Homework (Assignment Six)

Please read:

- This is your sixth assignment which you need to submit on the LMS. This is a homework on Testing.
- Work on the questions given below and be ready with your solution in CodeSandbox. You have to submit your CodeSandbox link below on this page.
- Late Submission: The deadline to submit this assignment is 19th February 2024, 6:00PM IST.

Important Instructions:

1. Make sure that you follow the rules of unit testing and write separate tests for each test case. This will help you build your testing muscle.
2. Do not copy from someone else as that would be cheating.

challenge

https://codesandbox.io/s/testing-homework-djjtll

map utility exercises

1. transformKeys

You have been given a utility function called `transformKeys`, which takes an object as input and returns an array containing the uppercase versions of all the keys in the object. Write test cases to ensure that this function behaves correctly.

Test Case	Input	Expected Output
transforms lowercase keys to uppercase	`{ name: 'John', age: 30, city: 'New York' }`	['NAME', 'AGE', 'CITY']

Test Case	Input	Expected Output
returns an empty array for an empty object	`{}`	[]
does not modify the original object	`{ key1: 'value1', key2: 'value2' }`	(Original object remains unchanged)

```
export function transformKeys(obj) {
  return Object.keys(obj).map((key) => key.toUpperCase())
}
```

COPY

2. reverseStrings

Write a test case that ensures the function handles an array with empty strings correctly and returns them as empty strings as well.

Test Name	Input	Expected Output	Matcher
Reverse multiple strings	['hello', 'world', 'jest']	['olleh', 'dlrow', 'tsej']	toEqual
Handle empty input array	[]	[]	toEqual
Reverse strings with spaces	['hello world', 'goodbye space']	['dlrow olleh', 'ecaps eybdoog']	toEqual
Original array remains unchanged	['abc', 'def']	(Original array remains unchanged)	toEqual
Reverse and check individual characters	['abc', '123']	['cba', '321']	toContainEqual

```
export function reverseStrings(arr) {
  return arr.map((str) => str.split('').reverse().join(''))
}
```

COPY

3. squareRoots

Test Name	Input	Expected Output	Matcher
Calculate square roots of positive integers	[4, 9, 16]	[2, 3, 4]	toEqual
Calculate square roots of positive floating-point numbers	[2.25, 0.25, 1.44]	[1.5, 0.5, 1.2]	toEqual
Handle empty input array	[]	[]	toEqual
Ensure original array remains unchanged	[4, 9, 16]	(Original array remains unchanged)	toEqual
Ensure each result is close to the actual square root	[25, 64, 100]	(Comparison with toBeCloseTo)	toBeCloseTo

Write a set of test cases for `squareRoots`. This function takes an array of numbers as input and returns a new array with the square roots of each number.

```
export function squareRoots(arr) {
  return arr.map((num) => Math.sqrt(num))
}
```

COPY

4. removeVowels

You are given a utility function named `removeVowels`. This function takes an array of strings as input and returns a new array with each string having all its vowels removed.

Your task is to write a set of test cases to ensure the correctness of this function.

Test Name	Input	Expected Output	Matcher
Remove vowels from single word strings	['hello', 'world']	['hll', 'wrld']	toEqual
Handle strings with mixed case vowels	['ApplE', 'OrAngE']	['ppl', 'rng']	toEqual
Handle empty strings	['', 'test', '']	['', 'tst', '']	toEqual
Handle strings with no vowels	['xyz', 'qrst']	['xyz', 'qrst']	toEqual
Ensure original array remains unchanged	['hello', 'world']	(Original array remains unchanged)	toEqual
Handle strings with all vowels	['aeiou', 'AEIOU']	['', '']	toEqual

```
export function removeVowels(arr) {
  const vowels = ['a', 'e', 'i', 'o', 'u']
  return arr.map((str) =>
    str
      .split('')
      .filter((char) => !vowels.includes(char.toLowerCase()))
      .join(''),
  )
}
```

COPY

filter utility exercises

1. filterLongStrings

Write tests for `filterLongStrings` which takes an array of strings and a minimum length as input and returns a new array containing only the strings that are longer than the specified minimum length.

