1.

Download and unzip the project skeleton located here. (already sent)

Stuck? Get a hint

2.

Open **index.html** in Google Chrome. Open the JavaScript console in Chrome DevTools and run `MatchGameTests.runTests()`. Tests for all of the empty methods will fail. As you implement these methods, more and more of the tests will pass and new tests may fail due to the ability to test previously untestable functionality. To complete this project, all of the tests must pass.

Stuck? Get a hint

Bootstrap HTML

3.

You will use Bootstrap to style your page. In **index.html** add a link to the Bootstrap CSS library.

Stuck? Get a hint

4.

Add a Bootstrap container with one row.

Stuck? Get a hint

5.

Within this row, add a div for the game instructions. Add the appropriate Bootstrap classes so this div takes up 1/4 of the container's width on devices with a screen width greater than 992 pixels and all of the container's width on devices narrower than 992 pixels.

Stuck? Get a hint

6.

Within the same row, add a div for the game. This div should take up 3/4 of the container's width on devices with a screen width greater than 992 pixels and all of the container's width on devices narrower than 992 pixels. Add an ID of `game` to this element.

Stuck? Get a hint

7.

Within the game instructions div, add heading and paragraph elements with the text from the design specs.

8.
Within the game div, add a Bootstrap row. This row will contain the game's cards.

9.
Add sixteen divs, representing cards, to the game row. Each of these elements should take up 1/4 of the row on all device sizes, and they should have a class of `card`.

10.
Finally, add the supplied favicon to the site. Congratulations, you've used Bootstrap to create and lay out all of the HTML you need! In the next section, you will finish styling the page with fonts and colors.

Stuck? Get a hint

Style HTML

11.
Before your Bootstrap link tag, link to a CSS reset.

Stuck? Get a hint

12.
Import the "Work Sans" font from Google Fonts and all font weights used in the design specs.

Stuck? Get a hint

13.
Create and link to an empty **style.css** file.

14.
Within **style.css**, set the text color, background color, and font family specified in the design specs.

15.
Add space to the top of the page (see specs).

16.
Apply the correct font sizes and weights to the instruction text (as described in the spec).

17.

Set the height, background color, border, and border radius of the cards.

18.
Style what the cards look like when face up. Add a number inside the first card. Center and style the text based on the design specs. Ignore the face-up colors for now. We will add those using jQuery.

Stuck? Get a hint

19.
In many browers, when you hover over the number on the card, your cursor turns into a text cursor. This doesn't look good. Use the CSS cursor property so the pointer cursor is displayed when users hover over the cards.

Stuck? Get a hint

20.
Congratulations, your board is now fully styled and ready to become interactive! Delete the fake number value you added to the first card.

Implement .generateCardValues()

21.
During the following tasks you will implement the `.generateCardValues()`method. `.generateCardValues()` is a function that creates and returns an array of 8 randomly-ordered matching pairs of values that will represent the order of cards on the game board.

Think about the steps needed to implement this function. When you've laid them out, move on to the next task.

22.
These are our steps to implement this function:

1. Create a sequentially-ordered — in ascending order — array with two copies of every number from 1 through 8
2. Randomly transfer those values to a new array
3. Return the randomly-ordered array

Compare these steps to the ones you created to see how the approaches differed.

23.
In the next few tasks, you will generate the array of all possible values in order. In **match-game.js**, start the `.generateCardValues()` method by creating an empty array representing the unplaced, in-order card values.

Stuck? Get a hint

24.
Write a loop to iterate through each card value, 1 through 8.

Stuck? Get a hint

25.
Inside the loop, add the current value to your array of unplaced values, twice.

Stuck? Get a hint

26.
In the next few tasks you need to transfer the values from the unplaced values array to the randomly-ordered array. Create an empty array representing the randomly-ordered card values.

27.
Create a `while` loop that runs until the sequentially-ordered array is empty. Be careful not to run your code yet — this loop will never terminate!

Stuck? Get a hint

28.
Within the loop, generate a random index in the array of in-order, unplaced values (a random number from 0 to the length of the unplaced values array). To accomplish this, use the Math.random method.

Stuck? Get a hint

29.
Access the value in the unplaced values array at the random index you just created. Add this value to the end of your randomly-placed values array.

Stuck? Get a hint

30.

Remove the element at the random index from the sequentially-ordered values array. Use the .splice() method to accomplish this.

Stuck? Get a hint

31.

After both loops are complete, you will have a randomly-ordered array of every value on the board. Return the randomly-ordered array from the function.

Stuck? Get a hint

32.

Open the JavaScript console and run your method `MatchGame.generateCardValues()` a few times to make sure it is working as intended. When you're confident, run `MatchGameTests.runTests()` to ensure that all of the `.generateCardValues()`tests now pass. Congratulations, you implemented all of the logic needed to create a random sequence of the game's cards!

Implement .renderCards()

33.

You now have a function that generates a board of random values. You need to turn these random values into visible cards on the web page.

In the next few tasks, you will implement a method called `.renderCards()` that takes an array of card values (`cardValues`) and a jQuery object representing the game (`$game`). `.renderCards()` will then create jQuery objects representing cards and add them to the game.

The method will store information about the value and color of each card as they are added. This information will be used by a different function that styles cards when they are flipped over.

Think about the steps needed to implement this function. When you''ve laid them out, move on to the next task.

34.

