Hybrid Desktop Apps

How and when to build desktop applications in HTML5

anirudh@razorflow.com

Hello there!

I'm **Anirudh**, and I've co-founded **RazorFlow** and have been working on it for 3 years.

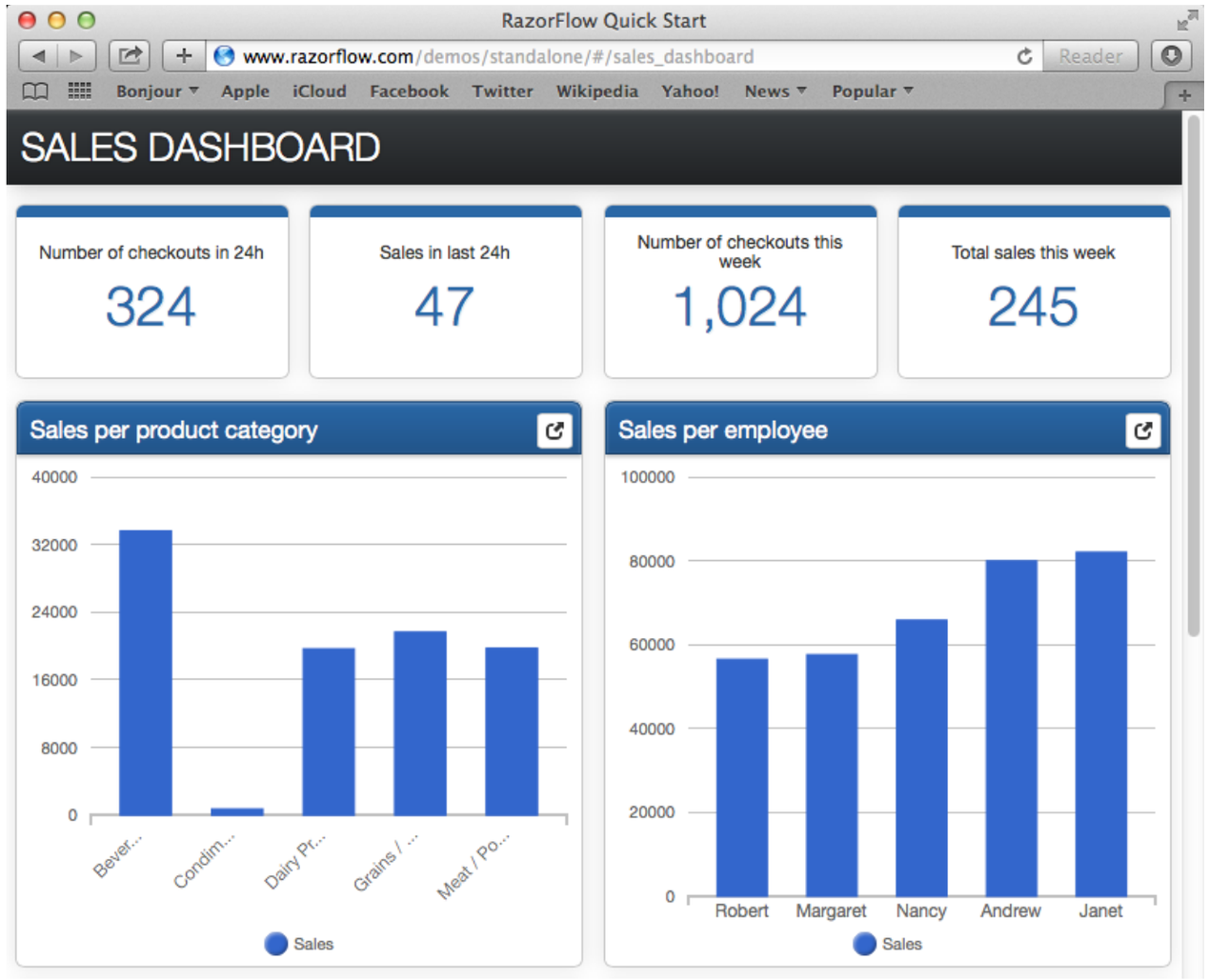
I've got considerable architecture and development experience for web frontend and backend tech.

You can email me at anirudh@razorflow.com

Why am I talking about this?

- Our product had a problem, which could be solved by a hybrid desktop app.
- We evaluated several solutions, and found some bumps along the way.
- Sharing our learnings with you.

What we do

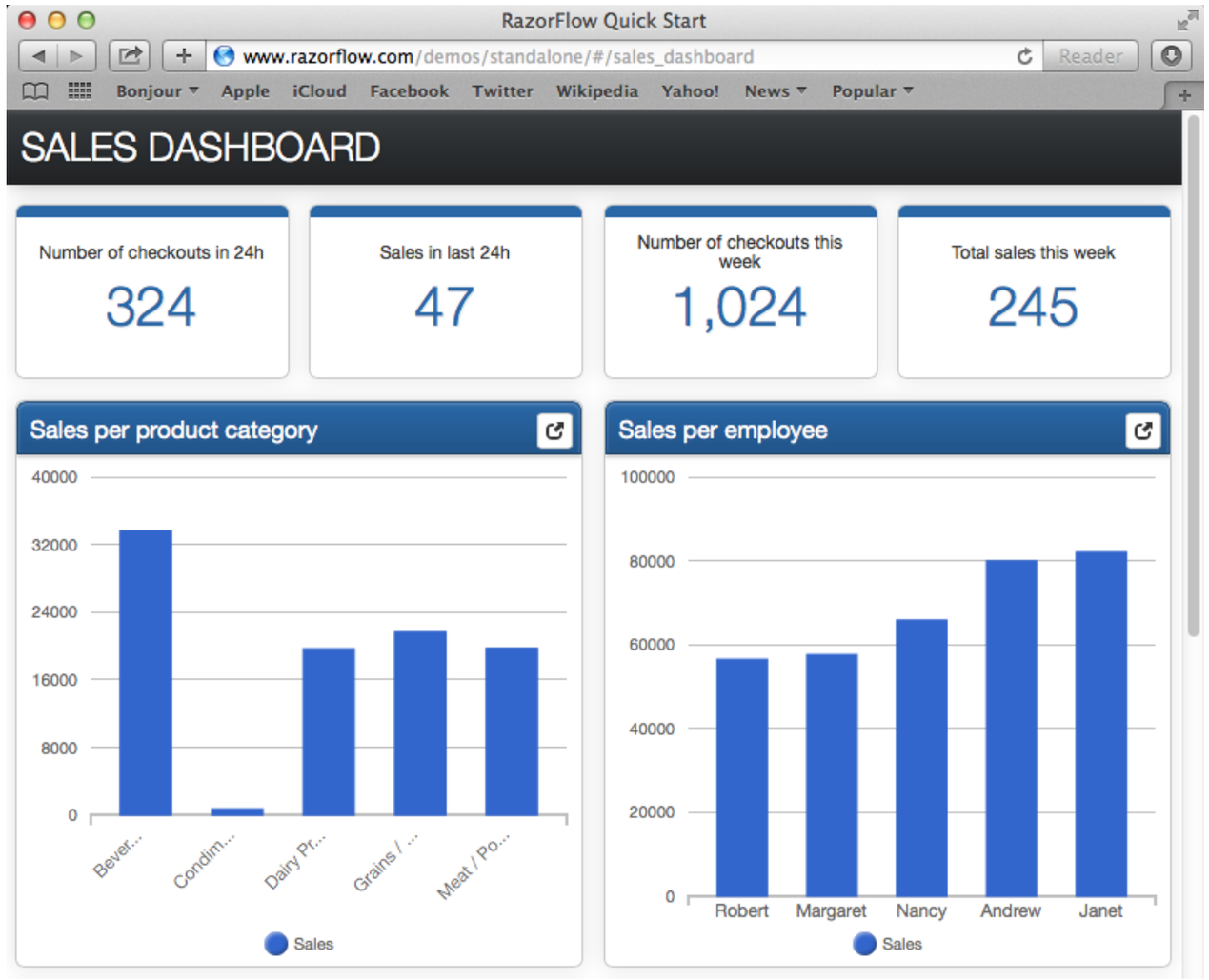


RazorFlow Dashboard Framework

Build HTML5 Dashboards that work across modern web browsers without the headache.

anirudh@razorflow.com

What we do



Basically:

Show a bunch of charts and tables and figures and stuff.