Test Case	Input	Expected Output	Matcher Used
Filters strings longer than minimum length	['apple', 'banana', 'cherry', 'date'], 5	['banana', 'cherry']	toEqual
Handles empty input array	[], 3	[]	toEqual
Handles empty output array	['cat', 'dog', 'rat'], 3	[]	toEqual
Handles negative minimum length	['hello', 'world'], -2	['hello', 'world']	toEqual
Ensures original array remains unchanged	['apple', 'banana', 'cherry'], 4	(Original array remains unchanged)	toEqual
Checks if the filtered array is empty	['apple', 'banana', 'cherry', 'date'], 10	Should be an empty array	toBeEmpty
Checks if the function throws an error with invalid input	'invalid', 5	Should throw an error	toThrow

```
function filterLongStrings(strings, minLength) {
  strings.filter((str) => str.length > minLength)
}
```

COPY

2. filterEvenAndPositive

Write tests for `filterEvenAndPositive`. It takes an array of numbers as input and returns a new array containing only the numbers that are both even and positive.

Your task is to write a set of test cases to ensure the correctness of this function.

Test Case	Input	Expected Output	Matcher Used
Filters even and positive numbers	[2, 4, -6, 8, 9, -10, 11]	[2, 4, 8]	toEqual
Handles empty input array	[]	[]	toEqual
Handles input with no even and positive numbers	[-3, -5, -7]	[]	toEqual
Handles input with only positive but odd numbers	[1, 3, 5, 7]	[]	toEqual
Checks if the output array contains only even and positive numbers	[2, 4, -6, 8, 9, -10, 11]	All elements should be even and positive	every
Checks if the output array length is correct	[2, 4, -6, 8, 9, -10, 11]	Should have length 3	toHaveLength
Checks if the filtered array does not contain negative numbers	[2, 4, -6, 8, 9, -10, 11]	Should not include negative numbers	not.toContain
Checks if the function throws an error with invalid input	'invalid'	Should throw an error	toThrow

```
function filterEvenAndPositive(arr) {
  arr.filter((num) => num % 2 === 0).filter((num) => num > 0)
}
```

COPY

3. isPalindromic

`isPalindromic` takes a number as input and determines if the number is palindromic (reads the same forwards and backwards).

Write a set of test cases for the `isPalindromic` function, as well as test cases for filtering palindromic numbers using the provided `numbers` array and the `filter` method.

Test Case	Input	Expected Output	Matcher Used
Check for a palindromic number	121	true	toBe
Check for a non-palindromic number	123	false	toBe
Check for a single-digit number	5	true	toBe
Filter palindromic numbers from the provided array	[121, 123, 1331, 454, 678, 898]	[121, 1331, 454, 898]	toEqual
Filter palindromic numbers from an empty array	[]	[]	toEqual
Filter palindromic numbers from an array with no palindromic numbers	[123, 456, 789]	[]	toEqual

Test Case	Input	Expected Output	Matcher Used
Ensure the filtered array contains only palindromic numbers	[121, 1331, 454, 898]	All elements should be palindromic	every
Check if the filtered array length is correct	[121, 1331, 454, 898]	Should have length 4	toHaveLength
Check if the filtered array is an array	[121, 1331, 454, 898]	Should be an array	toBeInstanceOf(Array)

```
function isPalindromic(num) {
  const strNum = num.toString()
  return strNum === strNum.split('').reverse().join('')
}

const numbers = [121, 123, 1331, 454, 678, 898]

const palindromicNumbers = numbers.filter(isPalindromic)
```

COPY

4. filterByProperties

You are provided a utility function named `filterByProperties`. It takes an array of objects and a criteria object as input, and it returns a new array containing only the objects that match the criteria.

Your task is to write a set of test cases for the `filterByProperties` function, as well as test cases for filtering items using the provided `items` array and the `criteria` object.

