Testing JavaScript

Martin Førre

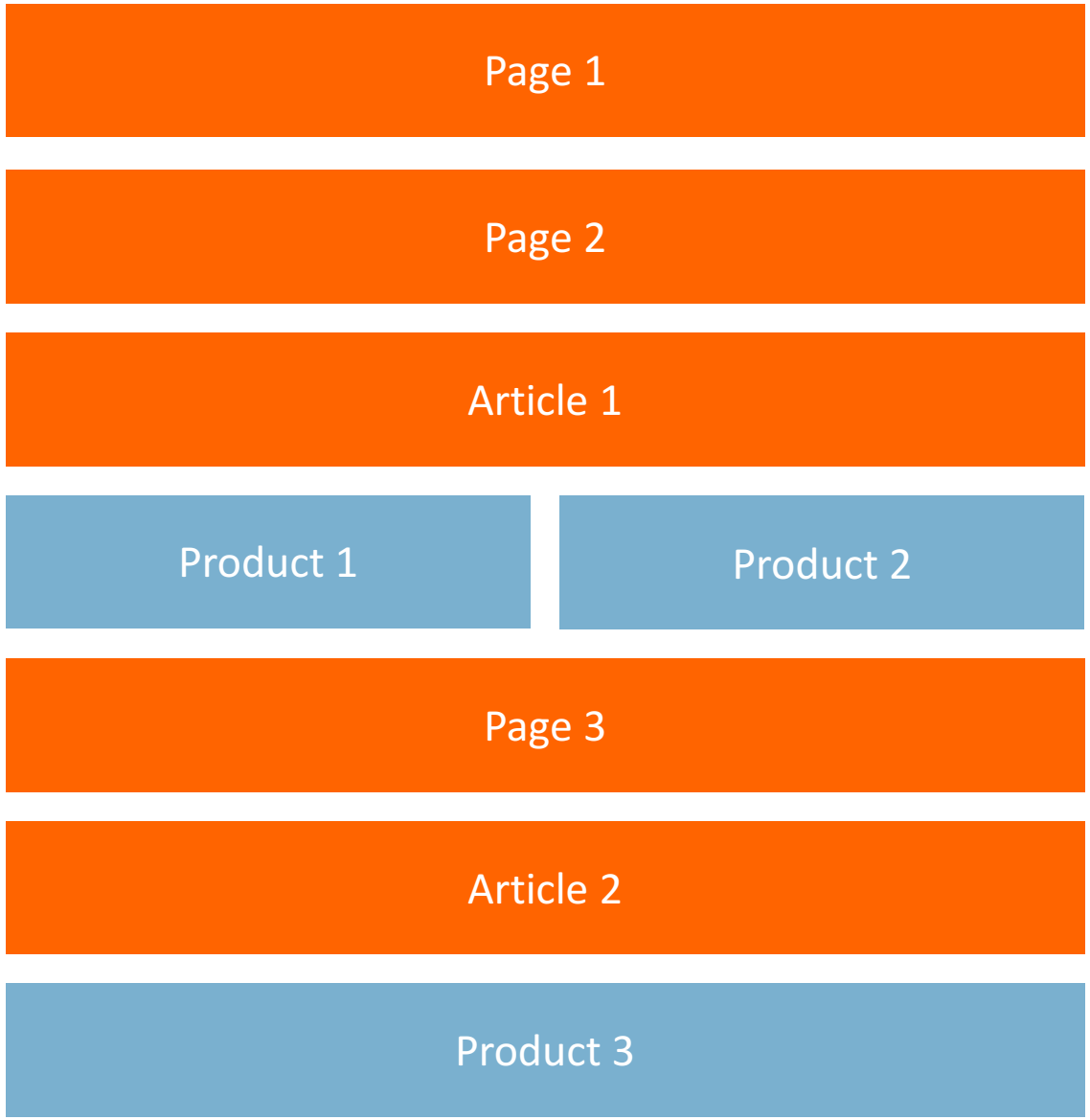
Group search results into rows

- Search result documents should be grouped into rows
- Pages should be on their own row
- Articles should be on their own row
- Products should be grouped into rows with a maximum of 2 products pr row

bouvet

Example

- Page 1
- Page 2
- Article 1
- Product 1
- Page 3
- Article 2
- Product 2
- Product 3



