SuperHi-Advanced-Week2

http://guides.superhi.com/advanced/jiffy#

Jiffy

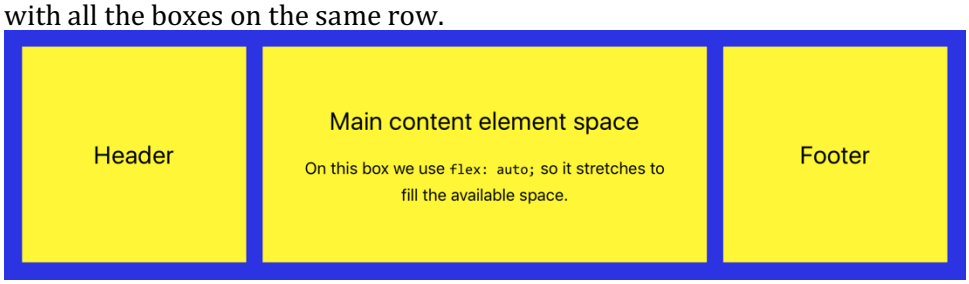
A gif searcher that uses the Giphy API

What we'll learn…

- More flexbox layout techniques
- Some fancy new CSS grid layout
- What are APIs and why they're awesome?
- Getting gifs from the Giphy API
- Creating HTML elements in JS
- Controlling the state of our page using events

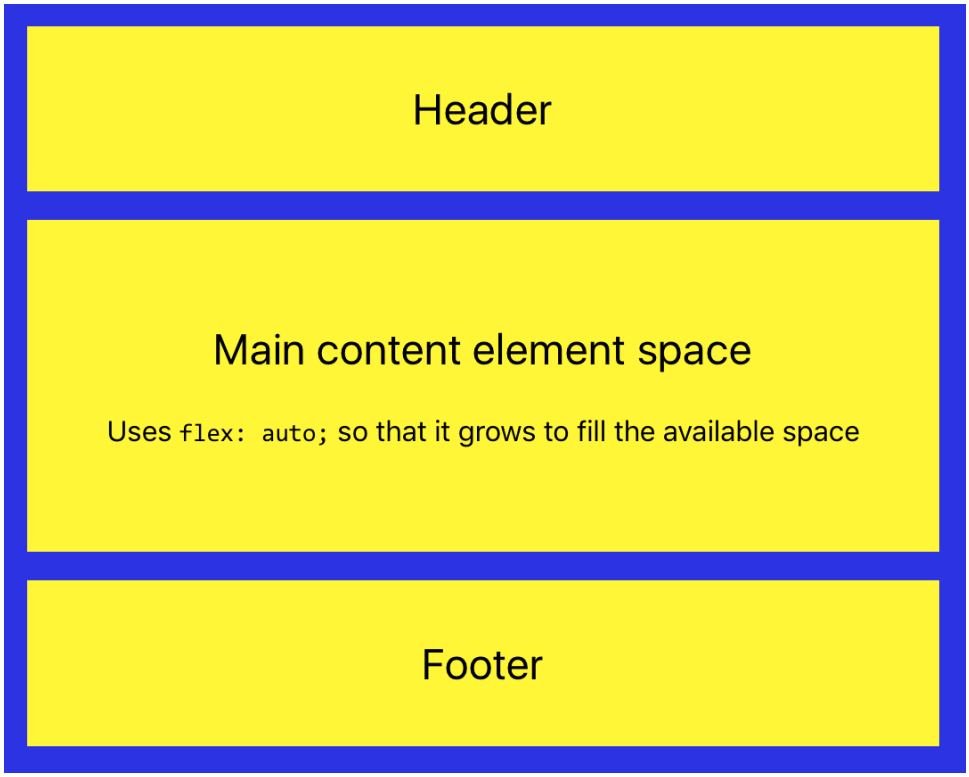
Layout using flexbox

Firstly we use `display: flex;` on the parent element. By default flexbox goes from left to right with all the boxes on the same row.



