

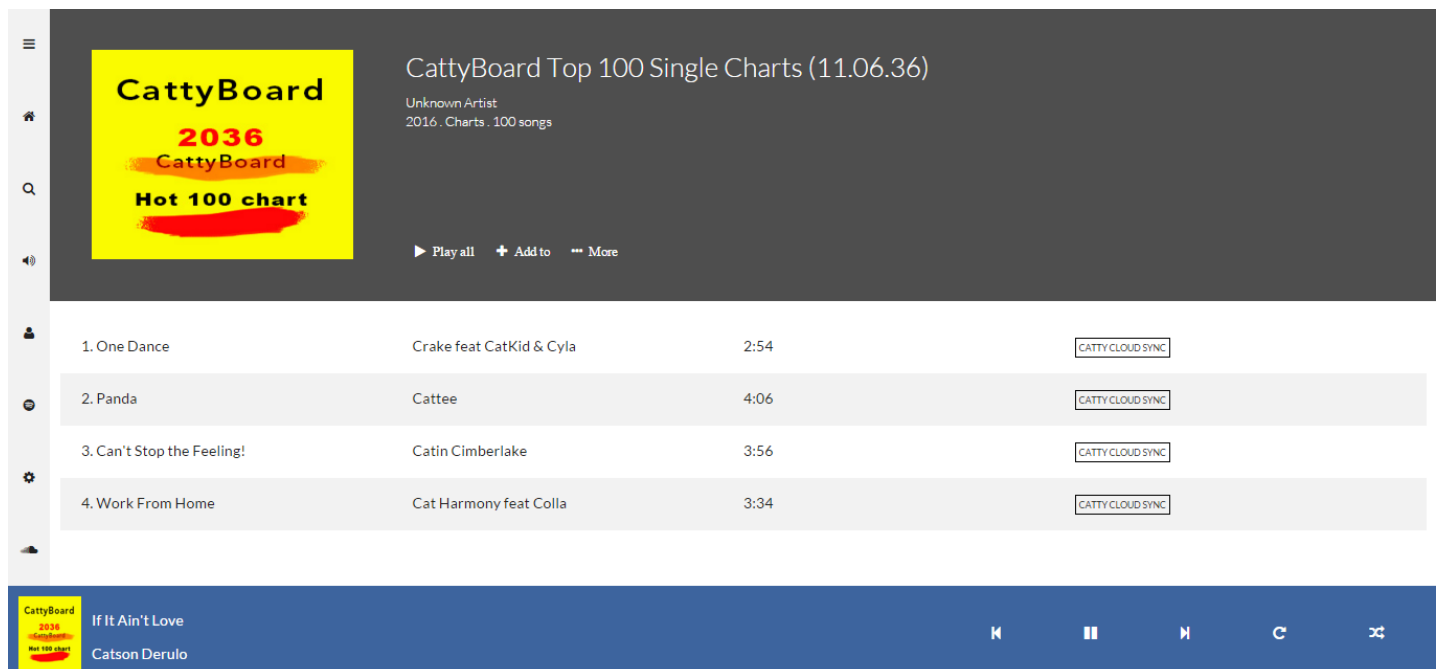
# Building a Music App Layout with Flexbox

After walking through the boring rigorous stuffs, you deserve some fun project.

It's time to walk through a practical example and apply your newly acquired *Flexbox skills*

It took me days to come up with a good project. Out of the lack of a creative option, I came up with a music app layout for cats. I call it catty music. Maybe by 2036, we'd have cats singing in rock bands somewhere in mars :-)

Here's what the finished layout looks like, and it is completely laid out with Flexbox.



You may view it online [here](#)

If you view that on a mobile device, you'll have a slightly different look. That's

If you view that on a mobile device, you'll have a slightly different look. That's something you'll work on in the responsive design section of this article.

I've got a confession to make though.

I've done something considered wrong by many.

I've completely built the overall layout with Flexbox.

For many reasons, this may not be ideal. But it's intentional in this scenario. I set out to show you all the things you can do with Flexbox, all wrapped up within a single project.

If you're curious as to when it's considered right or wrong to use the Flexbox model, you may check out my article on that. [Flexbox is awesome but it's NOT welcome here](#)

There, I got that off my chest. Now I'm sure no one's going to yell at me after taking this course.

Everything in Catty Music is laid out using the Flexbox model—this is intentional to show off what's possible.

So let's get this thing built!

As with any reasonable project, a bit of planning goes a long way sifting through inefficiencies.

Let me take you through a planned approach to building the catty music layout.