These are our steps to implement `.renderCards()`:

1. Empty the `$game` object's HTML
2. Generate jQuery objects for each card, including data about the value, color, and flipped status of each card
3. Add the card objects to the `$game`object

Compare these steps to the ones you created to see how the approaches differed.

35.
Start the method by emptying the HTML of the `$game` object.

Stuck? Get a hint

36.
In the next few tasks, you will generate jQuery card objects for each value in the `cardValues` array. Start by looping through each value in the `cardValues`array argument.

37.
Inside the loop, create a jQuery object for a new card. This object should be instantiated — created — with the same HTML code you used to render a card in your **index.html**file.

Stuck? Get a hint

38.
Using the data method, add a data attribute representing the card's value on the new card object, setting the value of the data attribute equal to the value at the current index in the `cardValues` array.

Stuck? Get a hint

39.
In addition to knowing the card's value, you need to know whether or not the card has been flipped. Add a data attribute to the jQuery card object representing whether or not the card has been flipped. This value should default to `false`.

40.

Finally, each card value will have a unique color. Go to the beginning of your `.renderCards()` method and create an array containing the eight hsl values listed in the design specs as strings.

41.

Add a data attribute to the card to store its color. Set its color by matching the card's value to the color at a position in the color array. Since the values of the cards are 1 through 8, subtract 1 from the card's value to find the index of the corresponding color in our array.

42.

Your card now has all of the information it needs. Append the card object to `$game`.

Stuck? Get a hint

43.

You can now generate the HTML for a game. You will use your `.generateCardValues()` and `.renderCards()` methods to render the randomly-arranged cards to the page.

Add a `$(document).ready` statement to the top of **match-game.js** and call `MatchGame.renderCards()` inside of its callback function. Make sure to pass the `.renderCards()` method an array of randomly-ordered card values and a jQuery object containing our `#game` HTML element, created by `MatchGame.generateCardValues()`, as parameters.

Stuck? Get a hint

44.

Congratulations, you created and rendered randomly-generated HTML using JavaScript! Since you are rendering the board in JavaScript, you can delete all of the HTML inside of the game HTML element in **index.html**. Refresh your page. The board should still be rendered even though there is no game HTML in the **index.html** file. Run the tests to ensure that all `.renderCards()` tests now pass.

Implement .flipCard() and Add Event Handler

45.

The final step is to implement `.flipCard()` which will add the dynamic behavior of flipping a card when it's selected and unflipping cards that don't match. This method takes two jQuery objects: the card being flipped, `$card`, and the game that is being played, `$game`. Think about the steps needed to implement this function. When you've laid them out, move on to the next task.

46.

These are our steps to implement this method:

1. Ensure that the card being flipped is not already flipped over
2. Modify the card being flipped so that it appears flipped over
3. Add the newly-flipped card to an array of flipped cards stored on the game object
4. Check if a match has been made when two cards have been flipped
5. Gray out the cards if a match is made
6. Flip the cards back over if a match isn't made
7. Clear the game's array of flipped cards to get ready for the next pair

Compare these steps to the ones you created to see how the approaches differed.

47.

To start, your game jQuery object will need information about which cards are flipped. At the top of the `.renderCards()`method, add a data attribute to `$game`which keeps track of the flipped cards. This should initialize — be initially set equal — to an empty array.

48.

In `.flipCard()`, you need to flip a card if it has not been previously flipped or do nothing if it has already been flipped.

Start the method by checking if the selected card is already flipped. This information should be stored in the data attribute that you specified in `.renderCards()`. If the card has already been flipped, return from the function so the function stops executing.

Stuck? Get a hint

49.

If the card has not been flipped, modify it so it appears flipped over. Change the background color of the card to be the color stored on the card, change the text of the card to be the value stored on the card. Then, update the data on the card to indicate that it has been flipped over.

Stuck? Get a hint

50.

At this point, flipping a card will render correctly, but you still need to check for a match when two cards have been flipped over. After you finish styling the flipped card in `.flipCard()`, push the card on to the end of the game object's array of flipped cards.

51.

Now, check if the game has two flipped cards.

52.

If two cards are flipped, check to see if they are the same value. If so, update the card's background color and color (see specs).

53.

If the two cards are not a match, flip them back over by resetting the background color, setting the text to an empty string, and updating the data to reflect that the card has not been flipped over.

54.

Finally, set the array of flipped cards on the game object to be a new, empty array, whether or not the cards were a match.

55.

Add an event listener at the end of `.renderCards()`. This listener should call `.flipCard()` whenever a card is clicked. Make sure to call `.flipCard()` with jQuery objects containing the card that was clicked and the `#game` element. Be careful to update just the card that was clicked, and not all elements with class `card`. It's important to create this event listener at the end of `.renderCards()` instead of

`document.ready` because we can only guarantee that the cards will be created at the end of `.renderCards()`. If you try to attach click handlers to elements that don't exist, they will never be created.

Stuck? Get a hint

56.

Refresh your page and try out the game. It should work, but you may notice that when a pair isn't found the cards automatically flip back over, not giving the user time to even look at the second card. Use .setTimeout()to add a delay to the flip.

Stuck? Get a hint

57.

Run the tests — they should all pass! Congratulations, you have a working card-matching game! Take some time customizing it to make it your own. Here are a few features you could add:

- Indicate that the user won when all pairs have been found
- Add a "Restart Game" button
- Only allow two cards to be visible at a time (currently the setTimeout allows users to click really quickly and see a few)
- Change card values to non-number values
- Add score or time
- Allow user to select from multiple board sizes
- Add sound effects
- Add flipping animations