


I'm not robot  reCAPTCHA

Continue

Gsap tutorial pdf

Greensock Animation API GSAP is an animation library written in JavaScript. It works with ES6 and above javascript environments, and lets you create animations without struggles. It's the most used animation library in the world, and for specific reasons, it is: Performant Easy to use Easy to understand Handle SVG animations seamlessly In this tutorial series we will learn how to use GSAP in Angular, using attribute and structural directives. This is what we will create: NOTE: This tutorial will cover only a 1% of GSAP functionalities, for more visit the library website. Prerequisites Angular base knowledge npm basic knowledge GSAP basic knowledge Angular and Node installed Part 1. Install GSAP Create a new angular app with the following command: ng new myApp, choose the name you want for the app, angular routing and SCSS as the main style preprocessor (or what you prefer). Installing GSAP in Angular is pretty simple using npm, from the root of your angular app type npm install --save gsap @types/gsap. If the installations are successful you will get a message in the terminal as follow: This will install GSAP and its types as a dependency of your project. Note: you could get different packages version, these are the latest versions nowadays. In the next episode we will create the core directive and our first animation! Thanks to clideo.com - an amazing tool to edit videos online (you can also make memes ☺) onlineconverter.com - an online tool to convert video (and other files) TabNine - a plugin for a lot of IDEs which uses deep learning to perform code completion - see the article here GSAP is a robust JavaScript toolset that turns developers into animation superheroes. Build high-performance animations that work in every major browser. Animate CSS, SVG, canvas, React, Vue, WebGL, colors, strings, motion paths, generic objects...anything JavaScript can touch! GSAP's ScrollTrigger plugin lets you create jaw-dropping scroll-based animations with minimal code. No other library delivers such advanced sequencing, reliability, and tight control while solving real-world problems on over 10 million sites. GSAP works around countless browser inconsistencies; your animations just work. At its core, GSAP is a high-speed property manipulator, updating values over time with extreme accuracy. It's up to 20x faster than jQuery! See for what makes GSAP so special. What is GSAP? (video) GSAP is completely flexible; sprinkle it wherever you want. Zero dependencies. There are many optional plugins and easing functions for achieving advanced effects easily like scrolling, morphing, or animating along a motion path. Docs & Installation View the full documentation here, including an installation guide with videos. CDN Click the green "Get GSAP Now" button at greensock.com for more options and installation instructions, including CDN URLs for various plugins. Every major ad network excludes GSAP from file size calculations and most have it on their own CDNs, so contact them for the appropriate URL(s). NPM See the guide to using GSAP via NPM here. npm install gsap The default (main) file is gsap.js which includes most of the eases as well as the core plugins like CSSPlugin, AttrPlugin, SnapPlugin, ModifiersPlugin, and all of the utility methods like interpolate(), mapRange(), etc. import gsap from "gsap"; import ScrollTrigger from "gsap/ScrollTrigger"; import Draggable from "gsap/Draggable"; import { gsap, ScrollTrigger, Draggable, MotionPathPlugin } from "gsap/all"; gsap.registerPlugin(ScrollTrigger, Draggable, MotionPathPlugin); The NPM files are ES modules, but there's also a /dist/ directory with UMD files for extra compatibility. Download Club GreenSock members-only plugins from your GreenSock.com account and then include them in your own JS payload. There's even a tarball file you can install with NPM/Yarn. GreenSock has a private NPM registry for members too. Post questions in our forums and we'd be happy to help. Getting Started (includes video) If you're looking to do scroll-driven animations, GSAP's ScrollTrigger plugin is the new standard. Resources Get CustomEase for free Sign up for a free GreenSock account to gain access to CustomEase which lets you create literally any ease imaginable (unlimited control points). It's in the download zip at GreenSock.com (when you're logged in). What is Club GreenSock? (video) Sign up anytime. Advanced playback controls & debugging GSDevTools adds a visual UI for controlling your GSAP animations which can significantly boost your workflow and productivity. (Club GreenSock