PHIL LEGGETTER

HEAD OF DEVELOPER EVANGELISM AT PUSHER
& REALTIME WEB CONSULTANT

Choosing your Realtime Web App Tech Stack

09 Dec 2013

In the previous parts of this three part series I covered:

- Part 1: History, Background, Benefits & Use Cases of Realtime
- Part 2: Fundamentals of the Realtime Web & Realtime Web Functionality.
- The FOWA 2013 Choosing your Realtime Web App Tech Stack video

On to part three...

In this final part (*sorry for the delay*) I'm going to cover how you should approach choosing your Realtime Web App Tech Stack.

My name is **Phil Leggetter**. I'm He
Developer Evangelism at Pusher.

I frequently write articles and give
Realtime Web Technologies and g
technology. Get in touch if you'd l
talk at your event or write an artic
I'm also very interested in develop
experience and productivity, APIs
customer service.

I'm also the co-author of the APre
beginners title "Realtime Web App

**Realtime Web Apps: With H
WebSocket, PHP, and jQue**

Buy the book I co-write with Jaso via Amazon.com or Amazon

Socket.IO has done an amazing job when it comes to making people aware of realtime web technology. And it's a great solution.

But, it shouldn't be the default choice when choosing your realtime web solution.

You should instead take a number of factors into account. Then, if it provides the functionality required to meet your requirements, use it. But there are **lots** of other fantastic realtime web solutions available that may be a better fit or your application's needs.

I would certainly recommend using an existing realtime web tech solution. Unless you have a very unique requirement there's no point going through the pain of buliding something from the ground up. If you really want to focus on adding realtime features to your apps then there is more than likely an existing solution that will enable you to start adding those features quickly.

Focusing on features: do you really want to have to write code to deal with connection fallback/upgrade strategy? Surely it's better to use a solution that has that in place.

If you use an existing solution you get support either from the company you've licensed the software from or/and from any community of developers that also use that product or service.

All software has to be maintained. Do you want to deal with the maintanance of the a realtime web solution as well as your application stack? If you use an existing solution that may be handled for you.

For example, when WebSocket came out Pusher handled all the pain as the WebSocket specification changed during numerous revisions and they managed and supported a number of protocol versions. I know I would rather focus on adding features than dealing with the intricacies of the WebSocket spec.

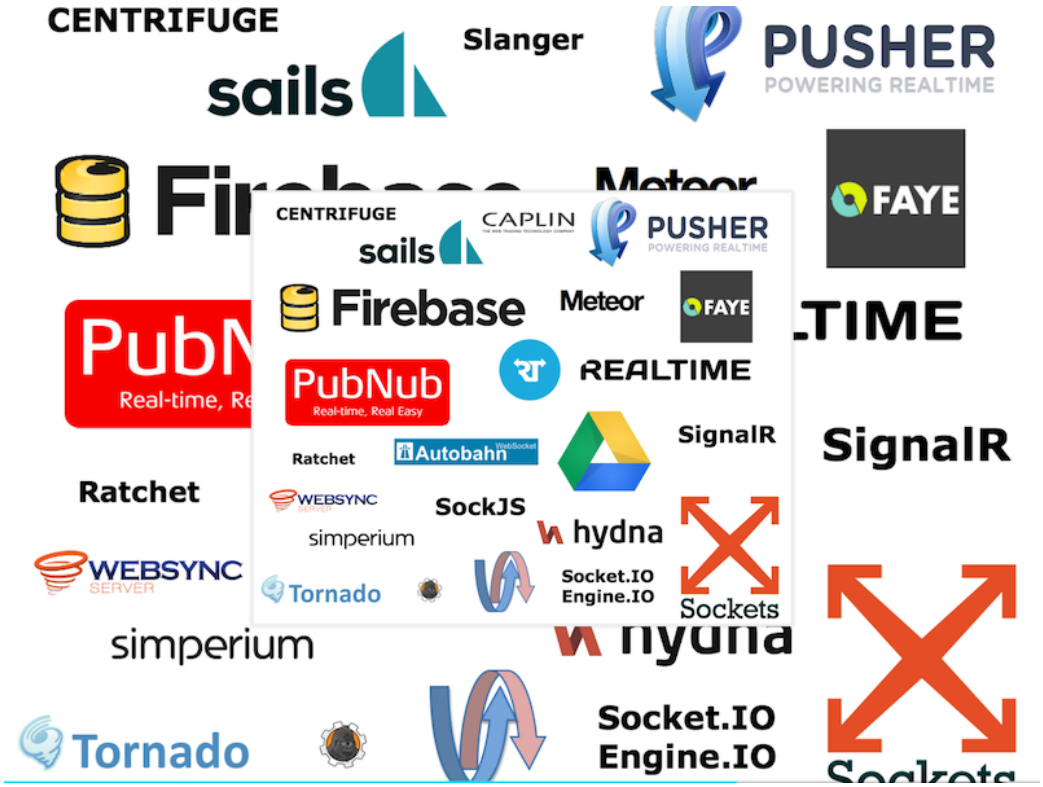
Realtime web tech is still evolving. For example. WebSocket provides support for sub-protocols. If one of these becomes popular it may be the case that the solution you use has that support added in with no effort required from you or your team. If a clear additional use for realtime web technologies raises it's head - such as acting as the middleman in WebRTC connection handshaking - then it may be that the solution you use decide to add this in.

