Gray Paper: QC Eats

Names: Rabsa Islam, Alexandra Chavez, Cody Frucht, Fahmida Khan, and Rachel Archer

Search Bar HTML

```
  <div class="search-bar">
    <input type="text" id="deliveryAddress" placeholder="Enter Delivery Address">
    <button id="deliverNow">Deliver Now</button>
     <button id="searchHere"><a href="restaurant.html">Search Here</a></button>
  </div>
</div>
```

1)<input type="text" id="deliveryAddress" placeholder="Enter Delivery Address”>

- The <input> element of type "text" allows users to input their delivery address. It has an ID of "deliveryAddress" and a placeholder text "Enter Delivery Address”.

2) <button id="deliverNow">Deliver Now</button>

- The <button> element with the ID "deliverNow" produces an immediate delivery action when clicked using async and wait which is done in JS.

3) <button id="searchHere">Search Here</button>

- The <button> element with the ID "searchHere" contains an <a> element pointing to "restaurant.html". When clicked, it will redirect users to a page where they can search for restaurants or food categories.

Search Bar: Asynch and Wait

```
async function fetchRestaurantData(address) {
 try {
   return await new Promise((resolve) =>
    setTimeout(() =>
      const restaurants = [
        { name: 'Zyara Restaurant', address: '733 Main St' }
        { name: 'Marthas Bakery', address: '246 Melbourne Ave' }
 ];
 resolve(restaurants);
  }, 2000);
 });
 } catch (error) {
  throw new Error('Error fetching restaurant data')  }}
```

4) To manage errors, a try...catch block is used. It waits inside for a Promise (setTimeout) to resolve after a 2-second delay. When the Promise is resolved, it generates an array of restaurants with names and addresses and uses this array to resolve the Promise.

```
document.addEventListener('DOMContentLoaded', () => {
  const deliverNowButton = document.getElementById('deliverNow');
  const addressInput = document.getElementById('deliveryAddress');
  async function handleButtonClick(event) {
    const address = addressInput.value;
    try {
      console.log('Typed Address:', address);
      const restaurants = await fetchRestaurantData(address);
      console.log('Fetched Restaurants:', restaurants);
    } catch (error) {
      console.error(error.message);
    }
  }
  deliverNowButton.addEventListener('click', handleButtonClick);
```

5) The address value is retrieved from the input field by the handleButtonClick.
Using the provided address and the fetchRestaurantData function, it tries to retrieve restaurant data. Once the restaurant data is successfully retrieved, the retrieved restaurants appear on the console. Any issues that arise during the fetching process are detected and reported to the console.

6) I needed help generating an output on the console. There appeared to be an issue with my DOM. After updating it to document.addEventListener ('DOMContentLoaded', (), I could finally observe the input value from the "Enter Address" search bar and view a list of restaurants.

Testimonials JS

```
const testimonials = [
  {
    name: 'Miyah Myles',
    position: 'Marketing',
    photo:

'https://images.unsplash.com/photo-1494790108377-be9c29b29330?ixlib=rb-0.3.5&q=80&fm=jpg&crop=entropy&cs=tinysrgb&w=200&fit=max&s=707b9c33066bf8808c934c8ab394dff6',
    text:
      "This food delivery service consistently wows me with its exceptional customer support. The user-friendly interface makes ordering a breeze, and their attention to detail, from order accuracy to careful packaging, ensures that every meal arrives fresh and delicious. With prompt and reliable deliveries, they've become my go-to choice for hassle-free dining at home.",
  },
```

```
    {
…
```

7) This array contains several objects, each representing a testimonial. Each object has properties such as name, position, photo, and text, holding details about a person's testimonial.

```
let idx = 0;
```

8) Initializes a variable idx (index) to keep track of the current testimonial being displayed.