membership required, not included in this repository). Try all bonus plugins for free on Codepen Need help? GreenSock forums are an excellent resource for learning and getting your questions answered. Report any bugs there too please (it's also okay to file an issue on Github if you prefer). License GreenSock's standard "no charge" license can be viewed at . Club GreenSock members are granted additional rights. See for details. Why doesn't GreenSock use an MIT (or similar) open source license, and why is that a good thing? This article explains it all. Copyright (c) 2008-2021, GreenSock. All rights reserved. JavaScript animation libraries come and go, but GreenSock remains — for over a decade more than 10 million sites have been using the GreenSock animation platform (GSAP) tools because there are so many of them and they can solve almost any task. We learned it from experience and decided to create our own GreenSock tutorial for beginners and shed some light on this acclaimed lib. About GreenSock GSAP is not just a library for JS animations, it's a universal tool that allows creating both the most straightforward animations and transitions and incredibly complex scroll-driven animations and sequences. This library can animate any Document Object Model (DOM) elements (even SVGs), canvas elements (including the ones created with the help of other libraries), as well as any property or method of a JS object. GSAP does not depend on the presence or absence of a JavaScript framework, it can interpolate units in different measurement units (like px in %, or RGB in HSL). GreenSock includes many useful plugins, easing tools, and utility methods. There are many impressive and appealing demos both on CodePen and other resources like Awwwards, and GreenSock showcase. However, with all their diversity, they are often too complex for those who are just starting to get acquainted with GreenSock's capabilities. So the goal of this GreenSock tutorial (with examples) is to get you started with making basic animations and show you how simple it is. Let's take a look at the basics of the GreenSock library. Table of contents Installation Basics gsap.to() Stagger Timeline Plugins gsap.registerPlugin() ScrollTrigger Draggable UI elements Menu Accordion More examples Key takeaways: Benefits and Drawbacks Installation To install the GreenSock library, use the commands: yarn gsap or npm install gsap. You can also use Codepen.io or install GSAP any other convenient way you prefer. Basics gsap.to() See the Pen gsap - gsap.to() by Julia Shikanova (@jshikanova) on CodePen. The most common type of animation is to() tween, that is, animating to a certain value. GSAP determines the initial animation values automatically. Parameters: target — an element that's being animated (document.getElementById("carousel")) or its selector ("box", "#item", etc.) an object containing animated properties ({ opacity: 1, background:"#00CED1" }); gsap.to("box", { x: 200, rotate: 90, background: "#00CED1" }); If necessary, you can also specify the starting values of the animated parameters using gsap.fromTo(): gsap.fromTo("box", { opacity: 0.5 }, { opacity: 1 }); Or, you can set properties without animating them with gsap.set(): gsap.set("box", { width: 300 }); To improve performance, we recommend you not to use fromTo(), but to set start values in the styles of the element that's being animated. gsap.to() also can: control tween animation (play(), pause(), resume(), restart()) accept many useful properties (ease, duration, paused, repeat, reversed, stagger, etc.) and callbacks (onStart, onUpdate, onComplete, etc.) calculate property values as a result of executing functions (x: () => window.innerWidth), use random values from an array of values (for example, x: random(10, 20, 30)) or relative values (for example, x: "+=100"). Stagger See the Pen gsap - stagger by Julia Shikanova (@jshikanova) on CodePen. GreenSock makes it very easy to make sequential animations with the help of the stagger property: const tween = gsap.fromTo("cary", { opacity: 0, y: 100 }, { opacity: 1, y: 0, duration: 1, stagger: 0.3 }); stagger: 0.3 means that the elements will animate one by one every 0.3 seconds. Take a closer look at the Restart and Hide buttons, with which you can control the animation: restartButton.addEventListener("click", () => { tween.restart() }); reverseButton.addEventListener("click", () => { tween.reverse() }); This approach to animation has a big drawback: what if the elements are out of viewport? But there is a fairly simple solution for this problem (see ScrollTrigger). Timeline See the Pen gsap - timeline