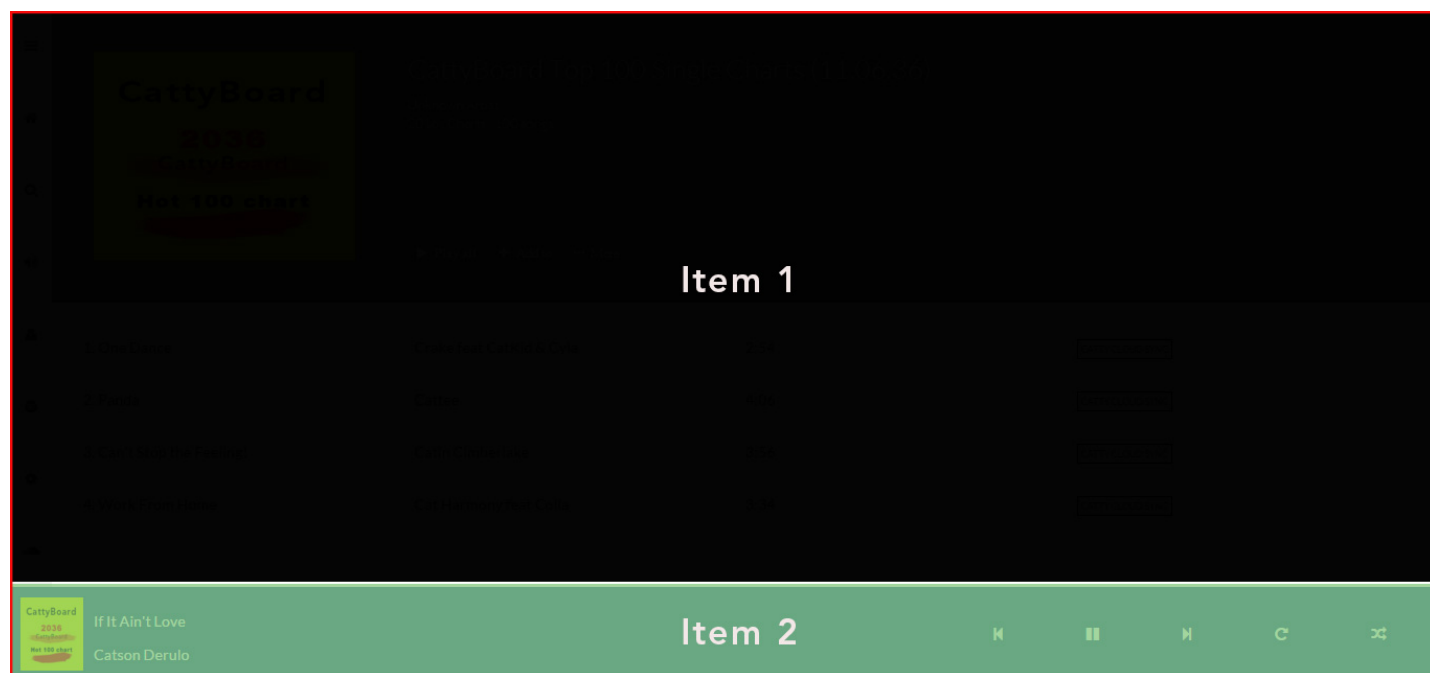
## So where do you start?

Whenever building a layout with flexbox, you should start by looking out for what sections of your layout may stand out as flex-containers. You may then leverage the powerful alignment properties flexbox makes available.

### The Breakdown

Take a look at the finished layout again.

You may have the overall containing body as a flex container (represented by the red border in the image below) and have the other sections of the layout split into flex-items (item 1 and 2).



This makes total sense, as item 1 contains every part of the layout other than the “footer”—the section that contains the music control buttons.

Did you know that a flex-item could also be made a flex-container?