We only support modern browsers, so we can do:

Offline access

Notifications and alerts

Automatic updates

Realtime streaming

Other stuff we haven't thought of

anirudh@razorflow.com

HTML5 is Awesome!

Offline access

Works!

Notifications and alerts

Works!

Automatic updates

Works!

Realtime streaming

Works!

Other stuff we haven't thought of

Probably works

Our product mostly caters to developers, but they cater to their users.

But strange as it may seem, some users simply don't use modern browsers.

People are fine downloading a new app, but never a new browser…

our solution?

Disguise a browser as a desktop app.

essentially:

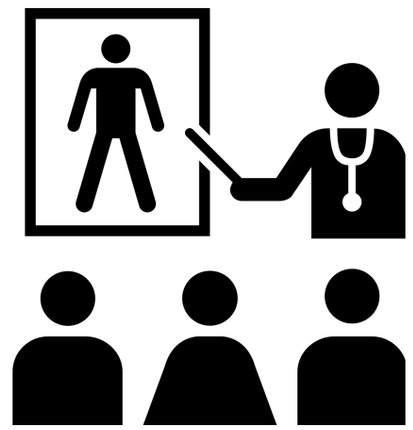
Users who want advanced functionality will be requested to download a special app which will run on their desktop.

Has it worked?

We don’t know. We haven’t launched this yet. But initial responses are good.

Enough about me, let's talk about you!

why would **you** want to consider building a hybrid app?



When to build a **Desktop app** rather than Web.

Extreme I/O

If you need low-latency high-bandwidth I/O, the hard disk is often better than the network.

Offline

If your user's work is primarily offline and works with files on their desktop, your app could run there.

Others

Use good judgement. Some applications are simply better running on the web and some aren't.

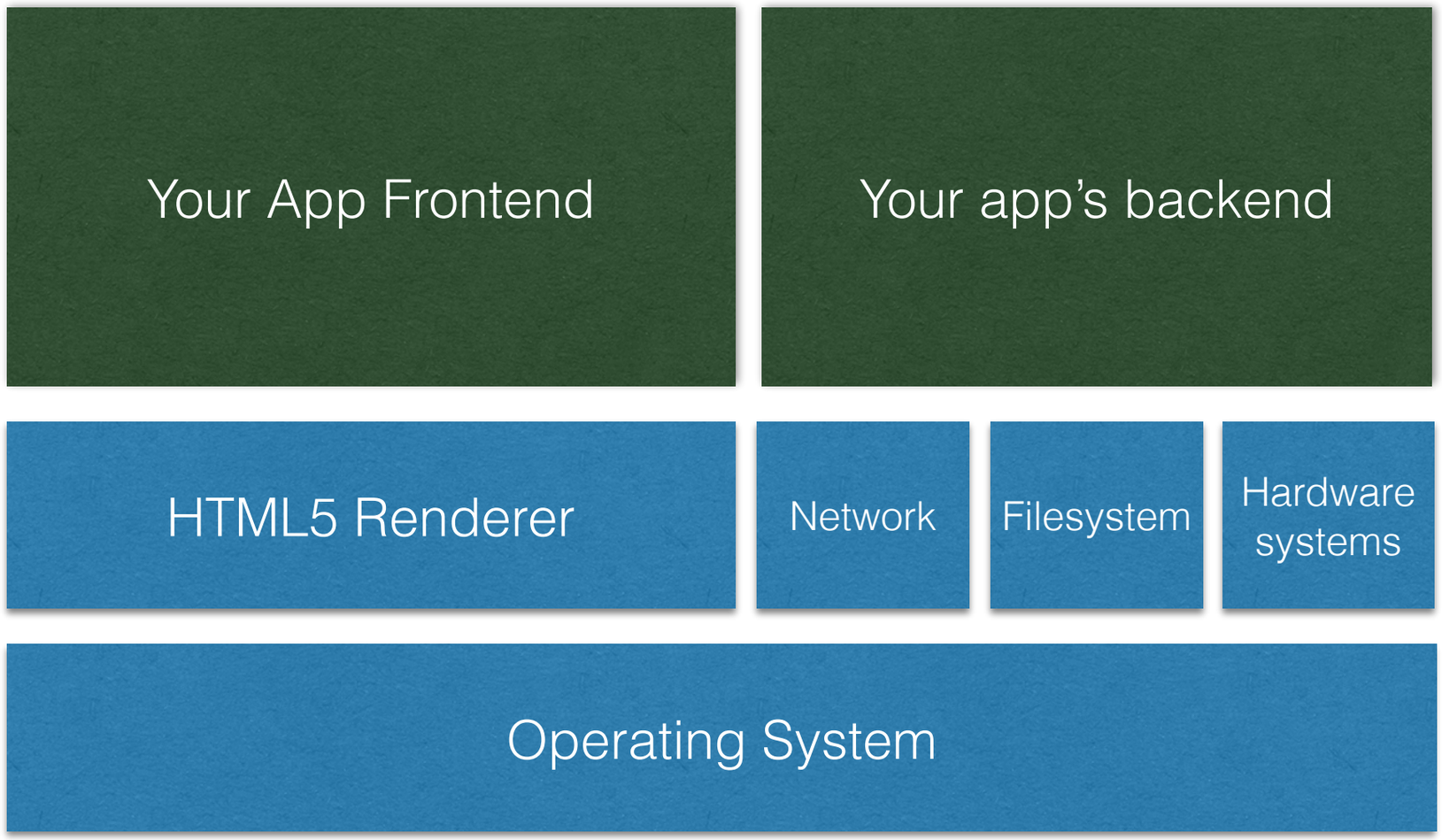
Beginner Friendly

Users don't understand what's Google Chrome? Give them a desktop app instead.

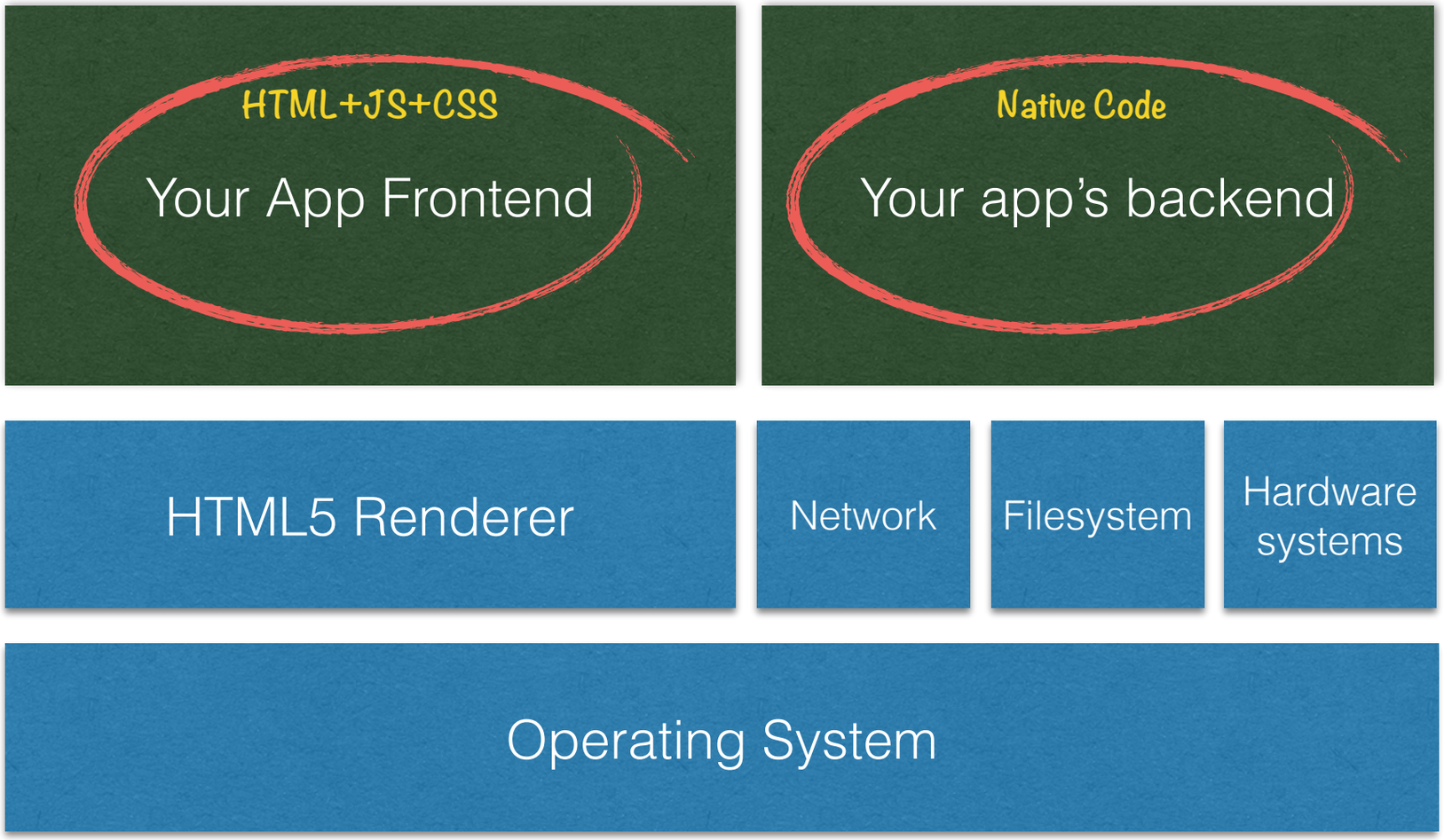
What is a hybrid desktop app?

