Part 1

> @author: @Guillaume Ferron
> @date: 04.03.2019
> @version: v1.0.0

```
TLDR;

- Reminder of flex
  - flex direction
  - justify content
  - align items
- Reminder of bootstrap classes
  - xs, sm, md, lg
  - container, row, col-
  - m, p, mx, my, px, py, mt, mr, ml, mb, pt, pr, pl, pb
  - mx-auto, ml-auto, mr-auto
  - d-flex, justify-content-..., align-items-...
  - d-none, d-block
  - position-relative, position-fixed, position-absolute
  - color, bg
  - border
  - w-..., h-...
  - text-uppercase
- Working with Font Sizes
  - rem, px, em : the good, the bad, the ugly
  - set the root font size as a percentage for accessibility and make the font sizes fluid
  - best practices
- Workflow
  - Mental image of the layout
    - Show example of layout and explain it
  - Skeleton
    - Build that mental image
  - Pixel perfecting with InVision
    - CSS now
  - Best practices
    - No duplicated code.
    - Try to code so that it makes sense.
    - As little css as you can.
    - As little media queries as you can.
    - Choose one method for margin and positioning. Top and left.
    - We mainly use col and col-md as we usually put phones and tablet in the same basket.
    - Keep margins coherent. If you use ml-1 between 2 elements, use mt-1 between others so that it makes sense.
    - If you have several time the same button for instance use css
- Conclusion
  - Always double check copies and layouts
  - If you don't know or you're not sure, ask
  - Front end is supposed to be quick paced
```

Flexbox

Cheatsheet here

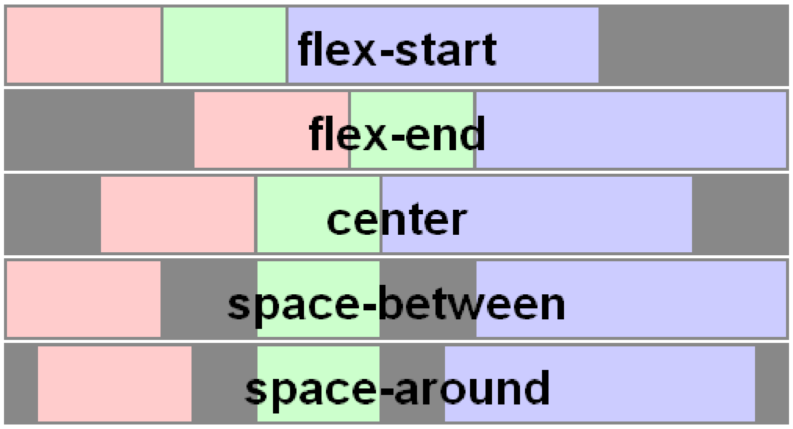
Flexbox is a module built in newer browsers that provides a more efficient way to lay out, align and distribute space among items in a container, even when their size is unknown and/or dynamic.

- `display: flex` is a property that sets a div to be a flex container.
- `flex-direction` can be set to `row (default)` or `column` if you want the children of the container to be layed out as a row or a column.
- `justify-content` sets the primary direction (row or column depending on whtat you set for `flex-direction`) spacing.
- `align-items` is the same as `justify-content` but on the secondary direction.

The two properties can be set to the same values with different effect depending on the direction you gave. If `flex-direction` is set to row, `justify-content` will set the horizontal axis, `align-items` the vertical. For the column, it will be the opposite.

Below, the effects will be decribed as `flex-direction: row` / `flex-direction: column`

Value	justify-content	align-items
flex-start	Everything on the left / top	Everything on the top / left
flex-end	Everything on the right / bottom	Everything on the bottom / right
center	Centered	Centered
space-between	Maximize space between the elements	Maximize space between the elements
space-around	Equal space around the elements	Equal space around the elements



Bootstrap

Cheastsheet Here

Bootstrap is a CSS toolkit for quick layout prototyping based on flexbox, a handling of screen sizes and columns. Reminder: we only use Bootstrap 4. Mostly, it helps writing quick code for webpages without extra CSS ▶️ better readability, maintenance and efficiency.

