# Open Source Experience: Storybook

## Issue #25818: Accessibility Addon Blur Preview Border

Allison Chan

## Introduction

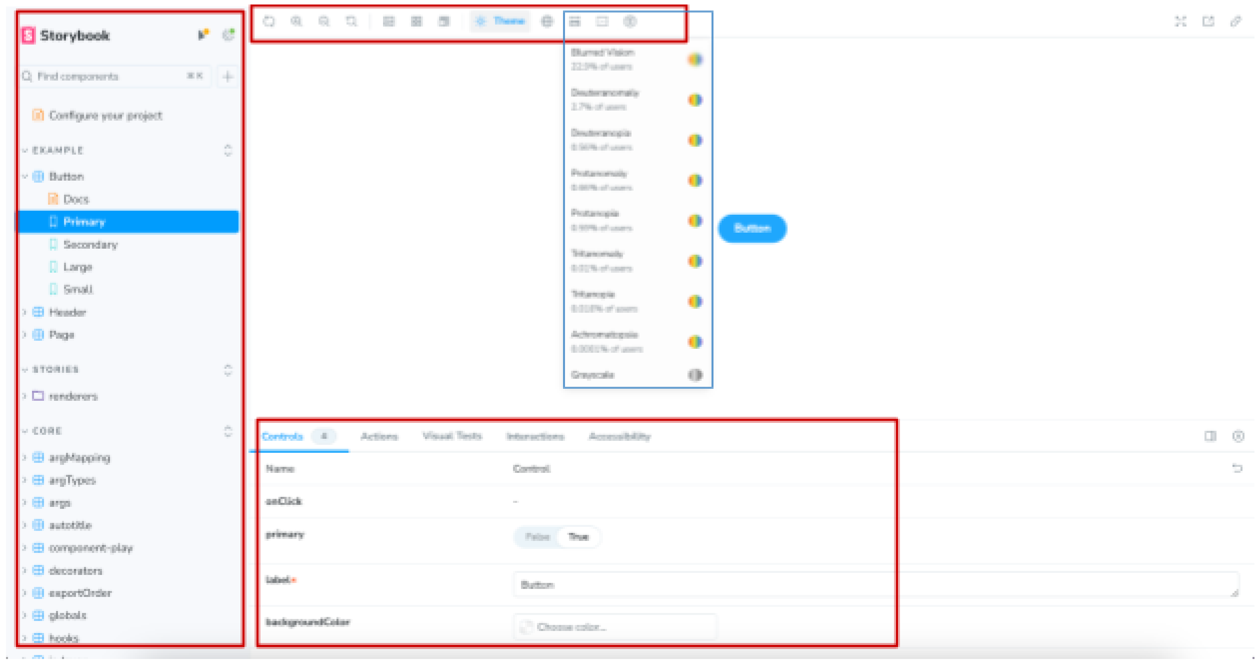
Over the last few weeks, I have worked with my team to contribute to the Storybook open source project. Our team was assigned to fix the issue of a preview border for the accessibility addon. In this article I will share my experiences of working with a team and mentor on the open source project. Through this project, I have gained significant knowledge and understanding towards the importance of collaboration and contributions to projects.

## About Storybook

Storybook is an open source frontend workshop created for building UI (user interface) components and pages insolation. With various tools and addons, Storybook creates an ideal environment for UI development, interaction testing, visual testing and documentation. Each UI component is referred to as a “story” where users are able to focus testing for hard to reach states and edge cases without having to run the whole app. With various add-ons and plugins, Storybook allows for easy customization of functionality and development of durable UI, documentation, and automated user interfaces. The motivation of developing UI faster with greater durability makes Storybook widely adopted across the industry used by hundreds of leading companies and thousands of frontend developers.

Once installed Storybook and added UI components, users are presented with this dashboard to help run and test. Below highlighted are a few key features of the storybook.