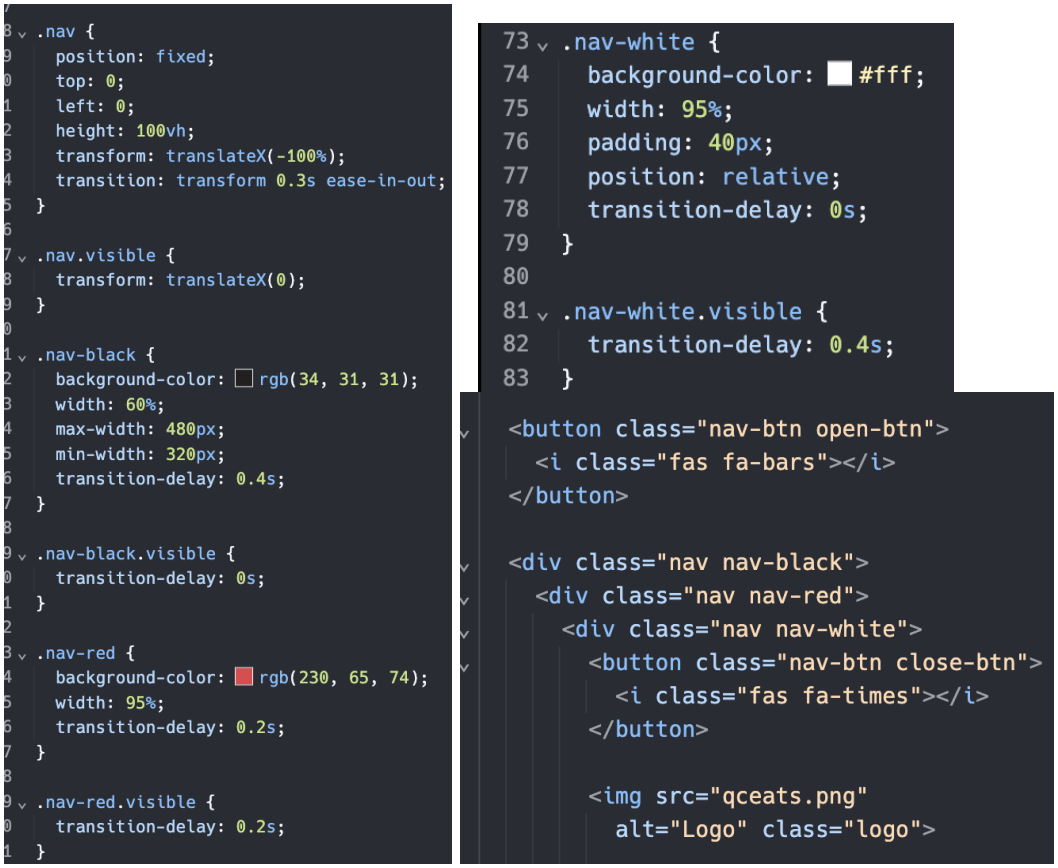
```
function updateTestimonial() {
  const testimonialElement = document.querySelector('.testimonial');
  const userImageElement = document.querySelector('.user-image');
  const usernameElement = document.querySelector('.username');
  const roleElement = document.querySelector('.role');

  const { name, position, photo, text } = testimonials[idx];

  testimonialElement.textContent = text;
  userImageElement.src = photo;
  usernameElement.textContent = name;
  roleElement.textContent = position;

  idx++;

  if (idx >= testimonials.length) {
    idx = 0;
  }
}
```

9) This function is responsible for updating the testimonial displayed on the webpage. It selects HTML elements related to the testimonial display:

- testimonialElement: Holds the text of the testimonial.
- userImageElement: Represents the image of the person giving the testimonial.
- usernameElement: Displays the name of the person giving the testimonial.
- roleElement: Displays the role or position of the person giving the testimonial.

```
updateTestimonial();
setInterval(updateTestimonial, 5000);
```

10) updateTestimonial is invoked once when the script loads to display the first testimonial immediately. updateTestimonial function is called when there is a set Interval every 5000 milliseconds (5 seconds) to cycle through the testimonials.



```
.nav {
  position: fixed;
  top: 0;
  left: 0;
  height: 100vh;
  transform: translateX(-100%);
  transition: transform 0.3s ease-in-out;
}

.nav.visible {
  transform: translateX(0);
}

.nav-black {
  background-color: rgb(34, 31, 31);
  width: 60%;
  max-width: 480px;
  min-width: 320px;
  transition-delay: 0.4s;
}

.nav-black.visible {
  transition-delay: 0s;
}

.nav-red {
  background-color: rgb(230, 65, 74);
  width: 95%;
  transition-delay: 0.2s;
}

.nav-red.visible {
  transition-delay: 0.2s;
}
```

```
.nav-white {
  background-color: #fff;
  width: 95%;
  padding: 40px;
  position: relative;
  transition-delay: 0s;
}

.nav-white.visible {
  transition-delay: 0.4s;
}
```

```
<button class="nav-btn open-btn">
  <i class="fas fa-bars"></i>
</button>

<div class="nav nav-black">
  <div class="nav nav-red">
    <div class="nav nav-white">
      <button class="nav-btn close-btn">
        <i class="fas fa-times"></i>
      </button>

      <img src="qceats.png"
        alt="Logo" class="logo">
```

For the side navigation bar it slides open with three different colored divs, one after the other. (Udemy)

11) **transform: translateX(-100%);** Affects how the navigation bar moves along the x-axis or moves horizontally. **** Shows icon for side nav button. **** Shows icon for close button to close navbar.