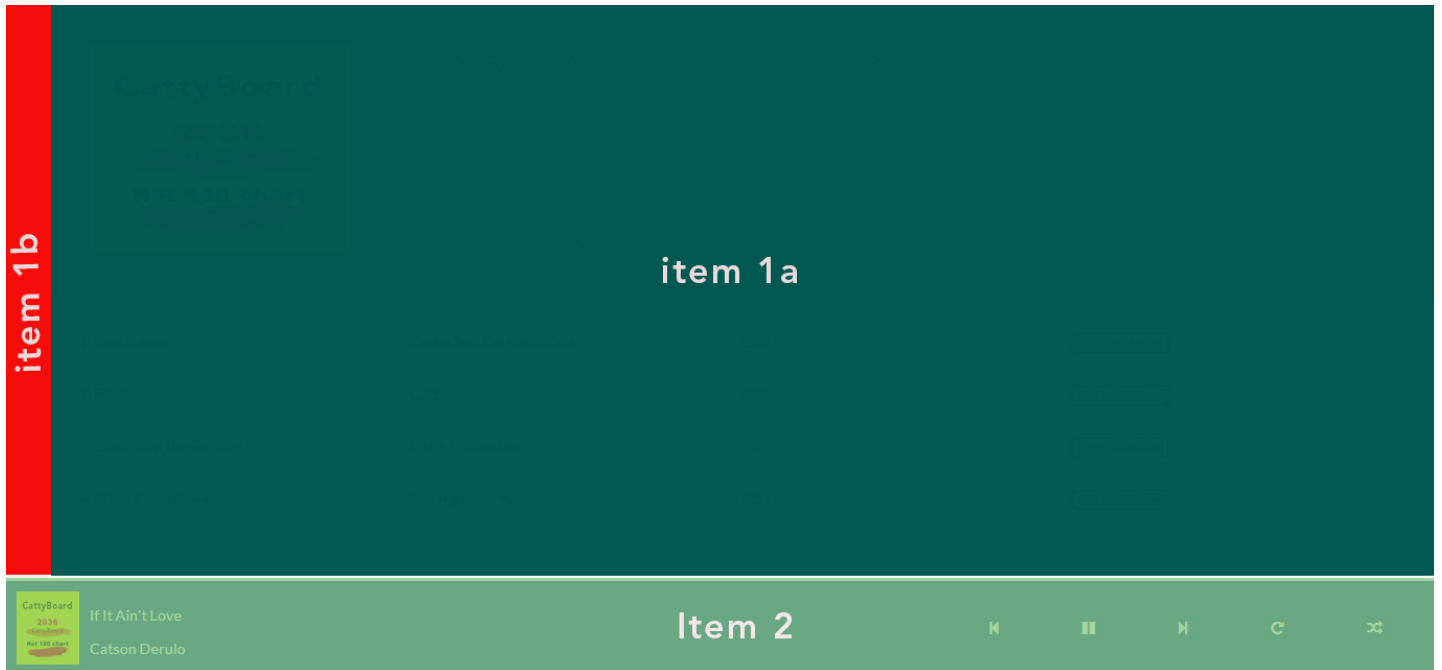
Yep, it’s possible!

You can nest as deep as you want (though the sane thing to do is to keep this to a reasonable level).

So, with that new revelation comes this...

Item 1 (the first flex-item) may also be made a flex container.

The sidebar(item 1b) and main section(item 1a) would then be flex-items.



You're still with me, right?

Decomposing your layout like this gives you a really good mental model to work with.

When you begin building even more complex layouts with the Flexbox model, you'd see how vital this is.

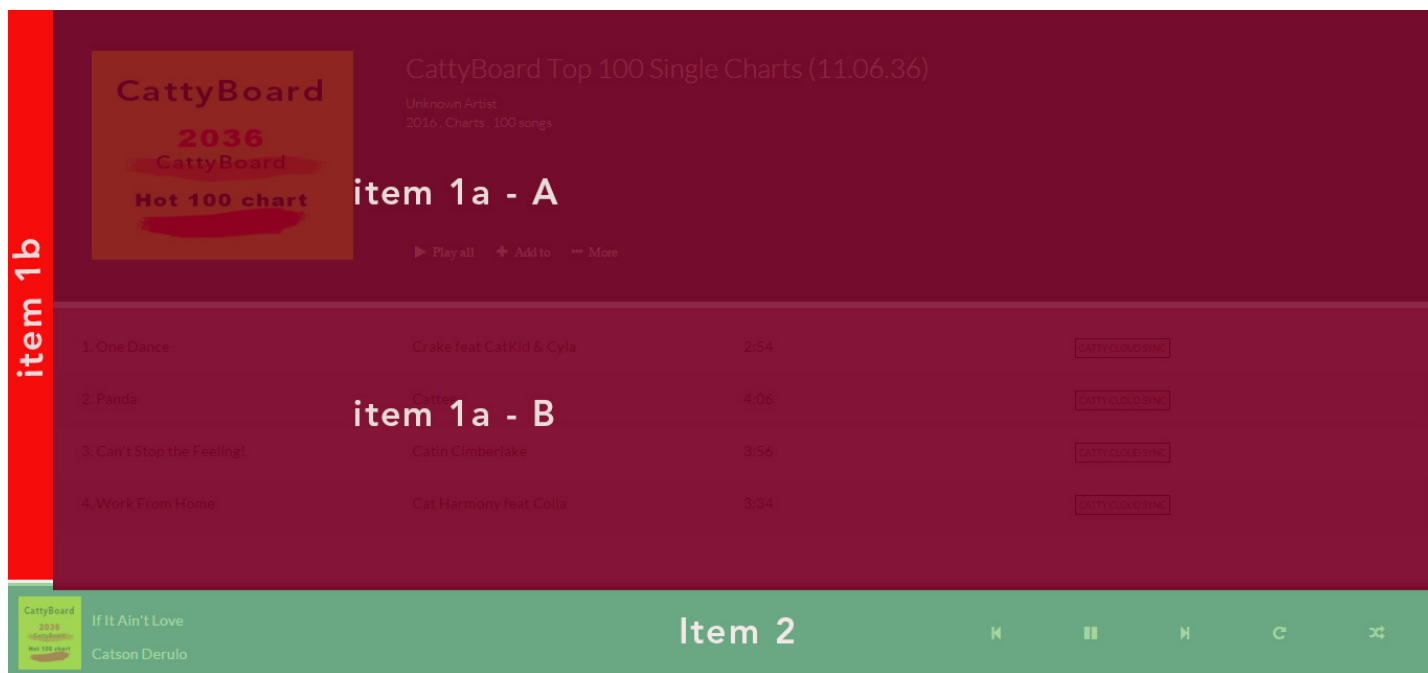
You do not need a fancy image like the ones above. A simple rough paper sketch should be just fine to get you going.

