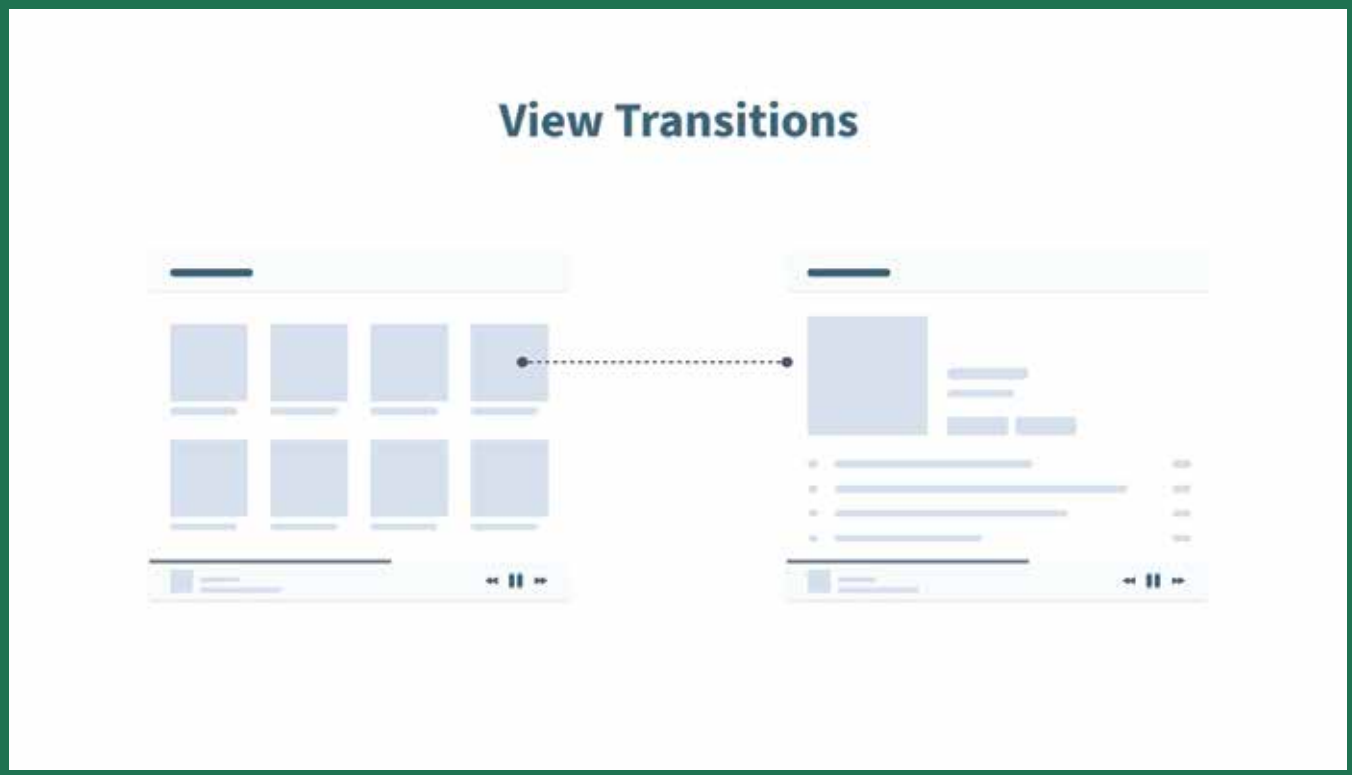
View Transition API

What are “view transitions”? How will they change the web? Can we use them now?

SHIBI BIJU GEORGE

Good afternoon everyone. Today we're going to talk about the View Transition API - a new web technology that's changing how we create animations on websites."
"We'll cover three main questions: What are view transitions? How will they change the web? And can we actually use them in now?"

WHAT ARE VIEW TRANSITIONS?



Imagine you're on a website showing a grid of photos. When you click on one photo, it smoothly expands into a larger, detailed view. That smooth movement - that's a view transition."

"Instead of the page suddenly jumping from one state to another, everything flows smoothly. It's like watching a film rather than flipping through still photos.

The View Transition API provides a mechanism for easily creating animated transitions between different website views.

How It Works (The 4-Step Process):

- Step 1: Capture

User triggers an action (clicks a button, navigates, filters a list)
Browser takes a snapshot of the current view state
All elements and their positions are recorded

- Step 2: Update DOM

Your JavaScript code runs and updates the DOM
Content changes, elements are added/removed, layout shifts
All happens in a single atomic update

"Step 1 is Capture. When someone triggers an action - like clicking a button - the browser takes a snapshot. Think of it like taking a photograph of the current webpage. It records where everything is positioned."

"Step 2 is Update DOM. Now your JavaScript code runs and updates the webpage. Content changes, elements might be added or removed, the layout might shift. All of this happens in one go - what we call an atomic update."