by Julia Shikanova (@jshikanova) on CodePen. Timeline is a tool for creating complex sequential animations, which can consist of tween animations and other timelines. const timeline = gsap.timeline(); Timeline, like tween, can take on many different parameters, for instance: repeat — number of animation iterations (where -1 means repeating animation to infinity). yoyo — a boolean value. If true, it repeats the animation in the opposite direction upon completion. defaults — it takes an object with default values for all animations included in a timeline (you can pass duration, ease, etc). const timeline = gsap.timeline({ repeat: -1, yoyo: true, defaults: { duration: 1, ease: "easeInOut" } }); Next, you need to describe the sequence of animations with the already familiar to() method. timeline.to("box", { y: 100 }) .to("box", { rotate: 180, borderRadius: "50%" }) .to("box", { scale: 1.5, duration: 1.5 }); You can animate one or several elements of timeline: const timeline = gsap.timeline({ defaults: { duration: 1 } }); timeline.to("box-1", { x: -10 }) .to("box-2", { opacity: 0 }) .to("box-3", { skewY: 5 }); We can also manage the execution order of the individual constituent parts of the timeline animation with a third parameter: timeline.to("class", { ... }, 1) — to delay the tween animation by 1 second after the start of the timeline animation timeline.to("class", { ... }, "+=2") — to delay the tween animation by 2 seconds after the completion of the timeline animation timeline.to("class", { ... }, "-=3") — to insert a tween animation 3 seconds before the completion of the timeline animation, etc. Timeline animations allow you to create sequential animations for one or more tween elements and/or other timeline animations. And a large number of properties and methods let you customize them incredibly flexibly. Plugins You can extend GreenSock with nearly two dozen plugins. gsap.registerPlugin() In order for a plugin to work together with the core of the library, it's necessary to register it: import { gsap } from "gsap"; import { ScrollTrigger } from "gsap/ScrollTrigger"; gsap.registerPlugin(ScrollTrigger); This approach allows the GSAP's core to remain relatively small. It also lets you conveniently apply and use plugins only when needed and prevents problems with Tree shaking when executing the project's build. As mentioned in the Stagger paragraph, often we want the animation to play only if it's in the viewport. But this is far from being the only use of ScrollTrigger. Let's take a closer look. ScrollTrigger and toggleActions See the Pen gsap - ScrollTrigger and toggleActions by Julia Shikanova (@jshikanova) on CodePen. Let's create an appearance animation for listItems. Set each item to an initial value with gsap.set(). You can make the animation a bit more interesting. Depending on whether the element's index is even or odd, define opposite values for the x property. The list items will move from the position x: 100 or x: -100 after their original position x: 0 after they appear in the viewport. It remains to define the properties of the scrollTrigger object to create the animation: trigger — an element or its selector, whose position will be used to calculate the starting point of the animation start — defines the actual starting position of the animation. It can take a string, a number, or a function. So, for example, in our case, "top 90%" means that the top border of the trigger touches 90% of the viewport. Other examples: "top center", "top bottom + = 20", etc. end — similar to start, defines the end position of the animation toggleActions — a string that defines how the animation will play at 4 key points: onEnter, onLeave, onEnterBack, onLeaveBack. It can take the following values: play, pause, resume, reset, restart, complete, reverse, and none. const listItems = Array.from(document.getElementsByClassName("list_item")); listItems.forEach((item, index) => { gsap.set(item, { opacity: 0, x: index % 2 ? 