You remember I said you could nest as deep as you wanted? It appears you may do one more nesting here.

Take a look at the main section above (Item 1a).

It could also be made a flex container to house the sections highlighted below.

*“Item 1a—A” and “Item 1a—B”*



You may decide not to make the main section (item 1a) a flex container and just put within it two “divs” to house the highlighted sections.

Yes that’s possible, since “Item 1a—A” and “Item 1a—B” are stacked vertically.

By default, “divs” stack vertically. It’s how the box model works.

If you choose to make the main section a flex-container, you get the powerful alignment properties at your disposal. Just in case you need them at any time.

The “flex” in Flexbox means flexible.

Flex-containers are by default flexible, kind off responsive.

This may be another reason to use a flex-container over regular “divs”. This depends on the case scenario though.

I’ll touch up on some other things as you build catty music. You should get to writing some code now.

We’ll start off with the basic html set up below:

```
<!DOCTYPE html>
<html>
<head>
<title>Catty Music</title>
</head>
```



```
</head>
<body>

<main></main> <!--to contain the main section of the app-->

<footer></footer> <!--to contain the music control buttons and song details-->

</body>
</html>
```

So style this ...

```
html,
body {
  height: 100%; /*setting this explicitly is important*/
}

body {
  display: flex; /*flex superpowers activated! */
  flex-direction: column; /*Stack the flex-items (main and footer elements) vertically NOT horizontally*/
}
```

The first step to using the Flexbox model is establishing a flex container.

This is exactly what the code above does. It sets the body element's display property to **flex**

Now you have a flex container, the body element.

The flex items are defined too (item 1 and item 2)—as in the breakdown earlier done.

Note that you should take another look at the images I showed in my initial breakdown earlier if this concept still seems fuzzy for you.

Keeping the image of the end in view, you should get the flex-items working.

## Make the footer stick to the bottom.

The footer which houses the music controls is to stick to the bottom of the page while the main section fills up the remaining space.

How do you do that?

```
main {  
  flex: 1 0 auto; /*fill the available space*/  
}  
  
footer {  
  flex: 0 0 90px; /*don't grow or shrink - just stay at a height of 90px.*/  
}
```

Thanks to the `flex-grow` property. It's relatively easy to have the main section fill the entire space. Just set the `flex-grow` value to 1. You should also set the `flex-shrink` property to zero. Why?

The reason may not be evident here because the flex-direction is changed.

In some browsers, there's a bug that allows flex-items shrink below their content size. It's quite a weird behavior.

The workaround to this bug is to keep the `flex-shrink` value at `0` NOT the default, `1` and also set the `flex-basis` property to `auto`.

It's like saying: *"Please take on an initial width based on your content size, but never shrink."*

With this shorthand value, you still get the default behavior of flex items.

The flex item would shrink upon resizing the browser. The resizing isn't based on the `shrink` property. It is based on the recomputing the width of the flex item automatically. `flex-basis: auto`

This will cause the flex-item to be at least as big as its width or height (if declared) or its default content size.

Please don't forget the framework for which I broke down the `flex-shorthand` properties. There's going to be a lot of shorthand stuffs coming up.

Now that you have things coming together, put in a bit of styling to define spacing, colors, etc.

```
body {  
  display: flex;  
  flex-direction: column;  
  background-color: #fff;  
  margin: 0;  
  font-family: Lato, sans-serif;
```

```

    color: #222;
    font-size: 0.9em;
}

footer {
  flex: 0 0 90px;
  padding: 10px;
  color: #fff;
  background-color: rgba(61, 100, 158, .9);
}

```

Nothing magical yet. Here's what we've got now:

HTMLCSSOutput

1html,

2body {

3height: 100%; /\*setting this explicit

4}

5

6body {

7display: flex; /\*flex superpowers a

8flex-direction: column; /\*Stack the

9background-color: #fff;

10margin: 0;

11font-family: Lato, sans-serif;

12color: #222;

13font-size: 0.9em;

14}

15

16main {

17flex: 1 0 auto; /\*fill the available space\*/

18}

19

20footer {

21flex: 0 0 90px; /\*don't grow or shrink - just stay

22padding: 10px;

23color: #fff;

24background-color: rgba(61, 100, 158, .9);

25}

output

Seeing how things are beginning to take shape, you'll make it even better.