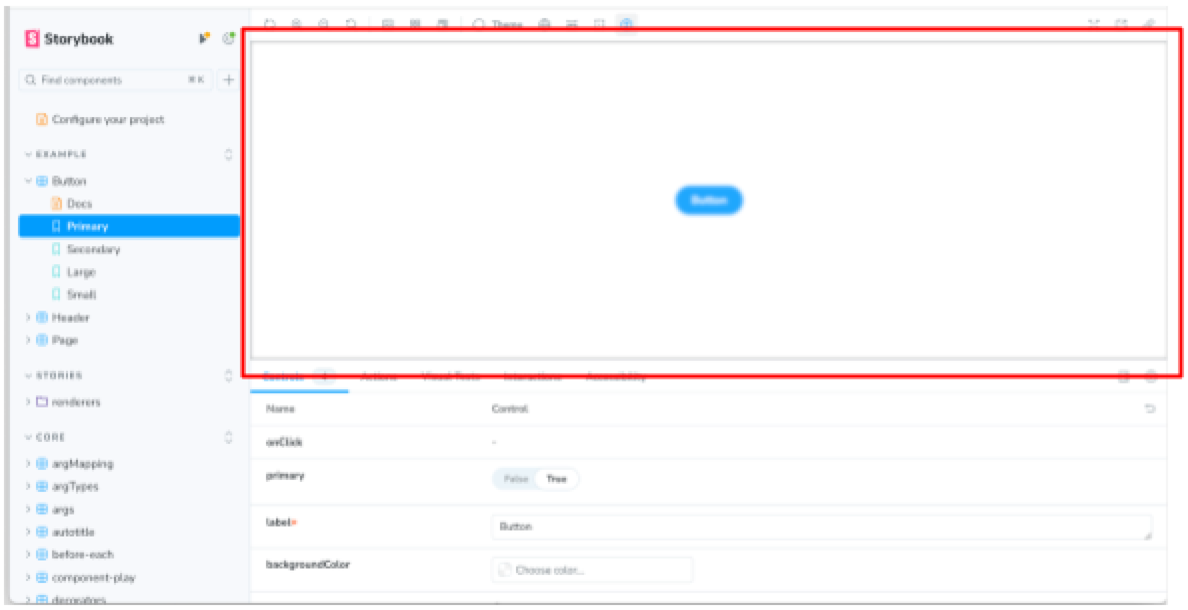
1. The left sidebar presents every UI component in the user’s project. This allows for easy organization of documentation and components through different categories, folders, and components.
2. The panel below includes the different modifications and settings users can customize to fit their requirements.
3. The top toolbar includes more modifications and any addons users utilize. Add-ons allow for users to customize Storybooks with core features. Below is the accessibility addon a11y installed shown through its icon in the toolbar and its tab in the panel.



*Storybook user dashboard*

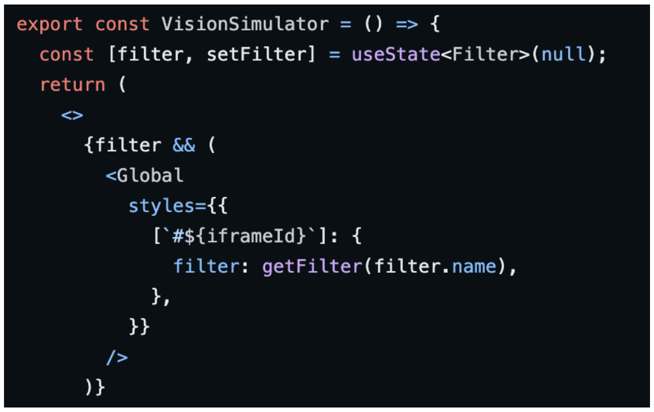
## Issue #25818: Accessibility Addon Border

Issue #25818 The accessibility addon a11y is an addon that allows for UI component testing for compliance with web accessibility standards through various vision filters such as blurred, deuteranomaly, grayscale, etc... When activating the blurred filter, the border of the frame containing the UI components is being blurred as well, adding unwanted visual artifacts in the preview frame. As a team, we determined that this is not intentional, as the frame is not part of the UI component users are testing, thus should be eliminated. Being able to test UI components without any external interference is key to ensure components are accessible to everyone. Storybook provides an easy way in improving the accessibility in web design allowing for all users to navigate, understand, and interact with websites.