It's an application which runs natively on your desktop but the UI is built using Web technologies like HTML5, JavaScript and CSS.

What is a hybrid desktop app?



What is a hybrid desktop app?



Benefits of **HTML** v/s Desktop toolkits

Re-use Skills

If you are already a web programmer you don't have much to learn and can start quicker.

Fluid Layouts

HTML is much better at handling text and image heavy fluid layouts, text wrapping and styling.

UI Libraries

Re-use open source and commercial UI libraries like ExtJS, Bootstrap, Kendo, YUI, etc.

Theme-able

CSS gives you much more control over looks, and you can quickly update the global theme of the app.

Graphics

SVG enables you build interactive graphical displays and Canvas provides fast 2D drawing.

MVVM

Model-View-ViewModel libraries like Backbone, Ember decouple your data and logic.

Benefits of **Desktop toolkits** v/s HTML

Consistency

Your app will feel familiar to new users because it looks and feels similar to other applications.

Better 3D/Gaming

DirectX/OpenGL, etc are faster than WebGL and plenty of libraries exist for large scale development.

Static layouts

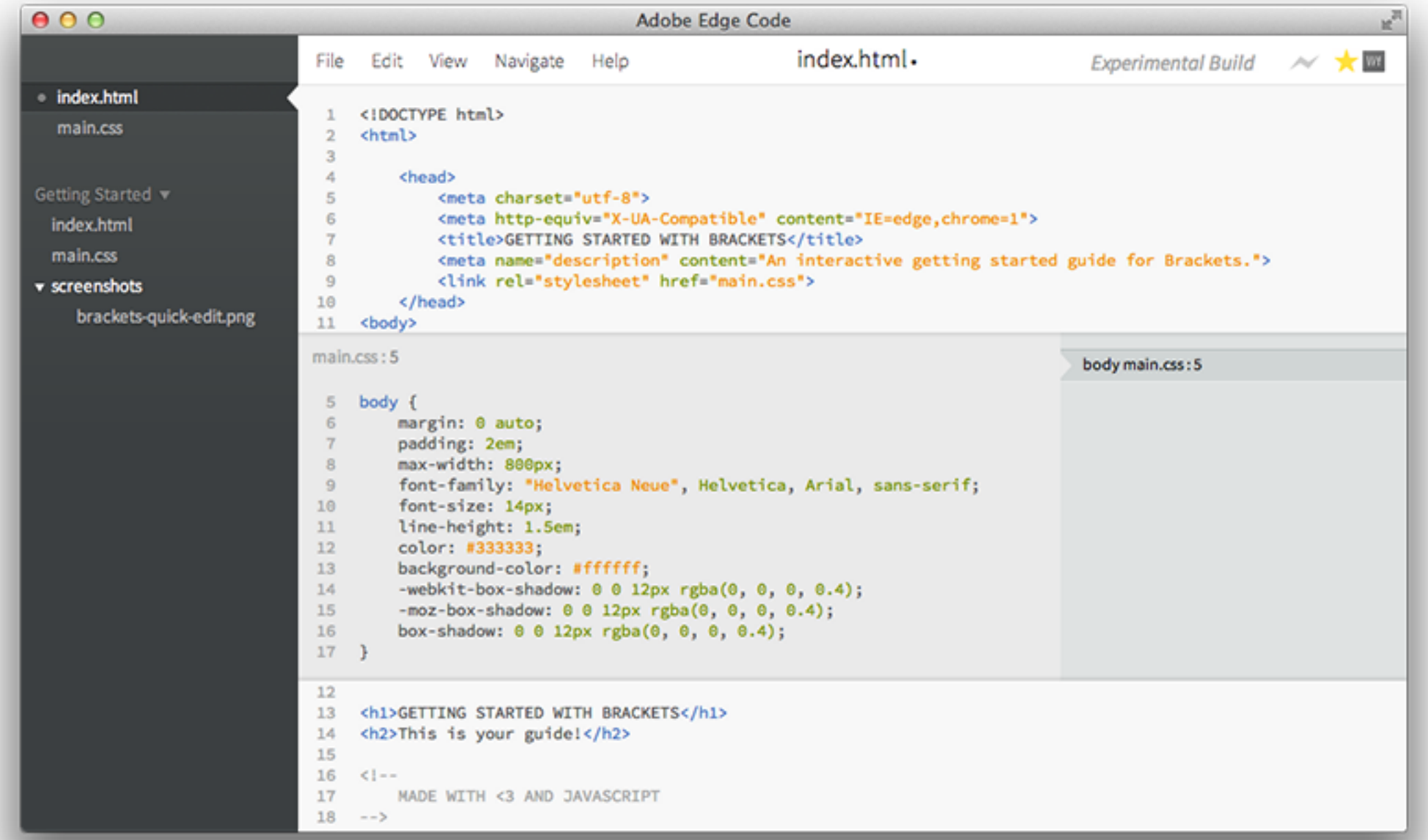
Desktop toolkits are much better at static layouts where buttons and controls are clearly laid out

Desktop apps with HTML UI

A few examples.

Adobe Brackets

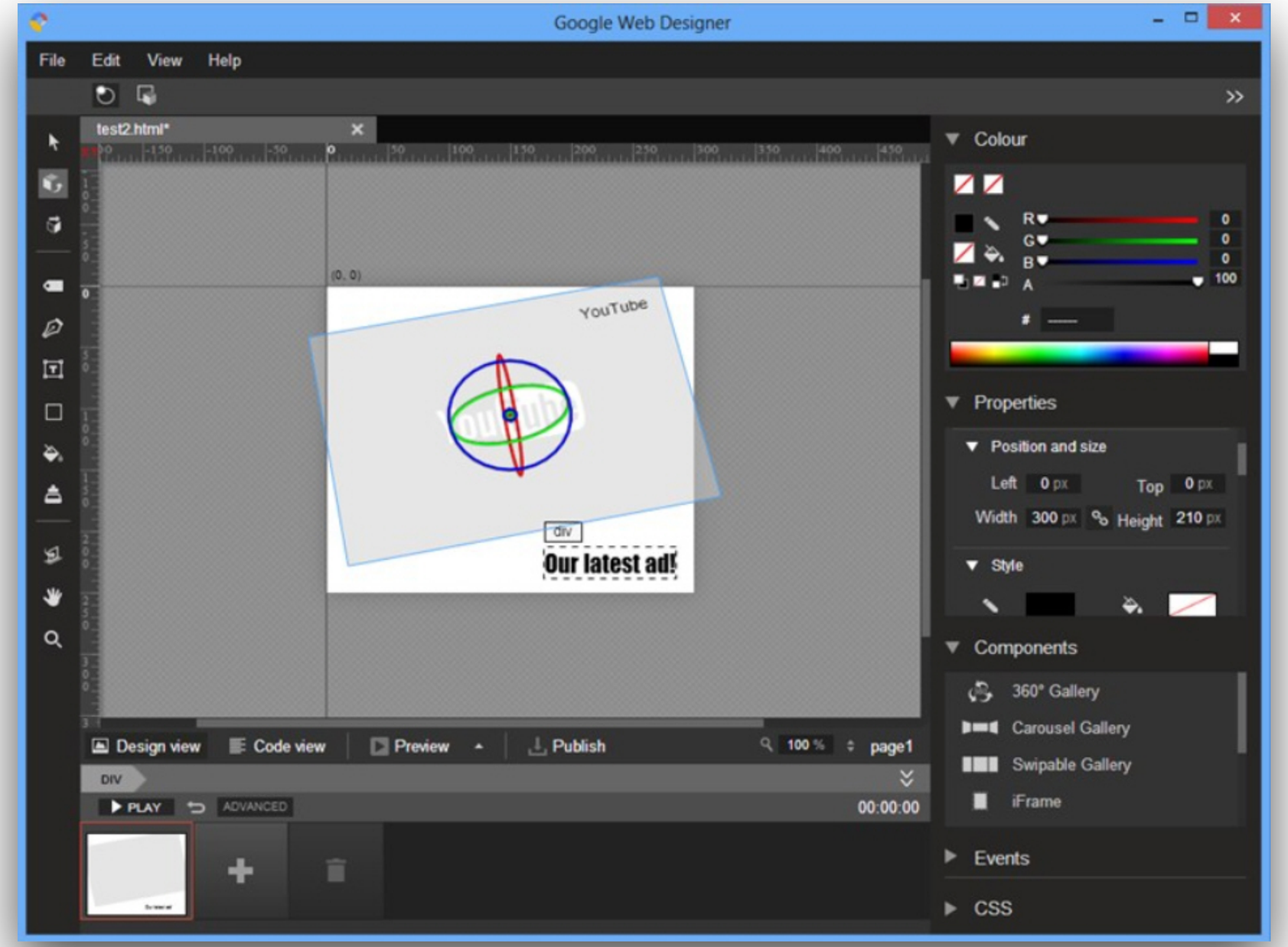
A text editor for HTML/CSS/JS. Also called *Adobe Edge Code*.