Test Case	Input	Expected Output	Matcher Used
Filter items based on criteria	items, `{ price: 10, category: 'A' }`	[Item 1, Item 3]	toEqual
Ensure original array remains unchanged	items, `{ price: 10, category: 'A' }`	(Original array remains unchanged)	toEqual
Check if filtered array includes certain items	items, `{ price: 10, category: 'A' }`	Item 1 and Item 3 should be included	toContainEqual
Check if filtered array does not include certain items	items, `{ price: 10, category: 'A' }`	Item 2 and Item 4 should not be included	not.toContainEqual
Check if the filtered array length is correct	items, `{ price: 10, category: 'A' }`	Should have length 2	toHaveLength
Check if the filtered array is an array	items, `{ price: 10, category: 'A' }`	Should be an array	toBeInstanceOf(Array)
Check if the filtered array is not empty	items, `{ price: 10, category: 'A' }`	Should not be an empty array	not.toHaveLength(0)

```
function filterByProperties(objects, properties) {
  return objects.filter((obj) => {
    for (const key in properties) {
      if (obj[key] !== properties[key]) {
        return false
      }
    }
    return true
  })
}

const items = [
  { name: 'Item 1', price: 10, category: 'A' },
  { name: 'Item 2', price: 25, category: 'B' },
  { name: 'Item 3', price: 10, category: 'A' },
  { name: 'Item 4', price: 15, category: 'C' },
]

const criteria = { price: 10, category: 'A' }
const filteredItems = filterByProperties(items, criteria)
```

COPY

reduce utility exercises

1. findMaxNumber

Write test cases for `findMaxNumber`. It takes an array of numbers as input and returns the maximum number from the array using the `reduce` method.

Test Case	Input	Expected Output	Matcher Used
Find maximum in a positive number array	[3, 7, 2, 9, 5]	9	toBe
Find maximum in a negative number array	[-3, -7, -2, -9, -5]	-2	toBe
Find maximum in an array with identical elements	[7, 7, 7, 7]	7	toBe
Ensure original array remains unchanged	[3, 7, 2, 9, 5]	(Original array remains unchanged)	toEqual
Find maximum in an array with decimal numbers	[3.5, 7.2, 2.1, 9.7, 5.3]	9.7	toBe
Find maximum in an empty array	[]	undefined	toBeUndefined

```
function findMaxNumber(arr) {
  return arr.reduce((max, curr) => (curr > max ? curr : max), arr[0])
}
```

COPY

2. countPositiveNumbers

`countPositiveNumbers` takes an array of numbers as input and returns the count of positive numbers in the array using the `reduce` method. Write a set of test cases for this function.

Test Case	Input	Expected Output	Matcher Used
Count positive numbers in an array with mixed numbers	[3, -7, 1, 9, -5]	3	toBe
Count positive numbers in an array with all positive numbers	[3, 7, 2, 9, 5]	5	toBe

Test Case	Input	Expected Output	Matcher Used
Count positive numbers in an array with all negative numbers	[-3, -7, -2, -9, -5]	0	toBe
Count positive numbers in an array with decimal numbers	[3.5, 7.2, -2.1, 9.7, -5.3]	3	toBe

```
function countPositiveNumbers(arr) {
  return arr.reduce((count, curr) => (curr > 0 ? count + 1 : count), 0)
}
```

COPY

3. flattenNestedArrays

You are provided a utility function named `flattenNestedArrays` which takes an array of arrays as input and returns a new array that is the result of flattening all the nested arrays. Your task is to write a set of test cases for the `flattenNestedArrays` function.

Test Case	Input	Expected Output	Matcher Used
Flatten nested arrays with mixed elements	[[1, 2], [3, 4], [5, 6]]	[1, 2, 3, 4, 5, 6]	toEqual
Flatten nested arrays with arrays of different lengths	[[1, 2], [3, 4, 5], [6]]	[1, 2, 3, 4, 5, 6]	toEqual
Flatten nested arrays with empty arrays	[[], [], []]	[]	toEqual
Flatten nested arrays with arrays containing non-numeric elements	[[1, 2], ['a', 'b'], [3, 4]]	[1, 2, 'a', 'b', 3, 4]	toEqual
Flatten an empty array of nested arrays	[]	[]	toEqual
Ensure original nested arrays remain unchanged	[[1, 2], [3, 4], [5, 6]]	(Original nested arrays remain unchanged)	toEqual
Check if the function throws an error with invalid input	'invalid'	Should throw an error	toThrow

```
function flattenNestedArrays(arrays) {
  return arrays.reduce(
    (flattened, currentArray) => flattened.concat(currentArray),
    [],
  )
}
```

COPY

4. groupByProperty

`groupByProperty` function takes an array of objects and a property name as input, and it returns an object where the keys are unique values of the specified property, and the values are arrays containing the objects that have that property value.