*When using the blur filter, the preview frame containing the UI component has a blur as well.*

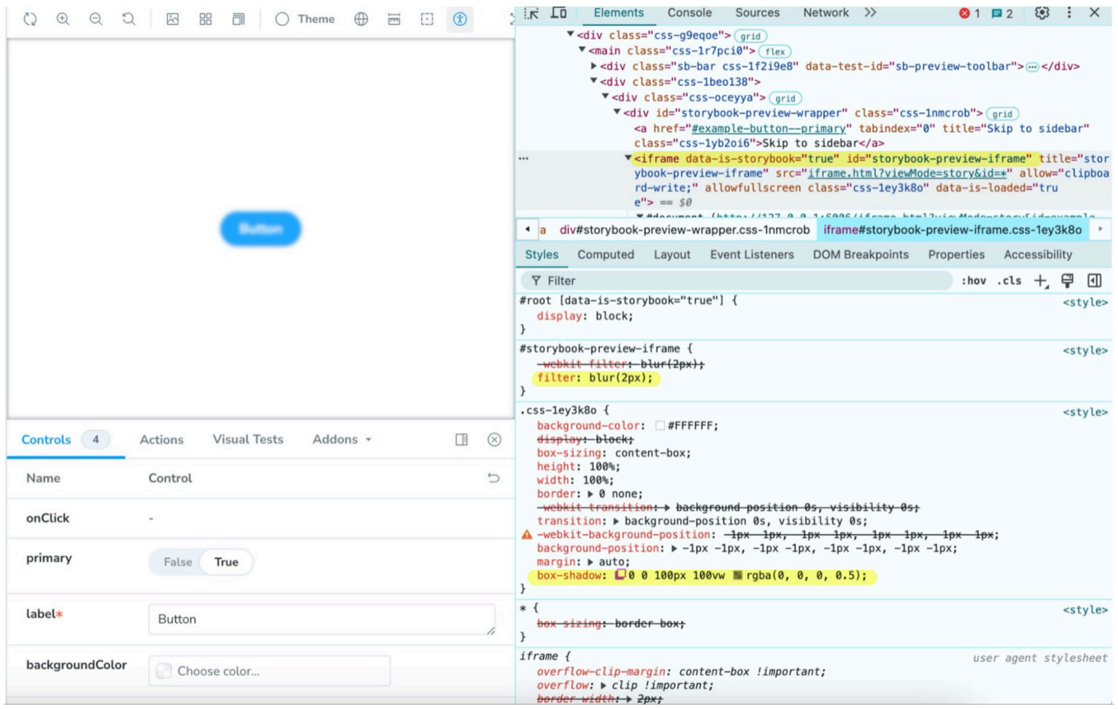
When looking at the Storybook codebase, we started in the accessibility a11y addon and found the *VisionSimulator.tsx* file where the filters are implemented to the iframe



```
export const VisionSimulator = () => {
  const [filter, setFilter] = useState<Filter>(null);
  return (
    <>
      {filter && (
        <Global
          styles={{
            [`#${iframeId}`]: {
              filter: getFilter(filter.name),
            },
          }}
        />
      )}
```

*Within the preview frame, the CSS filters are applied globally. This means not only the component inside the iframe has the blur filter, but also the iframe itself.*

Utilizing the inspect tool in Chrome, this confirmed our results. Here we can see that the blur filter is implemented to the entire iframe of the UI component. Additionally, the preview frame contains a box-shadow creating the border issue with the blur filter.