anirudh@razorflow.com

Google Web Designer

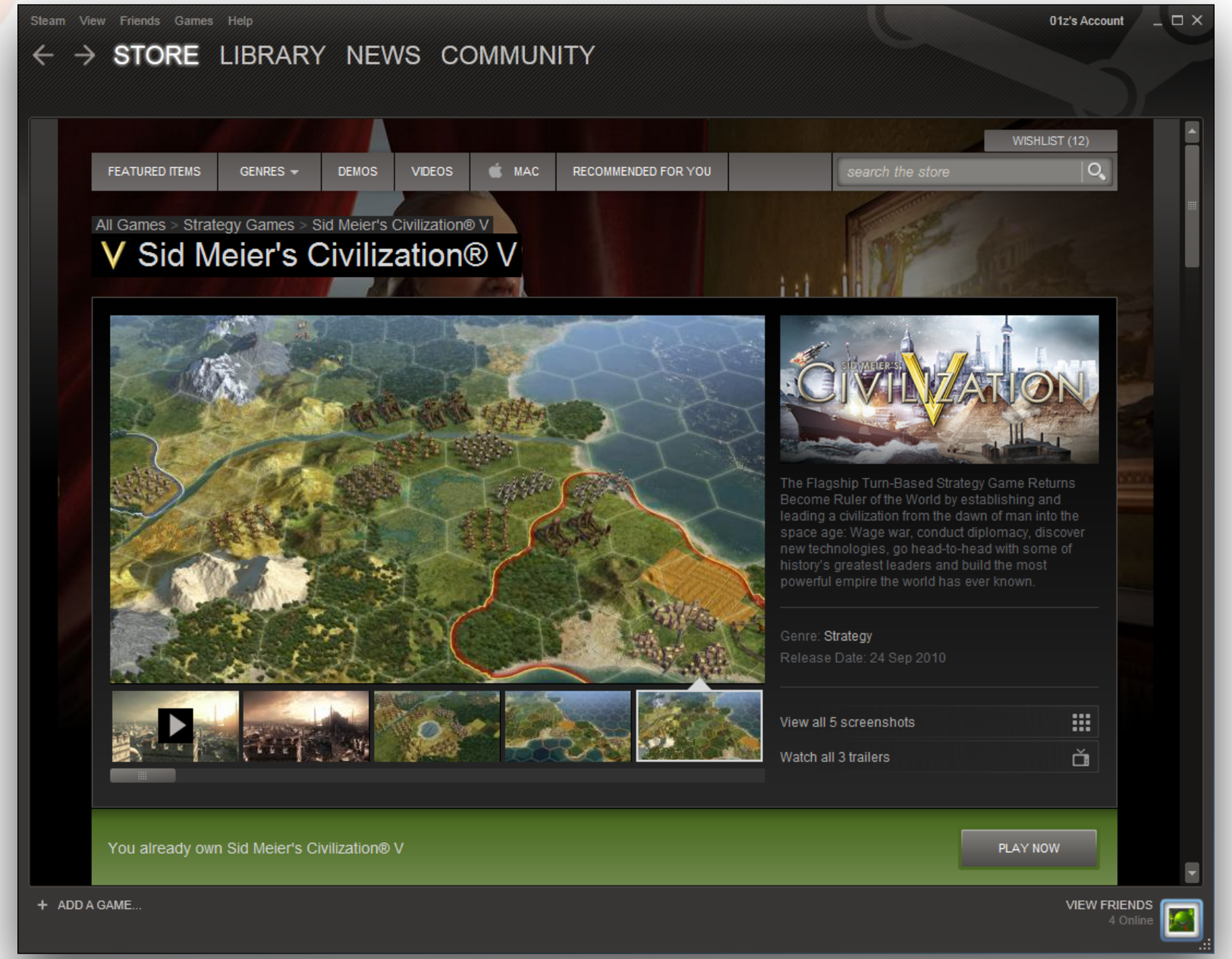
A tool to make HTML/CSS3 Animations.



anirudh@razorflow.com

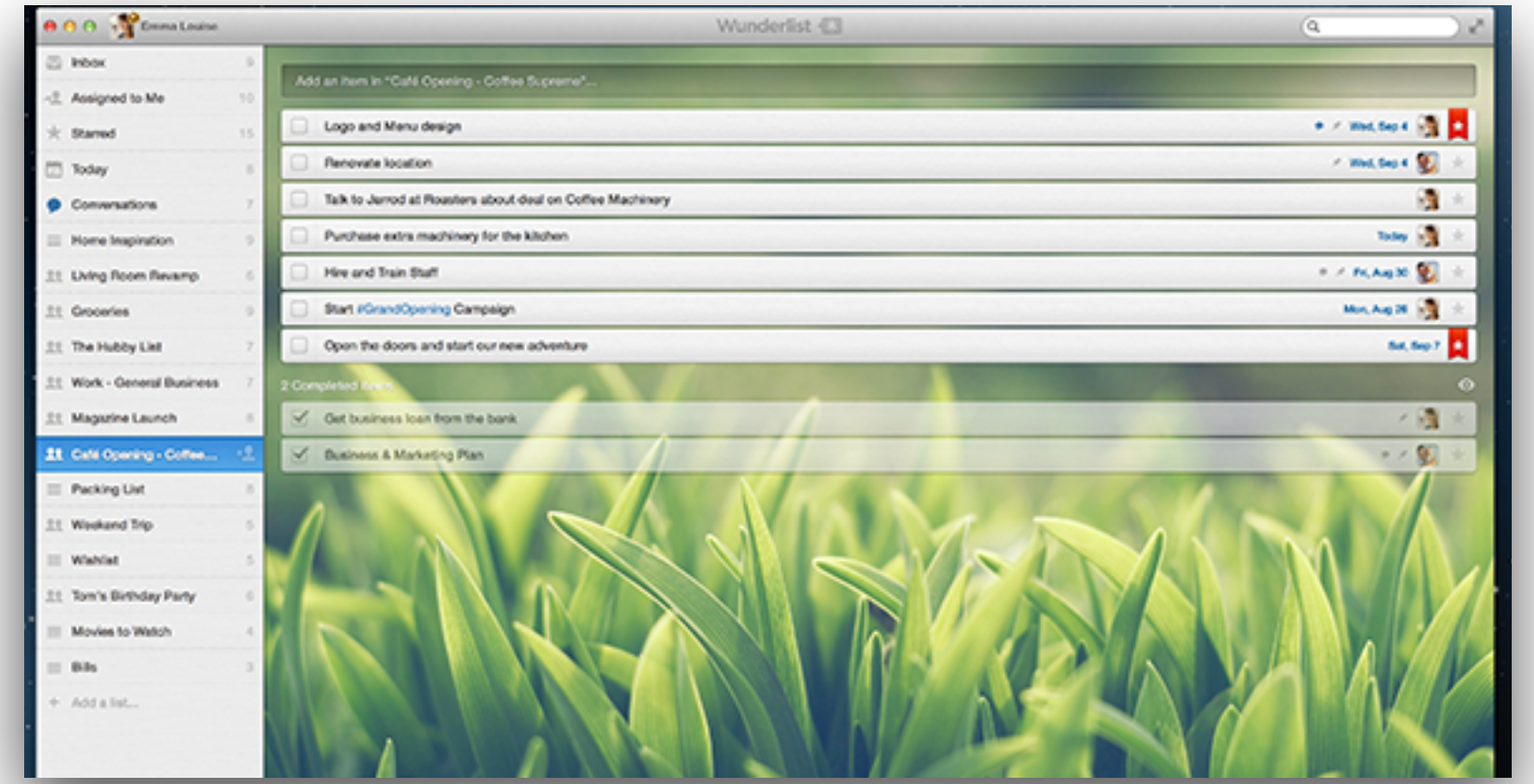
Steam

Valve's game purchase/delivery platform. Win/Mac/Linux.



