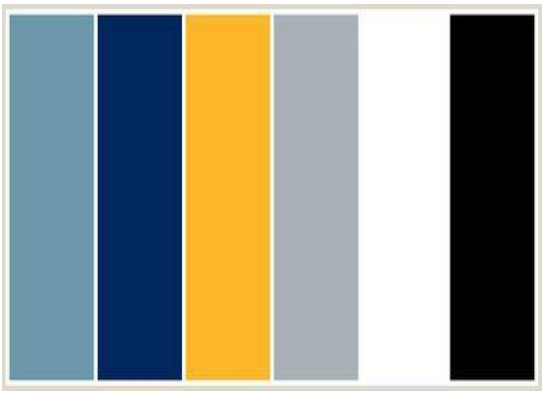
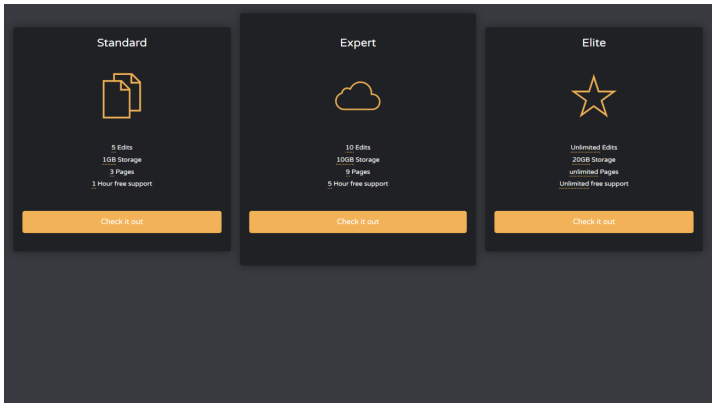
# $-CryptoMania-$

## 1) Initial Design:

After reading the description of the task I started my research and looked at online samples for crypto information websites. Not a lot of websites spoke about exchanges but all crypto related websites had a flashy and fancy touch to them.

This made my initial mood board go from -






To -

This change happened due to Crypto being a rather happening and dynamic topic. The feelings associated with it come from a sleek and financial background as it is a combination between modern and intelligent block chain technology added on to oldschool financial market systems. I simply did not want a boring solid color scheme after my mood board research

**Design conclusion -**

I will be going for a high definition background with glass like information profile cards displaying information about each Cryptocurrency exchange.

---

2) **Using the API**

The API was new to me so initially I did some research. Checked its compatibility, restrictions, call rate allowance (Did not want my coding to get paused due to hitting limitations). Once I figured it out, I made a call to it in my code.

```
/*axios needed to use api url*/
import axios from 'axios'
```

Axios was needed to handle the web requests but APIs can be pulled from without it too. I wanted to experiment and ended up learning something new while using Axios.

```
import React, {useState, useEffect} from 'react'
```

I imported useState and useEffect and here is how I used them. In addition to that click here to learn more about useState and useEffect -

```
/*use url and set data, throw error if it fails*/
useEffect(()=> {
    axios.get(url).then((response)=> {
        setData(response.data)
    }).catch((error)=> {
        console.log(error)
    })
}, [])


// Checking for error in API data extraction
if(!data) return null
```

Using this method , The array of values goes into data. We check for any failures in the if statement in the lines below the useEffect method.

---

3) **The code and its design**

Now that the data was ready to be used, we needed a beautiful way to display it.

Here I had two options, treat each card as a component or treat all the cards as one component. Due to time constraints and the physical limit being 10 exchanges, I decided to create just 10 cards and treat them all as one component.

This is not a good design approach for scalability. But I wanted to focus on the looks of the cards and how the data is represented. However, in any other scenario, iterating and passing the first 10 information arrays to each card component would be way better and scalable.

**My cards -**

All the cards came in one container so I can control their flex and wrap properties due to the instructed approach being mobile first. I had to ensure the mobile version looks complete and not like an omitted version of the desktop version.