The «naive» approach

```
const render = (searchResultDocuments: SearchResultDocument[]) => {
    const container = window.document.createElement("div");

    for (const document of searchResultDocuments) {
        if (document.type !== "product") {
            const row = window.document.createElement("div");
            row.className = document.type;
            container.appendChild(row);

            const documentElement = window.document.createElement("div");
            documentElement.innerText = document.type;
            row.appendChild(documentElement);
        } else {
            const lastRow = container.querySelector("> div:last-child");
            if (lastRow.className === document.type && lastRow.childElementCount === 1) {
                const secondProductElement = window.document.createElement("div");
                secondProductElement.innerText = "product";
                lastRow.appendChild(secondProductElement);
            } else {
                const newProductRow = window.document.createElement("div");
                newProductRow.className = document.type;
                container.appendChild(newProductRow);

                const firstProductElement = window.document.createElement("div");
                firstProductElement.innerText = "product";
                newProductRow.appendChild(firstProductElement);
            }
        }
    }
};
```

bouvet

The «naive» approach

```
const render = (searchResultDocuments: SearchResultDocument[]) => {
    const container = window.document.createElement("div");

    for (const document of searchResultDocuments) {
        if (document.type !== "product") {
            const row = window.document.createElement("div");
            row.className = document.type;
            container.appendChild(row);

            const documentElement = window.document.createElement("div");
            documentElement.innerText = document.type;
            row.appendChild(documentElement);
        } else {
            const lastRow = container.querySelector("> div:last-child");
            if (lastRow.className === document.type && lastRow.childElementCount === 1) {
                const secondProductElement = window.document.createElement("div");
                secondProductElement.innerText = "product";
                lastRow.appendChild(secondProductElement);
            } else {
                const newProductRow = window.document.createElement("div");
                newProductRow.className = document.type;
                container.appendChild(newProductRow);

                const firstProductElement = window.document.createElement("div");
                firstProductElement.innerText = "product";
                newProductRow.appendChild(firstProductElement);
            }
        }
    }
};
```

bouvet

The «naive» approach

```
const render = (searchResultDocuments: SearchResultDocument[]) => {
    const container = window.document.createElement("div");

    for (const document of searchResultDocuments) {
        if (document.type !== "product") {
            const row = window.document.createElement("div");
            row.className = document.type;
            container.appendChild(row);

            const documentElement = window.document.createElement("div");
            documentElement.innerText = document.type;
            row.appendChild(documentElement);
        } else {
            const lastRow = container.querySelector("> div:last-child");
            if (lastRow.className === document.type && lastRow.childElementCount === 1) {
                const secondProductElement = window.document.createElement("div");
                secondProductElement.innerText = "product";
                lastRow.appendChild(secondProductElement);
            } else {
                const newProductRow = window.document.createElement("div");
                newProductRow.className = document.type;
                container.appendChild(newProductRow);

                const firstProductElement = window.document.createElement("div");
                firstProductElement.innerText = "product";
                newProductRow.appendChild(firstProductElement);
            }
        }
    }
};
```

bouvet

The «naive» approach

```
const render = (searchResultDocuments: SearchResultDocument[]) => {
    const container = window.document.createElement("div");

    for (const document of searchResultDocuments) {
        if (document.type !== "product") {
            const row = window.document.createElement("div");
            row.className = document.type;
            container.appendChild(row);

            const documentElement = window.document.createElement("div");
            documentElement.innerText = document.type;
            row.appendChild(documentElement);
        } else {
            const lastRow = container.querySelector("> div:last-child");
            if (lastRow.className === document.type && lastRow.childElementCount === 1) {
                const secondProductElement = window.document.createElement("div");
                secondProductElement.innerText = "product";
                lastRow.appendChild(secondProductElement);
            } else {
                const newProductRow = window.document.createElement("div");
                newProductRow.className = document.type;
                container.appendChild(newProductRow);

                const firstProductElement = window.document.createElement("div");
                firstProductElement.innerText = "product";
                newProductRow.appendChild(firstProductElement);
            }
        }
    }
};
```

