Clean and Organized CSS using BEM Methodology

Introduction

Naming classes has never been a simple aspect of **CSS**.

Especially when working on a project that has a lot of **HTML selectors**, **rule sets**, and many **pages**.

Potential for issues with specificity and possibility that altering and modifying **CSS styles** in one place would break styles in another.

Maintaining and developing that code may be challenging.

BEM (stands for Block Element Modifier)

- We will examine how to write clean, organized and maintainable code using a **CSS naming technique** called **BEM**.
- We will examine what is **BEM** and why we should adopt it.
- We will examine how it functions and how to implement **BEM** successfully.

What is BEM?

This is a **front-end naming technique** used to group and name **CSS classes**.

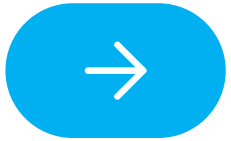
For **class names** in **HTML** and **CSS**, **Block**, **Element**, **Modifier** (or **BEM**) **technique** is widely used.

By abiding by guidelines, it makes writing clean **CSS** easier.

This **technique** adheres to "Do not Repeat Yourself" (DRY) and "Keep it Short and Simple" (KISS) principles.

BEM technique can help projects of any scale that use **CSS**.

Why BEM?



Without using any **naming techniques** in a large project with a large code base, there are numerous problems with class naming in **HTML** and **CSS**.

- Team members find it difficult and time consuming trying to understand functions of **class names** that were not specified by them.
- **Class names collisions** between team members.
- **Overriding classes** accidentally.
- Code is difficult to modify and alter without breaking something in entire project.
- How to deal and keep up with nesting is difficult.

Why BEM?

There are some reasons why we should adopt **BEM methodology** of **naming classes** in **HTML** and **CSS**.

Why BEM?

Organization and Scalability

Classes, **class names**, and **styling** will all become more organized using **BEM**, and organization in a project with a vast code base is a big plus.

Semantic accuracy & Readability

To explain why we did that or what controls what in code when we use a large number of random classes, we must use lots of comments.

BEM is self-documenting since its **naming system** is so readable and semantically accurate.

We don't require comments of any sort, it documents itself as we code.

Why BEM?

Avoids class name Collision

BEM provides a structure, which speeds up development and also implementation of new features.

CSS code base is more manageable over time and it makes teamwork simpler.

Easy to write and Debug

BEM methodology provides us with a **technique** that is basic enough for everyone involved in a project to understand.

Why BEM?

Avoids specificity issue

When **CSS** is not maintained and structured properly, larger projects can soon become extremely messy.

BEM encourages developers to write **HTML** and **CSS** without worrying about **inheritance** and specificity problems.

Modularization and Reusability

By using **BEM block styles**, we may avoid **cascade** issues in project and move blocks to other new projects.

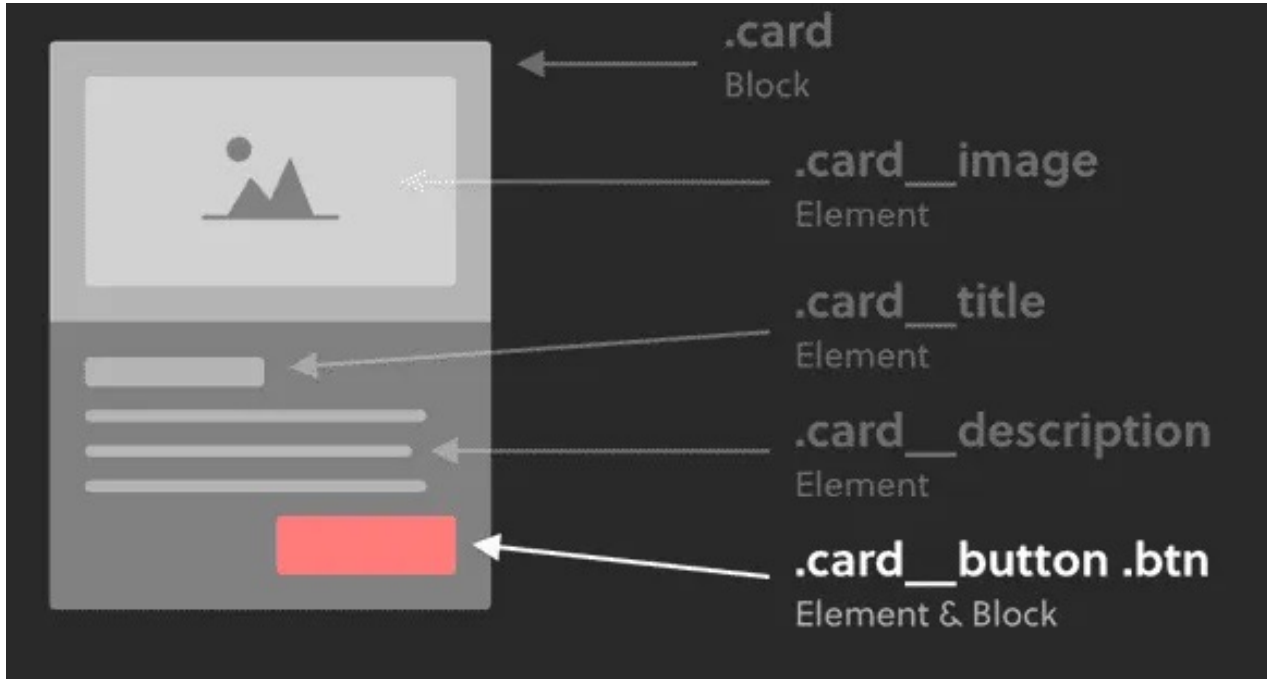
It reduces amount of **CSS** code we will write for a project and makes it easier for us to maintain.

How does BEM work?

Let’s take a look at what these components of **BEM** mean and how we can use them.

This is syntax of **BEM methodology**:

.block__element--modifier



How does BEM work?

Block is a standalone component or entity that is meaningful on its own.

A **block** can be seen as parent container that has other children and grandchildren under it.

When we are using **BEM methodology** of **naming classes** it is required that we specify a **block**.

A **block** could be a **card**, **header**, **footer**, **menu**, **container**.

In our image, class name given to **block** will be "**.card**".

How does BEM work?

An **element** is a part of a **block** that has no standalone meaning and it is semantically tied to its **block**.

An **element** can be seen as anything that goes into **block**, basically children of parent container.

To give an **element** a **class name**, **element's name** is separated from **block name** with a double underscore "**__**".

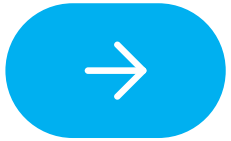
Using our image, we have identified a **block** with a **class name** of "**.card**", and inside this card we have an image, a description text and buttons as children within "**.cardcontainer**".

How does BEM work?