To ensure this I made sure the background image behaves and remains the same despite the screen size. This was to prevent any cropping or moving of the HD image.

```
/* setting image here as background, ensuring image remains centered no mat
html {
  background: url(../assets/bg6.jpg) no-repeat center center fixed;
  -webkit-background-size: cover;
  -moz-background-size: cover;
  -o-background-size: cover;
  background-size: cover;
}
```

I added this to the index.css to make it part of the root styling. As one can see, I ended up picking bg6, among many other shortlisted. I was rather indecisive, so

conducted user polls with all my friends. Then went with the technically right answer, which was the background that contrasts well with the glass cards. In terms of readability and aesthetics.

Accessing the data was easy once the data was fetched and I treated each card as an index in a 0-9 array with 10 positions. Based on the index, I looked up the data title - image, country , etc and accessed the information accordingly.

```
{/* The api info needed is for the first 10 cards only, stored in an array
<div className = 'container'>
    <div className='card' >

        {/*logo of exchange*/}
        <img id = "mylogo" src={data[0].image} alt=''/>
        {/*name of exchange*/}
        <h1>{data[0].name}</h1>

        {/*Highlighting data type for ease of read using bold*/}

        {/*Country based*/}
        <p><strong>Country</strong> - {data[0].country}</p>

        {/*url to exchange website*/}
        <p><strong>URL</strong> - {data[0].url}</p>

        {/*Ranking of the exchange*/}
        <p><strong>Trust Rank</strong> - #{data[0].trust_score_rank}</p>

        {/*adding this text for call to action*/}

        <h2>CLICK FOR MORE INFO</h2>

    </div>
```

The cards all come under the class card and are encompassed in the container class.

The design of the glass cards can be seen here -

```css
/*each card is exchange info*/
.card {
    /* glass panel look*/
    background: rgba(255, 255, 255, 0.4);
    border-radius: 18px;
    box-shadow: 0 4px 30px rgba(0, 0, 0, 0.9);
    backdrop-filter: blur(6px);
    -webkit-backdrop-filter: blur(5px);
    border: 1px solid rgba(255, 255, 255, 0.3);

    /*sizing*/
    width: 300px;
    height: 300px;

    /*content positioning*/
    justify-content: space-between;
    align-content: center;
    margin: 50px;
    padding-top: 20px;
    flex-wrap: wrap;



    /*transition timing setting*/
    transition: all 1s;
}
```

This ensured center aligned content and the low opacity glass look to the cards. I should have added a minimum width to the cards/ media width styling,  but upon mobile testing it did not give an issue, so I chose not to over complicate my code.

Keeping everything aligned and moving with the screen size was easy. The main container had to be designed in the following way -

```
.container {
    /*want to move cards to next line and maintain center alignment on
    display: flex;
    justify-content: center;
    flex-wrap: wrap;
    text-decoration: solid;

}
```

With the focus of data representation out of the way, I focused on sight interaction.

I could see that upon hovering, the cards looked very boring. So I began using the key words associated with cryptocurrencies and comparing them to design choices I liked personally. The crypto market is competitive and dynamic, implying that every exchange desires to be the top pick. This reminded me of character selections in combat games where even a slight hover over the character would make them do a signature move to grab our eye/attention. Thus I decided to treat the Logo of the companies as little video game sprites and make a little spin and hover on selection.

This was achieved by simply on hover property changes and rotate and transform usage.