If your application is going to be popular you're going to have to consider how you scale your solution. Some solutions such as Faye and SignalR have put a lot of thought into this. Documentation and features will be available to help you achieve that.

When I posted part two of this series to the LinkedIn Realtime Software & Tech Group the CTO of Migratory Data Systems, Mihai Rotaru, commented on my post:

> *I hope you agree that much more difficult to implement requirements, vital for any production deployment, should be considered when choosing a real-time web technology including:*
>
> - *Vertical scalability (and performance in general)*
> - *Horizontal scalability via fault-tolerant clustering*
> - *Guaranteed delivery of data in presence of failures (e.g. if the connection between a client and a messaging server is broken or if the messaging server serving the client goes down, the client will reconnect to another cluster member and will receive all messages published during the reconnection time)*
> - *Security of data including data entitlement*

Well put! Do you want to have to deal with all this? If not, use an existing solution (*and definitely consider using a hosted service*).

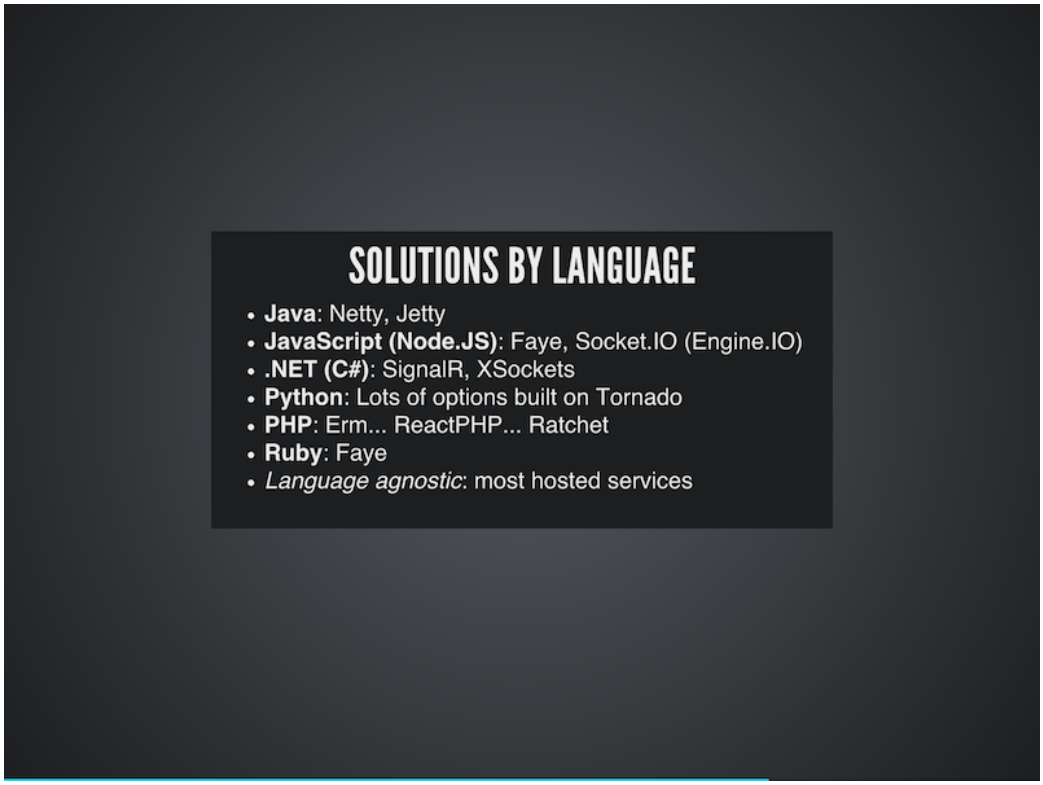


I did say there were lots of solutions - and the above images doesn't show them all!