Update the html document

```

<main>
  <aside> <!--This represents the sidebar and contained in it are icon sets from font-awesome-->
    <i class="fa fa-bars"></i>

```



```

<i class="fa fa-home"></i>
<i class="fa fa-search"></i>
<i class="fa fa-volume-up"></i>

<i class="fa fa-user"></i>
<i class="fa fa-spotify"></i>
<i class="fa fa-cog"></i>
<i class="fa fa-soundcloud"></i>
</aside>

<section class="content"> <!--This section will house everything other than the sidebar-->
</section>

</main>

```

The listing above is quite explanatory.

For the icon sets, I am using the popular [Font Awesome](#) library.

Having your desired icon is as simple as just adding a CSS class. This is what I’ve done within the aside tag.

As explained earlier, the “main” section above will also be made a flex container. The sidebar (represented by the aside tag), and the section will be flex-items.

```

main {
  flex: 1 0 auto; /*was a flex item*/
  display: flex; /*I just included this! - now a flex container with flex items: sidebar & main co
}

```

Alright, this is getting interesting, huh?

Now you have the main section as a flex container. Deal with one of its flex items, the sidebar.

There’s something cool happening here.

The sidebar has icons stacked vertically. You can make **aside** a flex-container and give it a flex-direction that lets all icons stack vertically and in position. This is explained below.

```

aside {
  flex: 0 0 40px; /*as a flex item: do not grow or shrink. Just stay fixed at 40px*/
  display: flex; /*Now you're a flex-container, you can decide how your flex-items are laid*,
  flex-direction: column; /*Stack my flex-item's vertically...change the default direction*/
  justify-content: space-around; /*Interesting...since direction is changed, this works on t
  align-items: center; /*direction is changed! This affects the horizontal direction*/
  background-color: #f2f2f2; /*make me pretty*/
}

/*font size for the icons*/

```

```
aside 1.fa {
  font-size: 0.9em;
}
```

I have obsessively commented through the code above and now see how pretty everything is laid out.



Super neat with few lines of codes.

Reasonable codes, no messy hacks.

HTML CSS Output

1 html,  
2 body {  
3 height: 100%; /\*setting this explic  
4 }  
5  
6 body {  
7 display: flex; /\*flex superpowers a  
8 flex-direction: column; /\*Stack the  
9 background-color: #fff;  
10 margin: 0;  
11 font-family: Lato, sans-serif;  
12 color: #222;  
13 font-size: 0.9em;  
14 }  
15  
16 main {  
17 flex: 1 0 auto; /\*was a flex item\*/  
18 display: flex; /\*I just included this! - now a flex  
19 }  
20  
21 footer {  
22 flex: 0 0 90px; /\*don't grow or shrink - just stay  
23 padding: 10px;  
24 color: #fff;  
25 background-color: rgba(61, 100, 158, .9);  
26 }  
27  
28 aside {  
29 flex: 0 0 40px; /\*as a flex item: do not grow or s  
30 display: flex; /\*Now you're a flex-container, you  
31 flex-direction: column; /\*Stack my flex-item's ver

output

## Adding content to the main section.

The main content section is empty but don't forget it's the second list-item. The sidebar is first.

Put in some stuff there. What do you think? You may take a look at the finished project again, so you don't lose sight of where we're headed. More importantly, it'd help you understand the next code listing.

```
<section class="content"> <!--This section was empty. Populating it with content-->

<div class="music-head"> <!--First list item: contains music details-->

     <!--Album art-->

    <section class="catty-music"> <!--other details of the album-->
        <div>
            <p>CattyBoard Top 100 Single Charts (11.06.36)</p>
            <p>Unknown Artist</p>
            <p>2016 . Charts . 100 songs</p>
        </div>

        <div> <!--Music controls-->
            <i class="fa fa-play"> &nbsp;Play all</i>
            <i class="fa fa-plus"> &nbsp;Add to</i>
            <i class="fa fa-ellipsis-h">&nbsp;&nbsp;&nbsp;More</i>
        </div>
    </section>
</div> <!--end .music-head-->

<ul class="music-list"> <!--Second list item: Contains a list of all songs displayed-->
    <li>
        <p>1. One Dance</p>
        <p>Crake feat CatKid & Cyla</p>
        <p>2:54</p>
        <p><span class="catty-cloud">CATTY CLOUD SYNC</span></p>
    </li>

    <li>
        <p>2. Panda</p>
        <p>Cattee</p>
        <p>4:06</p>
        <p><span class="catty-cloud">CATTY CLOUD SYNC</span></p>
    </li>

    <li>
        <p>3. Can't Stop the Feeling!</p>
        <p>Catin Cimperlake</p>
        <p>3:56</p>
        <p><span class="catty-cloud">CATTY CLOUD SYNC</span></p>
    </li>

    <li>
        <p>4. Work From Home</p>
        <p>Cat Harmony feat Colla</p>
        <p>3:34</p>
        <p><span class="catty-cloud">CATTY CLOUD SYNC</span></p>
    </li>
</ul>
</section>
```



Uhhh, I added a bit more than the last time but its pretty simple.

