Photo Gallery

Why we used span rather than div in following part

<h1>

<span class="photo">Photo</span><span class="gallery-text">Gallery</span>

</h1>

We used <span> to style parts of the heading individually without breaking them into different lines. If we used <div> , If you used <div>, each word ("Photo" and "Gallery") would appear on separate lines

<form id="search-form">

<input type="text" placeholder="Search here" class="search-input">

<button class="search-button" type="submit">

<ion-icon name="search-outline"></ion-icon>

</button>

<div class="error-message">Please enter a valid item</div>

</form>

Why did we used type submit attribute for button ?

if a <button> is placed inside a <form> without a type, it defaults to type="submit" but we still explicitly used it because it makes code safe ,easier to understand ,more professional

Why did we use a <button>?

We used a <button> to trigger the form submission.

When a user types something and clicks the button even then browser submits the form not only when ENTER is pressed

Without a button, the user could only press Enter to submit input value. A button makes it more user-friendly

Why we enclosed icon within the button element ?

To make user interface more friendly and visually appealing, instead of a boring text now we have a search icon , when we click on it form will be submitted.

Why did we use an id for the form?

We used an id="search-form" so that we can easily find the form using JavaScript.

Without an id, it would be harder to select the specific form, especially if there are multiple forms on the page.

 <div class="loading-text">Loading...</div>

 <a class="load-more" data-img="curated">Load More</a>

</div>

Why we used anchor tag rather than a load more button ?

If we put a <button> outside a form, or if it's inside another form unintentionally, clicking it could submit the nearest form in the DOM.

This could cause unexpected form submission if you have multiple forms on the page.

But <a> elements don’t submit forms ever.

So using <a> avoids accidental form submissions automatically.

Why we used data-img="curated" attribute ?

It tells JavaScript what kind of images to load when you click the “Load More” button.

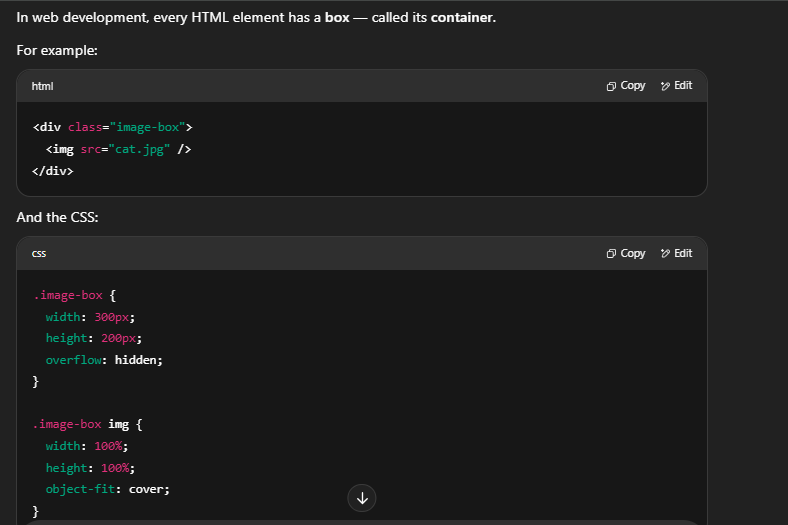
it means we get random images when we click on "Load More" by default. But if we have searched for a particular category of images, we get results based on that category.

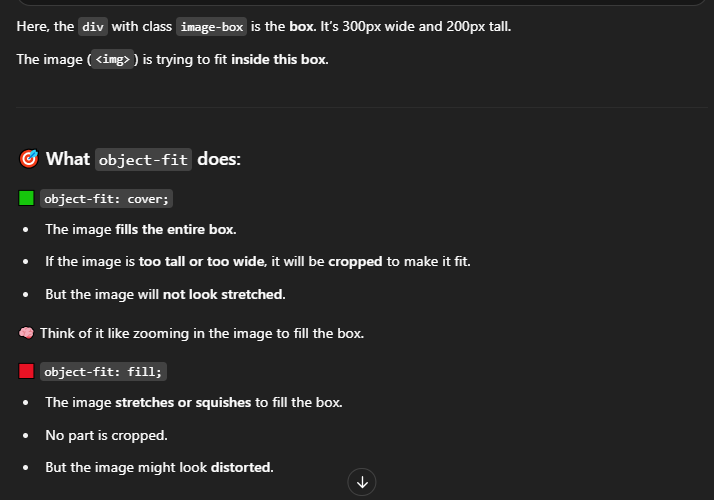
Now, if we want to load more images related to the search, the data-img attribute helps with that.

