View Transitions: Improving User Experience in Web Development

Ogbu Celestine Ogbu

19.03.25

Introduction to View Transitions.

What are View Transitions?

- Definition: View transitions refer to animations or effects that happen when switching between different states or views in a web application.
- Purpose: View Transitions are a way to make websites look smoother and more polished when you move from one page to another or when something on the page changes (like a list updating). Instead of things just popping in or out suddenly, the browser can create nice animations, like fading and sliding to make the change feel more natural.

Do they matter? Yes.

- View Transitions boost user engagement and enable applications appear more modern and professional.

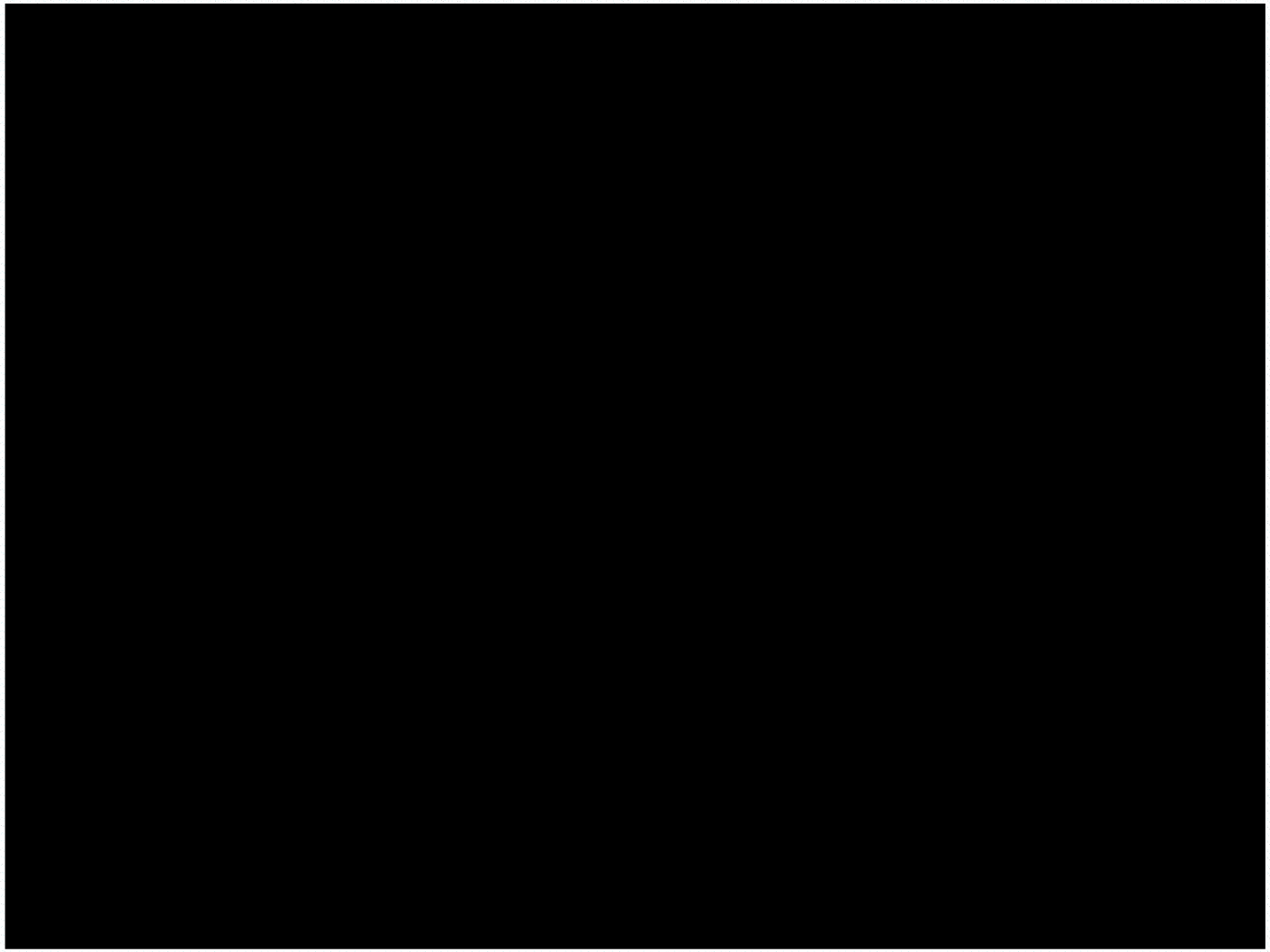
How it works

Imagine you're looking at a photo gallery. When you click on a photo to make it bigger, the browser can take a snapshot of the small photo, then smoothly animate it to the larger version. View Transitions help the browser handle these animations automatically, so that developers don't have to write a lot of complicated codes.