| Name | Control |
|---|---|
| onClick | - |
| primary | False [✓ True] |
| label* | Button |
| backgroundColor | Choose color... |

```
▼<div class="css-g9eqoe"> grid
  ▼<main class="css-1r7pci0"> flex
    ▶<div class="sb-bar css-1f2i9e8" data-test-id="sb-preview-toolbar">…</div>
    ▼<div class="css-1beo138">
      ▼<div class="css-oceyya"> grid
        ▼<div id="storybook-preview-wrapper" class="css-1nmcrob"> grid
           <a href="#example-button--primary" tabindex="0" title="Skip to sidebar"
           class="css-1yb2oi6">Skip to sidebar</a>
…         ▼<iframe data-is-storybook="true" id="storybook-preview-iframe" title="stor
           ybook-preview-iframe" src="iframe.html?viewMode=story&id=*" allow="clipboa
           rd-write;" allowfullscreen class="css-1ey3k8o" data-is-loaded="tru
           e"> == $0

a   div#storybook-preview-wrapper.css-1nmcrob   iframe#storybook-preview-iframe.css-1ey3k8o
```

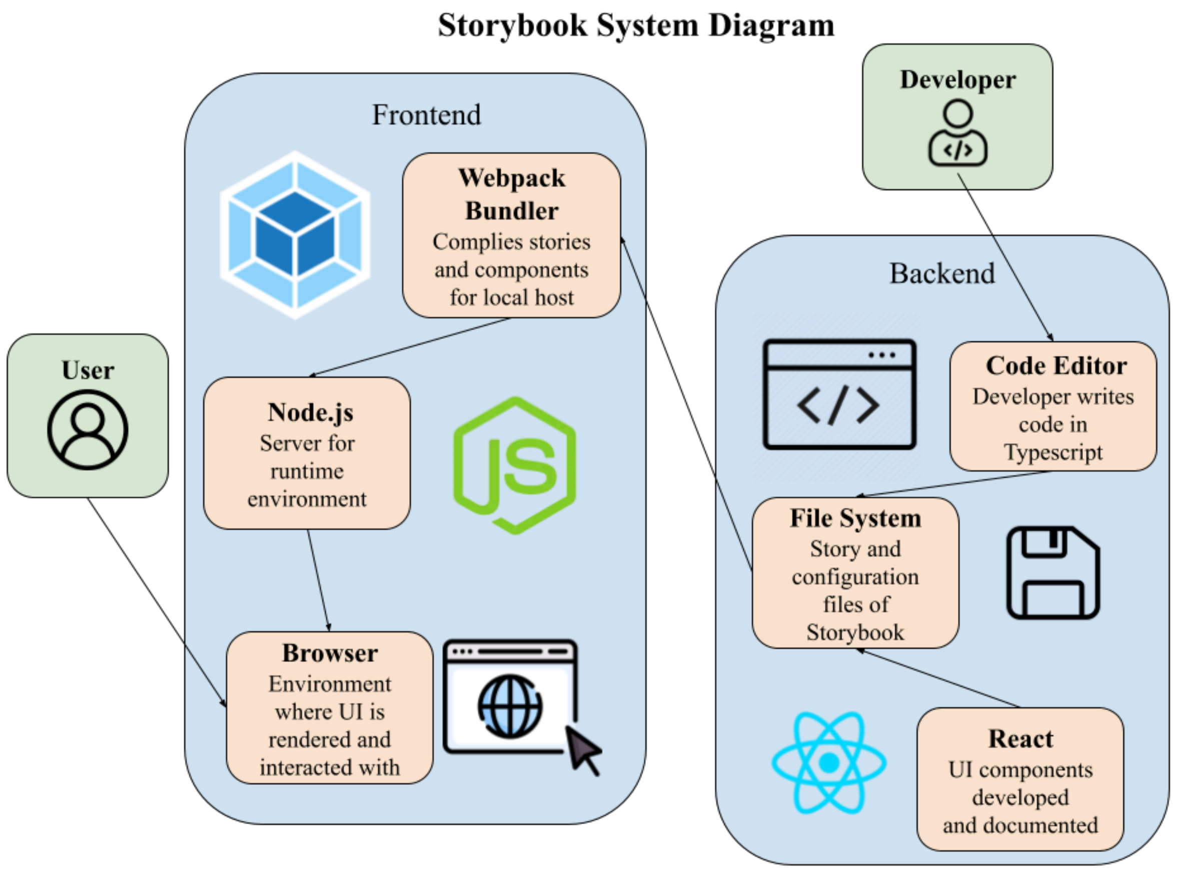
```
#root [data-is-storybook="true"] {                                    <style>
  display: block;
}
#storybook-preview-iframe {                                           <style>
  ~~-webkit-filter: blur(2px);~~
  filter: blur(2px);
}
.css-1ey3k8o {                                                        <style>
  background-color: #FFFFFF;
  ~~display: block;~~
  box-sizing: content-box;
  height: 100%;
  width: 100%;
  border: ▶ 0 none;
  ~~-webkit-transition: ▶ background-position 0s, visibility 0s;~~
  transition: ▶ background-position 0s, visibility 0s;
⚠ -webkit-background-position: ~~-1px -1px, -1px -1px, -1px -1px, -1px -1px~~;
  background-position: ▶ -1px -1px, -1px -1px, -1px -1px, -1px -1px;
  margin: ▶ auto;
  box-shadow: 0 0 100px 100vw rgba(0, 0, 0, 0.5);
}
* {                                                                   <style>
  ~~box-sizing: border-box;~~
}
iframe {                                                 user agent stylesheet
  overflow-clip-margin: content-box !important;
  overflow: ▶ clip !important;
  ~~border-width: ▶ 2px;~~
```