Step 3: Snapshot

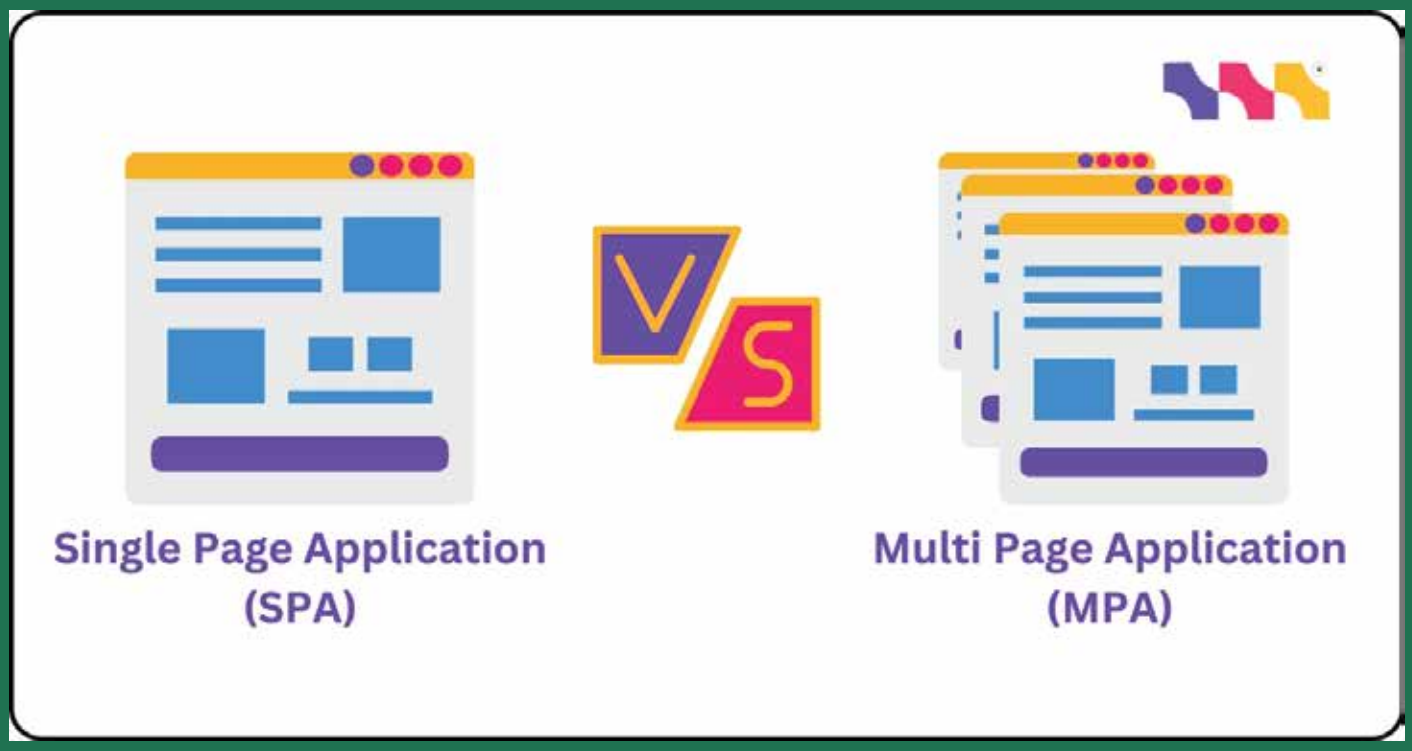
Browser captures the new view state
It knows where every element was before and where it is now
It calculates position and size differences

Step 4: Animate

Browser creates CSS animations that morph between old and new states
Elements appear to move, scale, fade, transform smoothly
All at 60fps using GPU acceleration
Then the browser cleans up the temporary pseudo-elements

Step 3 is Snapshot again - but this time of the new view. The browser now has two snapshots: the old state and the new state. It knows exactly where every element was before and where it is now. It can calculate how much things have moved, grown, or shrunk."
"Step 4 is Animate. This is where the magic happens. The browser creates smooth CSS animations that morph between the old and new states. Elements appear to move, scale, fade in and out - all smoothly at 60 frames per second using your computer's graphics processor. Then it tidies up afterwards."
"The brilliant thing is: all of this happens automatically. The browser does the hard work for you."

TWO TYPES OF VIEW TRANSITIONS:



We have Single Page Applications - or SPAs - where everything happens on one page. And we have Multi Page Applications - MPAs - where you navigate between different pages. The good news is that view transitions work for both.

Same-Document (SPA Transitions)

Used with document.startViewTransition()
For content changes within a single page
Examples: filtering, sorting, expanding sections, tab switching

Cross-Document (MPA Transitions)

Automatic on page navigation
Works between different pages
Both pages just need to opt-in with a CSS rule

For Single Page Applications, we use Same-Document transitions. You call a JavaScript function called document.startViewTransition. This is perfect for things like filtering a list, switching tabs, or expanding a section - all within the same page."
"For Multi Page Applications, we have Cross-Document transitions. These work automatically when you navigate between different pages. Both pages just need to opt in with a simple CSS rule. The browser handles everything else."