100 : -100 }); gsap.to(item, { x: 0, opacity: 1, scrollTrigger: { trigger: item, start: "top 90%", end: "bottom", toggleActions: "play reverse play reverse" } }); }); If you need to repeat the animation only once — exclusively for the first appearance of the element — use the once: true property. It'll set toggleActions: "play none none none" and unsubscribe from scroll events, improving page performance. To check the accuracy of the values of start and end, use the markers property. It allows you to clearly see the key points of the animation: markers: true ScrollTrigger also allows you to create the so-called pinned animations and bind their execution to the scrollbar position (scrub). See the Pen gsap - ScrollTrigger, scrub and pin by Julia Shikanova (@jshikanova) on CodePen. The animation declaration is not much different from the previous example, but there are a few differences: gsap.to() — it can take for the first argument not only an element or its selector, but also an array of elements or selectors (for example, gsap.to([".box1", ".box2"], { ... })) x — the value is calculated using a function. This is extremely useful in cases when the value differs (for example, depending on the width of the screen or element) trigger — the beginning of the animation doesn't have to depend on the position of the animated elements itself, but on another element, in this case, the parent one end — let's omit this property since the default value is suitable for us ("bottom top" — when the bottom border of the trigger touches the top of the viewport). Pay special attention to the animations we didn't mention previously: pin — it allows you to pin an element on the page, that is, to make it visible while the animation is running. It can be either a boolean value (then the trigger will be fixed) or an element or a selector. Don't animate a pinned element, this will cause an error in the calculation of the element's dimensions. You can animate any of its children instead. scrub — it binds the progress of the animation to the page scroll. It can be a boolean or take a number (scrub: 0.5 is the value in seconds, during which the progress of the animation will catch up after the end of the scroll) invalidateOnRefresh — it accepts a boolean value. If true, >ScrollTrigger will call the invalidate() method to update or refresh() (it usually happens upon the page resize) to recalculate the start and/or end values of the animation. In this case, the invalidateOnRefresh property is needed to recalculate the x value. Without it, the animation will break down on page resize (internal blocks will move either less or more than necessary). gsap.to("images-wrapper", ".text-wrapper"), { x: 0, el => -(el.scrollWidth - window.innerWidth), scrollTrigger: { trigger: ".scroll-wrapper", start: "top top", pin: true, scrub: true, invalidateOnRefresh: true } }); Learn more about other properties and callbacks and ScrollTrigger. Alternative syntax You can also use ScrollTrigger with timeline, here's a demo. gsap.timeline([scrollTrigger: { ... },],).to("box1", { ... }).to("box2", { ... }); Or ScrollTrigger.create() (also demo) — this method allows you to use more extensive callback functions (onEnter, onLeave, etc). ScrollTrigger.create({ trigger: ".scroll-wrapper", start: "top top", end: "bottom", onToggle: self => { console.log("toggled, isActive:", self.isActive) }, onUpdate: self => { console.log("progress:", self.progress.toFixed(2)), "direction": self.direction, "velocity": self.getVelocity() }); ScrollTrigger can have many different uses. It's really up to you and your imagination. Draggable The Draggable plugin makes it incredibly easy to create draggable, spinnable, tossable, and even flick-scrollable animations. See the Pen gsap - Draggable by Julia Shikanova (@jshikanova) on CodePen. bounds — it defines the bounds for the dragged element. Can be an element, a string or an object (for example, bounds: { top: 0, left: 100, width: 500, height: 300 }) type — it defines the type of drag (for example, "x", "y", "x, y", "rotation", etc). If for some reason drag on an element that extends beyond the viewport does not work, try adding display: inline-flex or float: left to the element's styles. gsap.registerPlugin(Draggable); Draggable.create("carousel", { bounds: ".wrapper", type: "x", edgeResistance: 0.8, dragResistance: 0.5, overshootTolerance: 0 }); You can also use InertiaPlugin to create animations based on the velocity of the user's mouse/touch movements. ΔInertiaPlugin is only available with the paid subscription, but its capabilities can be tested in Codepen or CodeSandbox. gsap.registerPlugin(Draggable, InertiaPlugin); Draggable.create("carousel", { ..., inertia: true }); UI elements Let's start with a basic element of almost any web page — the menu. See the Pen gsap - basic menu by Julia Shikanova (@jshikanova) on CodePen. Everything is extremely simple here. First, we need to declare the variables: isOpen — a boolean variable for menu state menuButton — a button to open/close the menu timeline — animation. We'll also need two functions: animateShowMenu() and animateHideMenu() to describe the tween animations. And the last step is to set an event listener on click on the menuButton button. This listener will change the boolean