So, how do you go about filtering that list of existing solutions down and choosing the right one for your app?

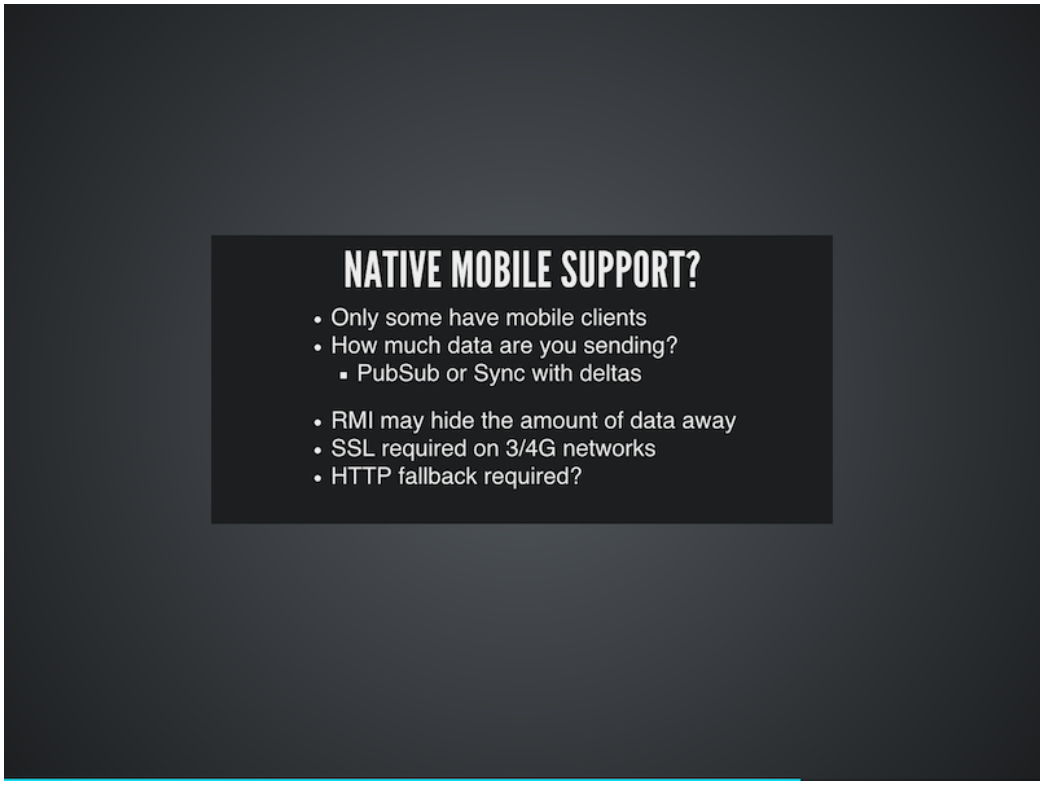
Here are some additional considerations that will help.

Sometimes you may need to dive into the code of the solution you've chosen. So it makes sense to choose one that is written in a language that you and your team are comfortable with. In some cases you can then even contribute to the solution with bug fixes and improvements.



The one language that there seems to be some limit with right now is PHP. For more information see my post on The current state of Realtime Web Tech for PHP.

As with most web technology the initial focus was desktop web clients - web browsers. Mobile is also very big in this space but only some solutions have good native mobile client support (Objective-C for iOS, Java for Android and .NET for Windows Phone).