I populated the empty content section with a `div` that holds the album art and some details of the catty album. What about the unordered list?

The `ul` holds a list of songs from the album. The song title, artiste, time and “catty cloud sync” are contained in individual paragraphs within the list.

So what are you going to do with styling? See what I did. Can you figure it out?

First off, you should make the main content section a flex container...

```
.content {  
  display: flex;  
  flex: 1 0 auto; /*this makes sure the section grows o fill the entire available space*/  
  flex-direction: column;  
}
```

You should also deal with it's flex-items:

```
.music-head {  
  flex: 0 0 280px; /*Same memo, don't grow or shrink - stay at 280px*/  
  display: flex;  
  padding: 40px;  
  background-color: #4e4e4e;  
}  
  
.music-list {  
  flex: 1 0 auto;  
  list-style-type: none;  
  padding: 5px 10px 0px;  
}
```

`.music-head` holds the album art and other related album details.

Same memo, do not grow or shrink but keep a height of 280px.

Height? Not width? Yes!

The parent element already had the flex-direction switched.

Oh, you're going to need this to be a flex-container later on too. So put in `display: flex`

`music-list` holds the list of songs and it fills up the remaining available space

`.music-list` holds the list of songs and it fills up the remaining available space shared with `.music-head` above.

This doesn't feel very pretty yet but c'mon you're doing great if still following.

Thumbs up.

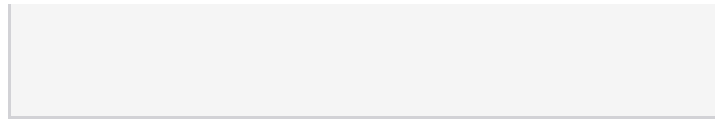
HTML CSS Output

12

css

output

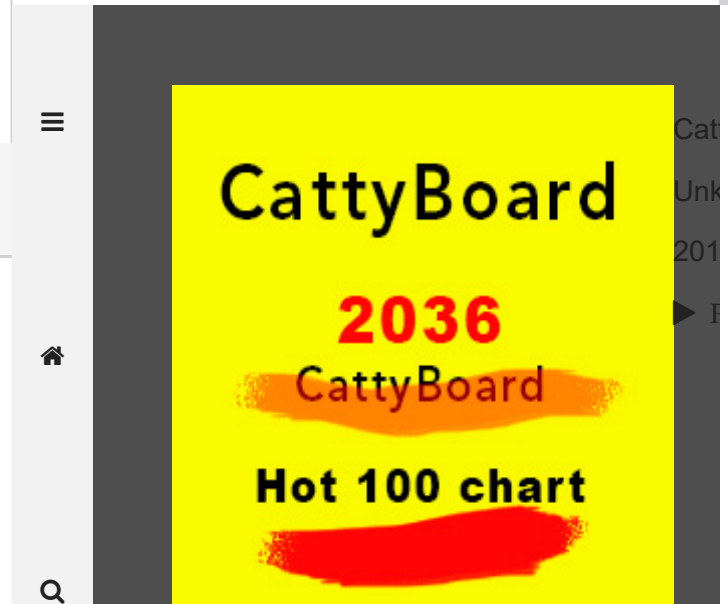
```
1  html,
2  body {
3    height: 100%; /*setting this explicitly is important*/
4  }
5
6  body {
7    display: flex; /*flex superpowers activated! */
8    flex-direction: column; /*Stack the flex-items (main container) vertically*/
9    background-color: #fff;
10   margin: 0;
11   font-family: Lato, sans-serif;
12   color: #222;
13   font-size: 0.9em;
14 }
15
16 main {
17   flex: 1 0 auto; /*was a flex item*/
18   display: flex; /*I just included this! - now a flex container*/
19 }
20
21 footer {
22   flex: 0 0 90px; /*don't grow or shrink - just stay the same height*/
23   padding: 10px;
24   color: #fff;
25   background-color: rgba(61, 100, 158, .9);
26 }
27
28 aside {
29   flex: 0 0 40px; /*as a flex item: do not grow or shrink - stay the same width*/
30   display: flex; /*Now you're a flex-container, you can have flex-items*/
31   flex-direction: column; /*Stack my flex-item's vertically*/
```



There are a few problems here.