Class names for **elements** image, title, description and button:

```
/*Image*/
.card__image
/*Tittle*/
.card__title
/*description*/
.card__description
/*Button*/
.card__button
```

How does BEM work?



A **modifier** defines appearance state or behaviour of a **block** or **element**.

This is a flag on **block** or **element** and they are applied to modify look of an **element** or a **block**.

To change look of a **block** or an **element** we will have to give them an additional **class name** where **modifier name** is separated with a "__" from **block** or **element's name**.

How does BEM work?

This is syntax of a **modifier**:

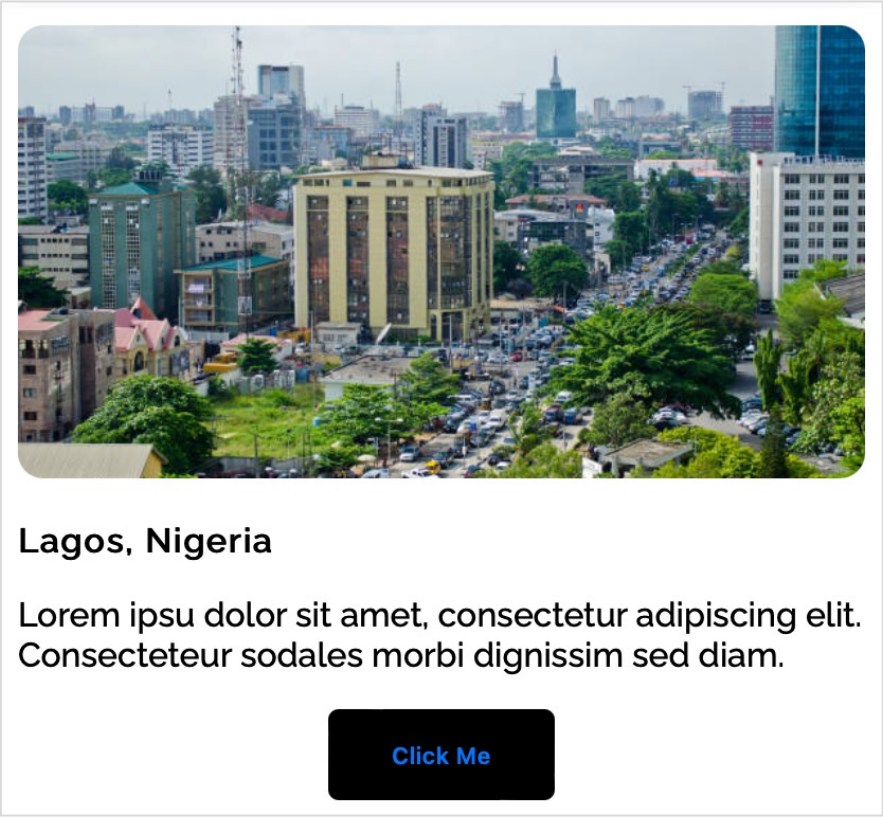
.block--modifier
.block__element--modifier

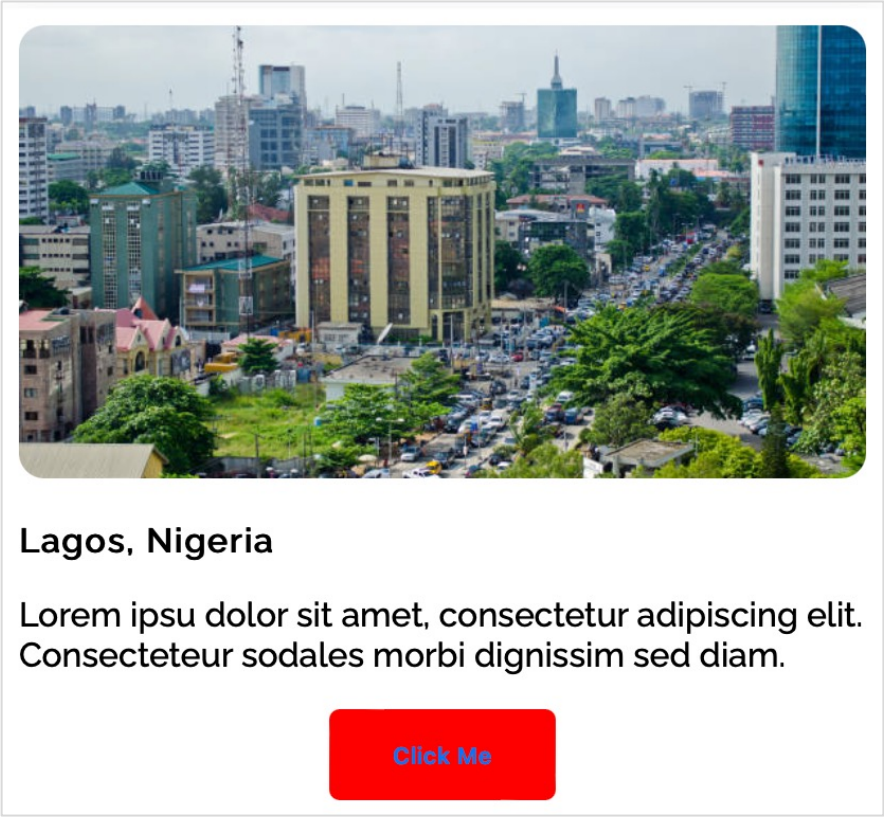
In our card component we have a button so we need to add a **modifier** to it **class name** of **button** since we want appearance of button to change when we hover around it:

.card__button--active

How does BEM work?

Let’s look up code for our card example.





```html
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Card Component</title>
</head>

<body>
  <div class ="card">
   <img src ="https://media.istockphoto.com/photos/image.jpg="
            alt="" class="card__image">
  <div class ="card__title">Lagos, Nigeria</div>
  <p class="card__dxescription">Lorem ipsu dolor sit amet, ...</p>
   <button class="card__button card__button--active">Click Me </button>
  </div>
</body>
```

```
body{
  background=color:#eaeff1;
  font-family: 'Raleway', sans-serif;
}
.card{
  max-width: 400px;
  margin: 0 auto;
  margin-top: 10vh;
  background-color:#fff;
  padding:8px;
  box-shadow:0 10px 25px rgba(92, 99, 105, .2);
}
.card__image{
  width:100%;
  border-radius:12px;
  height:214px;
  object-fit: cover;
}
...
```

```
...
.card__tag{
  padding: 4px 8px;
  border: 1px solid #e5eaed;
  border-radius:50px;
  font-weight:60px;
  color:#788697;
}
.card__title{
  font-sizw: 24px;
  font-weight: 600;
  margin-top:16px;
}
.card__description{
  font-size:14px;
  color:#7f8c9b;
  line-height:150%;
}
...
```

```
...
.card__button{
  border:none;
  padding: 15px 30px;
  border-radius:5px;
  font-weight:600;
  color:#0077ff;
  background-color:black;
  margin: 0 auto;
  display:block;
}
.card__button--active:hover{
  background-color:red;
}
```

Simple Example

In this example, we showcase an **HTML card component**.