Write test cases for the `groupByProperty` function using the provided `students` array and the property `'age'`.

Test Case	Input	Expected Output	Matcher Used
Group objects by an existing property	students, 'age'	Grouped object by age	toEqual
Group objects by an empty property	students, ''	Grouped object by empty key	toEqual
Group objects with no objects	[], 'age'	Empty object	toEqual

```
function groupByProperty(objects, property) {
  return objects.reduce((grouped, currentObject) => {
    const propValue = currentObject[property]
    if (!grouped[propValue]) {
      grouped[propValue] = []
    }
    grouped[propValue].push(currentObject)
    return grouped
  }, {})
}

const students = [
  { name: 'Alice', age: 25 },
  { name: 'Bob', age: 30 },
  { name: 'Carol', age: 25 },
]

const groupedByAge = groupByProperty(students, 'age')
```

COPY

find utility exercises

1. findFirstPositiveNumber

Write tests for `findFirstPositiveNumber` which takes an array of numbers as input and returns the first positive number from the array using the `find` method.

Test Case	Input	Expected Output	Matcher Used
Find first positive number	[3, 7, -2, 9, -5]	3	toBe
Find first positive number in an array with only negative numbers	[-3, -7, -2, -9, -5]	undefined	toBeUndefined
Find first positive number in an array with decimal numbers	[3.5, 7.2, 2.1, 9.7, 5.3]	3.5	toBe
Check if the function throws an error with invalid input	'invalid'	Should throw an error	toThrow

```
export function findFirstPositiveNumber(arr) {
  if (!Array.isArray(arr)) {
    throw new Error('Input must be an array.')
  }

  const positiveNumbers = arr.filter((num) => num > 0)

  if (positiveNumbers.length > 0) {
    return positiveNumbers[0]
  } else {
    return undefined
  }
}
```

COPY

2. findCommonElement

You are given a utility function named `findCommonElement`. This function takes two arrays as input and returns the first element that is common between both arrays using the `find` method and `includes` method.

Your task is to write a set of test cases for the `findCommonElement` function using the provided `array1` and `array2`.

Test Case	Input	Expected Output	Matcher Used
Find a common element	[2, 4, 6, 8, 10], [5, 7, 8, 10, 12]	8	toBe
Find a common element in arrays with no common elements	[2, 4, 6], [5, 7, 9]	undefined	toBe
Find a common element when one array is empty	[], [5, 7, 8, 10, 12]	undefined	toBe
Find a common element when both arrays are empty	[], []	undefined	toBe
Check if the function throws an error with invalid input	'invalid1', 'invalid2'	Should throw an error	toThrow

```
function findCommonElement(arr1, arr2) {
  return arr1.find((item) => arr2.includes(item))
}

const array1 = [2, 4, 6, 8, 10]
const array2 = [5, 7, 8, 10, 12]
const commonElement = findCommonElement(array1, array2)
```

COPY

ex03: testing reducer - exercises

1. Todo List Reducer:

Description: Write test cases for the `todoReducer`

- `todoReducer` manages a state containing an array of todos.
- It handles two types of actions: `"ADD_TODO"` and `"TOGGLE_TODO"`.
- For the `"ADD_TODO"` action, a new todo object is added to the `todos` array with the specified `id`, `text`, and `completed` set to `false`.
- For the `"TOGGLE_TODO"` action, the `completed` status of the specified todo is toggled.

```
const initialState = {
  todos: [],
}
```

```
function todoReducer(state = initialState, action) {
  switch (action.type) {
    case 'ADD_TODO':
      return {
        ...state,
        todos: [
          ...state.todos,
          {
            id: action.payload.id,
            text: action.payload.text,
            completed: false,
          },
        ],
      }
    case 'TOGGLE_TODO':
      return {
        ...state,
        todos: state.todos.map((todo) =>
          todo.id === action.payload.id
            ? { ...todo, completed: !todo.completed }
            : todo,
        ),
      }
    default:
      return state
  }
}
```