Screen Sizes

- `nothing` (for phones - screens less than 768px wide)
- `sm` (for tablets - screens equal to or greater than 768px wide)
- `md` (for small laptops - screens equal to or greater than 992px wide)
- `lg` (for laptops and desktops - screens equal to or greater than 1200px wide)

The system is based on a bottom to top parsing, meaning what you apply to xs will be applied to all the sizes above.

> Example: a class used for sm size will also be used automatically for md and lg sizes. On the other hand, a class applied to md won't be applied to smallest screen size and sm.

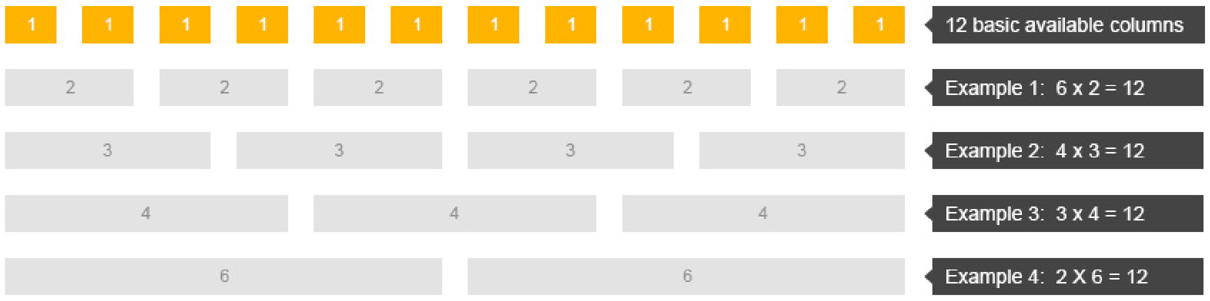
Keep in mind that if you ever have to do media queries, always use that bottom-to-top standard system with the bootstrap sizes.

For almost every classes reminded down below, the screen size can be added to it to have it applied only in certain sizes.

Columns

Bootstrap has three types of layouts that you use by adding its name in the class of the html tag.

- `container`. Higher hierarchy it is what you use to create sections or blocks in a page. A container can contain several containers or rows.
- `row`. Middle class, it is by default `display: flex` and `flex-direction: row` plus some extra padding and margin. A row can contain several col-, containers or rows..
- `col-`. Based on the column grid, it sets the width of the elements.
    - A grid has 12 columns so you can have up to that number of "columns", knowing that `col-1` will be 1 column, `col-5` will be 5 columns, `col-12` 12 columns... So you can have 12 `col-1`, or 6 `col-2`, or 4 `col-3`, or 2 `col-5` and 1 `col-2`...
    - You don't have to fill the 12 columns.
    - It is the core of bootstrap, it is what helps prototyping responsive pages. You can add the screen sizes to it, such that `col-12 col-md-6 col-lg-4` will make an element 12 columns wide on phone screens, 6 on small laptops and 4 on big screens.



Margins & Paddings

It is a general rule of thumb for development, but always try to use bootstrap classes instead of css code.

- `m` & `p` : classes used for margin and padding. It can be set to values from 0 to 5, 0 being none and 5 being the most. Adding the `m-3` or `p-3` classes will add margins or padding top, bottom, left and right to the div.
- `m-0` & `p-0` : Important to force the element to have 0 margin or padding.
- `my` & `py` : Same system as above, only for bottom and top.
- `mx` & `px` : Same system as above, only for left and right.
- `mb` & `pb` : Same system as above, only for bottom.
- `mt` & `pt` : Same system as above, only for top.
- `ml` & `pl` : Same system as above, only for left.
- `mr` & `pr` : Same system as above, only for right.
- `ml-auto`: Will push the div to the right as far as it can.
- `mr-auto`: Will push the div to the left as far as it can.
- `mx-auto`: Very important, allow to center an element in a row.
  - Remember that all of those classes can be used with screen sizes.

Flexbox

If you want to use flexbox, you can use the bootstrap classes instead of creating CSS. As said above, the rows are based on flexbox, that means that you can play with it.

- `d-flex`: display: flex.
- `justify-content-...`: justify-content property.The ... can be center, around, between, start, end.