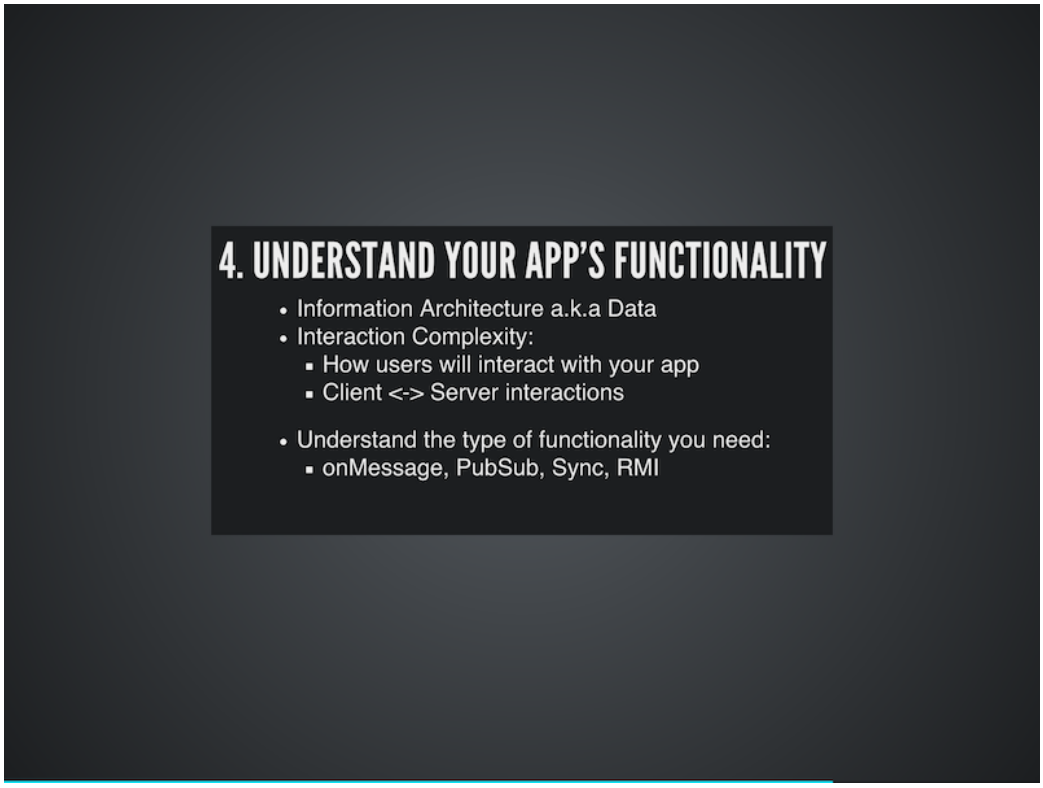


When building for mobile you also need to think about things such as the amount of data you're sending, how frequently data is being sent and the processing these things result in on the mobile client.

I think PubSub solutions make you think about messaging and Data Sync solutions that only send deltas (the changes to data) are the best solutions in this space. I feel that RMI may abstract things away too much and you might forget you're making calls 'over the wire'.

The solution is also going to have to provide SSL support as many mobile networks tamper with or drop persistent connections; HTTP Streaming and WebSockets. You may also need HTTP fallback on mobile.

I don't have a comprehensive list of solutions that have good mobile support, but I do know that self hosted solutions like Socket.IO, Faye and SignalR do have some mobile client libraries and hosted services like Firebase, PubNub and Pusher do too. I'll try to update the realtime web tech guide to highlight this information.



In order to make a decision on the right realtime web solution for your application you need a good understanding of the functionality that it offers.

You need to understand the structure of the data, how it's used in the application and how you expect it to be requested and transmitted between the client and the server. How complex is the data and how much data will your app have?

How complex are the interactions between your user and the application and how does that affect the interactions between client and server?

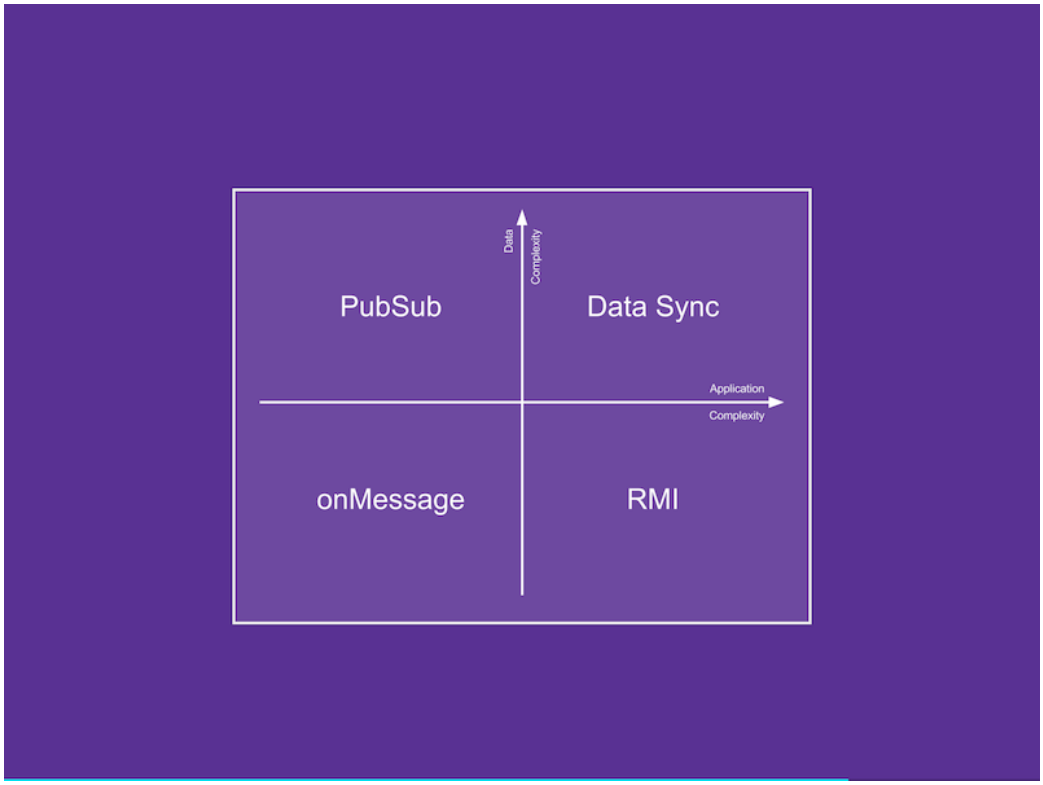
All this should feed into you being able to choose the type of functionality you need.