Another example:

- Imagine a to-do list app. When you check off an item, instead of it just disappearing, it could fade out smoothly. Or when you open a new page, the old page could slide out while the new one slides in. These are the kinds of effects View Transitions make easy to create.

- View Transitions make the web feel more fun to use!



Why It is Cool

- Better Experience: Websites feel more like apps, with smooth, fancy animations.
- Easier for Developers: Instead of spending time creating animations from scratch, developers can use built-in tools to make things look good.
- Works Everywhere: Once all browsers support it, websites will look consistent no matter what device or browser you use.

Types of View Transitions

- CSS-Based Transitions: basic animations using CSS properties such as transition and transform. Example: Gradually fading in a new view.
- JavaScript-Driven Animations: advanced animations using libraries like GSAP or Anime.js. Example: Custom animations for specific user actions.
- View Transitions API: a modern API(Application Programming
- Interface) designed to simplify transitions between DOM (Document Object Model) changes.
- Example: Smooth transitions between pages in single-page applications (SPAs).

Enabling automatic transitions

- The CSS @view-transition rule with the navigation: auto; property is used to trigger automatic view transitions for navigations within a web application:
- *@view-transition {*
  navigation: auto;
  }

- **@view-transition at-rule** defines the view transition settings.
- **Navigation: auto;** triggers automatic view transitions when a user navigates from one page to another, the browser automatically applies a transition effect between both pages thereby creating a smooth transition.
- The duration of the transition can be customised to make it slower:

```
@view-transition { navigation: auto; } /* Customize the transition */ ::view-transition-old(root), ::view-transition-new(root) { animation-duration: 1s; }
```

Advantages of View Transitions

- Improved User Experience
- Makes applications feel more responsive and interactive.
- Clear Visual Feedback
- Simplified Navigation
- Helps users follow the flow of the application more easily.

How will VT change the web?

- Faster and smoother websites. Example, a photo gallery with smooth transitions appears faster and more responsive.
- More engaging websites – users will be more interested and engaged with a more dynamic and interactive websites. Example, a shopping site with a smooth animated shopping cart when you click on them.
- Better user experience with a smoother and polished website, example, a seamless transition between pages.

Can we use VT now?

- Yes, but only in some browsers (like Chrome and Edge). Other browsers (like Firefox and Safari) are still catching up. If you're building a website, you can use View Transitions in supported browsers and provide a simpler experience for others.

Challenges and Considerations

- **Performance:**

  Complex animations may affect performance, particularly on lower-end devices.
- **Accessibility:**

  Ensure transitions do not create barriers for users with disabilities.
- **Browser Compatibility**:

  Not all browsers currently support the View Transitions API.

Best Practices

- **Simplicity is vital:**

  Avoid complicated animations that may distract or confuse users.

- **Cross-Device Testing**:

  Ensure transitions work smoothly across various devices and screen sizes.

- **Fallback Options**:

  Provide alternatives for browsers that do not support the View Transitions API.

Real-World Applications

- Single-Page Applications (SPAs):

  Smooth transitions between sections or pages.
- E-Commerce Platforms:

  Animations when adding items to the cart or navigating product pages.
- Social Media Sites:

  Transitions for actions like liking, commenting, or sharing posts.

Disabling view transitions for users who prefer reduced motion

- To create an accessible website for all users, use the following code for those who prefer reduced motion:
- *@media (prefers-reduced-motion) {*
  ::view-transition-group(*),
  ::view-transition-old(*),
  ::view-transition-new(*) {
  animation: none !important;
  }
  }

Conclusion

- Summary:

  View transitions are a powerful way to enhance user experience.

  They can be implemented using CSS, JavaScript, or the View Transitions API.

- Final Thoughts:

  When used effectively, view transitions can make web applications more engaging and user-friendly.

References

- https://developer.mozilla.org/en-US/docs/Web/CSS/transition
- https://developer.mozilla.org/en-US/play
- https://www.patterns.dev/vanilla/view-transitions/
- https://www.smashingmagazine.com/2023/12/view-transitions-api-ui-animations-part1/
- https://www.debugbear.com/blog/view-transitions-spa-without-framework
- https://developer.chrome.com/docs/web-platform/view-transitions/same-document