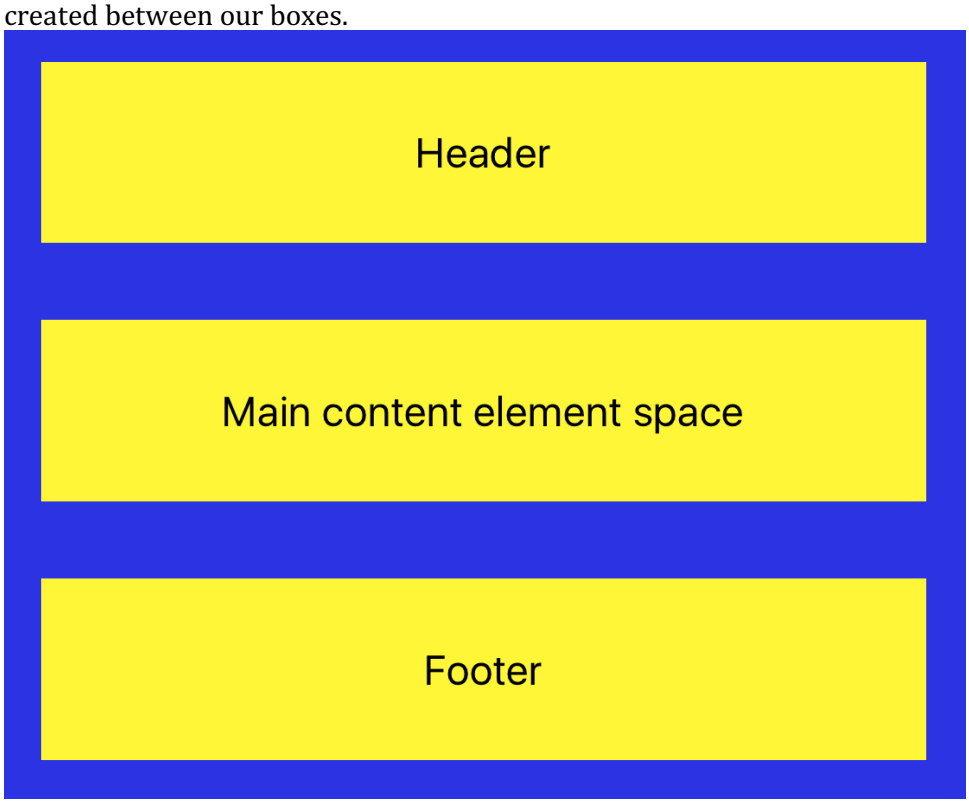
Stacking those boxes

Now we use `flex-direction: column;` on the parent element, so our boxes stack on top of each other instead.



Spacing those boxes

Now we use `justify-content: space-between;` on the parent element, so equal space is created between our boxes.



What is an API?

It's like a harbor. It's designed to move shipping containers of certain shapes and sizes in and out. It might be open to the public, or it might be for VIP use only.
— Taken from Sideways Dictionary
https://sidewaysdictionary.com/#/term/api

APIs in the wild...

SuperHi uses lots of APIs to do all kinds of stuff. For example here's what happens when a new student buys a course:

- **1**
  We call Stripe's API to make a 'payment token' from the user's card
- **2**
  We then use our own backend API to finalise payment and create the user in the database
- **3**
  Then we use Intercom's API to add that user to the weekly course email list
- **4**
  Using the user's email address, we create an auto-invite to Slack using their API
- **5**
  For a successful payment email we use SendGrid's API to send them email

We also do things like use Wistia's API to fetch all of our course videos and show them on our own site.

How we get stuff from Giphy

We're going to use Giphy's API to find us gif images that match a certain keyword. It requires us to make a HTTP `GET` request to their server using a token code for access.
Our search URL to Giphy is going to look like this:
https://api.giphy.com/v1/gifs/search?api_key=api_key&q=doggos&limit=25&offset=0&rating=PG&lang=en
It's broken down into lots of parts:
https://api.giphy.com/v1/gifs/search?api_key=api_key&q=doggos&limit=25&offset=0&rating=PG&lang=en

Looking at Giphy's data

Let's have a look at what data Giphy gives us back


```
[
 {
  "type": "gif",
  "id": "Z3aQVJ78mmLyo",
  "slug": "doggo-Z3aQVJ78mmLyo",
  "url": "https://giphy.com/gifs/doggo-Z3aQVJ78mmLyo"
 },
 {
  "type": "gif",
  "id": "TmzqWbXnmPHBS",
  "slug": "doggo-TmzqWbXnmPHBS",
  "url": "https://giphy.com/gifs/doggo-TmzqWbXnmPHBS"
 },
 {
  "type": "gif",
  "id": "3drMhKAGYF0QM",
  "slug": "am-doggo-3drMhKAGYF0QM",
  "url": "https://giphy.com/gifs/am-doggo-3drMhKAGYF0QM"
 }
]
```

Using fetch for Ajax

Using `fetch` lets us get and send data dynamically behind the scenes using Ajax. This means we can do things without our page reloading.
We're going to take our URL that Giphy gave us and put it into a `fetch` request to get data from the Giphy API behind the scenes.

```
// here we put our URL into fetch
fetch('https://api.giphy.com/v1/gifs/search?api_key=api_key&q=doggo&limit=50&offset=0&rating=PG-13&lang=en')
  // we add a .then() which lets us do something with it when it succeeds
  .then(response => {
    // here we convert the response into json so we can work with it easily in javascript
    return response.json()
  })
  // once the response is in json we can .then() it again to work with the json
  .then(data => {
    // our data is given to us as json and we can do something with it
  })
  .catch(error => {
    // lastly we can use .catch() to do something in case our fetch fails
  })
```

Handling the data in Javascript

Once we have our data returned to us as JSON, we can then navigate through it and pick out the bits we want. Let's make a variable for our array of images…

```
// our json data looks like this
{
  data: [
    // an array of all our images
  ],
  meta: {
    // info about the request
  },
  pagination: {
    // how many pages of data we have
  }
}
// here we make a data variable for our array of images
const data = json.data
```

Grabbing the first image

Again navigating through our data, let's grab the first image from our array and an image URL from it.