Is it a simple messaging solution where you just get a data blob in an onMessage callback?

Do you have data that you want to be able to partition and have users selectively subscribe (PubSub) to sections of data?

Are the interactions more like method invocations?

Or are you synchronising data structures between clients?



In the diagram above there's an X-Axis representing functional complexity and the Y-Axis represents data complexity.

If the data in your system is simple and the application functionality (around that data) is also relatively simple then it's possible that a simple messaging solution (onMessage) may fit the application's needs.

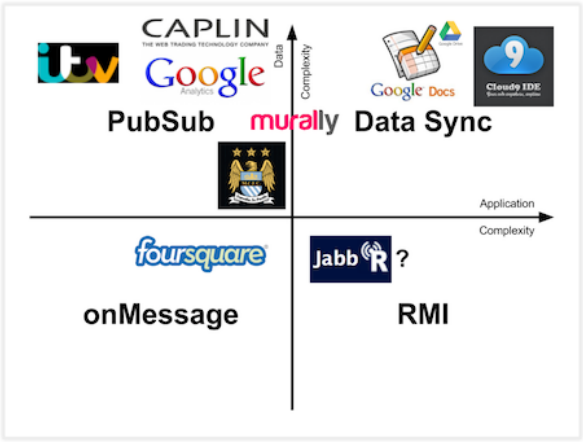
As the complexity of the data increases (and the amount of data) then a PubSub solution is more

likely to be a better fit.

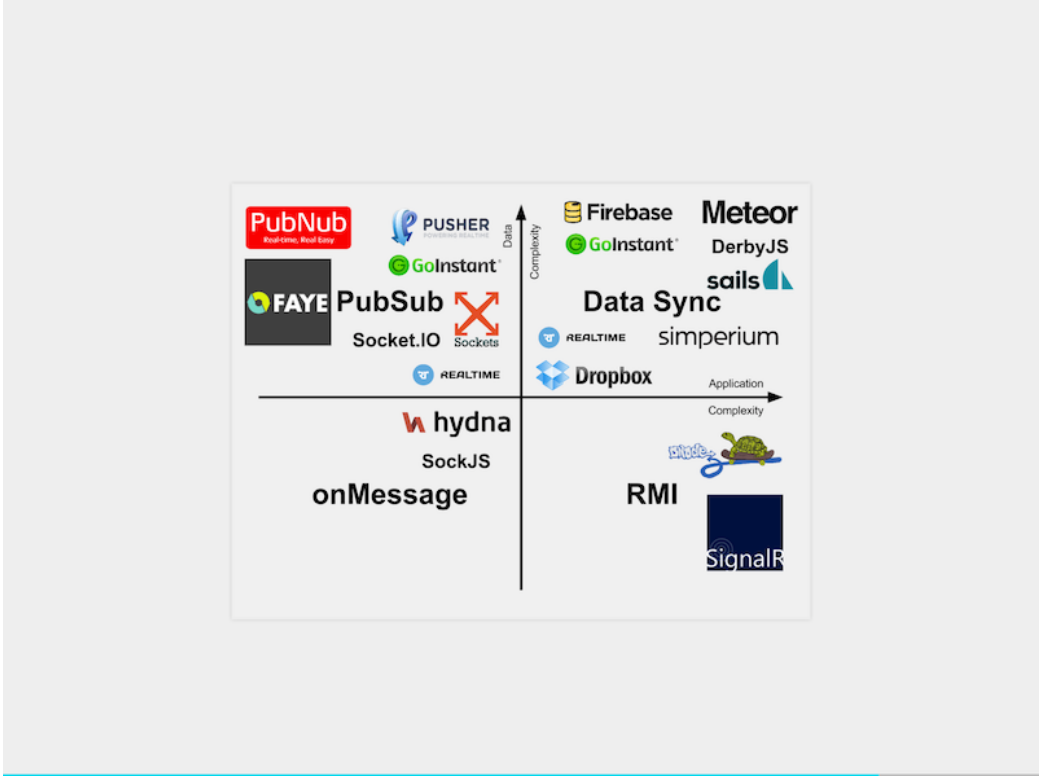
If the data is simple and the the data sizes of the objects representing the method parameters are small, but the interactions are complex, then a RMI solution may suit your needs.