Wunderlist

Shared TODO list manager.



anirudh@razorflow.com

So you want to build an app

What are your options?

- node-webkit
- Chromium Embedded Framework
- Adobe AIR
- Tide SDK
- CEF-Python
- Bellite
- QWebkit
- GtkWebkit
- and some others

Introducing **Braces**

To evaluate all options that we had, we decided to build one app in all platforms. This was called **Braces**.

Braces is a text editor which can let you (*gasp*) open, edit and save text files.

Braces covered all the basics:

1. Adding JS/CSS assets like codemirror to your app.
2. Interacting directly with the filesystem.

What factors do we consider?

1. Compatibility

- What platforms does it support?
- How much native functionality can we access (hardware, C libraries, kernel, etc)

What factors do we consider?

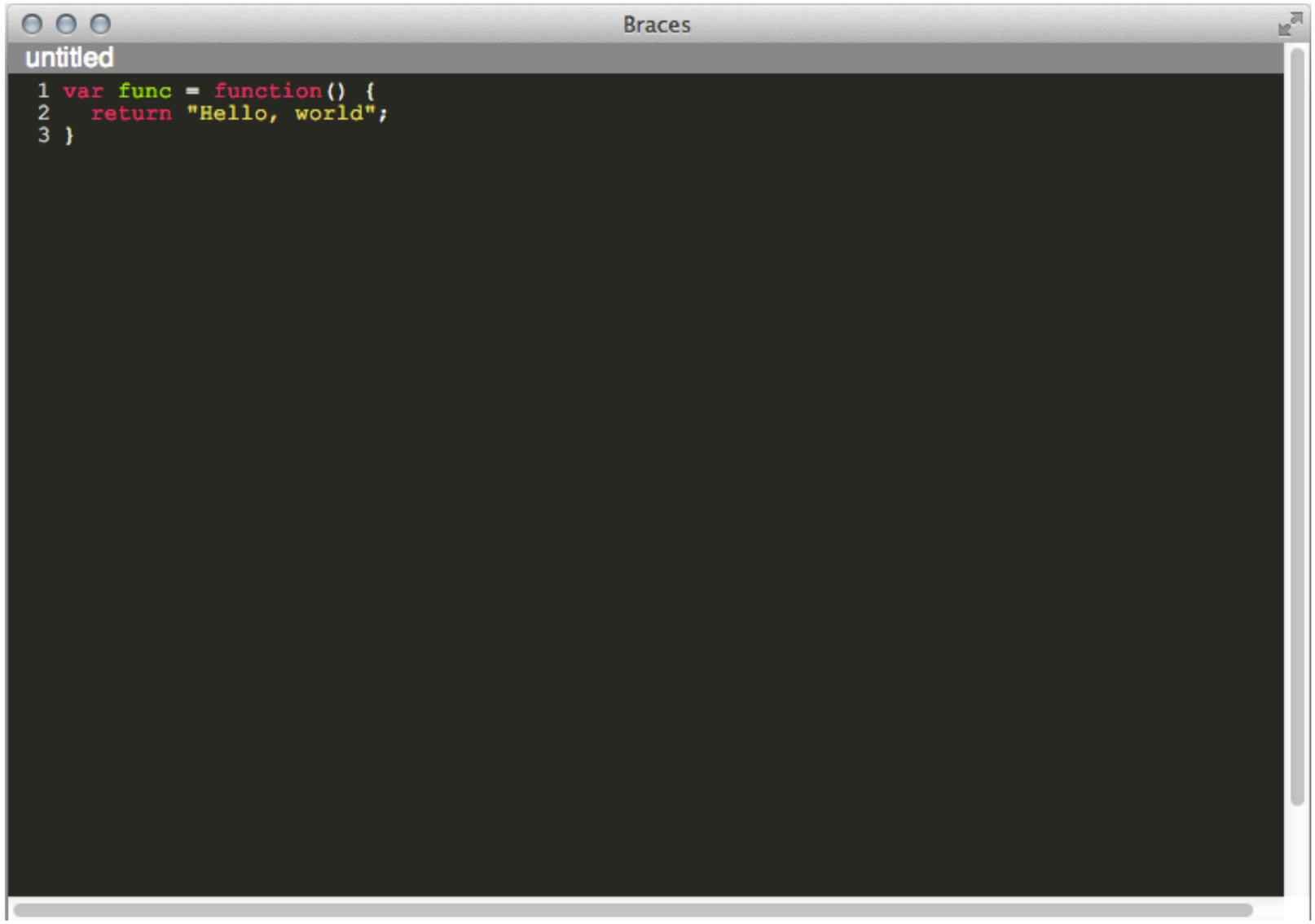
2. Productivity

- Can you update assets (JS/CSS) without compiling, etc?
- How easy is it to make verifiable changes?
- How well does it integrate with build systems and pre-processors like CoffeeScript, Less, etc.
- How fast can we get started?

What factors do we consider?

3. Deployment

- How much effort to package an application for deploying?
- Is it compatible with app stores?
- Will your users have to perform any steps apart from a download/installation?
- How easy is it to update your application?



Node-Webkit

The best of node, and the best of webkit.

Loading a file in node-webkit

```
var fileContents = fs.readFileSync(filePath, ‘utf-8’);
$(“#textContainer”).text(fileContents);
```

=

mind = blown

Node-Webkit **PROs**

- Ultra easy to get started.
- Straight-forward to package and deploy.
- Imports assets easily

Node-Webkit **CONs**

- You're constrained to node. If you need something that's not in node (hardware access, for instance), you can't do it.

- It's a little tricky importing other node modules, especially those that rely on native extensions.

- Cannot submit to mac app store.