Without it, clicking "Load More" would load random (curated) images again instead of continuing with the searched category.

But when we use data-img="search" (using js), the app knows that the user is viewing searched results, so it loads more images from the same category instead of random ones.

Why we use object-fit property ?





transform: translateY(-30%);

It moves the element upward by 30% of its own height.

30% of the element’s own height (not the parent’s height).

grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));

repeat(auto-fit, ...)

This tells the browser to repeat as many columns as will fit in the container.

auto-fit automatically adjusts the number of columns based on the available space.

minmax(250px, 1fr)?

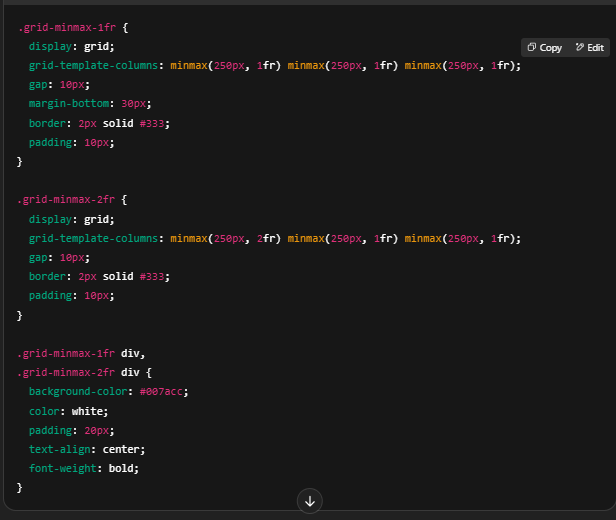
Each grid column must be at least 250 pixels wide (minimum),

but can grow bigger to fill extra space if available (maximum).

The maximum size is 1fr, which means 1 fraction of the free space. It divides remaining space evenly among all columns.

fr is a CSS Grid unit meaning "fraction" of the available free space.

So if you have leftover space, each column gets an equal share (1 fraction each).



In .grid-minmax-1fr, all three boxes have the same minimum width of 250px, and grow equally when there’s extra space.

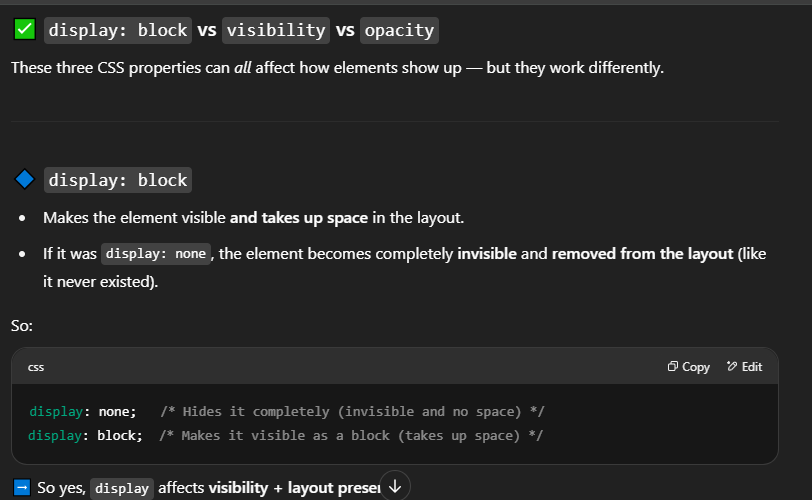
In .grid-minmax-2fr, the first box gets twice the extra space compared to the other two boxes, but all boxes stay at least 250px wide.