- `align-items-...` : align-items property.The ... can be center, around, between, start, end.
  - If you apply the justify-content or align-items classes to a row, it will work as row has a d-flex prebuilt.
  - Remember that all of those classes can be used with screen sizes.

Display & Positionning

You can use the classes to display and position some elements

- `d-block` : display:block.
- `d-none` : display: none. Very useful to hide some elements in small screen sizes for instance by using `d-none d-sm-block` that will hide an element in small sizes but let it appear on sm and above sizes.
  - Those classes are very useful for responsiveness as it happens a lot that you need to hide some element in certain sizes, and show in others.
- `position-relative` , `position-fixed` , `position-absolute` : position property.
  - Keep in mind that we try to keep the number of absolute and fixed positionned elements as low as possible as it gets them out of the page thread.

Width & Height

- `w-…` & `h-…` : Used to set the width and height to a percentage. Can be 25, 50, 75 or 100.
  - Very useful for `h-100` that we use often to make sure an element is going to fill the parent's height.

Miscellaneous

**There are plenty of other classes but these up top are the main ones that we use on a daily basis. **

- `text-…` & `bg-…` are used to set the color for text and background. Poorly used in client projects, but you can have look here
- `border-…` sets a border to a thickness and style.
- `text-uppercase` , `text-lowercase` , `text-truncated` . Pretty explicit, the last one works when you set a width or max-width sot that the text will end with ... if too long.

Emmet

Cheatsheet Here

Emmet is a great way to make quick HTML code by using selectors and tab. The indication below are made for divs, but they work with any html tag name as input, button, a, section… It also includes prebuilt attribute taht are required as href for a. To enable it, follow guide to install the plugin in your code editor.

- To create a `div` with a class, use `.` : `div.class-name` will create `<div class= "class-name"></div>` when pressing tab.
- To create a `div` with an id, use `#` : `div#id-name` will create `<div id="id-name"></div>`
- To create adjacent `div`, use `+` : `div+div` will create `<div></div><div></div>`
- To create children, use `>` : `div>div` will create `<div><div></div></div>`
- To add content in a div, use `{}` : `div{Test}` will create `<div>Test</div>`
- To multiply the number of `div`, use `*` : `div*3` will create `<div></div><div></div><div></div>`
- To go up in the arborescence of the html, use `^` : `div.lvl-1>div.lvl-2^div.lvl-1-aswell` will create `<div class="lvl-1"><div class="lvl-2"></div></div><div class="lvl-1-aswell"></div>`
    - Overall, you can add custom attributes and whatnot, but the main ones you will use are above.
    - You can mix all of these property, try to practice a bit to be even quicker.

Working with Font Sizes

Font sizes might be the most tedious part of front end. As soon as it comes to responsiveness, it never works as expected, we always end up doing crazy media queries to implement proper breaklines and sizes. We'll see below how to create a better system for copies handling.

The good, the bad and the ugly: rem, em & px

A first lesson is regarding units. Three types exist and are `rem`, `em`, and `px`. **rem and em should not be used for dimensions. Use vw or % if you want to make a dimension responsive.**

- `rem` : the good 👍 : allows to have a font size relative to the root HTML element. You use it as a multiplying factor: `2em` is equivalent to `20px` if the root font size is `10px`. Can get messy, and no one properly changes the root font size with responsiveness. But if correctly used, it is a powerful ally in the font size responsiveness war.
- `px`, the bad 👎 : most commonly used. Not the best as it is a fixed absolute value and you will need media queries to implement responsiveness.

- `em`, the ugly ⛔ : it does the exact same thing as `rem`, but with respect to the first relative parent. Which means nested font sizes. If you implement a `div` with a font size of `10px`, a `2em` child, and a grandchild of `1.5em`. The grand child will get a font size of 10 * 2 * 1.5= `30px`. That's already a calculation, so picture it in a 50 level dom tree (which is not that hard to believe). You could see the good in the bad as you could cluster each section to not overlap with an other section's font sizes. The issue with that is that usual websites use uniform font sizes throughout all sections, and you will find yourself trying to hack your way around it, like doing `2em`, then `0.5em` for its child to go back to the main font size...

Accessibility & Responsiveness

As said above, the `rem` is relative to the root font size.

