in nuxt config file:

```
export default defineNuxtConfig({
    //Other settings
    app: {
        baseURL: '/<name of the repository>/'
        buildAssetsDir: "<new name for the folder>"
    }
})
```

10.2 Vercel and Supabase

GH Pages works only with static pages, so it's not ideal for displaying dynamic content. Instead, we can use hosting services such as **Vercel**, which offers a free plan that suits our needs. For managing a Postgres db, we can use **Supabase** as an alternative to **Firebase**. Vercel simply is **automagically** and just need to be connected to the GitHub repo. Supabase is great for its easy-to-use database. No SQL is necessary, and data can be imported from various sources like csv files, Excel, or Google Sheets. Connecting to the database only requires a few simple steps:

```
npm install @nuxtjs/supabase --save-dev
```

and add the module in the nuxt config file:

```
export default defineNuxtConfig({
    //other settings ..
    modules: ['@nuxtjs/supabase']
})
```

Create a `.env` file in the root folder of your project with these values. The `.env` is used to store credentials (remember to not upload them).

```
SUPABASE_URL="https://example.supabase.co"
SUPABASE-KEY="<your_key>"
```

and finally, install the Supabase integration from the Vercel Marketplace, add the integration to your Vercel project and set the environment variable for Supabase in your project settings.

11 SEO

Search Engine Optimization (**SEO**) is the process of making your website better for search engines (aka Google). Your website is given a score based on some known and some unknown parameters evaluated by the crawlers. The important thing for the crawler is that the page is complete and filled with the content the moment it's going to be analysed. Nuxt facilitates it by easily changing the rendering of the project. The head tag in HTML gives information and metadata to describe the content for crawlers. NuxtJS is a framework that manages a SPA. To distinguish pages, Nuxt offers two ways to add metadata: globally in the `nuxt.config.ts` file or locally in the script or template section of each page. For custom and reactive metadata, it's possible to set up the tag inside each page by using `useHead()` or `useSEOMeta()` in the script or template section. `useHead()` accepts an object similar to the one used for global metadata (title, meta...) while `useSEOMeta()` accepts an object where metadata is placed as a field of the object. Both `useHead()` and `useSEOMeta()` should work properly

when called inside the `asyncData()`. The alternative is to use some special tags directly inside the template section of the `SFC` file.

12 Web Accessibility

> "The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect." - Tim Berners-Lee, inventor of the WWW

The World Wide Web Consortium (W3C) and the Web Accessibility Initiative (WAI) developed technical specifications, guidelines, techniques and supporting resources that describe accessibility solutions. Overall, accessibility guidelines and techniques are based on four core principles:

- **Perceivable**: be perceivable to everyone, media should provide captions and content should be easy to see (and hear)
- **Operable**: users should easily navigate, find content, and determine where the are (even through keyboard and not the mouse for example)
- **Understandable**
- **Robust**: it means that the website supports and can be accessed on a variety of devices, including assistive technologies.

Some "techniques" to use to achieve accessibility:

- **Ranking of headers** is important since there are software that re-organize the appearence of the website using the headers (paragraphs, sub-paragraphs) you use
- **Alt text** are a lot important: it is a written description of an image that screen readers can read out loud. Note that there exists *decorative images* which are images that doesn't add information to what is already written. In this case alt text is not necessary
  - https://webaim.org/techniques/alttext/
  - https://supercooldesign.co.uk/blog/how-to-write-good-alt-text
  - https://www.w3.org/WAI/tutorials/images/
- **ARIA** (Accessible Rich Internet Applications) is a set of roles and attributes that define ways to make web content and web applications more accessible to people with disabilities. ARIA provides a framework for adding attributes to identify features for user interaction, how they relate to each other and their current state. For example a written description of a progress bar. From HTML5 some of the functionalities of ARIA are already implemented inside the semantic tags. "*No ARIA is better than bad ARIA*".