Why did we use display:block property in

.error-message.show {

  display: block;

}



javaScript

Why did we enclose whole code in a class ?

To nclose all related data and functions in one place

→ This keeps the code clean, organized, and reusable.

To easily create multiple instances in the future

→ For example, we can create other galleries using different APIs, pages, or settings by just changing the values(e.g API key).

const gallery1 = new PhotoGallery(); // Uses Pexels API

const gallery2 = new PhotoGallery('AnotherKey'); // Maybe for a different API or section

What happens if we omit this line

const gallery = new PhotoGallery();

This line does three important things:

Creates an object from the PhotoGallery class.

Runs the constructor() method automatically.

The constructor() calls this.eventHandle() — which adds all the event listeners (like fetching images, handling search, loading more).

If we omit it Nothing happens when the page loads, or when you search or click "load more"

Why did we enclose just a part of code in a constructor while the rest is outside it?

Why can't we use ” this” written explicitly directly in the class body outside constructor or any function?

Because the class body runs before any object is created.

So JavaScript doesn't let us use this directly in class body cause it will give error as object does not exist yet to which this refers to

Object is created when we execute following statement

const obj = new MyClass();

So, until new MyClass() is called, there's no object — and no this to refer to.

Where can we use this?

You can only use this inside a method, like:

constructor()

Other class methods like showName()

Why can we use this inside methods like constructor() or sayHi(), even though the object isn't created yet?

Technically, the object is created right before the constructor() runs.

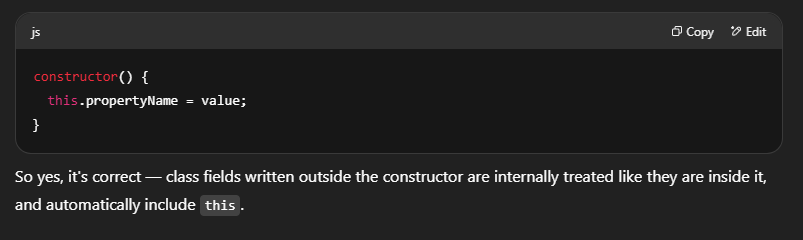
That’s why you can safely use this inside the constructor() and all class methods — because by the time those run, the object already exists and this points to it.

this.property = "value" is a statement, and only method definitions are allowed directly in the class body — not statements.

We can only write methods (functions) definitions outside the constructor, not regular code like variable assignments using “this”.

Can we omit “this” keyword and constructor ?

Even if we remove the this keyword from all these properties and delete the constructor() function, we can write the properties directly in the class body. JavaScript will implicitly treat them as if they were written inside a constructor with “this”. In other words, these properties will still belong to the object created by the class, because JavaScript automatically handles them as:



But in this case, the eventHandle() function will not run automatically, because you removed the constructor where it was called.

you'll need to call it manually after creating the object.

const gallery = new PhotoGallery();

gallery.eventHandle(); // ✅ call the method manually here

If you want it to happen automatically:

You should keep the method call only inside the constructor

If we had used let or var before these variables (this.logo , this.searchForm etc), would it have caused an error, or would they not have been considered properties of the class?

Using let or var instead of this. would cause the variables to become local to the constructor function and not properties of the class instance.

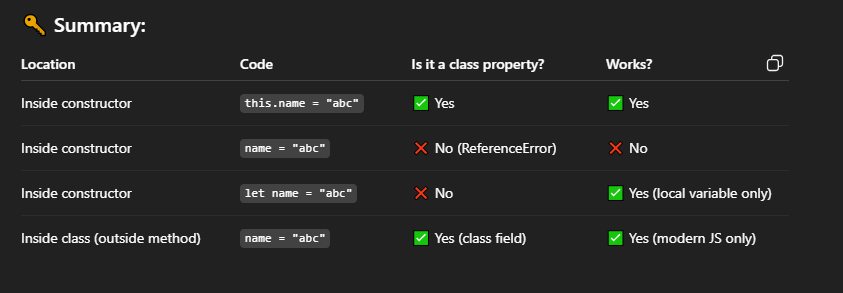
let API\_KEY = '...';

API\_KEY is just a local variable inside the constructor method.