> If the root font size is 10px, 1.5rem is 15px, 2rem is 20px...

What is good practice would be to set the root font size to `10px`, so that any calculation would be simplified to a simple division by 10.

What you need to do at the beginning of a project is then to set your tags `h1`, `h2`, `h3`, `h4`, `p`... to values in rem. Some of them will get overwritten as you might need a special font size for a main title or so, but in the long run you will save time.

Now that uniformization has been tackled, what about accessibility ? Indeed if you set the root to `10px`, a zoom won't change the value of `2rem`.

The overall best practice is to set the HTML root font size to a percentage of the browser default font size, usually being `16px`. By setting `font-size: 62.5%` , you will then get a root HTML font size of 16 * 0.65= `10px` . We then get the `10px` that simplifies the calculation as what we talked about above.

Finally, responsiveness is the main issue with copies. What I would recommend is to use `calc([minimum size] + ([maximum size] - [minimum size]) * ((100vw - [minimum viewport width]) / ([maximum viewport width] - [minimum viewport width])))`

when you're setting your tags font size values. That will allow to have a *fluid* implementation that will depend on the browser size. Keep in mind that for really tight design projects, you will have to overwrite those font sizes sometimes to make sure you're having proper breaklines.

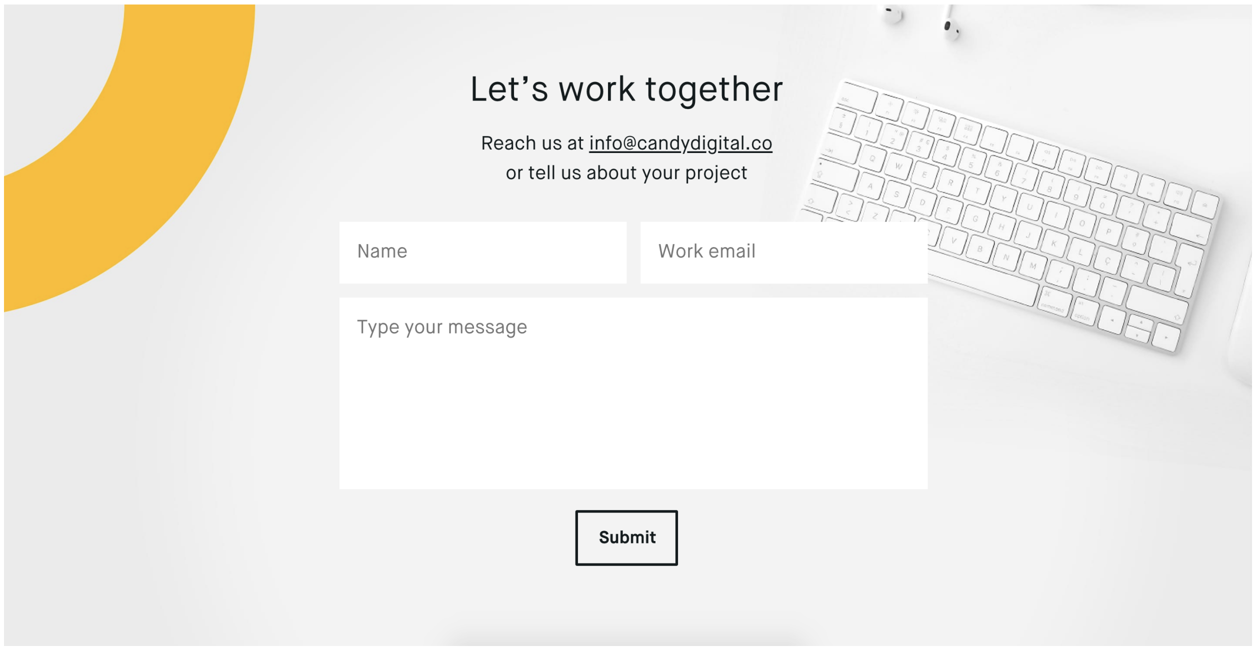
Best practices

- Do NOT set manual line breaks with `br`. It will lead to a terrible responsiveness workflow among the team as those line breaks won't be the same on all devices.
- To have proper break lines, play around with the container width. You can often exact breaks according to design by playing with the width of the container.