*Inspect tool on Storybook preview frame*

## Codebase Overview

### Tech Stack

| | |
|---|---|
| React | Javascript library used to build user interface components with its own structure, styling and behavior. |
| Typescript | An extension of Javascript used as the main coding language of Storybook for stories, components, and addons |
| Node.js | Used to run Storybook's development server and build processes |


Storybook System Diagram
Frontend
Webpack Bundler
Complies stories and components for local host
Node.js
Server for runtime environment
Browser
Environment where UI is rendered and interacted with
User
Developer
Backend
Code Editor
Developer writes code in Typescript
File System
Story and configuration files of Storybook
React
UI components developed and documented

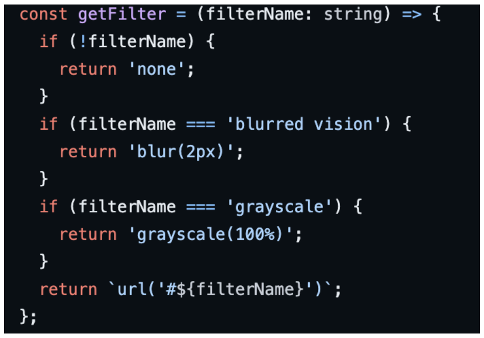
## Choosing A Solution

Looking at the codebase, my team and I came up with 2 different solution ideas. We determined that the issue was caused by the interactions between the filter blur() and the boxShadow CSS. The first one was getting rid of the box shadow. The different filters of the iframe were implemented to the entire iframe, not just the components inside, including the border. Thus eliminating the box shadow will result in the box border no longer being blurred as well, fixing the issue. Using the Inspect tool, we confirmed our hypothesis as when unchecking the box-shadow, simulating deleting it, the border issue was fixed. The other solution we thought of was applying the filter to only the components inside the iframe, not the iframe itself. With this, the box shadow will no longer be affected by the blur filter and not create the border issue.