Other class methods like fetchImages() cannot access it using this.API\_KEY — and will throw an error or return undefined.

this.API\_KEY = '...';

this.API\_KEY becomes a property of the class instance, accessible anywhere in the class using this.API\_KEY.



Why we do not enclose method’s definitions in constructor as well ?

best practice is

Properties / variables? → In constructor with this.property

Functions / methods? → Define outside constructor in the class body

we can enclose function definitions in the constructor,but it is not a good practice unless you have a specific reason.

1. You are allowed to define methods (functions) directly in the class
2. Keeps the code clean and organized
3. Reuses the same function across all instances (better performance)

Every time you do:

const obj1 = new MyClass();

const obj2 = new MyClass();

It creates a new copy of the functions defined in constructor for each object.

Drawbacks of writing methods outside constructor

Memory waste if many objects are created.

Each object gets its own separate function, even though it does the same thing.

Benefit of writing methods outside constructor

Methods are stored once in memory.

All objects of the class share the same function.

We can choose any name for the properties like (API\_KEY & other properties names )

What is ( this.pageIndex = 1 ) property referring to , why we are using it ?

It is creating a property called pageIndex inside the class and setting its value to 1.

When the page loads for the first time

this.pageIndex = 1; → It loads images from page 1 of pexel website

When you click "Load More":

let index = ++this.pageIndex;

this.pageIndex becomes 2, then 3, then 4, etc.

So the next API call becomes:

https://api.pexels.com/v1/curated?page=2&per\_page=12

https://api.pexels.com/v1/curated?page=3&per\_page=12

This way, you always get new images from the next page.

we didn’t use this.pageIndex:

Every time "Load More" is clicked, the app would keep loading page 1 again and again, showing the same images repeatedly.

What is this.searchValueGlobal = '';?

It’s a property of the class that is used to:

store the user’s last search keyword

It starts empty ('') because nothing has been searched yet and

We put it in the constructor to initialize the property.

Why Not Just Use a Variable?

Yes, you could use a regular variable like

let searchValueGlobal = '';

But in a class like PhotoGallery, it's better to use an object property like:

this.searchValueGlobal = '';

Here's Why We Use a Property (this.searchValueGlobal):

A regular let variable inside a method (like inside getSearchedImages) is local — it disappears when the method ends. But this.searchValueGlobal stays available in the whole object, in any method.

You need searchValueGlobal in multiple methods (e.g., in getSearchedImages() and getMoreSearchedImages()), so it must live as part of the object — not just inside one function.

Why do we need it?

When the user searches for something (like “nature”), we want to:

Save that keyword.

Use it again when the user clicks “Load More”, so we load more results for the same search (not curated images).

how it works:

1. User searches:

this.searchValueGlobal = 'nature';

2. Then they click “Load More”:

const baseURL = `https://api.pexels.com/v1/search?query=${this.searchValueGlobal}&page=${index}&per\_page=12`

Now it becomes:

https://api.pexels.com/v1/search?query=nature&page=2&per\_page=12

So the app(Photot Gallery web ) knows it has to keep loading more 'nature' images.

this.galleryDIv.appendChild(item)

inserts the item element as the last child of this.galleryDIv.

Place the item right before the closing </div> tag of the .gallery element.

Why do we only call this.eventHandle(); in the constructor, and not the other functions?

How do other functions (like getImg(), getSearchedImages(), etc.) still work even though we never called them directly?

They are called inside the eventHandle() function, which is called from the constructor.