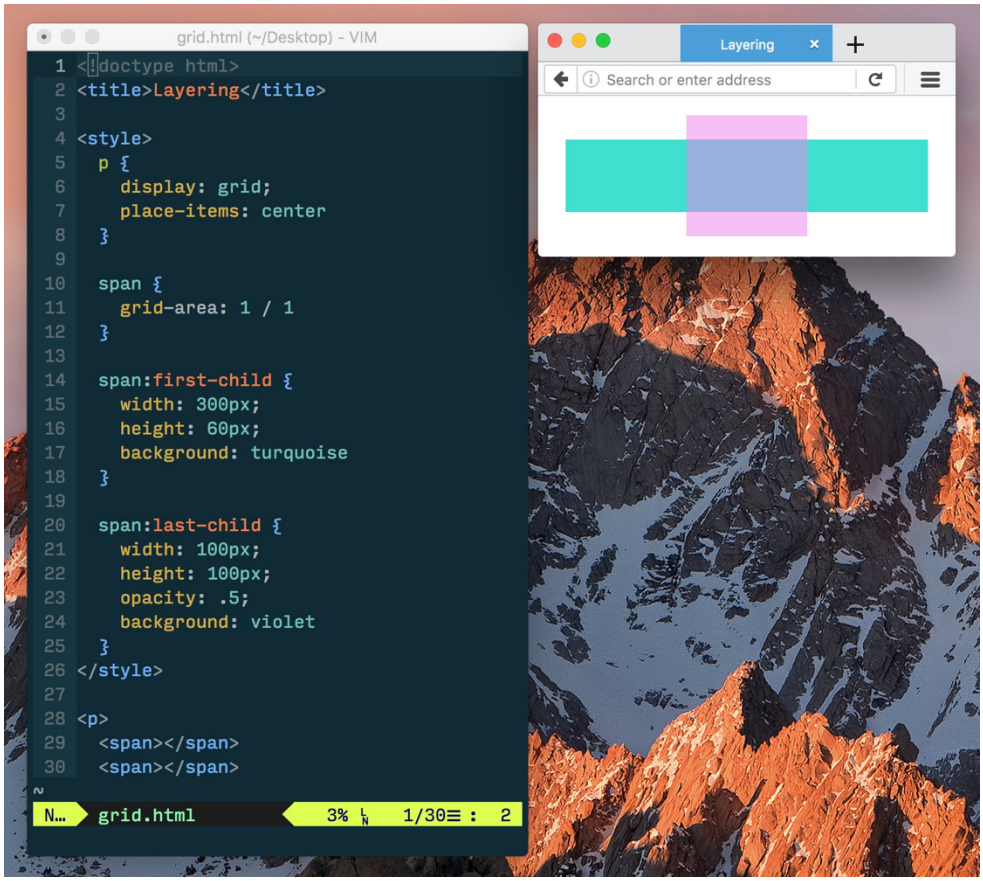
```
// our array of images looks like this
[
 {
  id: "Z3aQVJ78mmLyo",
  images: {
   original: {
    mp4: "https://media2.giphy.com/media/Z3aQVJ78mmLyo/giphy.mp4"
   }
  },
  title: "title",
  // etc...
 }, {
  // our second image
 }, {
  // our third image
 }
]
// here we make a variable for the downsized_large image from the first image
const src = data[0].images.original.mp4
// this gives us back "https://media2.giphy.com/media/Z3aQVJ78mmLyo/giphy.mp4"
```

Creating a video element

Let's now create a video element on our page, set its URL to be our src along with autoplay and loop attributes, and then add it into our HTML.

```
// our src variable from the previous step
const src = data[0].images.original.mp4
// here we grab our videos element from the page
const videos = document.querySelector('.videos')
// we create a video element using document.createElement
const video = document.createElement('video')
// we set the src to our src url varable
video.src = src
// we make it so our video autoplays
video.autoplay = true
// add the loop attribute to our video
video.loop = true
// lastly, we add our video inside of our .videos element
videos.appendChild(video)
```

Using CSS grid for stacking elements



CSS Grid Layout is a powerful new layout system available in CSS. It lets us place items onto a grid that smartly adapts to screen size and solves a lot of common layout issues.