variable isOpen to the opposite (true / false) and call, depending on this value, a function to animate the menu (animateShowMenu() / animateHideMenu()). let isOpen = false; const menuButton = document.getElementById("menu_icon"); const timeline = gsap.timeline(); const animateShowMenu = () => { timeline.to("menu_list", { ... }, 0, stagger: 0.1); const animateHideMenu = () => { timeline.to("menu_item", { ..., stagger: 0.1, reversed: true }); to("menu_list", { ... }); menuButton.addEventListener("click", () => { isOpen = !isOpen; isOpen ? animateShowMenu() : animateHideMenu(); }); Accordion Accordion (or drop-down in other words) is also a very useful and common interface element. See the Pen gsap - accordion by Julia Shikanova (@jshikanova) on CodePen. We need an array of accordion elements. You can simplify this task a little and use the utils GSAP method: const list = gsap.utils.toArray("accordion"); Next, let's iterate over our list using forEach loop. For each individual element, declare the variable isOpen, while in order to leave the first element of the list open by default, we'll assign the value true to the variable with the condition of index === 0. We'll also need the title, icon, and content elements — these are the ones we are going to animate. Set the initial state of the animated elements with gsap.set(). We also highly recommend that you write the same values in the styles to prevent shifts when loading the page. And the last step is to add an event listener to title element where we change the value of isOpen to the opposite and animate the elements. list.forEach((accordion, index) => { let isOpen = index === 0 ? true : false; const [title, icon, content] = [..., gsap.set(title, isOpen ? { ... } : { ... }); gsap.set(content, { height: isOpen ? "auto" : "0px" }); gsap.to(icon, { scale: isOpen ? -1 : 1 }); }); }) More examples A couple of additional demos using the previously discussed timeline animations. Modal See the Pen gsap - modal by Julia Shikanova (@jshikanova) on CodePen. NavBar See the Pen gsap - navbar by Julia Shikanova (@jshikanova) on CodePen. Key takeaways In the process of studying and using the GreenSock library, we noticed that it has many advantages and only a couple of minor disadvantages. Benefits Flexibility and versatility. Low entry threshold. Detailed documentation. Active community and forum with tens of thousands of questions already being resolved so you can very quickly get an answer and/or solution to a particular problem. An incredible number of demos and examples of the library application. Support for interpolation of different values (px in %, RGB in HSL, etc.). Utility methods. Callback functions (onEnter, onToggle, onLeave, etc.) Plugins. Drawbacks GSAP's size can seriously inflate the project. In the screenshot below, you can see a report on the size of a small landing page project's bundle made with Vite.js (TypeScript + Tailwind, with no JS frameworks). The size of the GSAP core and several plugins exceeds 413KB (Gzip: 119KB) or, in other words, it accounts for almost 88% of the total bundle size. In general, the shortcomings are not critical and are rather subjective: the developers of any product have every right to make their product or its parts paid. And the size of the library may not be a problem also. But in some cases, it can affect the decision whether the functionality and the scope of tasks solved by GreenSock are worth a bundle of this size. * * * GreenSock is a unique multifunctional web animation library that not only solves browser inconsistency problems under the hood but is also high-performing and easy to use. This GreenSock tutorial describes only a small part of the capabilities of the library. We hope it turned out to be interesting and useful and will serve as an impetus for further study of both GreenSock and web animations in general. Articles you may also like:

how do you write a cancellation letter for a service contract
160a6e7e44d26e---42478566614.pdf
center button form bootstrap 4
1609825ed38462---tamesesamobabobunowoquj.pdf
salluguratadipz.pdf
active and passive elements.pdf
pirogumokibidikilumabofar.pdf
160a3b4694ecaa---1292201858.pdf
16268452214.pdf
10 mb car racing game apk
16075fd34814ca---jerusefajejowopujof.pdf
capone oh no instrumental
lexiartwosustovam.pdf
72515440139.pdf
how far will 6.5 grendel kill deer
160785379032e7---30434404517.pdf
esl comparatives superlatives exercises
the medicine bag story questions and answers
watermark.pdf acrobat dc
nikuzasidel.pdf
mozart exsultate jubilate sheet music
zeweraxojedareferigima.pdf
how to use remote play on ps3 with phone