Workflow

This section will sum up a workflow to quickly achieve front end tasks. The goal is to improve efficiency and make tasks less tedious. This is NOT the perfect workflow as no such thing exists, but one that works pretty well.

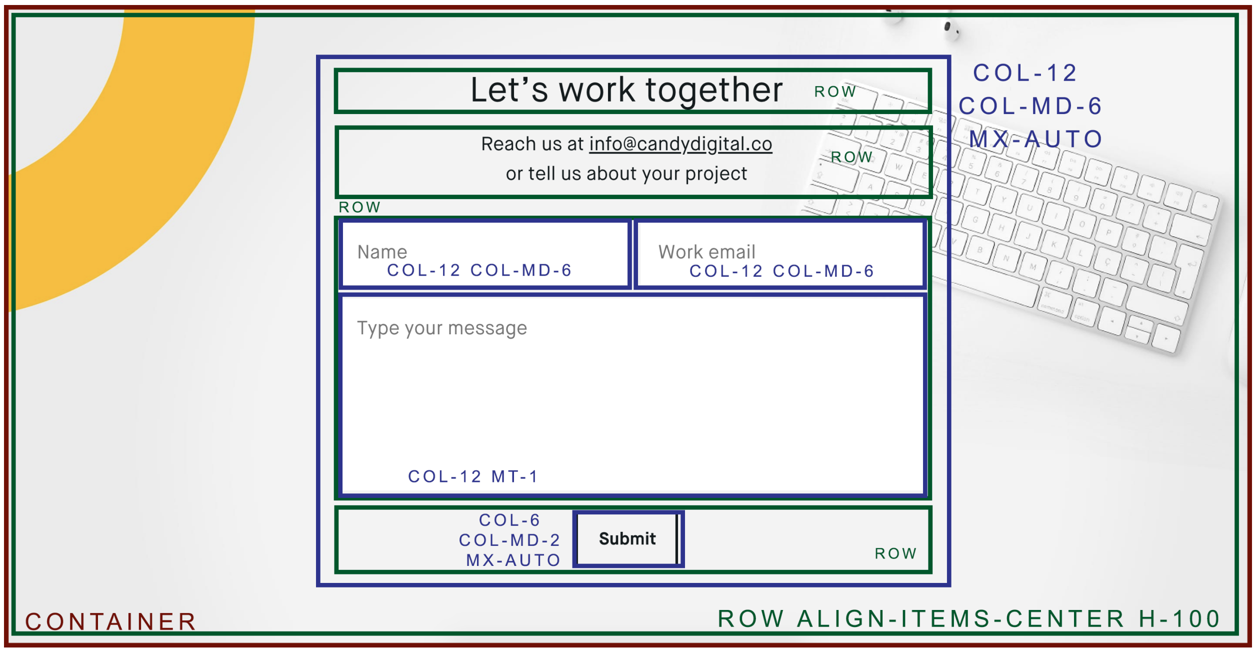
> We will use an example section from CandyDigital website throughout that whole workflow.


Let’s work together

Reach us at info@candydigital.co
or tell us about your project

1. Mental Layout

The bigger job is prior any type of code. That should be quick, but this is needed to start and head toward the correct direction. You should be able to see what bootstrap elements you're going to use to build the skeleton.Below would be the breakdown of the section :



Thought Process

1. First, the main parent element is going to be a `container`.
2. Inside it, a `row` will contain the `col-` that will contain the content. that row will also get the `align-items-center h-100` class to make it 100% height of the container, and center vertically the elements in it.
3. The `col` inside the `row` should be set as `col-12 col-md-6 mx-auto` because we want to have the form taking the whole screen on phones (makes sense, it'd be to small otherwise), and to be half the screen and centered on bigger screens.
4. Inside the `col` ware going to use row that will take the whole width, and since the first two only have text, we will use a `p` tag in it class.
5. The `p` tags should get the `mx-auto text-center` class to center it, and `col-12 col-md-5 text-center` for the second one to restrict its width.
6. The form part should be a row as well, and contain two `col-12 col-md-6 `. We want the inputs to be half of the row on big screens and be full width on small screens.
7. The textarea will be `col-12 mt-1`. The `mt-1` is to have a bit of spacing between elements.
8. Lastly, the last `row` will contain a `button` which will have `col-6 col-md-2 mx-auto` to have the button not too small on small screens and not too big on big screens, and center it.