Before View Transitions (Complex):

```javascript
// Without view transitions, you might do something like:
async function updateList(filter) {
  // Animate out old content
  await animateOut();

  // Update DOM
  filteredItems = items.filter(item => item.type === filter);
  renderList(filteredItems);

  // Animate in new content
  await animateIn();

  // Handle focus, accessibility, event listeners...
}
```

With View Transitions (Complex):

```javascript
function updateList(filter) {
  document.startViewTransition(() => {
    filteredItems = items.filter(item => item.type === filter);
    renderList(filteredItems);
  });
}
```

That's the entire difference. The browser handles all the animation logic.
Customizing the Animation (CSS):

```css
/* Default fade transition */
::view-transition-old(root) {
  animation: fadeOut 0.5s ease-out;
}

::view-transition-new(root) {
  animation: fadeIn 0.5s ease-in;
}

/* Or customize specific elements */
.product-image {
  view-transition-name: product-hero;
}

::view-transition-old(product-hero) {
  animation: zoomOut 0.4s cubic-bezier(0.4, 0, 1, 1);
}

::view-transition-new(product-hero) {
  animation: zoomIn 0.4s cubic-bezier(0, 0, 0.2, 1);
}
```

"On the left, you see the old way. You had to manually animate things out, update your content, then animate things back in. You had to worry about focus, accessibility, event listeners - it was complicated."
"On the right, with View Transitions, look at how simple it is. You literally wrap your update function with document.startViewTransition. That's it. Three lines of code instead of dozens."
"And if you want to customise how things animate, you use CSS - which you already know. The examples on the right show how to control fade speeds and zoom effects. Clean, simple, powerful."

WHY IS THIS IMPORTANT?

Before View Transitions:

You needed to learn complicated animation libraries (GSAP, Framer Motion)
These libraries add extra code (50KB+) to your website
Different frameworks used different tools
Only professional developers could do it well
Websites often had no animations because it was too hard

After View Transitions:

Anyone can add smooth animations
Animations just work in the browser
No special libraries needed
Much less code to write
Even beginners can make professional-looking animations

Before View Transitions came along, if you wanted animations, you needed to learn complicated animation libraries like GSAP or Framer Motion. These are powerful tools, but they add extra code - often 50 kilobytes or more - to your website. Different frameworks used different tools. Only experienced developers could really do it well. As a result, many websites had no animations at all because it was just too hard."
"After View Transitions? Anyone can add smooth animations. The animations are built right into the browser, so you don't need special libraries. There's much less code to write. Even beginners can make professional-looking animations. This is democratising web animation."

HOW DOES THIS CHANGE THE WEB?

Change 1: Less Code to Write

Before: Write 50 lines of animation code
Now: Write 3 lines with View Transitions
You can actually focus on the real stuff instead of animation complexity.

Change 2: Websites Feel Better

Before: Many websites had no animations (too complicated)
Now: Anyone can add smooth animations
Regular websites will look as polished as big company websites.

Change 3: Works Everywhere

Before: React websites used one animation tool, Vue websites used another, vanilla JavaScript did something different
Now: Everyone uses View Transitions. Same tool for everyone.

"Change number one: Less Code to Write. We're talking about writing 3 lines of code instead of 50. This isn't just about saving time - it's about being able to focus on the actual functionality of your website instead of wrestling with animation complexity."
"Change number two: Websites Feel Better. Before, many websites had no animations because it was too complicated. Now anyone can add smooth animations. This means regular websites - not just those built by big companies - will feel polished and professional."

Change 4: Web Apps Feel Like Phone Apps

Before: Web apps felt different from phone apps (less smooth)
Now: Web apps can be just as smooth as phone apps
Users won't feel the difference anymore.

Before View Transitions, web applications felt different from phone apps - they were less smooth, less polished. Now, with View Transitions, web apps can be just as smooth as native phone applications. Users won't feel the difference any more. This is huge for web development.

CAN WE USE THEM NOW?

Desktop Browsers:

Chrome: 65%+ support
Firefox: 25%+ support (new!)
Safari: 10%+ support
Edge: (included in Chrome stats)

Mobile Browsers:

Chrome Mobile: 65%+ support
Firefox Mobile: 25%+ support (new!)
Safari iOS: 10%+ support (iOS 18+)
Samsung: 60%+ support

TOTAL: 90%+ of modern browsers support View Transitions

Current support is not widely adopted yet, as of the present moment, the support for the View Transition API is exclusively in Chrome version 111 and beyond.

LIMITATIONS

What View Transitions Are Good For:

- DOM state changes within a page
- Page navigations (same-origin)
- Filtering, sorting, expanding content
- Switching between tabs or sections
- Gallery/lightbox interactions

What They're NOT For:

- Streaming/real-time updates (they need discrete state changes)
- Complex 3D animations (use WebGL for that)
- Very large DOM changes (might impact performance)

View Transitions are excellent for certain things. They're perfect for DOM state changes within a page - things like filtering, sorting, or expanding content. They work brilliantly for switching between tabs or sections. They're great for gallery interactions and lightbox effects. And they're ideal for page navigations within the same origin."
"However, they're not designed for everything. They're not suitable for streaming or real-time updates because they need discrete state changes - a clear before and after. They're not for complex 3D animations - for that, you'd want to use WebGL instead. And if you're making very large DOM changes, you might see performance impacts.