Top-level is set with "**.card**" **class name** and this is a **block**.

Child items (or **elements**) inside of .card is prefixed with "**.card__**" where "**.card__description**" is declared.

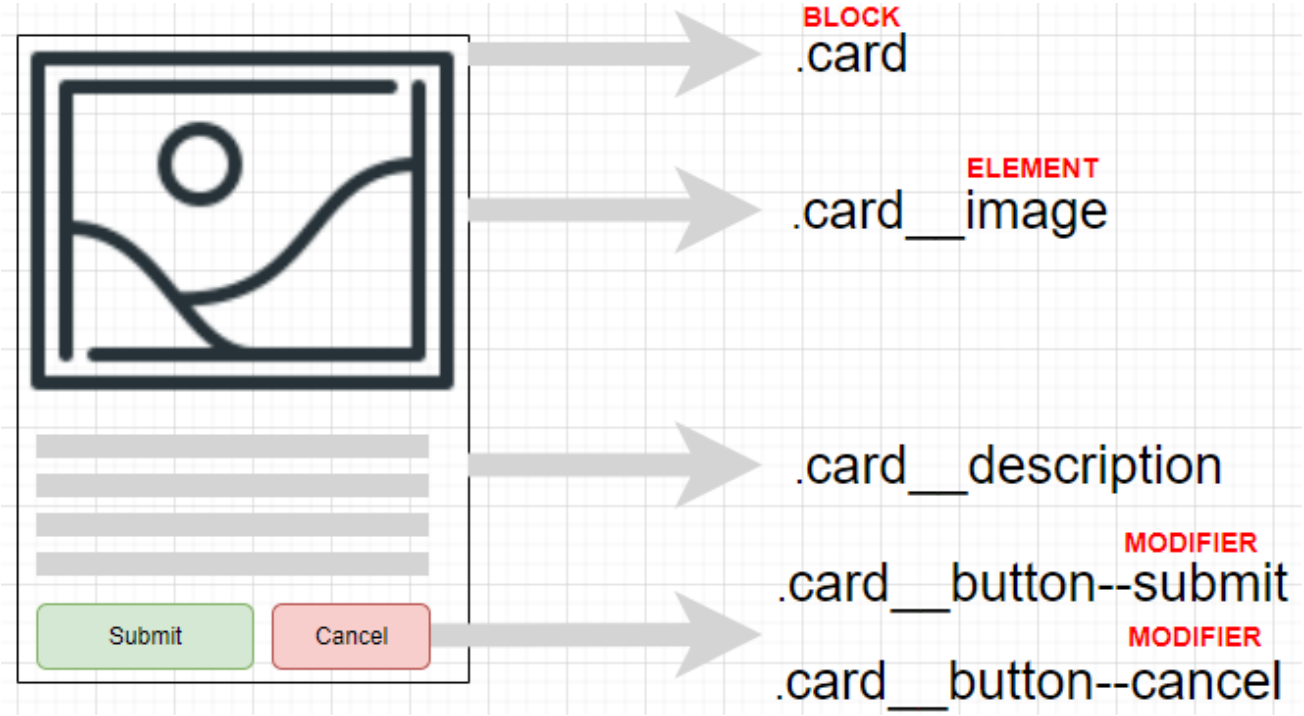
Simple Example

Next, both buttons have a similar look and feel, but only difference is background colors, and width of buttons.

With these different button variations, we can then use **modifiers** conventention in **BEM**.

We prefix **modifier** with "**.card__button**", and then **modifier name**; for example, "**.card__button-submit**" and "**.card__button-cancel**".

Simple Example




```
<div class="card">
   <div class="card__image"></div>
   <p class="card__description"></p>
   <button class="card__button--submit"></button>
   <button class="card__button--cancel"></button>
</div>
```

SCSS Input

```
.card {
  &__image {
    display: block;
  }
  &__description {
    display: block;
  }
  &__button {
    &--submit {
      display: inline-block;
    }
    &--cancel {
      display: inline-block;
    }
  }
}
```

CSS Output

```
.card__image {
  display: block;
}
.card__description {
  display: block;
}
.card__button--submit {
  display: inline-block;
}
.card__button--cancel {
  display: inline-block;
}
```

Advance Example

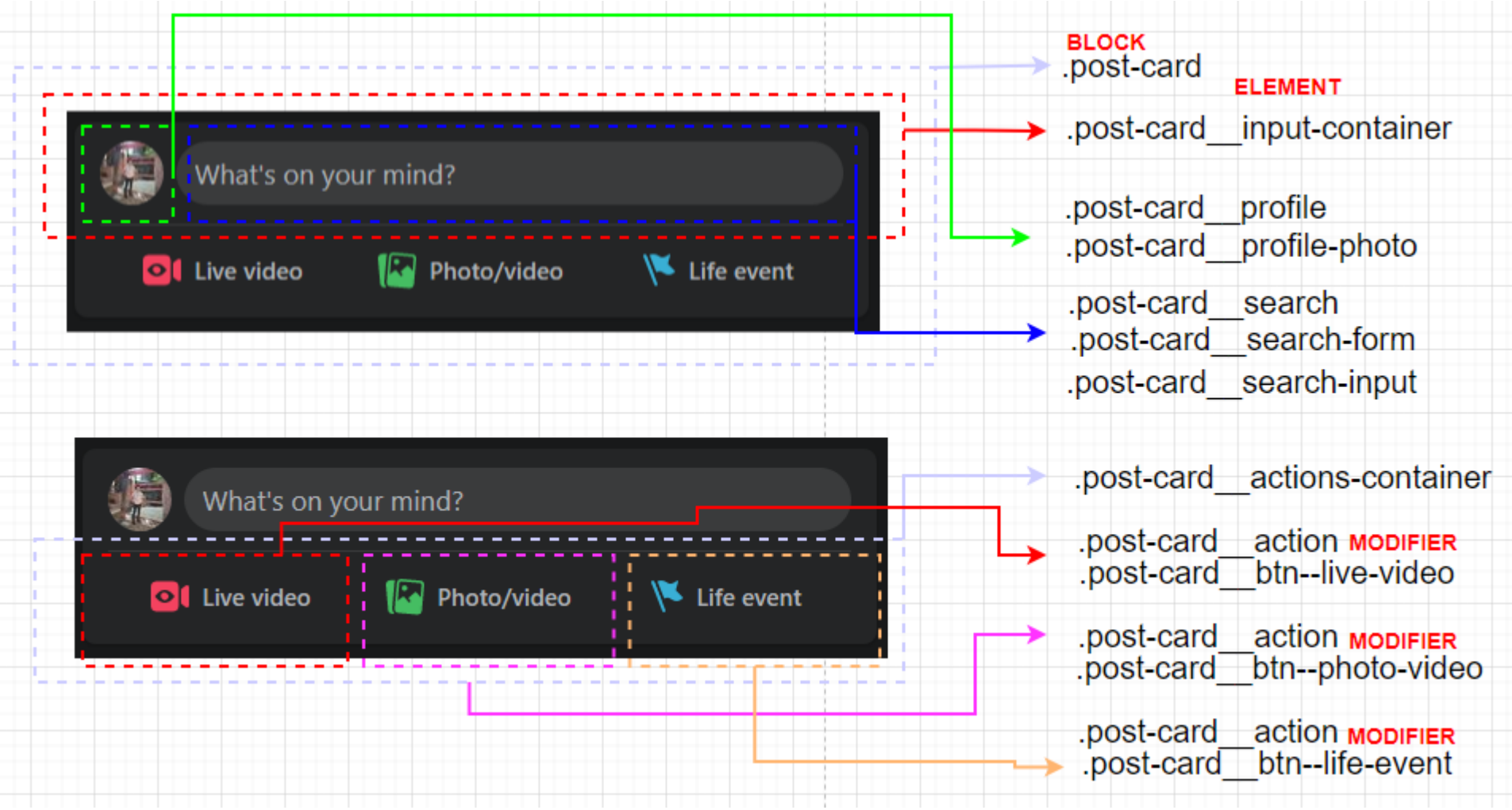
In this new more complicated example, we showcase an **HTML post-card component**.