Conclusion

Keep in mind that this is a mental process, so might not be perfect, even some wrong classes might have been added but this is the general flow.

One thing that bootstrap doesn't handle well is heights, so what will be done will require no CSS code, apart from height settings and text styling.

All the elements in the `col` should get a `mt-2` to get proper spacing between them.

---

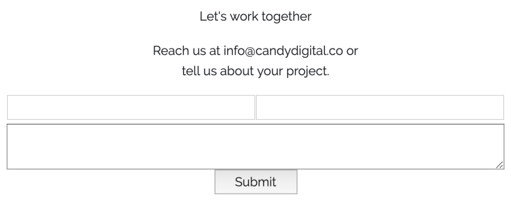
2. Build

That part is the actual coding of that mental image. As you'll gain more experience, the first step will blur in the second and you'll code the skeleton as you're thinking about how to do it. We will disregard the image for now, but if we follow our mental process, we should get :

```
<section class="container">
  <div class="row align-items-center h-100">
    <div class="col-12 col-md-6 mx-auto">
      <div class="row">
        <p class="mx-auto text-center">Let's work together</p>
      </div>
      <div class="row">
        <p class="col-12 col-md-6 mx-auto text-center">
          Reach us at info@candydigital.co or tell us about your project.
        </p>
      </div>
      <div class="row">
        <input class="col-12 col-md-6"/>
        <input class="col-12 col-md-6"/>
        <textarea class="col-12 mt-1"></textarea>
      </div>
      <div class="row">
        <button class="col-6 col-md-2 mx-auto">
          Submit
        </button>
      </div>
    </div>
  </div>
</section>
```

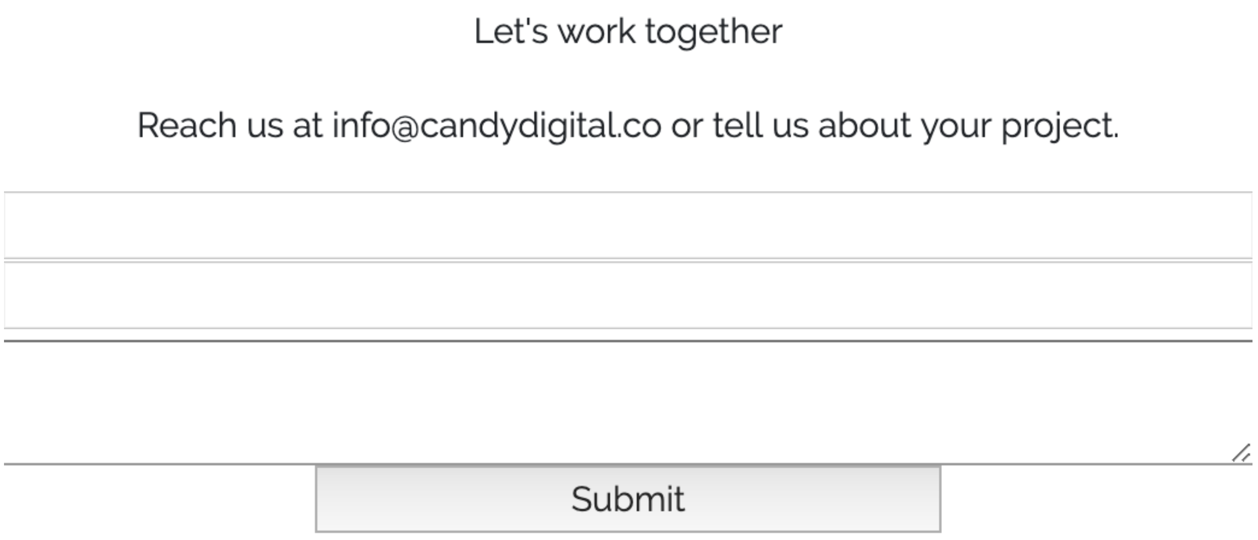
> You can also do it with emmet, try : `section.container>div.row.align-items-center.h-100>div.col-12.col-md-6.mx-auto>div.row>p.mx-auto.text-center{Let's work together}^div.row>p.col-12.col-md-6.mx-auto.text-center{Reach us at info@candydigital.co or tell us about your project.}^div.row>input.col-12.col-md-6*2+textarea.col-12.mt-1^div.row>button.col-6.col-md-2.mx-auto`

What will be displayed is that :


Let's work together

Reach us at info@candydigital.co or
tell us about your project.