When unsure of which solution to proceed, we sought feedback from our mentor and Storybook maintainers. They agreed that the better solution would be applying the filters to the components within the iframe. This would be a more permanent solution as future additions of new filters may result in a similar issue.

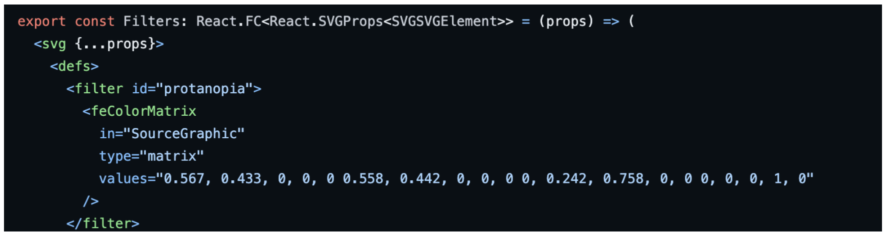
## Different Filter Implementation

Within the accessibility addon, there are two types of filters implemented. The blur and grayscale filters utilize the CSS element methods blur() and grayscale() to create the visual changes. These methods are implemented in the *VisionSimulator.tsx.* Other filters such as deuteranomaly however, are implemented as a SVG (scaleable vector graphic) with custom values to create the visual changes. These values are stored in a separate file called *ColorFilters.tsx.*



```
const getFilter = (filterName: string) => {
  if (!filterName) {
    return 'none';
  }
  if (filterName === 'blurred vision') {
    return 'blur(2px)';
  }
  if (filterName === 'grayscale') {
    return 'grayscale(100%)';
  }
  return `url('#${filterName}')`;
};
```

*Blur and grayscale filters implemented in VisionSimulator.tsx*

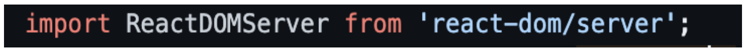


```
export const Filters: React.FC<React.SVGProps<SVGSVGElement>> = (props) => (
  <svg {...props}>
    <defs>
      <filter id="protanopia">
        <feColorMatrix
          in="SourceGraphic"
          type="matrix"
          values="0.567, 0.433, 0, 0, 0 0.558, 0.442, 0, 0, 0 0, 0.242, 0.758, 0, 0 0, 0, 0, 1, 0"
        />
      </filter>
```

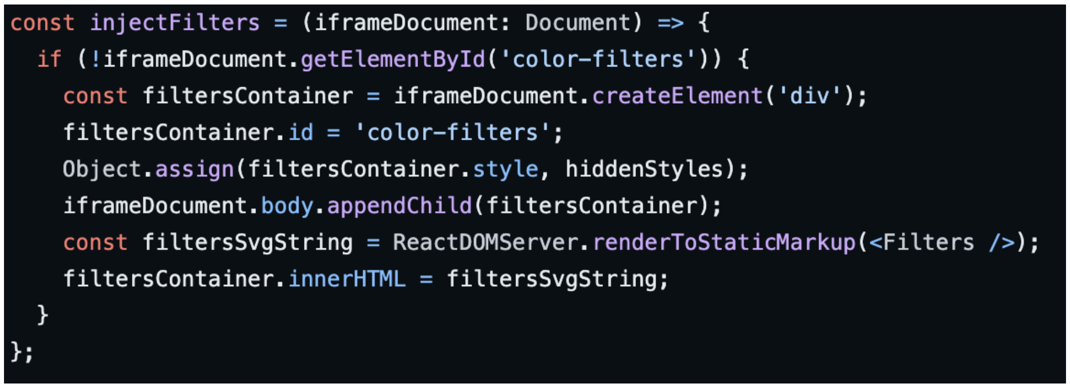
*Example filter implementation with custom values in ColorFilters.tsx*

When creating our solution, our initial solution worked only for the blur and grayscale filters. When reviewing, we realized that previously the filters were applied directly to the iframe as a global component. In our solution, we used the filters to the components within the iFrame, dynamically, not globally, thus the file no longer has access to the other filter values.