Top-level is set with "**.post-card**" **class name** and this is a **block**.

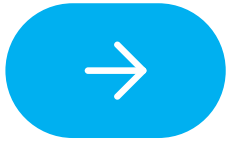
Child items (or **elements**) inside of "**.post-card**" is prefixed with "**.post-card__**".

There are many **elements** inside of this **HTML component**; for example, "**.post-card__input-container**", "**.post-card__profile**", "**.post-card__profile-photo**".

Advance Example



Advance Example



There are some buttons for this given **component** such as live-video, photo-video, and live-event buttons.

These buttons are similar and have a slight variation to them so we can use **modifiers** for these buttons.

Prefixed by "**.post-card__btn**", we now append **CSS class names** like "**.post-card__btn–live-video**", "**.post-card__btn–photo-video**", and "**.post-card__btn–life-event**".

Advance Example

```
<div class="post-card">
  <div class="post-card__input-container">
    <div class="post-card__profile">
      <img class="post-card__profile-photo" />
    </div>
    <div class="post-card__search">
      <form action="" class="post-card__search-form">
        <input type="text" class="post-card__search-input" />
      </form>
    </div>
  </div>
  ...
```

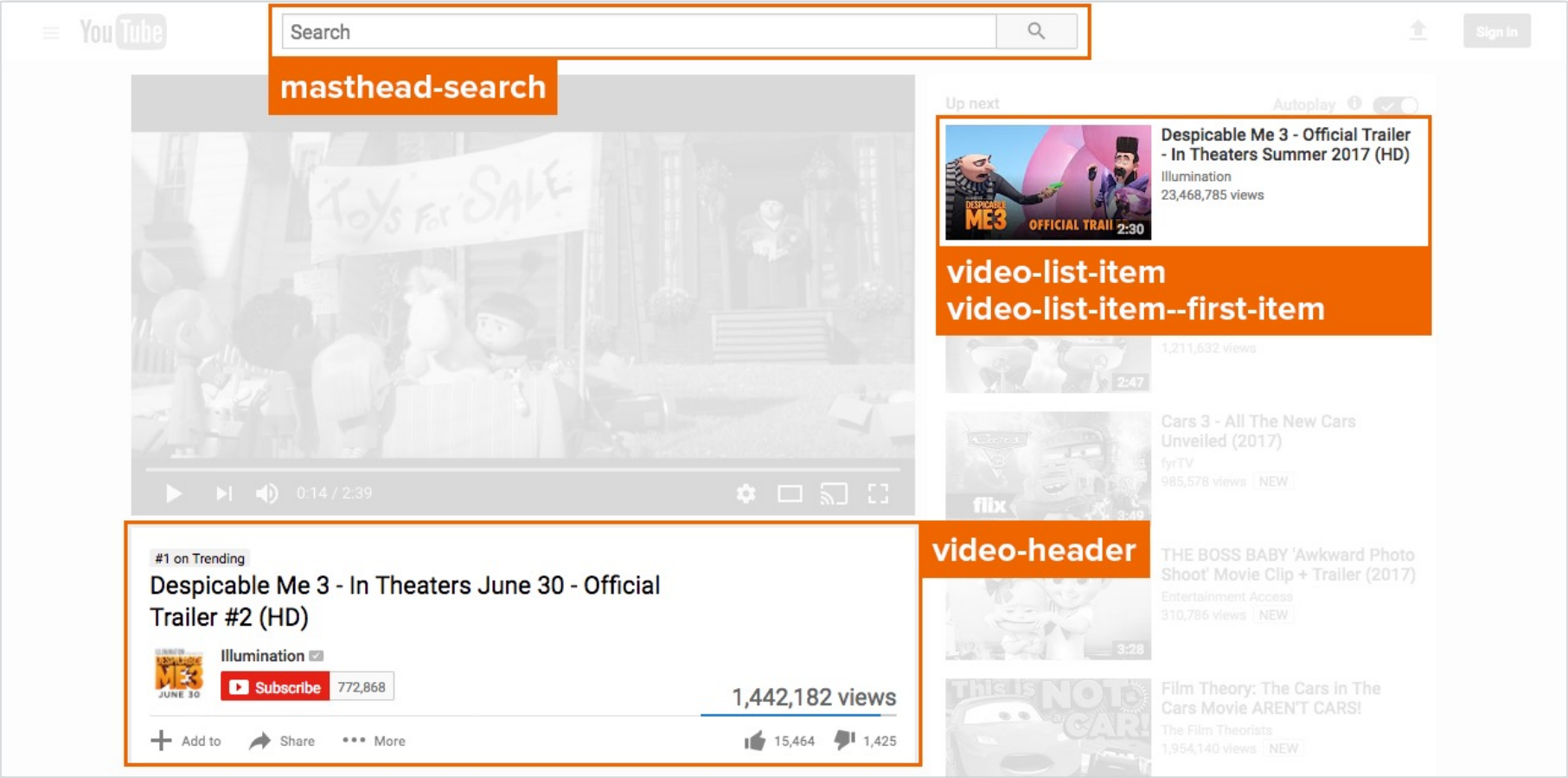
Advance Example

```
...
<div class="post-card__actions-container">
  <div class="post-card__action">
    <button class="post-card__btn--live-video"></button>
  </div>
  <div class="post-card__action">
    <button class="post-card__btn--photo-video"></button>
  </div>
  <div class="post-card__action">
    <button class="post-card__btn--live-event"></button>
  </div>
</div>
</div>
```

```
.post-card {
  display: block;
}
.post-card__input-container {
  display: flex;
}
.post-card__profile, .post-card__search {
  width: 50%;
}
.post-card__profile {
  display: block;
}
.post-card__profile-photo {
  display: block;
}
.post-card__search {
  display: block;
}
.post-card__search-form {
  display: block;
}
.post-card__search-form {
  display: block;
}
.post-card__actions-container {
  display: flex;
}
.post-card__action {
  width: 33%;
}
.post-card__btn--live-video {
  display: block;
}
.post-card__btn--photo-video {
  display: block;
}
.post-card__btn--life-event {
  display: block;
}
```

YouTube Example

To better illustrate **blocks**, **elements**, and **modifiers**, let's identify potential **blocks** and **elements** on this YouTube video page.