On mobile :


Let's work together

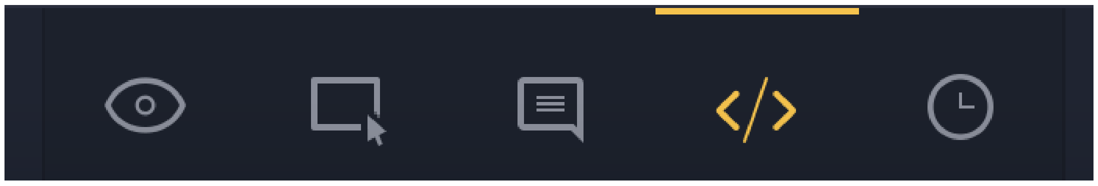
Reach us at info@candydigital.co or tell us about your project.

Of course we're far from done, but the layout is here.

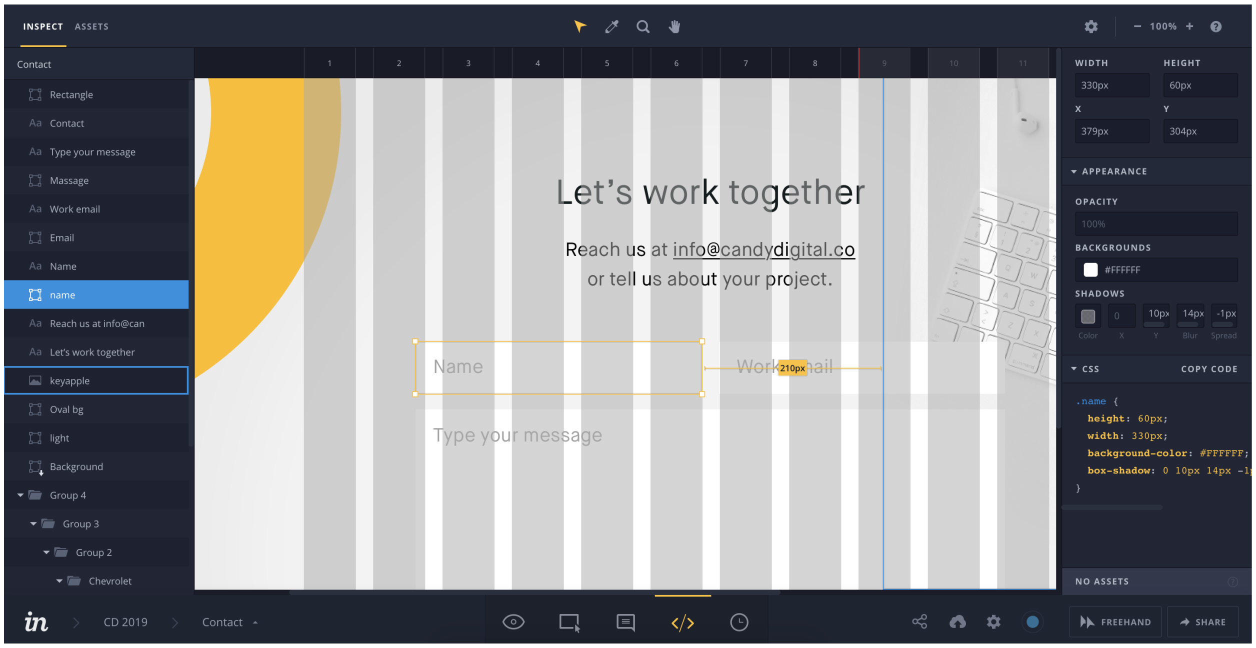
3. InVision

You can now create CSS code to style the form and tweak some things. You should use the InVision to export CSS codes and make sure that the styling are right.

