**1. We're looking for people with a real passion for collaboratively creating great software. Please give an example of a software component you have designed and written from concept to deployment, outlining the steps you took.**

-I had to create a new component for the kingsonline bootstrap tool, used by the content developers

to generate html letting them copy it.

-The component had to be based on their guidance: https://keats.kcl.ac.uk/mod/book/view.php?id=4599788

-I followed the previous pattern, to keep consistency.

I needed to first create a card containing:

-an image (optional to show or hide, when not an image src added, display a template). Default show

-a header (optional to show or hide). Default show

-text (Always show)

-a connector to the following card, with a label (optional to show or hide). Default hide

-Arrow down

-Arrow up

-Double arrow

-Relation

The card had to give the option to add more cards if needed(max of 8) and a default number of 3.

There had to be:

-An options area form.

-A dropdown with forms for all the cards to fill up.

-A preview area(showing a preview of the component built) where the users could see the changes on the go

-A code area(showing the html markup) where users could see the changes on the go

The initial task was to show or hide all images and headers. I suggested to do it individual. Final version:

https://iddkingsonline.com/ko-bb-dev1/builder.php# (select process builder)

**2. Using the example that you provided above, tell us about a significant decision you made to solve a technical challenge. Give details of technologies that you chose and why you chose them.**

Initially, the code was all in one file. The whole system was structured that way. Depending on the menu selection, the code would generate the forms and preview and code panels for each one of the components. All of them linked to one only .js file on jquery.

I had to then re-think the way every function worked to keep a bit of consistency for the way everything was set up.  
  
The approach seemed to be working fine with the first component. As I was adding new ones, I kept finding new ways to implement my code to make it more reusable and cleaner.  
  
Most of my code was done in JavaScript. I had to manipulate the DOM and every component was limited to the client side.  
  
  
**3. Using the example that you provided above, tell us about how you ensured your software was fit for purpose and of high quality. What did you learn and what would you do differently next time to do a better job**?  
  
For this project, I assisted meetings every week to share suggestions with every department involved and to keep track of its progress.   
  
Communication is the best way to make sure everything and everyone is keeping up with the requirements. It also gives you a starting point to make it better  
  
I have learned that most of the times, simple is good. Keeping communicative at all times. Next time I have to think, plan, structure, test and then code.

**DECIMAL TO ROMAN APPROACH**

1: I, 4: IV, 5: V, 9: IX, 10: X, 40: XL, 50: L, 90: XC, 100: C, 400: CD, 500: D, 900: CM, 1000: M

I need a number between 1 and 3999 (restricted to this)  
  
X = number  
  
1.- Compare the number against the list above.

400 >= **222** >= 100 => C (X = X – 100)

2.-check if X >= aboveList.keyValue (maybe create another list with the matches)

3.- Find the maximum value from that list and subtract it from X till X = 0  
  
4.- For every subtraction, a symbol/sign will be added to another container.

I also thought about de-structuring the number first and then work with the result. For example 358:   
  
 100 + 100 + 100 + 50 + 5 + 1 + 1 + 1

**CODE**

class Number {

constructor(number) {

this.number = number;

}

toRoman() {

if ( !(this.number > 0 && this.number < 4000) || (isNaN(this.number)) ) {

return 'The value must be a number between 1 and 3999'

}

const romanRelation = {

1: "I", 4: "IV", 5: "V", 9: "IX", 10: "X", 40: "IL", 50: "L",

90: "XC", 100: "C", 400: "CD", 500: "D", 900: "CM", 1000: "M"

}

const keys = Object.keys(romanRelation) // decimal values

let roman = ""

let n = this.number

while (n > 0){

let max = Math.max(...keys.filter(val => val <= n)) // maximum of the resulting filter array

roman = roman + romanRelation[max]

n = n - max

}

return `${this.number} is ${roman} in roman numbers`

}

}

222 is CCXXII in roman numbers

The value must be a number between 1 and 3999

const test1 = new Number(0)

console.log(test1.toRoman())

/\* The value must be a number between 1 and 3999 \*/

const test2 = new Number(444)

console.log(test2.toRoman())

/\*444 is CDILIV in roman numbers \*/

const test3 = new Number(3999)

console.log(test3.toRoman())

/\* 3999 is MMMCMXCIX in roman numbers \*/

const test4 = new Number('444')

console.log(test4.toRoman())

/\* 444 is M in roman numbers

const test5 = new Number(222)

console.log(test5.toRoman())

const test6 = new Number('££££')

console.log(test6.toRoman())