As data complexity increases along with application complexity then a Data Synchronisation solution may be a better fit.

This isn't a one-size-fits-all diagram, but it provides a nice simple way of considering which solution may suit your application's needs.

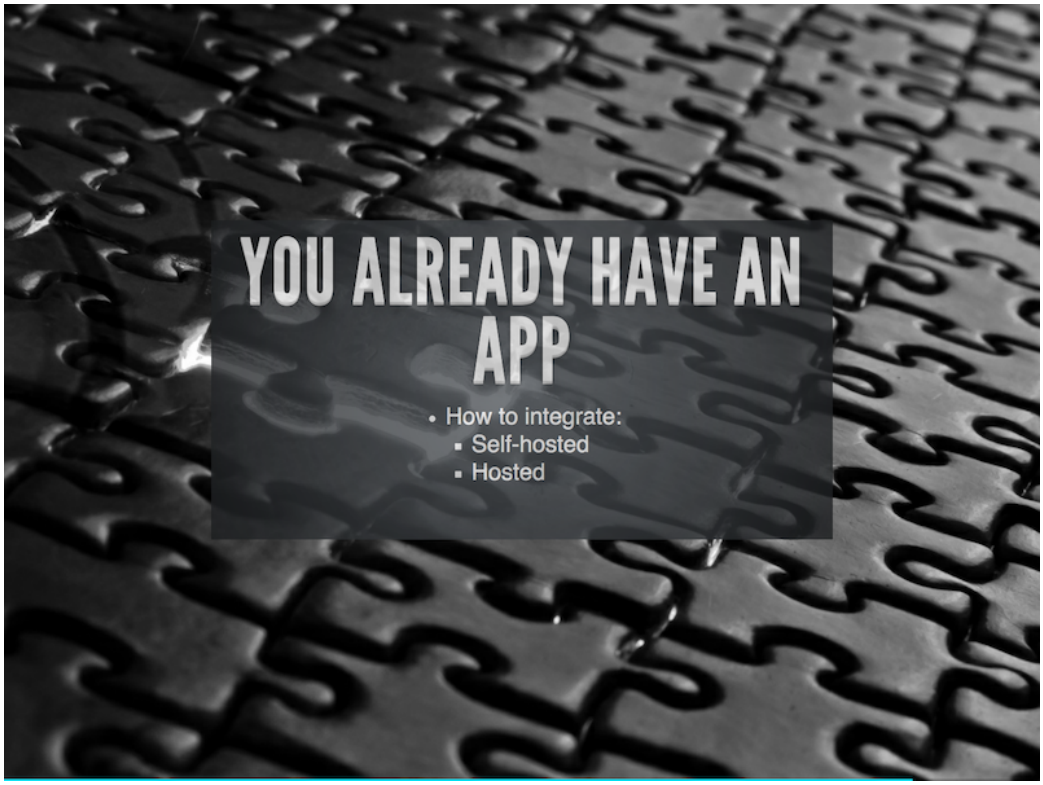


The diagram above shows how the examples I provided in the Benefits & Use Cases section of part one fit into the functionality quadrants.