Using CSS grid for stacking elements

Before, using absolute position

```
/* we'd set a height on our parent
and give it relative position */
.parent {
  height: 500px;
  position: relative;
}

/* we position the child absolutely
inside the parent element */
.child {
  position: absolute;
  left: 50%;
  top: 50%;
  /* offsets our box to it appears in the middle */
  transform: translate(-50%, -50%);
}
```

After, using CSS grid layout

```
.parent {
  /* enable css grid */
  display: grid;
  /* tell it to center all child items */
  place-items: center;
}
.child {
  /* tell each child to take up full
  width and height of the grid area */
  grid-area: 1 / 1;
}
```

Keyboard events on inputs

Using `addEventListener` and the `keyup` event, we can watch for when users type in our input.

```
// we grab the input from the html
const searchEl = document.querySelector('.search-input')
// we listen out for 'keyup' events
searchEl.addEventListener('keyup', event => {
  // now we can do something with each 'event' snapshot like getting the key
  console.log(event.key) // gives us the name of the key
})
```

Separating our event function

Another way we can write our `addEventListener` code is to separate out the function and call to it by its name. This is called a callback function.

```
// we grab the input from the html
const searchEl = document.querySelector('.search-input')
const doSomething = event => {
  // here we check if the search box has more than 2 characters in
  // and also whether the enter key has been pressed using && (and)
  if (searchEl.value.length > 2 && event.key === 'Enter') {
    // do something only when both of the conditions match
  }
}
// we listen out for 'keyup' events, and fire the doSomething function
searchEl.addEventListener('keyup', doSomething)
```

Controlling our page state

Our page has three states it can be in. These are controlled using Javascript and CSS by adding and removing classes from our body element.

`.show-hint`

- Shows the search hint at the bottom and hides the hint
- Only shown when we have two or more characters in the search box
- Changes the hint to say 'Hit enter to search'

`.loading`

- Shows the loading spinner at the bottom
- Displays every time the fetch function runs and stops when it finishes
- Uses a `toggleLoading` function to change between the two states

`.has-results`

- When we have results on our page
- Displays the close button at the top and hides the title
- Hint updates to say 'Hit enter to see more'

Adding our search term into fetch

We can make our `fetch` code a bit smarter by wrapping it up inside a function that we can pass a search term to. Using the URL Giphy gives us, we'll combine it to make a proper search that'll be dynamic every time.

```
// separate our API_KEY as a constant at the top of the file
const API_KEY = 'o7IyuSKkLiR728rSCOE3Pov4refIv10F'
// wrap up our fetch code into a searchGiphy function
const searchGiphy = searchTerm => {
 // we now have searchTerm as a variable which we bake into our giphy url
 fetch(

`https://api.giphy.com/v1/gifs/search?api_key=${API_KEY}&q=${searchTerm}&limit=50&offset=0&rating=PG-13&lang=en`
 )
  .then(response.json())
  .then(json => {
   // do something with our code
  })
}
// we can then call our function like this
searchGiphy('doggos')
```

Check when our video loads

Video elements have their own special events that we can listen out for. We want to check when our video has loaded before making it visible on the page.

```
video.addEventListener('loadeddata', event => {
 // the loadeddata event fires when the video has loaded
})
```

Transition the video elements

```
video.addEventListener('loadeddata', event => {
 // add a visible class to trigger our transition effect
 video.classList.add('visible')
 // change the hint text
 hintEl.innerHTML = `Hit enter to see more ${term}`
 // change our page state to has-results
 document.body.classList.add('has-results')
 toggleLoading(false)
})

/* our video is initially invisible and 0 in scale */
.video {
```

```
  opacity: 0;
  transform: rotate(0deg) scale(0);
  transition: all 0.5s ease;
}
/* adding the visible class makes it full size and faded in */
.visible {
  opacity: 1;
  transform: rotate(0deg) scale(1);
}
```

Clearing our search results

Inside of our clear function we are going to be removing the `.has-results` class from our body element and then clearing out all of the results by setting their `innerHTML` to be empty.

```
const clearSearch = event => {
  // remove the results state
  document.body.classList.remove('has-results')
  // empty the videos element
  videosEl.innerHTML = ''
  // empty the hint text
  hintEl.innerHTML = ''
  // empty the search input value
  searchEl.value = ''
  // focus the cursor back into the search input
  searchEl.focus()
}
// fire the function when we click our clearEl
clearEl.addEventListener('click', clearSearch)
```

Listening for events anywhere

If we use `addEventListener` on our `document`, we can listen and respond to events that occur anywhere on our page. Let's wire up a `keyup` event that looks for when we use the `Escape` key.

```
document.addEventListener('keyup', event => {
  if (event.key === 'Escape') {
    clearSearch()
  }
})
```