node-webkit scorecard

✓

OPEN SOURCE

✓

WELL MAINTAINED

JavaScript

BACKEND CODE

Easy

INITIAL SETUP

Moderate

PACKAGING DISTRIBUTION

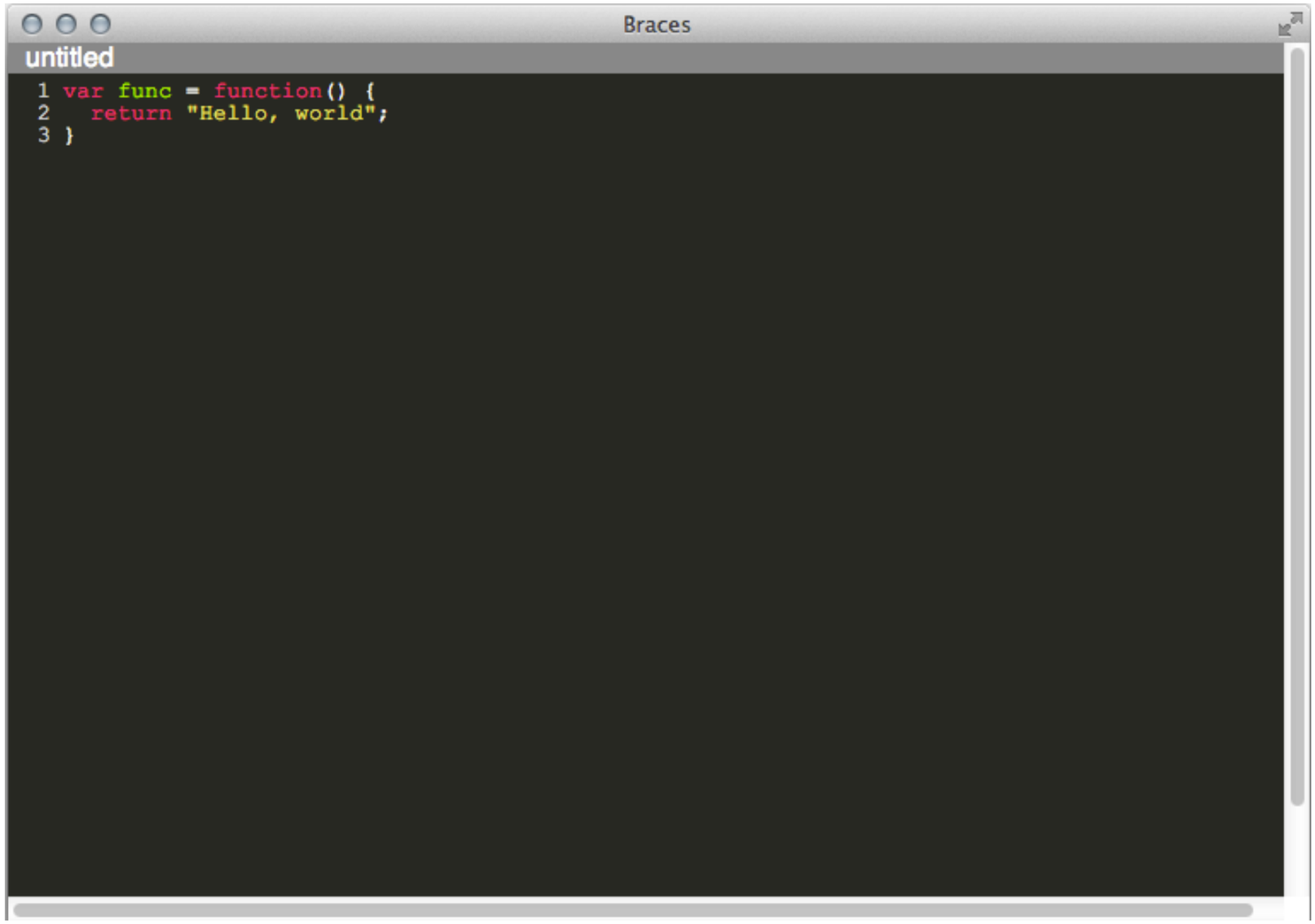
M/W/L

Mac+Windows+Linux

PLATFORM SUPPORT

✓

SHARED BACKEND CODE

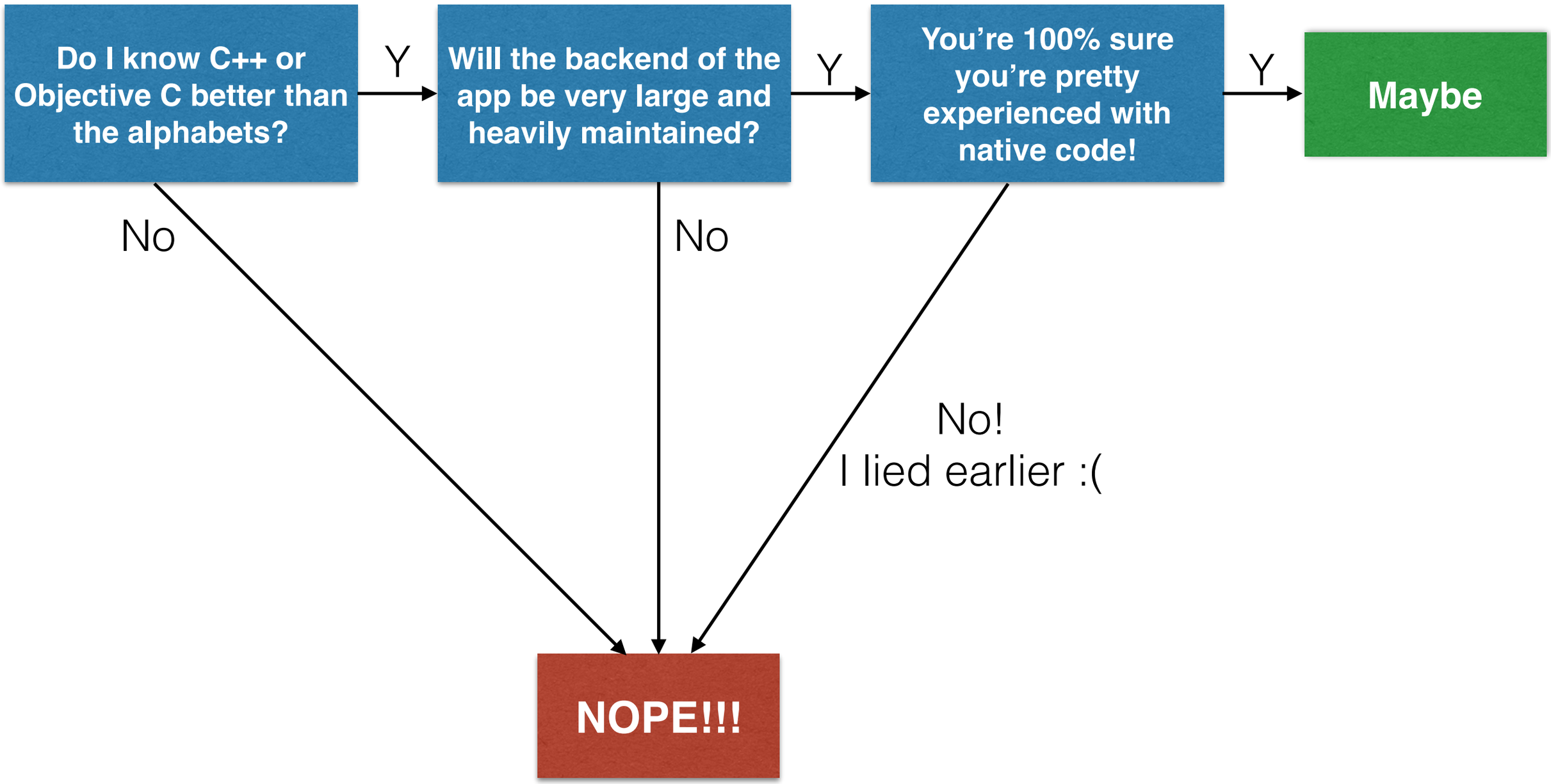


Chromium Embedded Framework

Industrial-strength hybrid app development.

Should I use CEF?

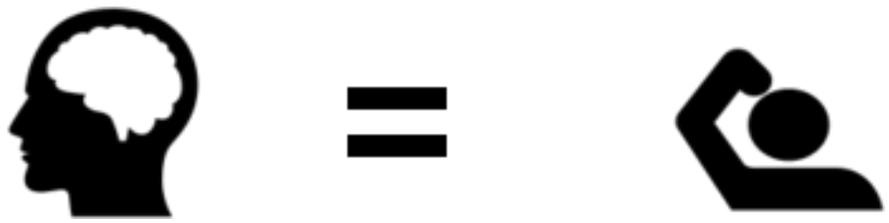
A handy flowchart



Loading a file in Chromium embedded framework into a HTML textarea

```
// It technically should be possible. But we couldn't
// figure it out.

// TODO: figure it out.
```