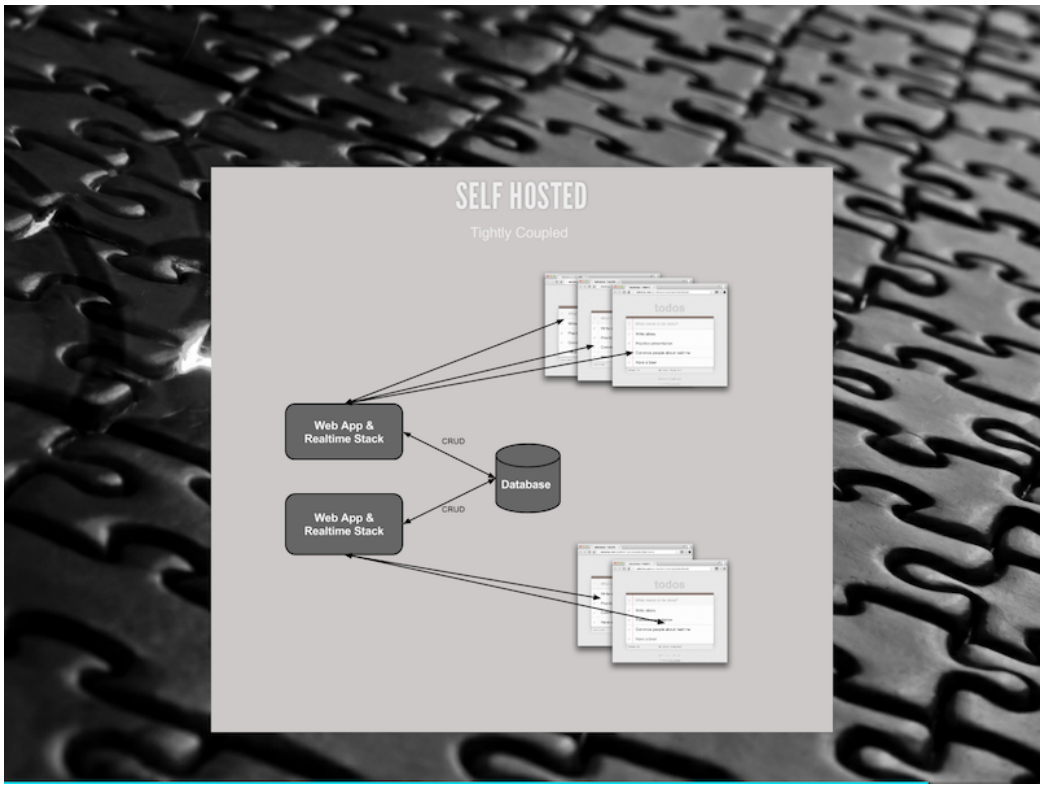


This diagram is an attempt to show the functionality offered by some of the available realtime web solutions. This may not be 100% correct but I plan to add this to the realtime web guide in order to make it as easy as possible for others to make contributes and fixes.

The stage in development that you are at can also affect what the most viable realtime web solution is for you.