COPY

2. Polling System Reducer with Dynamic Polls:

Description: Write test cases for the `pollReducer`

- `pollReducer` manages a state containing an array of polls.
- It handles three types of actions: `"CREATE_POLL"`, `"ADD_OPTION"`, and `"VOTE"`.
- For the `"CREATE_POLL"` action, a new poll object is added to the `polls` array with the specified `id`, `question`, and an empty `options` array.
- For the `"ADD_OPTION"` action, a new option with the given text and zero votes is added to the options array of the specified poll.
- For the `"VOTE"` action, the vote count of the specified option in the specified poll is incremented by 1.

```
const initialState = {
  polls: [],
}

function pollReducer(state = initialState, action) {
  switch (action.type) {
    case 'CREATE_POLL':
      return {
        ...state,
        polls: [
          ...state.polls,
          {
            id: action.payload.id,
            question: action.payload.question,
            options: [],
          },
        ],
      }
    case 'ADD_OPTION':
      return {
        ...state,
        polls: state.polls.map((poll) =>
          poll.id === action.payload.pollId
            ? {
                ...poll,
                options: [
                  ...poll.options,
                  { text: action.payload.optionText, votes: 0 },
                ],
              }
            : poll,
        ),
      }
    case 'VOTE':
      return {
        ...state,
        polls: state.polls.map((poll) =>
          poll.id === action.payload.pollId
            ? {
                ...poll,
                options: poll.options.map((option) =>
                  option.text === action.payload.optionText
                    ? { ...option, votes: option.votes + 1 }
                    : option,
                ),
              }
            : poll,
        ),
      }
    default:
      return state
  }
}
```

COPY

3. Cart Reducer w/ discount & promotions

Description: Write test cases for the `cartReducer` which manages a shopping cart containing items, discounts, and promotions.

- `cartReducer` manages a state containing an array of items, their total price, total quantity, and an array of discounts.
- It handles six types of actions: `"ADD_TO_CART"`, `"REMOVE_FROM_CART"`, `"UPDATE_QUANTITY"`, `"ADD_DISCOUNT"`, `"APPLY_PROMOTION"`, and `"REMOVE_DISCOUNT"`.
- For the `"ADD_DISCOUNT"` action, a discount is added to the cart and the total price is recalculated.
- For the `"APPLY_PROMOTION"` action, a promotion is added to the cart and the total price is recalculated.

- For the "REMOVE_DISCOUNT" action, a discount is removed from the cart and the total price is recalculated.

```
const initialState = {
  items: [],
  totalPrice: 0,
  totalQuantity: 0,
  discounts: [],
}

function cartReducer(state = initialState, action) {
  switch (action.type) {
    case 'ADD_TO_CART':
    // refer the code done in class if needed

    case 'REMOVE_FROM_CART':
    // refer the code done in class if needed

    case 'UPDATE_QUANTITY':
    // refer the code done in class if needed

    case 'ADD_DISCOUNT':
      const newDiscounts = [...state.discounts, action.payload.discount]
      const newTotalPriceWithDiscounts = calculateTotalPrice(
        state.items,
        newDiscounts,
        state.totalQuantity,
      )
      return {
        ...state,
        discounts: newDiscounts,
        totalPrice: newTotalPriceWithDiscounts,
      }

    case 'APPLY_PROMOTION':
      const newPromotions = [...state.discounts, action.payload.promotion]
      const newTotalPriceWithPromotions = calculateTotalPrice(
        state.items,
        newPromotions,
        state.totalQuantity,
      )
      return {
        ...state,
        discounts: newPromotions,
        totalPrice: newTotalPriceWithPromotions,
      }

    case 'REMOVE_DISCOUNT':
      const remainingDiscounts = state.discounts.filter(
        (discount) => discount.id !== action.payload.discountId,
      )
      const newTotalPriceWithoutDiscounts = calculateTotalPrice(
        state.items,
        remainingDiscounts,
        state.totalQuantity,
      )
      return {
        ...state,
        discounts: remainingDiscounts,
        totalPrice: newTotalPriceWithoutDiscounts,
      }
    default:
      return state
  }
}

function calculateTotalPrice(items, discounts, totalQuantity) {
  const totalDiscount = discounts.reduce(
    (sum, discount) => sum + discount.value,
    0,
  )
  const itemTotalPrice = items.reduce(
    (sum, item) => sum + item.price * item.quantity,
    0,
  )
  const totalPrice = itemTotalPrice - totalDiscount
  return totalPrice
}
```

COPY

ex04: let's try TDD - exercises

📢 Make sure you write the test cases first. And then write the reducer. You need to submit the final codesandbox link with all the tests and the reducers.