bouvet

Split «application logic» and rendering

Initial tests

```
Run | Debug | Show in Test Explorer | Run | Debug
describe("groupByType", () => {
    Run | Debug | Show in Test Explorer | Run | Debug
    ✓it("Creates a list of rows with pages", () => {
        const rows = groupByType([page1, page2, page3]);

        expect(rows).toStrictEqual([[page1], [page2], [page3]]);
    });

    Run | Debug | Show in Test Explorer | Run | Debug
    ✓it("Creates a list of rows with articles", () => {
        const rows = groupByType([article1, article2, article3, article4]);

        expect(rows).toStrictEqual([[article1], [article2], [article3], [article4]]);
    });

    Run | Debug | Show in Test Explorer | Run | Debug
    ✓it("Groups products in rows whith a maximum of 2 products in each row", () => {
        const rows = groupByType([product1, product2, product3]);

        expect(rows).toStrictEqual([[product1, product2], [product3]]);
    });
});
```

bouvet

Initial implementation

```
const groupByType = (searchResultDocuments: SearchResultDocument[]) => {
    return searchResultDocuments.reduce<SearchResultDocument[][]>((rows, document) => {
        if (document.type !== "product") {
            return [...rows, [document]];
        }

        if (rows.some((row) => row.some((doc) => doc.id === document.id))) {
            return rows;
        }

        const allProducts = searchResultDocuments.filter((doc) => doc.type === "product");
        const indexOfCurrentProduct = allProducts.indexOf(document);
        const nextProduct = allProducts[indexOfCurrentProduct + 1];

        if (nextProduct) {
            return [...rows, [document, nextProduct]];
        }

        return [...rows, [document]];
    }, []);
};
```

TPP and open-closed

- Recursion
- Expression -> function
- Open-closed

```
const getDocumentsWithSameType = (
    documents: SearchResultDocument[],
    type: SearchResultDocumentType,
    maxNumberOfDocuments: number
) => {
    return documents.filter((document) => document.type === type).slice(0, maxNumberOfDocuments);
};

const groupByType = (
    documents: SearchResultDocument[],
    groupBy: Partial<Record<SearchResultDocumentType, number>>,
    rows: SearchResultDocument[][] = []
) => {
    const [currentDocument, ...restDocuments] = documents;

    if (!currentDocument) {
        return rows;
    }

    if (!groupBy[currentDocument.type]) {
        return groupByType(restDocuments, groupBy, [...rows, [currentDocument]]);
    }

    const documentsOfSameType = getDocumentsWithSameType(
        restDocuments,
        currentDocument.type,
        groupBy[currentDocument.type] - 1
    );

    const restWithoutUsedDocuments = restDocuments.filter(
        (document) => documentsOfSameType.indexOf(document) === -1
    );

    return groupByType(restWithoutUsedDocuments, groupBy, [
        ...rows,
        [currentDocument, ...documentsOfSameType],
    ]);
};

export default groupByType;
```

bouvet

```
const render = (searchResultDocuments: SearchResultDocument[]) => {
    const container = window.document.createElement("div");

    for (const document of searchResultDocuments) {
        if (document.type !== "product") {
            const row = window.document.createElement("div");
            row.className = document.type;
            container.appendChild(row);

            const documentElement = window.document.createElement("div");
            documentElement.innerText = document.type;
            row.appendChild(documentElement);
        } else {
            const lastRow = container.querySelector("> div:last-child");
            if (lastRow.className === document.type && lastRow.childElementCount === 1) {
                const secondProductElement = window.document.createElement("div");
                secondProductElement.innerText = "product";
                lastRow.appendChild(secondProductElement);
            } else {
                const newProductRow = window.document.createElement("div");
                newProductRow.className = document.type;
                container.appendChild(newProductRow);

                const firstProductElement = window.document.createElement("div");
                firstProductElement.innerText = "product";
                newProductRow.appendChild(firstProductElement);
            }
        }
    }
};
```