YouTube Example

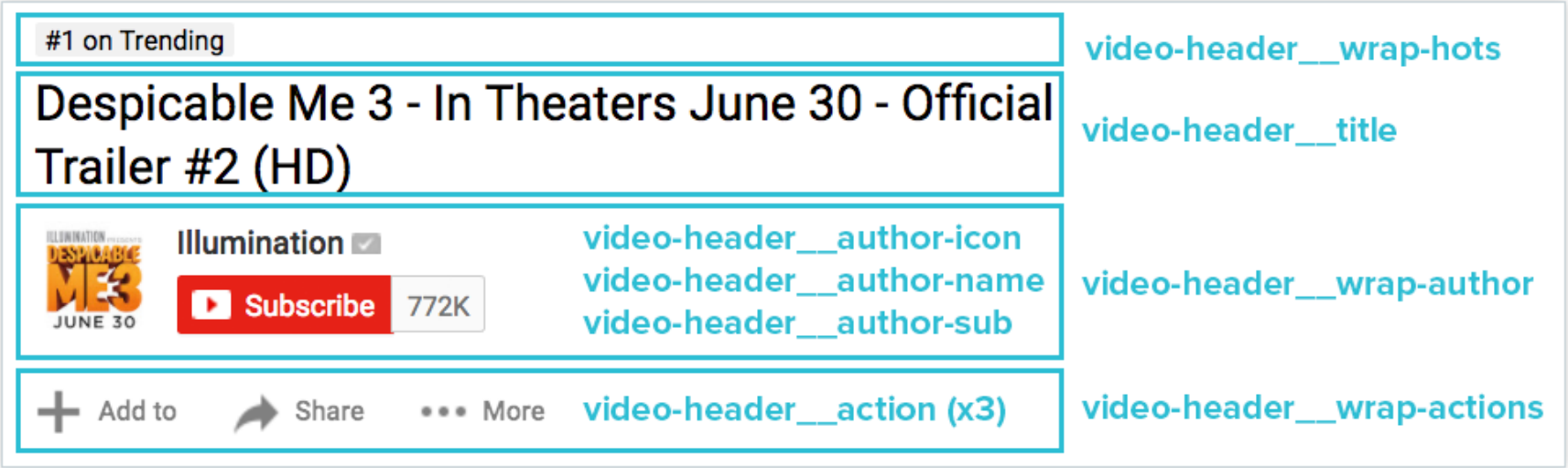
Starting with **blocks**, the **masthead-search** and **video-header blocks** appear once on this page.

Block video-list-item is repeated for each video in sidebar.

We also have a **modifier** to first one to allow for different styling.

YouTube Example

This image identifies potential **elements** within **video-header block**.



YouTube Example

This example defines an abstract **block** "**.block**" that has an **element** identified by double-underscores "**.block__element**".

Block also defines a **modifier** identified by double-hyphens "**.block--modifier**".

Additionally, "**.block__element**" also defines a new **modifier** "**.block__element--modifier**".

YouTube Example

```
<!-- a block with two elements, one has a modifier -->
<div class="block">
  <div class="block__element"></div>
  <div class="block__element block__element--modifier"></div>
</div>

<!-- a block with a modifier that contains one element -->
<div class="block block--modifier">
  <div class="block__element"></div>
</div>

.block {};
.block__element {};
.block--modifier {};
.block__element--modifier {};
```

A More Complex Example

Phillip Dodson

I heart hamburgers.

Dustin Woehrmann

I heart Grace (woof).

```
<div class="c-bio c-bio--latest">
  <div class="c-bio__name">Phillip Dodson</div>
  <div class="c-bio__excerpt">I heart hamburgers.</div>
</div>

<div class="c-bio">
  <div class="c-bio__name">Dustin Woehrmann</div>
  <div class="c-bio__excerpt">I heart Grace (woof).</div>
</div>
```

A More Complex Example

```
.c-bio {
  background-color: #eee;
  border: 1px solid #ddd;
  border-radius: 5px;
  padding: 1em;
  font-family: sans-serif;
}

.c-bio--latest {
  background-color: #333;
  color: white;
}
```

```
.c-bio__name {
  font-size: 2em;
}

.c-bio__excerpt {
  border-top: 1px solid #ddd;
  margin-top: .5em;
  padding-top: .5em;
}
```

Complicated Grandchildren Structure

Working with **BEM**, all **class names** are flat, meaning that there will only be allowed with once "__".

Because **block structure** should be flattened, we do not need to reflect nested **DOM structure** of **block**.

Complicated Grandchildren Structure

```
<div class="block">
  <div class="block__elem-1">
    <div class="block__elem-2">
      <div class="block__elem-3">
        <div class="block__elem-4">
          <button class="block__elem-5--red">
            <span class="block__elem-6"></span>
          </button>
          <button class="block__elem-5--blue">
            <span class="block__elem-6"></span>
          </button>
        </div>
        ...
```

Complicated Grandchildren Structure

```
        ...
        <div class="block__elem-7">
          <ul class="block__elem-8">
            <li class="block__elem-9"></li>
            <li class="block__elem-9"></li>
            <li class="block__elem-9"></li>
          </ul>
         </div>
      </div>
    </div>
  </div>
</div>
```