1. Bookmark management

Write test cases for the `bookmarkReducer` which manages a state containing an array of bookmarks with their titles, URLs, and tags.

- `bookmarkReducer` manages a state containing an array of bookmarks.
- It handles six types of actions: `"ADD_BOOKMARK"`, `"REMOVE_BOOKMARK"`, `"UPDATE_TAGS"`, `"FILTER_BOOKMARKS_BY_TAG"`, `"ADD_TAG"`, and `"REMOVE_TAG"`.
- For the `"ADD_BOOKMARK"` action, a new bookmark with the provided details is added to the bookmarks array.
- For the `"REMOVE_BOOKMARK"` action, a bookmark is removed from the bookmarks array.
- For the `"UPDATE_TAGS"` action, the tags of a bookmark are updated.
- For the `"FILTER_BOOKMARKS_BY_TAG"` action, bookmarks are filtered to include only those with the specified tag.
- For the `"ADD_TAG"` action, a new tag is added to the tags of a bookmark.
- For the `"REMOVE_TAG"` action, a tag is removed from the tags of a bookmark.

2. Comment Reducer

Write test cases for the `commentReducer` which manages a state containing an array of comments with their text, votes, and replies.

- `commentReducer` manages a state containing an array of comments.
- It handles six types of actions: `"ADD_COMMENT"`, `"REMOVE_COMMENT"`, `"UPVOTE_COMMENT"`, `"ADD_REPLY"`, `"REMOVE_REPLY"`, and `"DOWNVOTE_COMMENT"`.
- For the `"ADD_COMMENT"` action, a new comment with the provided details is added to the comments array.
- For the `"REMOVE_COMMENT"` action, a comment is removed from the comments array.
- For the `"UPVOTE_COMMENT"` action, the vote count of a comment is increased.
- For the `"ADD_REPLY"` action, a new reply is added to the replies of a comment.
- For the `"REMOVE_REPLY"` action, a reply is removed from the replies of a comment.
- For the `"DOWNVOTE_COMMENT"` action, the vote count of a comment is decreased.

3. Products Reducer

Write test cases for the `productReducer` which manages a state containing an array of products with their details and filters for category, search query, sorting, and price range.

- `productReducer` manages a state containing an array of products and filter options.
- It handles five types of actions: `"SET_CATEGORY_FILTER"`, `"SET_SEARCH_QUERY"`, `"SET_SORT"`, `"SET_PRICE_RANGE"`, and `"TOGGLE_AVAILABILITY"`.
- For the `"SET_CATEGORY_FILTER"` action, the category filter is updated.
- For the `"SET_SEARCH_QUERY"` action, the search query filter is updated.
- For the `"SET_SORT"` action, the sorting options are updated.
- For the `"SET_PRICE_RANGE"` action, the price range filter is updated.
- For the `"TOGGLE_AVAILABILITY"` action, the availability of a product is toggled.

```
const initialState = {
  products: [
    {
      id: 1,
      name: 'Phone',
      category: 'Electronics',
      price: 500,
      inStock: true,
    },
    { id: 2, name: 'Shirt', category: 'Clothing', price: 20, inStock: true },
    {
      id: 3,
      name: 'Laptop',
      category: 'Electronics',
      price: 1000,
      inStock: true,
    },
    { id: 4, name: 'Jeans', category: 'Clothing', price: 40, inStock: false },
    // ... more products
  ],
  filters: {
    category: 'All',
    searchQuery: '',
    sortBy: 'price',
    ascending: true,
    priceRange: { min: 0, max: 1000 },
  },
}
```

COPY

All the best. We hope you complete and submit your assignment in time.

Click on the Share button on your CodeSandbox, then click on Copy link button and submit that link here in the submission form below. Make sure the access is public.