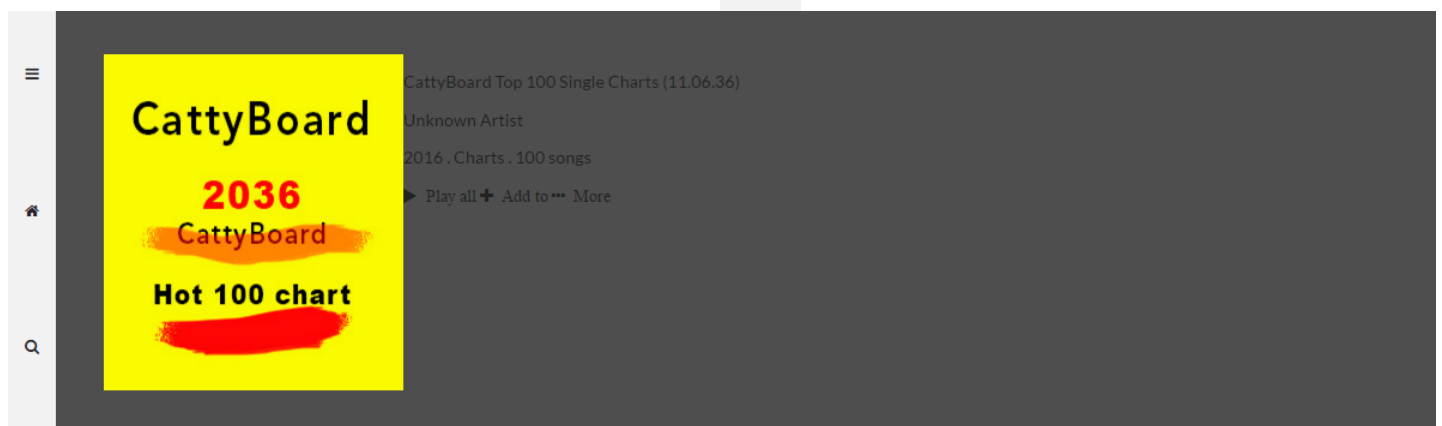
## 1. The list of songs looks terrible.

- 1. One Dance  
Crake feat CatKid & Cyla  
2:54  
CATTY CLOUD SYNC
- 2. Panda  
Cattee  
4:06  
CATTY CLOUD SYNC
- 3. Can't Stop the Feeling!  
Catin Cimperlake  
3:56  
CATTY CLOUD SYNC
- 4. Work From Home  
Cat Harmony feat Colla  
3:34  
CATTY CLOUD SYNC



## 2. The section containing the music art has really ugly looking texts.

- 3. Can't Stop the Feeling!  
Catin Cimperlake  
3:56  
CATTY CLOUD SYNC
- 4. Work From Home  
Cat Harmony feat Colla



Again, I'd walk you through solving these problems

Again, I'd walk you through solving these problems.

Below are the solutions I propose.

## Dealing with the list of songs

Each list of songs contain 4 paragraphs. Song title, artiste, duration, and “catty cloud sync”.

There's got to be a way to put all of this in one line with each paragraph taking up equal space along this line.

Flexbox to the rescue!

The concept here is the same employed in many grid systems.

Translate that to code.

```
li {  
  display: flex; /*Targets each list containing the paragraphs*/  
  padding: 0 20px;  
  min-height: 50px;  
}  
  
li p {  
  flex: 0 0 25%; /*This is the sweet sauce*/  
}
```



You see what's happening there with the paragraphs?

`flex: 0 0 25%`

*“Don't grow or shrink but each paragraph should take up 25% of the available space”.*

The space is shared equally among the paragraphs.

HTML

CSS

Output

```
1  html,  
2  body {
```

CSS

output

```

3     height: 100%; /*setting this explicitly is important*/
4 }
5
6 li {
7     display: flex; /*Targets each list container*/
8     padding: 0 20px;
9     min-height: 50px;
10 }
11
12 li p {
13     flex: 0 0 25%; /*This is the sweet spot*/
14 }
15
16 body {
17     display: flex; /*flex superpowers are here*/
18     flex-direction: column; /*Stack the elements*/
19     background-color: #fff;
20     margin: 0;
21     font-family: Lato, sans-serif;
22     color: #222;
23     font-size: 0.9em;
24 }
25
26 main {
27     flex: 1 0 auto; /*was a flex item*/
28     display: flex; /*I just included this*/
29 }
30
31 footer {

```



# CattyBoard

## 2036

### CattyBoard

### Hot 100 chart

- |                            |                          |   |
|----------------------------|--------------------------|---|
| 1. One Dance               | Crake feat CatKid & Cyla | 2 |
| 2. Panda                   | Cattee                   | 4 |
| 3. Can't Stop the Feeling! | Catin Cimperlake         | 3 |
| 4. Work From Home          | Cat Harmony feat Colla   | 3 |