With guidance from our mentor, we used the injectFilter function to inject the SVG filters into the VisualSimulator.tsx, giving the file access to the filter values. injectFilter() is a typescript function for adding the SVG filter definitions into the iframe document.  Since iframe itself does not use React, thus the ReactDomServer must be imported to convert a react component, the filters, into a static HTML string. With the addition of injectFilter(), our solution now works for all filters of the accessibility addon.



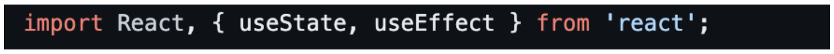
```
import ReactDOMServer from 'react-dom/server';
```



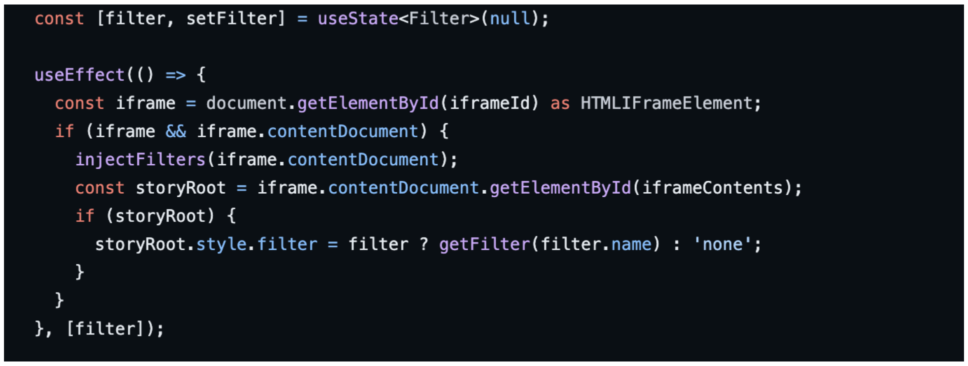
```
const injectFilters = (iframeDocument: Document) => {
  if (!iframeDocument.getElementById('color-filters')) {
    const filtersContainer = iframeDocument.createElement('div');
    filtersContainer.id = 'color-filters';
    Object.assign(filtersContainer.style, hiddenStyles);
    iframeDocument.body.appendChild(filtersContainer);
    const filtersSvgString = ReactDOMServer.renderToStaticMarkup(<Filters />);
    filtersContainer.innerHTML = filtersSvgString;
  }
};
```

## The Final Solution

Here are the final code changes we have made for the issue. Using resources from our mentor, we utilized the react hook useEffect() to implement the filters to the components within the iframe. In order to use useEffect, the React core library must be imported first.