A Portfolio's Landing Page

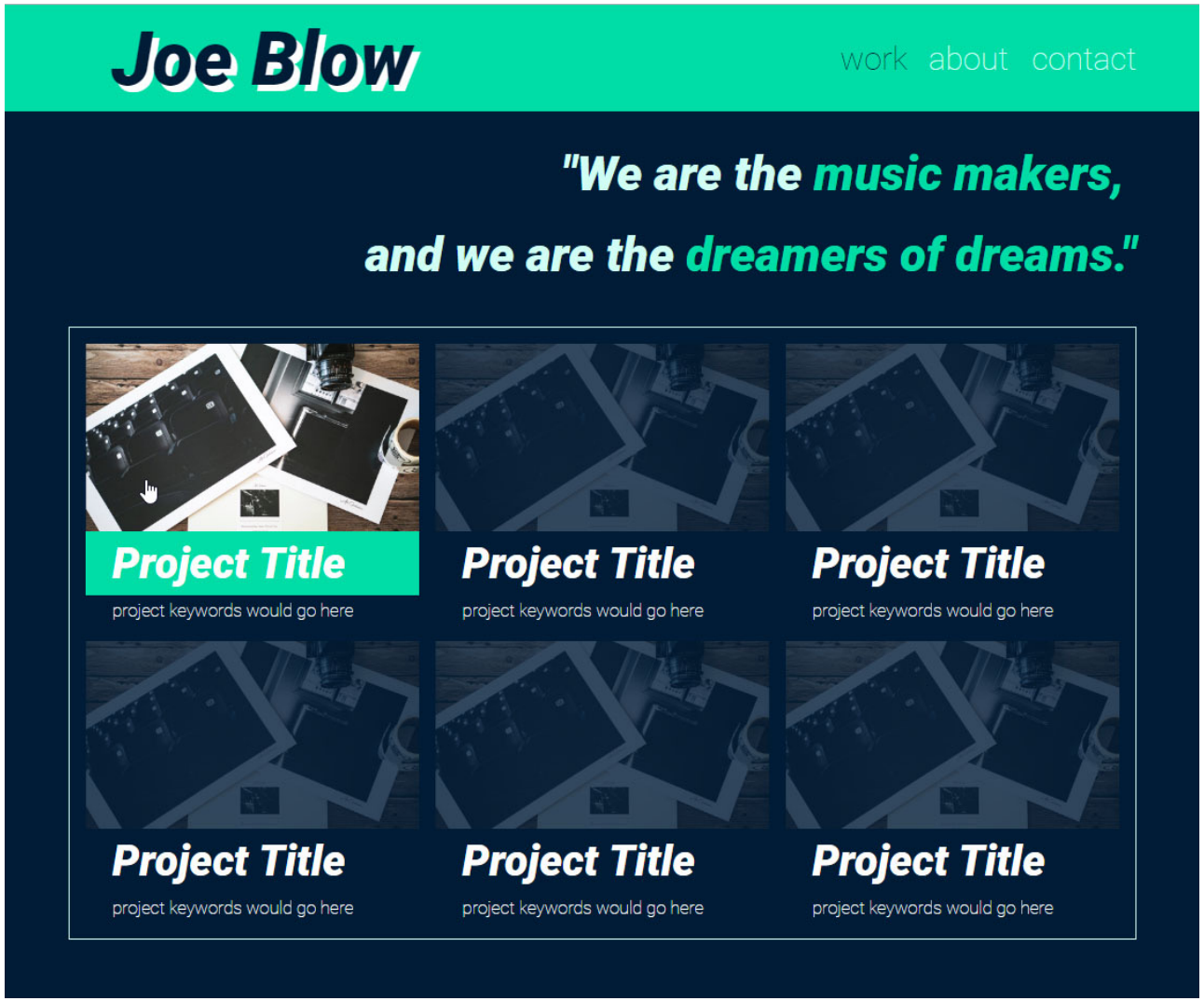
A **block** is a **component**, or **section**, of a **page** that can stand alone and function independently from rest of page.

This **block** could be a **header**, **container**, **menu**, or **button**.

Key is that we could strip away all of surrounding website and it would still make sense.

A Portfolio's Landing Page

Let’s take a look at a portfolio site that we'll be building where we have a logo, navigation, quote, and project grid.

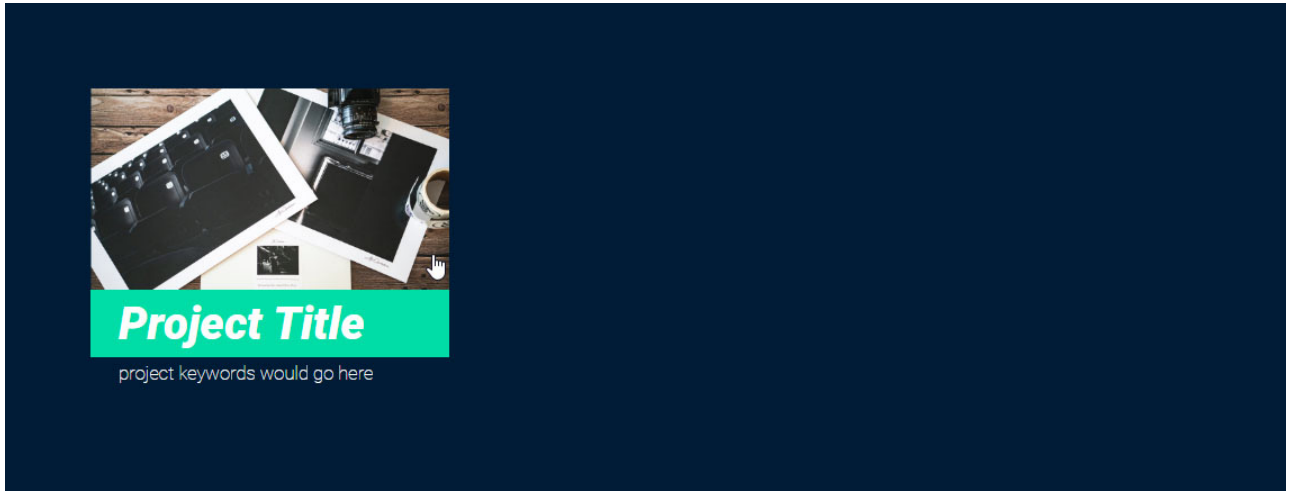


A Portfolio's Landing Page

If we were to isolate project preview, it still works.

There is an image with a title and descriptive text.

It makes sense on its own so, in **BEM terminology**, this project preview is a **block**.



A Portfolio's Landing Page

Let’s name **block selector** for our project preview **.proj-prev**, and only assign rules that are specific to setup of **block**.

Don't worry about giving title its padding or opacity of image.

Instead, we can set font to be white which all of elements in **block** use.

By assigning it to **block**, its **elements** will inherit color.

We also need to apply a margin to bottom of **block**.