We don't call all the functions in the constructor because:

Some functions should only run in response to user actions or events, not immediately when the object is created.

this.searchForm.addEventListener('submit', (e) => {

this.pageIndex = 1;

this.getSearchedImages(e);

});

What is event object check out cutom-cursor project documentation  
  
 Why we set this.pageIndex = 1; again in the submit event:

When a new search is submitted (e.g., from "cat" to "dog"), we want to start from the first page of the results — not continue from the previous page (e.g., page 4 of "cat").

What would happen if we didn’t set pageIndex = 1?

Let’s say the user searched for "cats" and clicked "Load More" 3 times → now pageIndex = 4.

Now they type "dogs" and submit — but if pageIndex is still 4, it will fetch page 4 of dog results, skipping the most relevant first pages.

By default, when you submit a form in HTML, the browser reloads the current page.

e.preventDefault() is a method of the Event object.

It's available in almost all browser events.

You use it when you want to stop the browser’s default behavior for that event., regardless of event type.

However, it is only useful (i.e., has an actual effect) on events that have a default browser behavior to stop — like:

submit → reloads the page

click on a link → navigates

keydown → scrolls or types characters

contextmenu → opens right-click menu

For events like mouseover, mouseenter, or mousemove — they do not have default actions, so calling preventDefault() does nothing, even though it's available.

If we do not use this method I current scenario

When we type anything in search input field e.g

You type "cat" and hit Enter.

The browser automatically reloads the page (because that’s what forms do by default).

When the page reloads:

All JavaScript variables are lost.

The .gallery becomes empty again.

Your script restarts from the top.

The code never gets a chance to fetch cat photos, because the page reload interrupts everything.

const input = e.target.querySelector('input');

const searchValue = input.value.trim();

e.target refers to the form element that triggered the submit event

querySelector('input') finds the input field inside that form.

So this line gets the actual <input> where the user typed the search term.

✅ const searchValue = input.value.trim();

input.value gets the text the user entered in the input field.

.trim() removes any extra spaces from the start and end (e.g., ' cat ' → 'cat').

This cleaned-up value is stored in searchValue.

Why did we used return statement in getSearchedImages method ?

To Stop further code execution (like calling the API) because input was invalid.

if (data.photos.length === 0) {

this.showLoading('no results found for your search');

setTimeout(() => {

this.hideLoading();

this.getImg(1);

}, 2000);

} else {

this.GenerateHTML(data.photos);

this.hideLoading();

}

e.target.reset();

}

data.photos is an array. If it's empty (length === 0), it means no search results were found.

e.target.reset() clear the search input field after the form is submitted

After a search is submitted, the input box becomes empty, so the user doesn’t have to manually delete the old search term before typing a new one.

async getMoreSearchedImages(index) {

this.showLoading();

const baseURL = `https://api.pexels.com/v1/search?query=${this.searchValueGlobal}&page=${index}&per\_page=12`

const data = await this.fetchImages(baseURL);

if (data.photos.length === 0) {

this.showLoading('No results found.');

} else {

this.GenerateHTML(data.photos);

this.hideLoading();

}

}

Why we used this.searchValueGlobal in baseURL , Why not use a new variable or take search value from input?

You shouldn’t use a new variable or take the input value again here, because:

1. User has already submitted the search

The input field might be empty now (we reset it), or the user might have typed something else.

We don’t want to depend on the current input anymore — we want to continue fetching more results for the same search.

2. We saved the search term globally

In getSearchedImages(e), we saved the user's input in:

So now we can re-use it anywhere across all methods of class, like in getMoreSearchedImages.

showLoading(message = 'Loading...') {

const loadingDiv = document.querySelector('.loading-text');

if (loadingDiv) {

loadingDiv.textContent = message;

loadingDiv.style.display = 'block';

}

this.loadMore.style.display = 'none';

}

showLoading(message = 'Loading...')

This function takes one optional parameter: message.

If no value is passed, it defaults to 'Loading...'.

if (loadingDiv)

If the element((loadingDiv) exists

this.loadMore.style.display = 'none';

While loading, we hide the "Load More" button (to avoid double clicks or confusion).