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

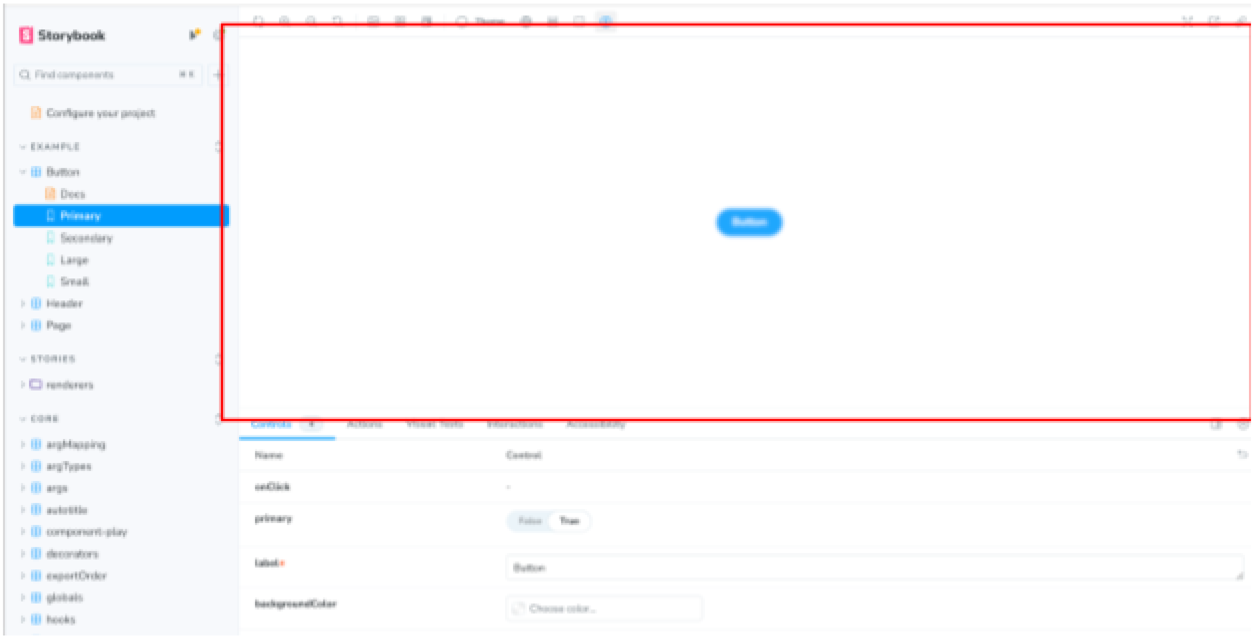


```
  const [filter, setFilter] = useState<Filter>(null);

  useEffect(() => {
    const iframe = document.getElementById(iframeId) as HTMLIFrameElement;
    if (iframe && iframe.contentDocument) {
      injectFilters(iframe.contentDocument);
      const storyRoot = iframe.contentDocument.getElementById(iframeContents);
      if (storyRoot) {
        storyRoot.style.filter = filter ? getFilter(filter.name) : 'none';
      }
    }
  }, [filter]);
```

useEffect() dynamically applies the filters to the components, eliminating the preview border being impacted by the filters. Firstly a state variable called filter is created and set to null. Then as mentioned above, the SVG filters are injected into the iframe document through injectFilters(). To apply the filter to the components inside the iframe, when the root element is found, the filter applies the filter using getFilter() creating the visual changes. If no filter is selected, the filter is set to ‘none’ to remove any existing filters.

Here is the user interface with our code changes. As you can see, the blur filter is being applied. Only the button component is being affected by the filter and box shadow no longer is being blurred.



To test the solution, we firstly looked into the existing tests within the *VisualSimulatorTest.tsx*. However looking carefully and running the test, we realized that the visual simulator tests are all skipped and not being tested. Since our changes have visual outcome, we manually tested through all the different filters within the accessibility a11y addon. We saw that each filter accurately reflected the values of the given filter with our code changes, verifying our solution of implementing the filter to the components. To ensure our code is compatible with Storybook's codebase, we ran Storybook's testing suites for end to end testing . The test report states a few tests have failed, however looking more closely, the tests that failed also failed before our code changes were implemented, verifying our solution once again.

**Pull Request Link:** [https://github.com/storybookjs/storybook/pull/28844](https://github.com/storybookjs/storybook/pull/28844)

## Pull Request Update

Soon after our pull request we heard back from the maintainers of Storybook. Our solution proved to be ineffective as it doesn't follow the use case of stories being embedded with each other. The maintainers provided great information about how our stories only work with stories of the same origin. They mentioned the preferred solution of changing the addon to be similar to the background addon, making the filters act as a decorator. We are currently in communication with the maintainers about a rough outline of how many new changes will be made, following their instructions.

## Takeaways

Through this experience, I have developed my technical and soft skills. At the end of this experience, we did not expect that we were missing crucial information for our solution. It highlights the importance of having good communication with the maintainers and understanding the project fully. There are multiple solutions to the same issue, each having their unique advantages and disadvantages, and ultimately it is up to the maintainer on how they would like to proceed.