**How to represent large numbers in your Node.js app**

Author: Joseph Mawa

Date: April 14, 2023

It is difficult for computers to represent numbers with several significant digits accurately without loss of precision. Integers that exceed the maximum safe integer limit in JavaScript lose precision when you store them as ordinary integers.

In the JavaScript ecosystem, you can use BigInt to work with large integers. However, you can also use third-party packages with features similar to BigInt.

This article will be a complete guide to managing large numbers using BigInt and popular packages that offer similar features. We will also compare the third-party packages’ use cases, strengths, and weaknesses.

**How does JavaScript encode numbers?**

The challenge of precision loss when representing large numbers is not unique to JavaScript. Internally, JavaScript uses the double-precision binary floating-point format to represent numbers.

**The double-precision binary floating-point format**

The double-precision binary floating-point format is defined by IEEE standard 754. It uses 64 bits to represent a signed floating point number. A number expressed in double-precision binary floating-point notation is comprised of three parts: the sign, mantissa, and exponent.

The double-precision binary floating-point format distributes the 64 bits among these three parts. It uses one bit to encode the sign, 11 bits for encoding the biased exponent, and 52 bits for the mantissa or significand.

The example below shows the internal double-precision binary floating-point number representation of the decimal number -1.7976931348623157e+308. I have used the • character to separate the encoding for the three parts.

The first bit encodes the sign. Because we are encoding a negative number, its value is one. If we were encoding a positive number, its value would be zero. The subsequent 11 bits encode the biased exponent, and the last 52 encode the mantissa.

**Build a SwiftUI customizable segmented control**

Author: Alejandro Ulate Fallas

Date: April 14, 2023

During 2022, there were more than 4.5M apps and games in the App Store. The sheer volume of options makes it important for apps to provide a fresh, novel experience or find other ways of standing out.

Creating a unique interface can help make an app more attractive and go a long way toward increasing its user acquisition and retention. Customizing the default look and feel of the controls or creating unique tabs or a switch for dark/light themes can help make an application more appealing.

In this tutorial, we’ll explore how to create a customizable segmented control in SwiftUI. First, we’ll explore this scenario by discussing how to use the SwiftUI Picker with a segmented style. Then, we’ll explore its limitations. Finally, we’ll build our own picker that allows for easy customization and usage.

**Prerequisites**

To follow along with this guide, you should have the following:

* Working knowledge of iOS development and Swift
* Experience creating user interfaces using SwiftUI

**Getting started**

To follow along with this SwiftUI Picker showcase project, see the open source code on GitHub. If you feel like jumping straight into a specific section, you can check out the different Git branches. There’s a specific branch for each section of this article.

Here’s a quick rundown of some important files you should be aware of:

* Domain: Defines business logic objects, in this case, enum structures; you won’t need to change anything in this directory
* Extensions: Contains a couple of handy extensions for this article; you won’t need to change anything in this directory
* UI: Stores all reusable components in the project. In this case, it contains PickerPlus, the customizable picker that we’ll build in the tutorial
* PickerLimitations.swift, StyledPicker.swift, DefaultPickerExamples.swift: SwiftUI Views that will help visualize the theory explained in this article

**Using the SwiftUI Picker**

Before we start creating custom component Views, let’s take a closer look at how the SwiftUI default Picker works. This will help us better understand how to build Views that support multiple configurations.

The SwiftUI Picker is a control for selecting from a set of mutually exclusive values. Let’s look at a few Picker examples: simple, complex, and segmented.

**11 Figma widgets to speed up your workflow**

Author: Daniel Schwarz

Date: April 14, 2023

Figma widgets are the new thing in Figma. Think of them as plugins except that they become a part of the shared workspace, so if you do something in a document using a widget, everyone that has access to that document will see it too (unlike plugins, which are for private use).

Product teams can use widgets to create notes, create to-do lists, link to external webpages, and so on.

Let’s take a look at some widgets that you and your team might appreciate — they’re intended to help you save time and make your job easier.

**Simple Vote**

Simple Vote enables product teams to vote on ideas (or anything really). There are several Figma widgets that accomplish the same thing, but in my opinion Simple Vote works best because it shows the number of votes (naturally) and the avatars of those who voted (which produces a nice community/collaborative vibe). Anonymous voting, secret (i.e., reveal later) voting, and multiple voting can be enabled in the settings.

To use the widget (or any Figma widget), navigate to Resources (shift + I) > Widgets from the horizontal toolbar, search for the widget you’d like to use (Simple Vote in this case), and then click on the Run button.

Next, specify the settings you’d like to use, then click on the Start button to have the widget placed onto the canvas. If you have any objects selected prior to running the widget, then each selection will get its own widget — there’s no need to run the widget for every voting option.

**DateStamp**

Every list has one “Huh? Why that?” item — DateStamp is that item. I’m not sure what the intended use case is, but I find it useful for letting others know when a design style or component was last updated. Although you can find out this information using design handoff tools, design system tools, or even Figma’s own versioning features, this little stamp places the information right there on the canvas.

**lil notes**

lil notes does exactly what you think it does: it enables users to write notes on the canvas. As you can imagine, there are quite a few Figma widgets that do the same thing, but this one in particular is very minimal and thus more pleasant to use.

You’d probably want to use it to write low-key documentation, personal notes that you’re happy for others to see (sort of like thinking aloud), and anything else that you don’t want to appear in the document’s “official” comment log.