When clicking on a screen in InVision, switch to inspect mode :



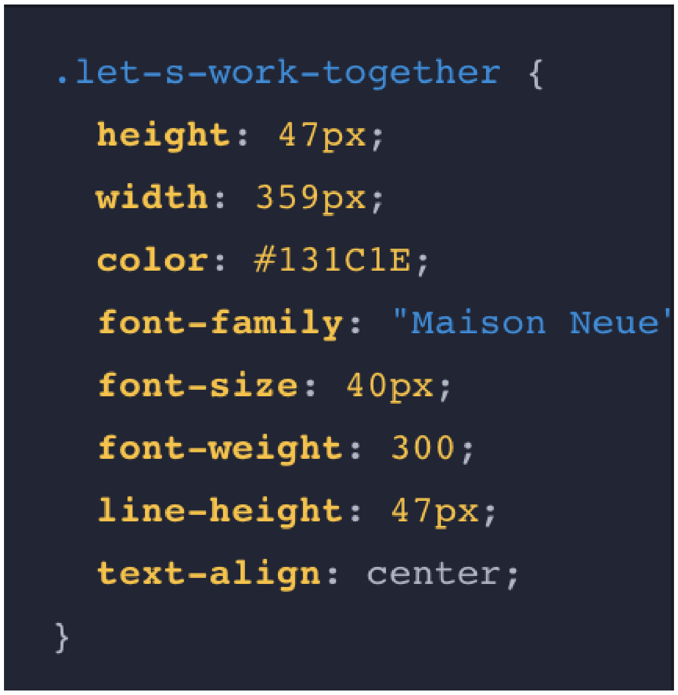
When clicking on an element (here an input), you should see CSS panel on the right :



WIDTH 330px HEIGHT 60px
X 379px Y 304px
APPEARANCE
OPACITY 100%
BACKGROUNDS #FFFFFF
SHADOWS Color ▨ X 0 Y 10px Blur 14px Spread -1px
CSS COPY CODE

```css
.name {
  height: 60px;
  width: 330px;
  background-color: #FFFFFF;
  box-shadow: 0 10px 14px -1
}
```

The width shouldn't be copied here for instance as it is already set by `col-6`. It goes the same for texts :



```css
.let-s-work-together {
  height: 47px;
  width: 359px;
  color: #131C1E;
  font-family: "Maison Neue'
  font-size: 40px;
  font-weight: 300;
  line-height: 47px;
  text-align: center;
}
```

Overall that step is for styling and pixel perfecting. This is an important part as it makes the product look slick and professional.

Overall Best Practices

- **NO DUPLICATED CODE**. I can't stress that enough. If you ever find yourself writing twice the same thing, that means that it should be refactored. Don't be lazy and do it, you will save time in the long run, and the more you do it, the better you will get at predicting those cases and making them generic right of the bat.
- **TRY TO CODE IN A SMART WAY**. If several elements in a section have the same margins right and left, maybe you should just set the section to have padding left and right for example.
- **AS LITTLE CSS AS YOU CAN**. Most of the time, the big layout and overall styling can be done through bootstrap classes.
- **EVEN LESS MEDIA QUERIES**. Those make code maintenance a drag, try to use the bootstrap size classes instead.
- **CHOOSE ONE METHOD FOR POSITIONING**. I personally use the *Latin Waterfall* method (name completely made up by myself). It means from top to bottom, and left to right. If you have to position an element, do it in regards to the element on the left, and the one on top. That means that you would use `margin-top` instead of `margin-bottom` if you have a choice, and same for `margin-left` and `margin-right`.
- **USE `COL` and `COL-MD`**. We usually put phones and tablet in the same basket.
- **BE COHERENT AND UNIFORM**. If you're using a padding for a section, other sections should probably have the same. If you have margins between elements, it should be the same.
- **BE GENERIC**. If a complicated styling is created as per a design, then you should create a class that can be reused throughout your webpage.

Conclusion

- Always double check copies and layouts.
- If you don't know or you're not sure, ask.
- Front end is supposed to be quick paced.