Building desktop applications with Node.js

Option 1: Electron

This is the most popular and well-supported option. Electron is a combination of Node.js and Chromium Embedded Framework, and so it will give you access to the feature sets of both. The main tradeoff is that it doesn't give you much direct control over the window or the system integration.

Benefits

- Cross-platform
- Well-supported, with a large developer base and a lot of (third-party) documentation
- Works pretty much out of the box, and lets you use HTML and CSS
- Can use native Node.js modules

Drawbacks

- Relatively high baseline memory usage; expect 50-100MB of RAM before running any application code. This is fine for most applications, but probably not for tiny utilities.
- Somewhat restrictive; does not give you much control over the system integration, instead has a default setup that's okay for most purposes and abstracts away platform-specific things for the most part.
- Limited OpenGL support; only WebGL is available.

Option 2: SDL

Using https://www.npmjs.com/package/@kmamal/sdl and

https://www.npmjs.com/package/@kmamal/gl, you can use SDL and OpenGL directly from Node.js. This will take care of window creation, input handling, and so on - but you will have to do all the drawing yourself using shaders.

A full (low-level) example is available here, and you can also use regl to simplify things a bit.

For text rendering, you may wish to use Pango or Harfbuzz, which can both be used through the node-gtk library (which, despite the name, is a generic GObject Introspection library rather than anything specific to the GTK UI toolkit).

Benefits

- Direct OpenGL access
- Does not enforce any particular structure on your project
- Good selection of examples

Drawbacks

- You have to do all of the drawing yourself; there are no widgets, there is no CSS, and so on. You will be writing OpenGL shaders. There is support for canvas-style drawing, but it is not fast.
- More research required to understand how to use it; not a lot of people use these libraries, and there are not very many tutorials.

Option 3: FFI bindings

You can also use an existing UI library that's written in C, C++ or Rust, by using a generic FFI library that lets you call the necessary functions from Javascript code in Node.js directly.

For C, a good option is Koffi, which has excellent documentation. For Rust, a good option is Neon, whose documentation is not quite as extensive as that of Koffi, but still pretty okay.

Option 4: GTK

The aforementioned node-gtk library can also be used to use GTK directly. Very little documentation is available about this, so you'll likely be stuck reading the GTK documentation (for its C API) and mentally translating to what the equivalent in the bindings would be.

Revision #2
Created 2025-02-28 00:28:54 UTC by joepie91
Updated 2025-02-28 00:44:08 UTC by joepie91