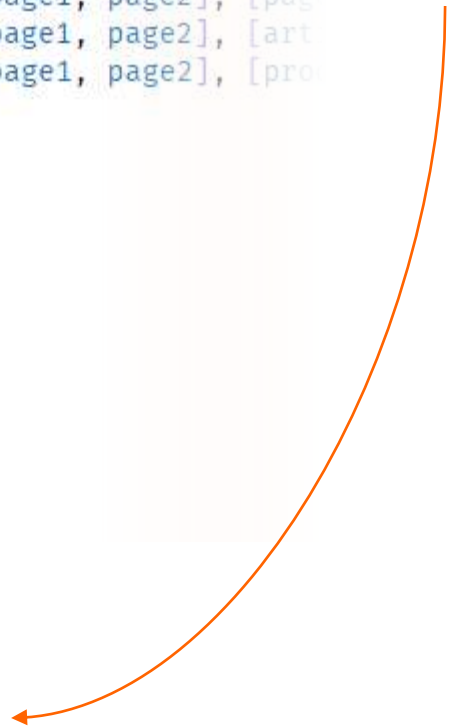
```ts
const getDocumentsWithSameType = (
    documents: SearchResultDocument[],
    type: SearchResultDocumentType,
    maxNumberOfDocuments: number
) => {
    return documents.filter((document) => document.type === type).slice(0, maxNumberOfDocuments);
};

const groupByType = (
    documents: SearchResultDocument[],
    groupBy: Partial<Record<SearchResultDocumentType, number>>,
    rows: SearchResultDocument[][] = []
) => {
    const [currentDocument, ...restDocuments] = documents;

    if (!currentDocument) {
        return rows;
    }

    if (!groupBy[currentDocument.type]) {
        return groupByType(restDocuments, groupBy, [...rows, [currentDocument]]);
    }

    const documentsOfSameType = getDocumentsWithSameType(
        restDocuments,
        currentDocument.type,
        groupBy[currentDocument.type] - 1
    );

    const restWithoutUsedDocuments = restDocuments.filter(
        (document) => documentsOfSameType.indexOf(document) === -1
    );

    return groupByType(restWithoutUsedDocuments, groupBy, [
        ...rows,
        [currentDocument, ...documentsOfSameType],
    ]);
};

export default groupByType;
```

```ts
const renderDocument = (document: SearchResultDocument) => {
    const element = window.document.createElement("div");
    element.innerText = document.id;
    return element;
};

const renderRow = (documents: SearchResultDocument[]) => {
    const row = window.document.createElement("div");

    documents.map(renderDocument).forEach(row.appendChild);

    return row;
};

const render = (documents: SearchResultDocument[]) => {
    const container = window.document.createElement("section");

    const rows = groupByType(documents, { product: 2 });
    rows.map(renderRow).forEach(container.appendChild);

    return container;
};
```

Test cases

```
it.each`
    documents                                                                                  | expectedRows
    ${[page1, page2, page3]}                                                                   | ${[[page1, page2], [pag
    ${[page1, article1, page2, article2, page3]}                                               | ${[[page1, page2], [art
    ${[page1, product1, article1, page2, product2, article2, product3, page3, article3]}       | ${[[page1, page2], [pro
`(
    "Groups pages and articles in rows whith a configurable max amount of each in every row",
    ({ documents, expectedRows }) => {
        const rows = groupByType(documents, { page: 2, article: 3 });

        expect(rows).toStrictEqual(expectedRows);
    }
);
```

```
expectedRows
${[[page1, page2], [page3]]}
${[[page1, page2], [article1, article2], [page3]]}
${[[page1, page2], [product1], [article1, article2, article3], [product2], [product3], [page3]]}
```



Twitter: @martinf_rre
E-mail: martinforre@gmail.com
Tinder: Classified
Lat/Long: 58.9846513, 5.7062923,17

bouvet

Thanks for listening

And you're welcome? I guess?