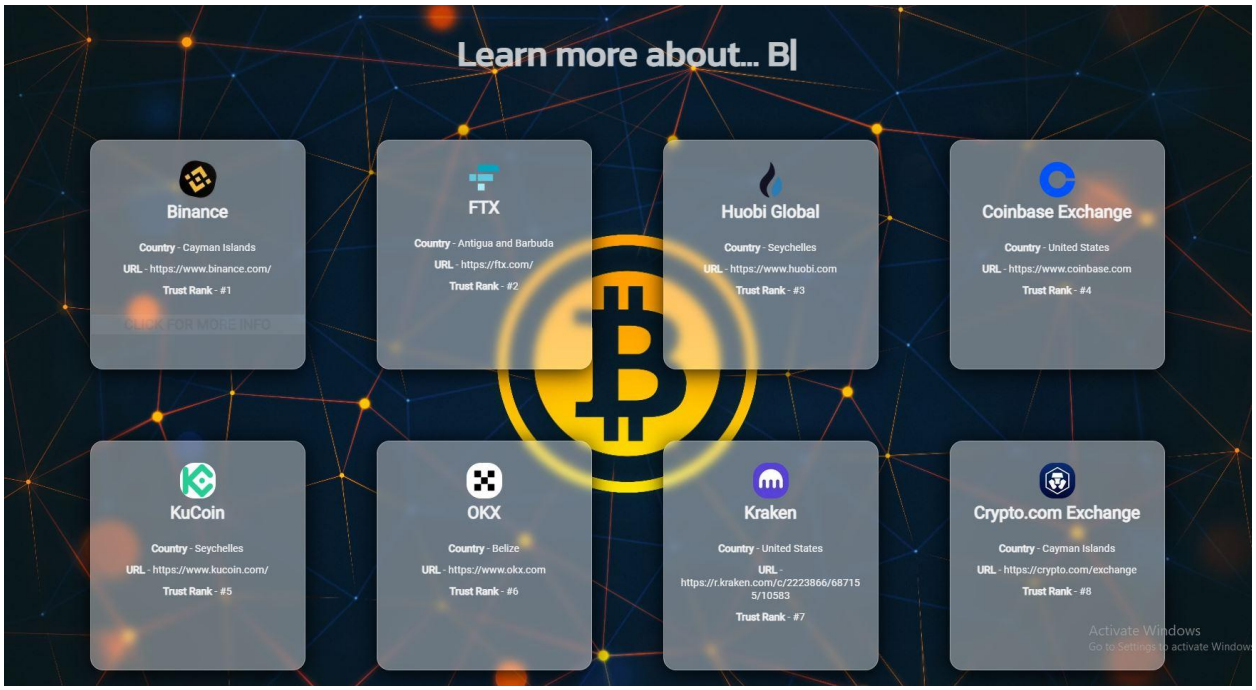
```
.card:hover img {
  /*rotate image, raise image, add shadow to show floating effect*/
  transform: translateY(-24px) rotate(360deg);
  box-shadow: 0 8px 2px 4px #414a4c;

  transition: box-shadow 1.9s transform 0.7s;


}
```

To help users navigate the page I added a call to action on hover which was a slow appearing text asking the user to click for more information. This was achieved simply by changing text opacity on hover.