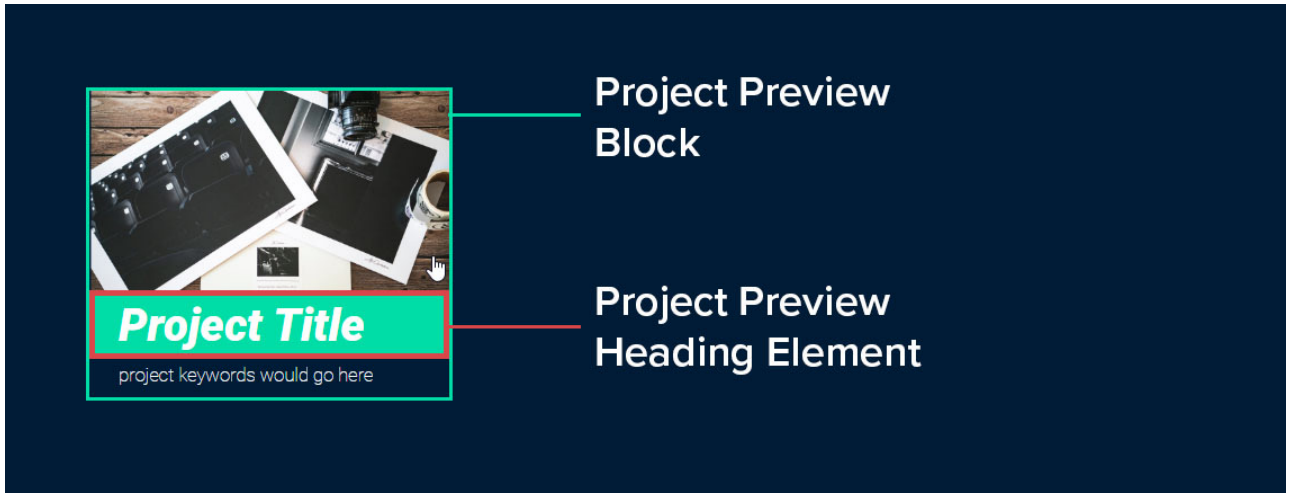
A Portfolio's Landing Page

No matter what we fill project preview with, it will need to use a white font and have a margin at bottom.

```
.proj-prev {
   color: #fff;
   margin-bottom: .25rem;
}
```

A Portfolio's Landing Page

Since text for project title doesn’t function independently, but rather forms an integral part of **block**, it's an **element** of **block**.



A Portfolio's Landing Page

We name a **block** by describing its purpose, for example, **.proj-prev** for our project preview.

Name of an **element** should identify its parent **block**, and then also purpose of this **element**.

Since it's heading for project preview, we'll name it **.proj-prev__heading**.

```
.proj-prev__heading {
   font-size: 4rem;
   padding-left: 2.5rem;
   margin: 0;
   line-height: 6rem;
}
```

A Portfolio's Landing Page

Modifiers changes appearance of a **block** or **element**.

Think of them as **selectors** that produce different versions of **blocks** and **elements**.

Need to change size, color, typography of an **element**.

A Portfolio's Landing Page

Standard font color for **block** is white, but for certain projects, we want to highlight them by using a minty color instead.

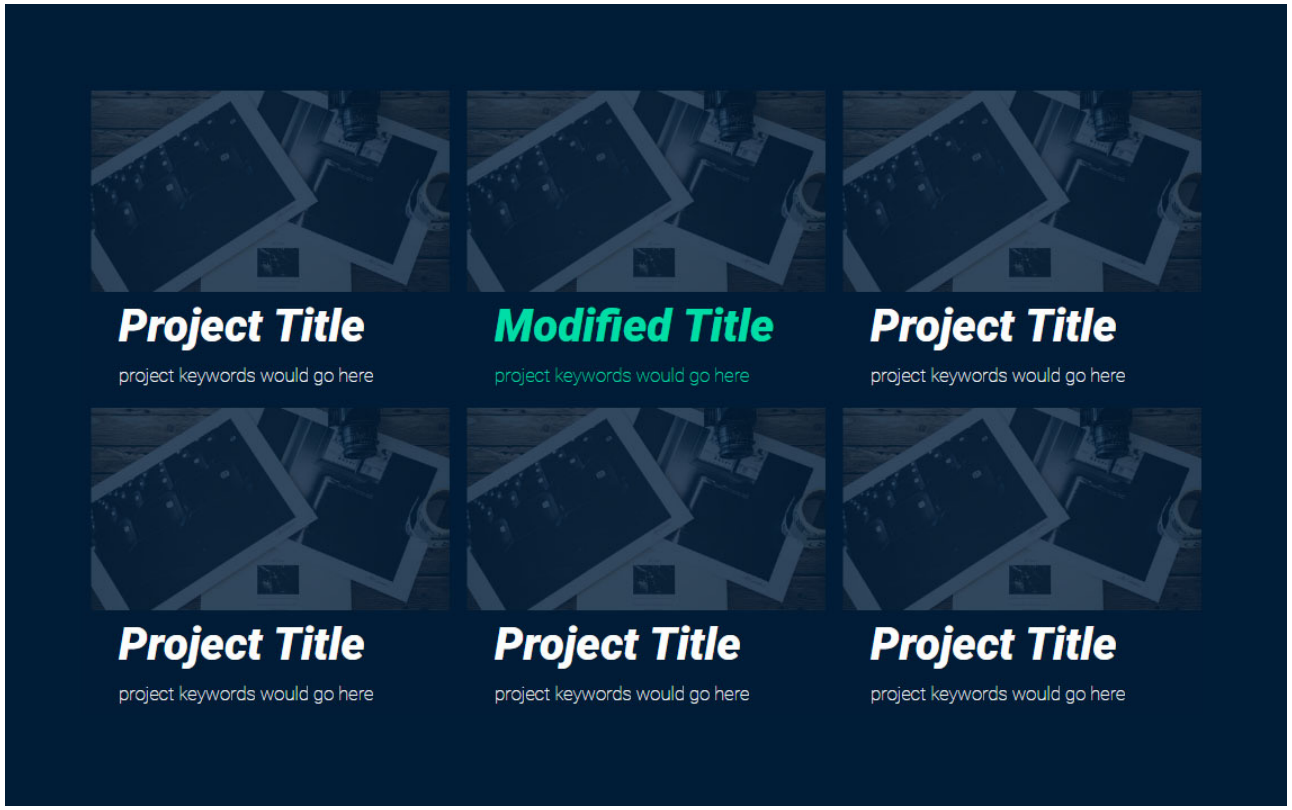
We'll create a **modifier** for our **.proj-prev block** and, to name it, we'll first identify what it's modifying, and then visual style of **modifier**.