12) **transition: transform 0.3s ease-in-out;** The transition eases in the navigation bar for it to appear on the page in 0.3 seconds.

13) **transition-delay: 0.4s;** The black section of the navigation bar comes in on a delay of 0.4s, while the red section has a delay of 0.2 seconds and the white section has a delay only when visible of 0.4 seconds. This allows for the transition between sections, black, then red and finally to ease in slowly, also for the three colors to be seen still when the nav bar is open. Then as the nav bar is closed, it goes from white, to red then black.

```
function slideShow(n) { //displays the images to play one after another
  var i;
  var slides = document.getElementsByClassName("slidesImage");
  var photo = document.getElementsByClassName("column1");
  var captionText = document.getElementById("caption");
  if (n > slides.length){
    slideIndex = 1
  }
  if (n < 1){
    slideIndex = slides.length
  }
  for (i = 0; i < slides.length; i++) {
    slides[i].style.display = "none";
  }
  for (i = 0; i < photo.length; i++) {
    photo[i].className = photo[i].className.replace("active", "");
  }
  slides[slideIndex-1].style.display = "block";
  photo[slideIndex-1].className += "active";
  captionText.innerHTML = photo[slideIndex-1].alt;
}

let index = 0;
displaySlides();
function displaySlides() {
  let i;
  const images = document.getElementsByClassName("slidesImage");
  for (i = 0; i < images.length; i++) {
    images[i].style.display = "none";
  }
  index++;
  if (index > images.length) {
    index = 1;
  }
  images[index-1].style.display = "block";
  setTimeout(displaySlides, 1000);
}
```

The slideShow function allows for the images to be displayed in the lightbox
14) **if (n > slides.length){slideIndex = 1}** If n is greater than the length of the slides then the index of the slides is equal to one.

15) **if (n < 1){slideIndex = slides.length}** If n is less than one then the index is equal to the length of the slides. **for (i = 0; i < slides.length; i++) { slides[i].style.display = "none";}** This for loop goes through the images and sets the style display to “none”.

16) **for (i = 0; i < photo.length; i++) { photo[i].className = photo[i].className.replace("active", "");}** This for loop goes through the images and replaces active with an empty string. **slides[slideIndex-1].style.display = "block";** The other images are blocked and hidden until they are clicked on or until played in the lightbox.

17) **photo[slideIndex-1].className += "active";** The current image is active and being shown.

18) **captionText.innerHTML = photo[slideIndex-1].alt;** This takes the alt for the image and makes it the caption for the current image being displayed.

19) **setTimeout(displaySlides, 1000);** The images transition one second between each other. For the displaySlides function the images are played automatically without having to press the next or prev buttons. The for loop and if statements are similar to the ones for the lightbox function.

20) A challenge I had was having trouble displaying the lightbox by placing the images in an array and then using appendChild to display them. I was able to show the images but they were not appearing in a row and the sizing was not working. The overlay wasn't displaying as well. Instead I created divs in HTML instead of Javascript for the images to appear in the lightbox, functions for the slideshow part to work, and for the setTimeout to transition between them automatically.

Double heart (Udemy)

HTML

```
<i class = "fas fa-heart" id ="grey"></i>
```

CSS

```
.fa-heart {
 color: red;
}
.fa-heart#grey .fa-heart{
 position: absolute;
 animation: grow 0.6s linear;
 transform: translate(-50%, -50%) scale(0);
}
@keyframes grow {
 to {
  transform: translate(-50%, -50%) scale(10);
  opacity: 0;
 }
}
```