**Final look of the website -**




---

4) **Testing**

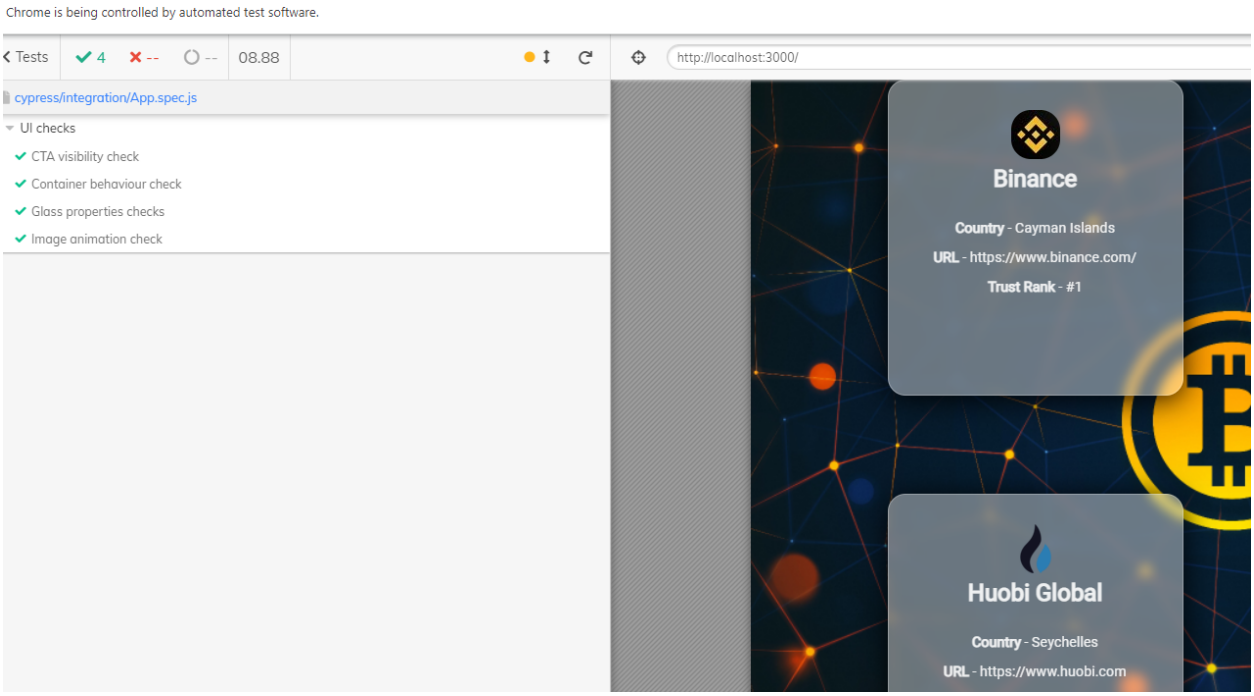
First I installed cypress using npm and then edited the dependencies in my package.json file.

After coding in my tests for the landing page UI and some mouse events , these are my final range of tests.

**Tests for the ideal case -**

1) Make sure hidden elements not show on initial load
2) Make sure containers are loaded with the correct css properties
3) Make sure cards have the right css properties
4) Checking image animation on hover
5) Checking appearing text on hover

## Passed tests -




## Sample code for tests -

```
describe("UI checks", () =>  {
    beforeEach(()=> {
        cy.visit('http://localhost:3000')

    })

    it('CTA visibility check', () => {

        cy.get('h2').should('have.css', 'opacity', '0')
    })

    it('Container behaviour check', ()=> {
        cy.get('.container').should('have.css', 'display', 'flex')
        cy.get('.container').should('have.css', 'justify-content', 'center')
        cy.get('.container').should('have.css', 'flex-wrap', 'wrap')

    })

    it('Glass properties checks', ()=>{
        cy.get('.card').should('have.css', 'width', '300px')
        cy.get('.card').should('have.css', 'height', '300px')
        cy.get('.card').should('have.css', 'align-content', 'center')

    })

    it('Image animation check', ()=> {
        cy.get('#card1').trigger('mouseenter')
        cy.wait(2000)
        cy.get('#mylogo1').should('have.css', 'box-shadow',  'rgb(65, 74, 76) 0px 8
        cy.wait(2000)
        cy.get('h2').should('have.css', 'opacity', '0.5')


    })
```

---

5) **Final checklist**

- ☑ ~~**Use coingecko API**~~
- ☑ ~~**List first 10 exchanges - name, country, url, logo, trust rank**~~
- ☐ **On click open new page with - name**
- ☑ ~~**Github**~~
- ☑ ~~**Documentation**~~
- ☑ ~~**Testing with Cypress**~~
- ☑ ~~**Code formatting with Prettier**~~

---

6) **What could have been done differently and what was left in the task list?**
The mistake I made with the components was unfortunate. Due to there being just one component I was not able to use Routing and Linking to open a new page with the current card's information passed as the props to the Newpage component.

Due to my initial design, any other loophole to link to the new page could be unreliable based on the browser. This is why I was not able to implement the new page functionality.

However I would implement it with a Routing and Linking import.

With the target being the new page in case a click event was registered on the card itself. What did not work was calling the newpage component in the tag of the card itself after the onclick property. I wanted my code to compile, so I had to omit that method.