Chromium Embedded Framework **PROs**

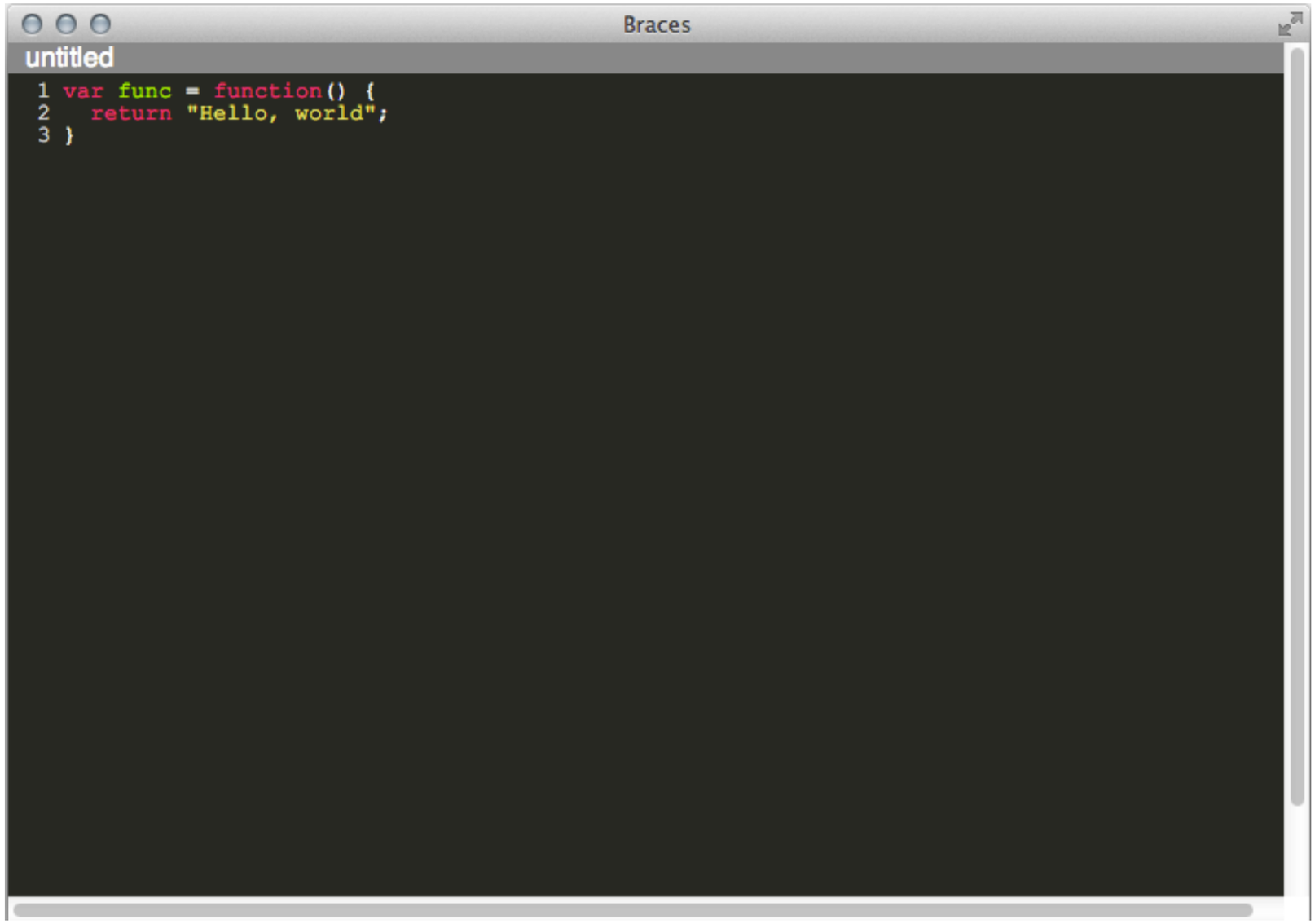
- All the big shots seem to use it.
- Gives you extreme flexibility and control.
- Have near-complete access to native features and libraries.

Chromium embedded framework **CONs**

- Documentation not so great. 
- Need very strong native app background.
- No mac app store support (according to some sources but we couldn’t find a definitive yes/no answer)

Chromium Embedded Framework (CEF)

✓ OPEN SOURCE	✓ WELL MAINTAINED	**C++/Objective C** BACKEND CODE	
HARD INITIAL SETUP	**Moderate** PACKAGING DISTRIBUTION	**M/W/L** Mac+Windows+Linux PLATFORM SUPPORT	× SHARED BACKEND CODE



Tide SDK

This used to be titanium desktop

anirudh@razorflow.com

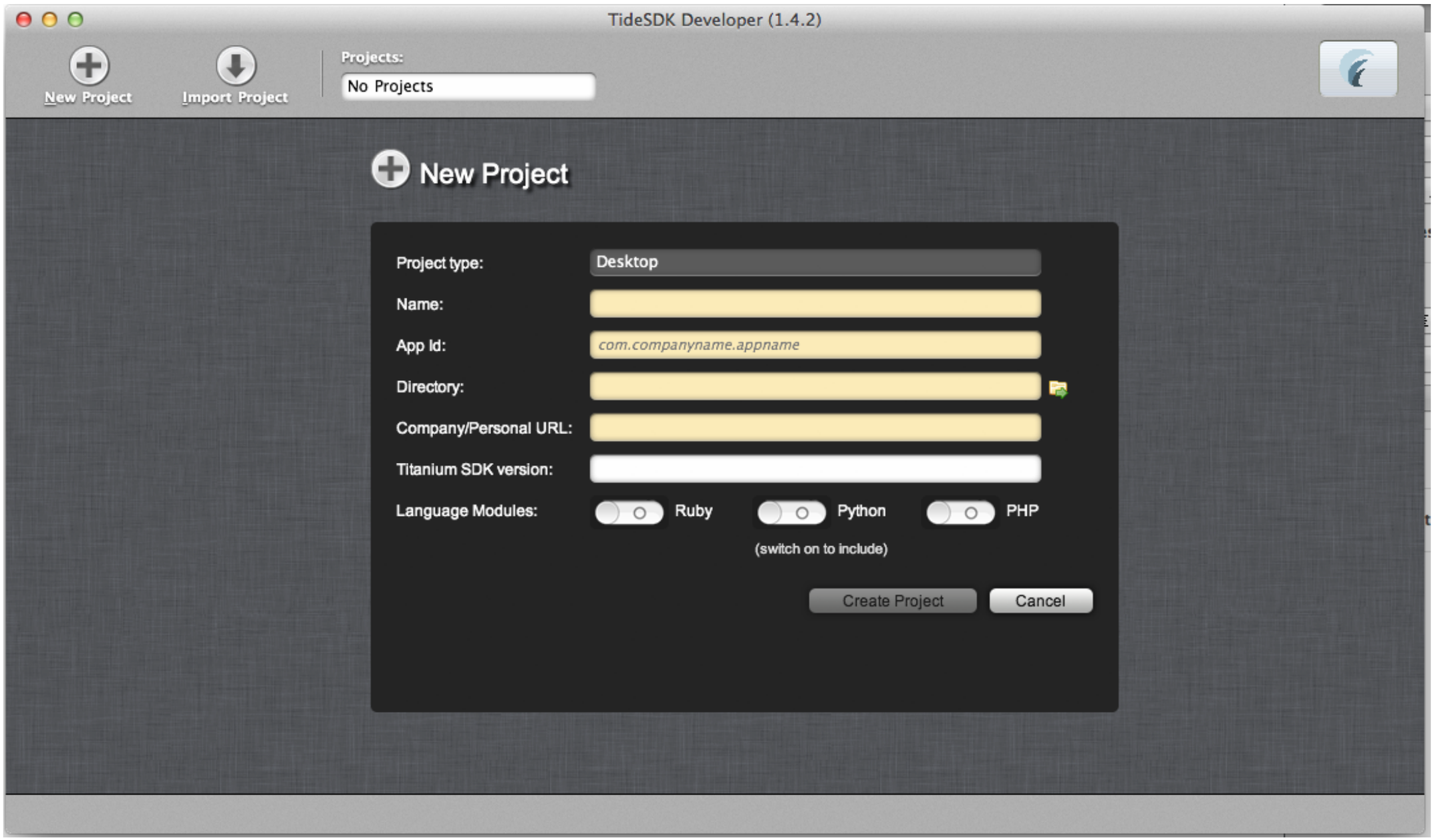
Tide SDK

Claim to fame

- Supports PHP, Python and Ruby on backend.
- Easy to get started with a good app manager tool.
- Used to be Titanium SDK Desktop before, now a community project.