Js

```
const heart_btn = document.querySelectorAll('.fa-heart#grey');
heart_btn.forEach(heart_btn => {
  heart_btn.addEventListener('click', (e) => {
    if (heart_btn.classList.contains('clicked')) {
      changeHeartColor(heart_btn, 'lightgrey');
    } else {
```

```
            createHeart(e);
            changeHeartColor(heart_btn, 'red');
        }
        heart_btn.classList.toggle('clicked');
    });
});
//—------------------------------
const createHeart = (e) => {
    const heart = document.createElement('i');
    heart.classList.add('fas');
    heart.classList.add('fa-heart');

    e.target.appendChild(heart);

    setTimeout(() => heart.remove(), 1000);
};
//—------------------------------
const changeHeartColor = (heart_btn, color) => {
    heart_btn.style.color = color;
};
```

21) The class **"fas fa-heart"** is the heart that is used as a button. **animation: grow 0.6s linear;** applies the keyframe in 0.6 seconds in a linear state. **transform: translate(-50%, -50%) scale(10);** Changes the position and size of the heart.

22) **heart_btn.forEach(heart_btn =>....)** identifies the purpose of each click of the constant “heart_btn”. If the button has been clicked once (**.contains('clicked')**) then once it is clicked again it will trigger the functions **changeHeartColor(heart_btn)**changing the button to its original state.Else if the button does not contain “clicked”, then when the button is clicked it will trigger **createHeart(e);changeHeartColor(heart_btn, 'red');**

23) When the constant **createHeart** is called, the button is taken and the animation/heart-transformation is added to it(**e.target.appendChild(heart);**). The heart-transformation is removed once the animation is finished, making it a temporary effect due to **setTimeout(() => heart.remove(), 1000);**

24) In the original udemy code, users would double click on the image to create the heart animation. My challenge was that the heart animation would appear outside of the image. I decided to add a button under the page, apply the animation to the button instead of the image

and added **heart_btn.classList.toggle('clicked');**so the button can be used to heart and un-heart. I added changeHeartColor to change the state of the button.

FAQ Collapse

HTML:

```
<div class="faq">
      <h3 class="faq-title">
        How do I purchase QC Eats Premium?
      </h3>
       <p class="faq-text">
        Acquiring an QC Eats membership is conveniently done within the app. If using the QC Eats app… you retain the flexibility to cancel and oversee your membership settings at any point within the app.
      </p>
      <button class="faq-toggle">
        <i class="fas fa-chevron-down"></i>
        <i class="fas fa-times"></i>
      </button>
    </div>
```

CSS:

```
.faq.active::before,
.faq.active::after {
  content: "\f075";
  font-family: "Font Awesome 5 Free";
  color: grey;
  font-size: 7rem;
  position: absolute;
  opacity: 0.2;
  top: 20px;
  left: 20px;
  z-index: 0;
}

.faq.active::before {
  color: rgb(239,74,74);
  top: -10px;
  left: -30px;
  transform: rotateY(180deg);
}
```

JS:

```
const toggles = document.querySelectorAll(".faq-toggle");

toggles.forEach((toggle) => {
  toggle.addEventListener("click", () => {
    toggle.parentNode.classList.toggle("active");
  });
});
```

25) The div class “ ” contains all other classes known as faq-title, faq-text, faq-toggle. The div class separates everything into its own section. There is also a class known as “”. This provides an arrow pointing down. For this, “”. This provides a multiplication symbol (x). These are all the different classes and phrases used, in order to create the FAQ Collapse. It allows the user to interact and be able to press the arrow or exit out.

26) This ”.faq.active::before, .faq.active::after {“ adds the css pseudo selector for faq-active. You are able to add a content value, font-family, opacity, z-index, etc. This class ”.faq.active::before{“, you are able to individually change a specific area. Now, this is going to change the faq.active BEFORE it becomes active. This allows to specify exactly what to change and what css should be applied to certain sections.

27) - const toggles = document.querySelectorAll(".faq-toggle"); : bring in all toggles with the class of .faq-toggle
-toggles.forEach((toggle) => { : take the node list and loop through using for each
- toggle.addEventListener("click", () => { :takes that specific toggle and adds an event listener (click) event then it runs a function when that happens.
- toggle.parentNode.classList.toggle("active"); : accesses parent node with the class list and toggles active class.

28)

```
require('dotenv').config();
const express = require('express');
const helmet = require('helmet');
const mongoose = require('mongoose');
const { body, validationResult } = require('express-validator');
const app = express();
const port = process.env.PORT || 3000;
app.use(helmet());
mongoose.connect(process.env.MONGODB_URI, {
  useNewUrlParser: true,}); const formDataSchema = new mongoose.Schema({
emailPhoneInput: String,
});
```

using an environment variable to store database URI, also includes helmet to enhance security for http headers.