Style a bit more by giving the lists alternating colors and deal with the “catty cloud sync” label too.

```

li span.catty-cloud {
    border: 1px solid black;
    font-size: 0.6em;
    padding: 3px;
}

li:nth-child(2n) {
    background-color: #f2f2f2;
}

```



So, you’re killing it, and really getting to understand the flex lingo better! This is what you should have now.



```

1  html,
2  body {
3    height: 100%; /*setting this explic
4  }
5
6  li {
7    display: flex; /*Targets each list co
8    padding: 0 20px;
9    min-height: 50px;
10 }
11
12 li p {
13   flex: 0 0 25%; /*This is the sweet sa
14 }
15
16 li span.catty-cloud {
17   border: 1px solid black;
18   font-size: 0.6em;
19   padding: 3px;
20 }
21
22 li:nth-child(2n) {
23   background-color: #f2f2f2;
24 }
25
26 body {
27   display: flex; /*flex superpowers a
28   flex-direction: column; /*Stack the
29   background-color: #fff;
30   margin: 0;
31   font-family: Lato, sans-serif;

```

output

☰

🏠

🔍

🔊

👤

🎵

⚙️

☁️

CattyBoard

2036

CattyBoard

Hot 100 chart

1. One Dance

Crake feat CatKid & Cyla

2

2. Panda

Cattee

4

3. Can't Stop the Feeling!

Catin Cimperlake

3

4. Work From Home

Cat Harmony feat Colla

3

## Using this Technique

This technique is invaluable. You can use it to create unequal content areas. Say, a 2 column view.

One section can take up 60% of the available space, and the other 40%

```
.first-section: 0 0 60%; .second-section: 0 0 40%;
```

You can use this technique for making grid systems.

The second problem will be dealt with now.

## Making the album details text look prettier.

Really simple stuff going on below

```
.catty-music{
  flex: 1 0 auto;
  display: flex;
  flex-direction: column;
  font-weight: 300;
  color: #fff;
  padding-left: 50px;
}

.catty-music div:nth-child(1){
  margin-bottom: auto;
}

.catty-music div:nth-child(2){
  margin-top: 0;
}

.catty-music div:nth-child(2) i.fa{
  font-size: 0.9em;
  padding: 0 0.7em;
  font-weight: 300;
}

.catty-music div:nth-child(1) p:first-child{
  font-size: 1.8em;
  margin: 0 0 10px;
}

.catty-music div:nth-child(1) p:not(:first-child){
  font-size: 0.9em;
  margin: 2px 0;
}
```

and you did it.

You're pretty much done.

HTML

CSS

Output

```
1  html,
2  body {
3    height: 100%; /*setting this explicitly is importa
4  }
5
```

CSS

output

```

6  li {
7    display: flex; /*Targets each list containing the pa
8    padding: 0 20px;
9    min-height: 50px;
10 }
11
12 li p {
13   flex: 0 0 25%; /*This is the sweet sa
14 }
15
16 li span.catty-cloud {
17   border: 1px solid black;
18   font-size: 0.6em;
19   padding: 3px;
20 }
21
22 li:nth-child(2n) {
23   background-color: #f2f2f2;
24 }
25
26 body {
27   display: flex; /*flex superpowers a
28   flex-direction: column; /*Stack the
29   background-color: #fff;
30   margin: 0;
31   font-family: Lato, sans-serif;

```



# CattyBoard

## 2036

### CattyBoard

### Hot 100 chart

1. One Dance

Crake feat C

2. Panda

Cattee

3. Can't Stop the Feeling!

Catin Cimb

4. Work From Home

Cat Harmor


## Quick Exercise

I've saved the footer for you to work on as an exercise.

Try fixing the footer yourself. Just employ the same techniques. You can do this you know?

If you get stuck, you can always check out the full source code for catty music. (You'll find the link in the last lesson)

You may break the entire footer into flex-items too, and get going from there.




## CattyBoard Top 100 Single Charts (11.06.36)

Unknown Artist  
2016 . Charts . 100 songs


▶ Play all + Add to ⋮ More

1. One Dance	Crake feat CatKid & Cyla	2:54	<a href="#">CATTY CLOUD SYNC</a>
2. Panda	Cattee	4:06	<a href="#">CATTY CLOUD SYNC</a>
3. Can't Stop the Feeling!	Catin Cimperlake	3:56	<a href="#">CATTY CLOUD SYNC</a>
4. Work From Home	Cat Harmony feat Colla	3:34	<a href="#">CATTY CLOUD SYNC</a>




If It Ain't Love

Catson Derulo



Item 2a



Item 2b

Wow. I can't believe you got to this point. That's great! You're becoming a Flexbox ninja now.

Next, you will see how Flexbox can help with responsive designs.