Since we are modifying **.proj-prev block** to have a mint font color, we can use **.proj-prev—mint**.

```
.proj-prev--mint {
   color: #15DEA5;
}
```

A Portfolio's Landing Page

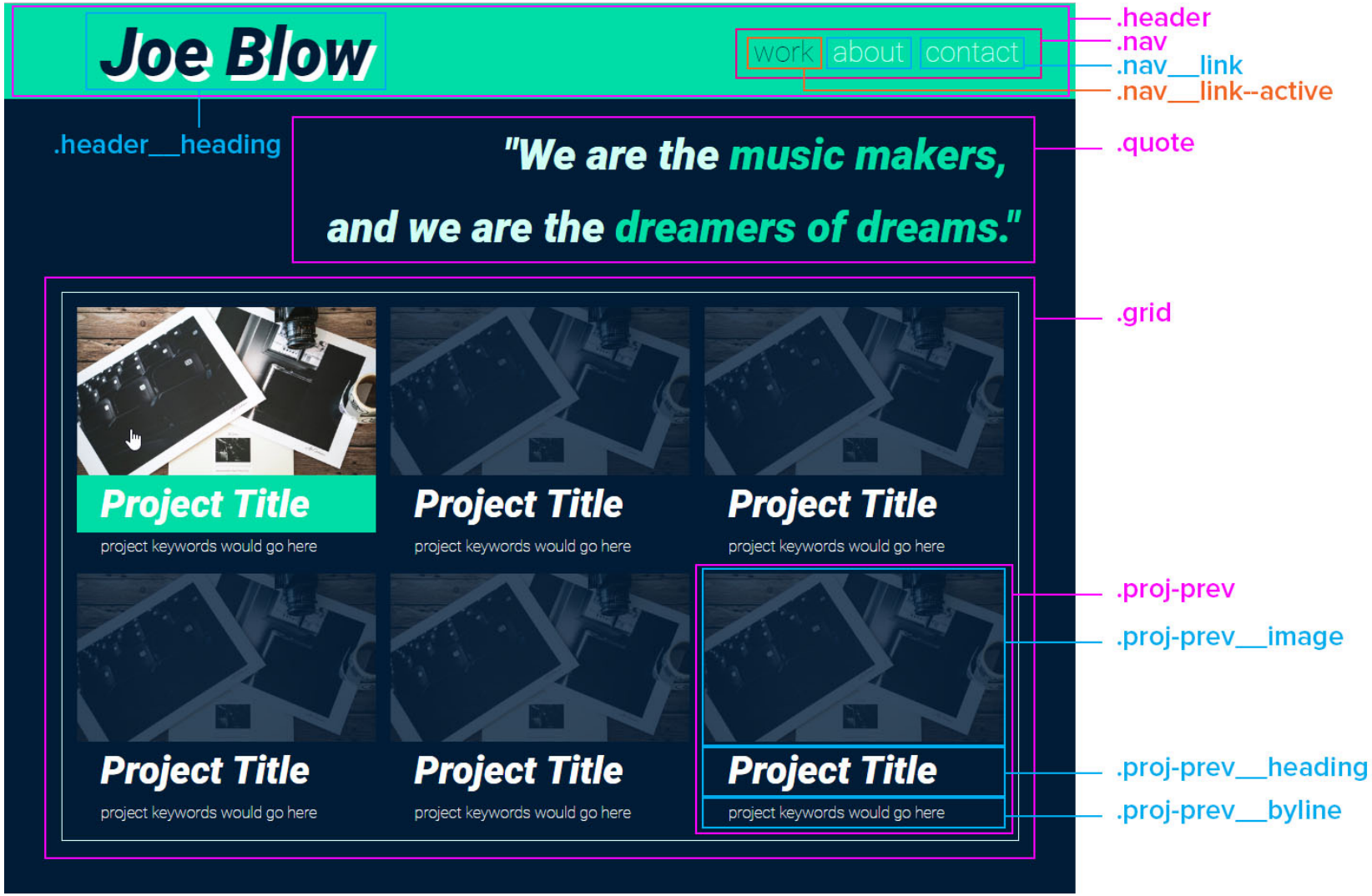
When we check out rendered **HTML**, we see that there is a special project highlighted.



A Portfolio's Landing Page

```
<section class="proj-prev proj-prev--mint">
  <div class="proj-prev__image">
    <img src="/public/img/photo_1280w.jpg" alt="project title goes here" >
  </div>
  <h1 class="proj-prev__heading">
    Project Title
  </h1>
  <p class="proj-prev__byline">
  project keywords would go here
  </p>
</section>
```

A Portfolio's Landing Page



A Portfolio's Landing Page

```
<nav class="nav">
   <ul>
      <li class="nav__link nav__link--active">work</li>
      <li class="nav__link"> <a href="/about.html">about</a> </li>
      <li class="nav__link"> <a href="/contact.html">contact</a>  </li>
   </ul>
</nav>
```

A Portfolio's Landing Page

```
.nav {
   padding-right: 6rem;
   text-align: right;
}

.nav__link {
   display: inline;
   font-size: 3rem;
   padding-left: 1.5rem;
}
```

```
.nav__link a {
   text-decoration: none;
   color: #D6FFF5;
}

.nav__link--active {
   color: #001534;
}

.nav__link a:hover {
   color: #fff;
}
```

Conclusion

BEM is an ideal way to writing **CSS** because it provides a **naming structure** that is semantically accurate and useful when working on a large project with lots of developers.

Conclusion

- **Block**: an independent **component** that can be reused (e.g. with **class name** "**.nav**").
- **Element**: a child within a **block** that cannot be used separately from that **block** (e.g. with a **class name** "**.nav__item**").
- **Modifier**: a variation in style of either a **block** or **modifier** (e.g. with class name "**.nav--dark**").

Conclusion

```
<!-- BLOCKS: INCORRECT -->
<div class="large-red-box">
 <img src="...">
 <h2>...</h2>
 <p>...</p>
 <a>...</a>
</div>
<style>
 .large-red-box {}
</style>
```

```
<!-- BLOCKS: CORRECT -->
<div class="card">
 <img src="...">
 <h2>...</h2>
 <p>...</p>
 <a>...</a>
</div>
<style>
 .card {}
</style>
```

Conclusion

```
<!-- ELEMENTS: INCORRECT -->
<div class="card">
 <img src="...">
 <h2>...</h2>
 <p>...</p>
 <a>...</a>
</div>
<style>
 .card {}
 .card img {}
 .card h2 {}
 .card p {}
 .card a {}
</style>
```

```
<!-- ELEMENTS: CORRECT -->
<div class="card">
 <img class="card__img" src="...">
 <h2 class="card__title" >...</h2>
 <p class="card__description" >...</p>
 <a class="card__button">...</a>
</div>
<style>
 .card {}
 .card__img {}
 .card__title {}
 .card__description {}
</style>
```

Conclusion

```
<!-- INCORRECT -->
<div class="card--dark">
 <img src="...">
 <h2 class="card__title--large">...</h2>
 <p>...</p>
 <a>...</a>
</div>
<style>
 .card--dark {}
 .card__title--large {}
</style>
```

```
<!-- CORRECT -->
<div class="card card--dark">
 <img src="...">
 <h2 class="card__title card__title--large">
   ...
 </h2>
 <p>...</p>
 <a>...</a>
</div>
<style>
 .card {}
 .card--dark {}
 .card__title {}
 .card__title--large {}
</style>
```