Tide SDK

Claim to fame



Loading a file in Tide SDK

```
var fs = Ti.Filesystem;
var file = fs.getFile(filepath);
var fileContents = file.read().toString();

$(“#textContainer”).text(fileContents);
```

mind

=

reading API docs

Tide SDK **PROs**

- Easy to get started
- Straight-forward to package and deploy.
- Imports assets easily
- Can be submitted to mac app store
- Commercial support can be purchased

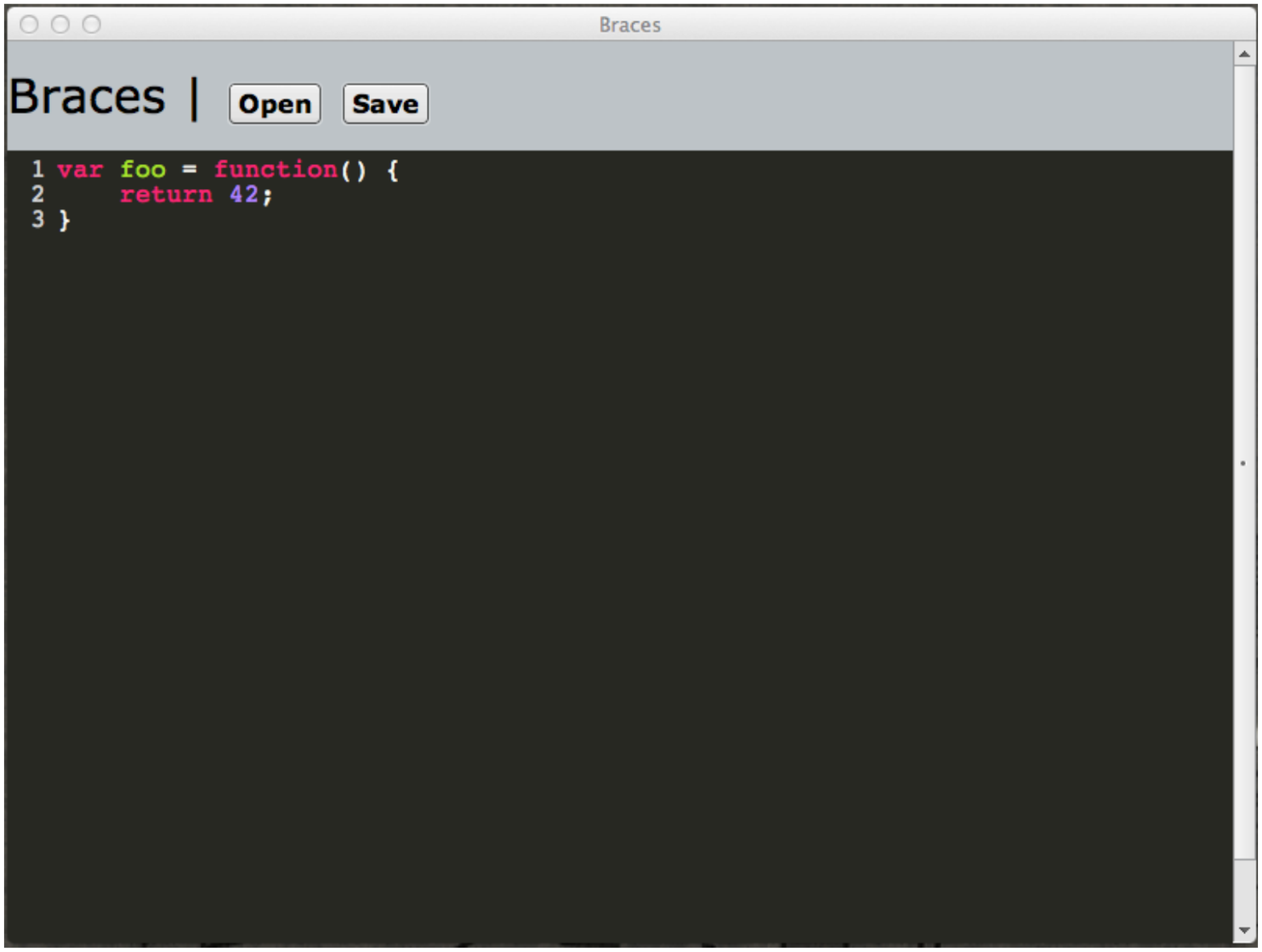
TideSDK **CONs**

- Limited by Tide's API
- No native app integration.

Tide SDK

✓ OPEN SOURCE	✓ WELL MAINTAINED	**PHP/Python/Java or more** BACKEND CODE	
Easy INITIAL SETUP	**Very Easy** PACKAGING DISTRIBUTION	**M/W/L** Mac+Windows+Linux PLATFORM SUPPORT	✓ SHARED BACKEND CODE

anirudh@razorflow.com



Adobe AIR

The grandfather of modern hybrid apps

Adobe AIR

Claim to fame

- This was amongst the first ones available (2008).
- Includes flash player
- Richer API
- Supports mobile as well.

Loading a file in Adobe AIR

```
var myFileStream = new air.FileStream();
myFileStream.addEventListener(air.Event.COMPLETE, completeHandler);
myFileStream.openAsync(myFile, air.FileMode.READ);

function completeHandler(event) {
  code_str = myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
  braces.setValue(code_str);
}
```

Short Version: It's pretty straightforward and uses conventional JavaScript techniques

Adobe AIR **PROs**

- More platforms supported than others.

Adobe AIR **CONs**

- Your users need to install the runtime first, and then your package.

- We couldn't find good debugging and profiling options.

Adobe AIR

COMMERCIAL SOFTWARE

$250+

✓

WELL MAINTAINED

JavaScript

BACKEND CODE

Easy

INITIAL SETUP

Cumbersome for users

PACKAGING DISTRIBUTION

M/W/L

Mac+Windows+Linux

PLATFORM SUPPORT

✓

SHARED BACKEND CODE

A few more options

QT's QWebView

- Available as part of QT Toolkit, which is a cross-platform toolkit for building GUI applications.

- Take the liberty of combining the amazing QT library for native widgets, and using QWebView to show your HTML UI, and mix it up.

- Embed resources using QT's QResources module.

GTK GtkWebkit

- A Webkit widget is available as part of the GTK UI Toolkit.

- GTK is cross-platform but works best on Linux.

- Use many different languages on your backend like C, C++, Vala, Python, C# thanks to GObject Introspection.

- Distribution of these apps are a little complex because of autotools and resource embedding.

CEF-Python

- Built on top of Chromium Embedded Framework.
- While CEF makes you question if you even know anything about computers, CEF-python is slightly more approachable.
- Slightly complex to install and get your first app running.
- Mix it up with other UI toolkits like wxWidgets, Gtk, etc.
- Easier to handle browser events in python.

One size doesn't fit all.

Requirements vary and one solution won't satisfy all of them however, we've isolated 3 major use cases.

What do you want to build today?

Heavyweight	**Desktop Edition™**	**Quick n' Effective**
Your app does intense I/O, uses several core libraries, modifies webkit itself, and the UI needs to be fast.	You've got existing code and want to re-use it to make a desktop alternative of your web app.	Your build fast and iterate fast. You don't need kernel and other core APIs. Just to execute some code.

Winner of the **Heavyweight** category

Chromium Embedded Framework

Though it's very hard to get started with, and you need to heavily modify it to work on different platforms - CEF gives you the closest access to the core OS without the use of any wrappers or sandboxes.

With a rock solid Chromium heritage and a ridiculously good codebase, this is what most large projects choose.

But with virtually zero tutorials and documentation this isn't recommended unless you are very experienced with Native languages like C++/Obj-C.

Winner of the **Desktop Edition™** category

TideSDK

Tide SDK makes it very quick to start out, and an easy tool to package and distribute applications.

Moreover, with a little careful planning, you can share large parts of your codebase between a web app and a desktop version of the application.

If you're building from scratch, you can even have PHP, Python and Ruby backends. Tide SDK also comes with paid commercial support which is useful.

Winner of the **Quick n' Effective** category

node-webkit is beautiful, blissfully simple and elegant. You can build an entire app with desktop functionality in a single file.

Access Node.JS functions and modules directly from the same code that modifies your DOM. It rarely gets any simpler than this.

However you're limited by what node.js can do, and can't break out of it's sandbox without significant effort. Even if you never build desktop apps, it's worth learning node-webkit as a valuable tool.

What did **we** finally choose for the hybrid app technology?

What did **we** finally choose for the hybrid app technology?

Why did we choose QT?

- Stable quality and good commercial support.
- APIs to interact with the system tray, minimize to tray, etc.
- Some of the UI around the browser is native buttons/menus.
- On windows, installs can be scripted using installation builders.

Problems encountered with QT

(so far)

- The webkit-to-native bridge isn't very strong, and native objects don't translate very easily to JS Objects.
- QWebkit and it's functionality isn't very well documented.
- We could not get the JavaScript debugger working properly.
- The program has to be re-compiled every time there is a change in the JS/CSS code.
- Have to manually manage memory, coming from a JavaScript background that's a little hard to get used to.

Special thanks

Thanks to the RazorFlow team - Ameen, Narendra, Selwyn and Joe - for doing all the work of evaluating all the options, building sample apps and writing the notes.

See the code

You can see all the evaluation code written for different platforms at this URL:

https://github.com/RazorFlow/metarefresh_braces

You can also see our detailed notes with more details about each tool in this page:

https://github.com/RazorFlow/metarefresh_braces/wiki

Thank you.