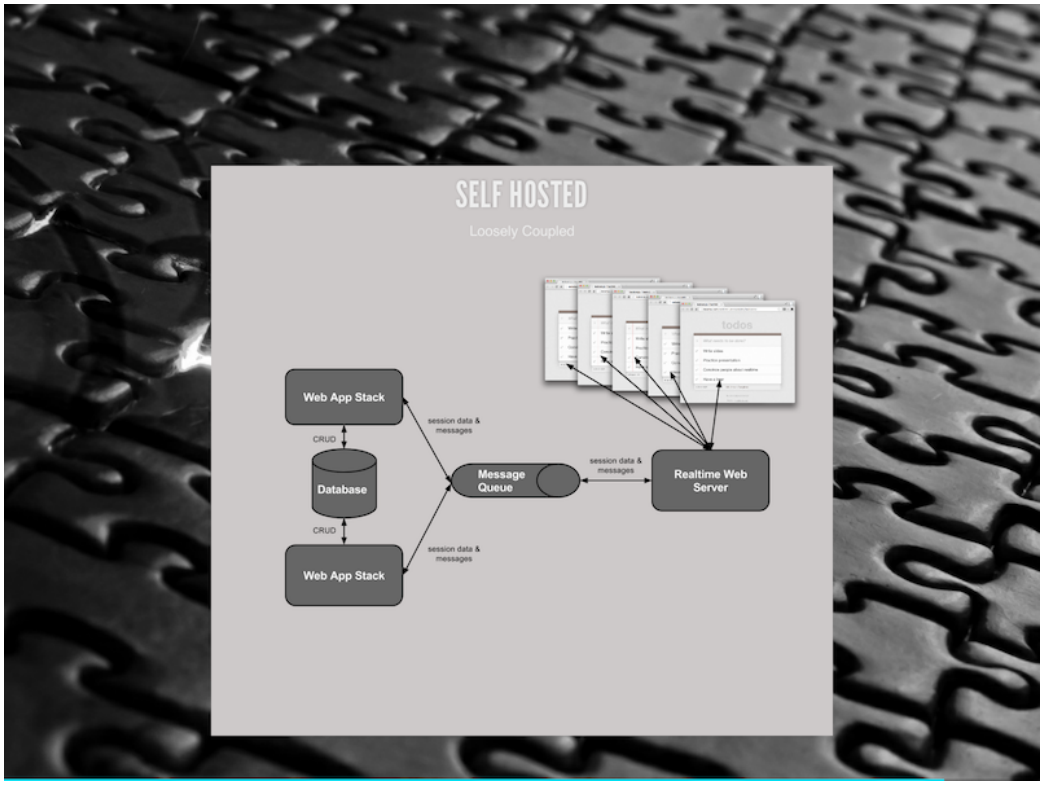


If you already have an application then you'll want to choose a solution that can easily be integrated with your existing application, and it's architecture.



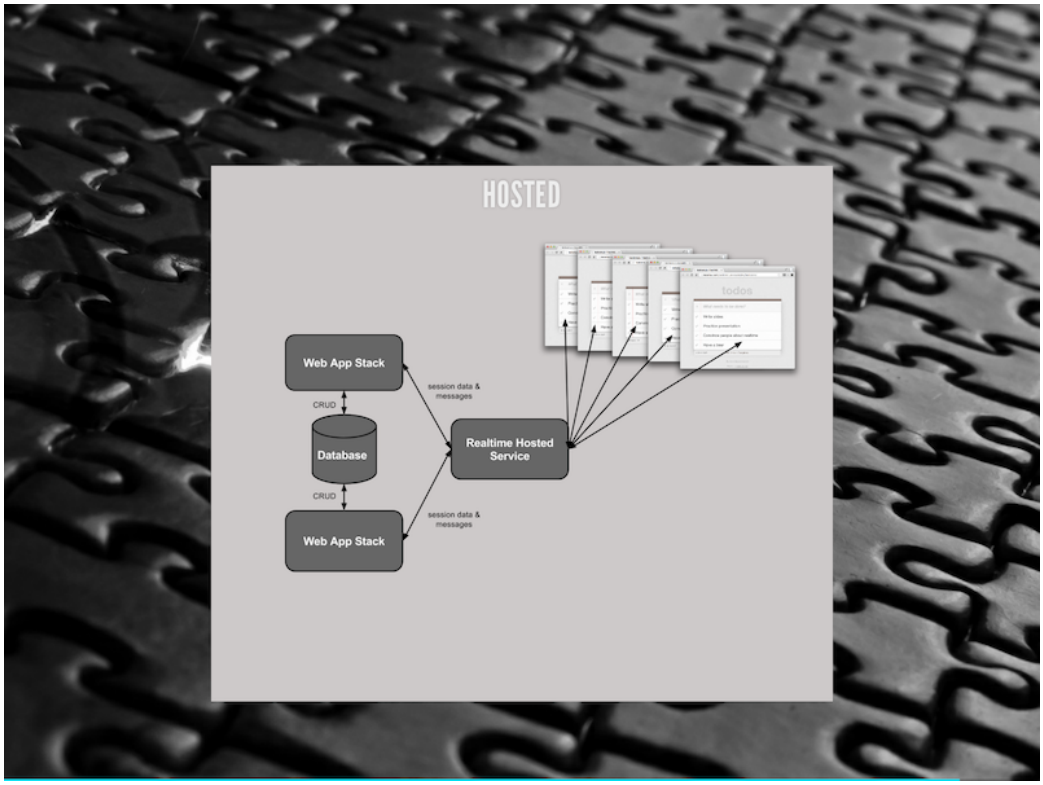
It may be possible to integrate a self-hosted solution in a tightly coupled way. By that I mean the applicaiton web server handles both the HTTP requests and the persistent connections (e.g. WebSocket and HTTP fallback connections) and the logic for your web apps is integrated with your realtime web functionality.

It's possible that there would be less initial overhead in doing this (but not guaranteed if you're really shoe-horing the functionality in). However, as the project grows the complexity of your codebase will increase and you're highly likely going to start to experience problems caused by this complexity. Realtime vs standard web functionality is a good place to exercise separation of concerns and keep that functionality loosely coupled.



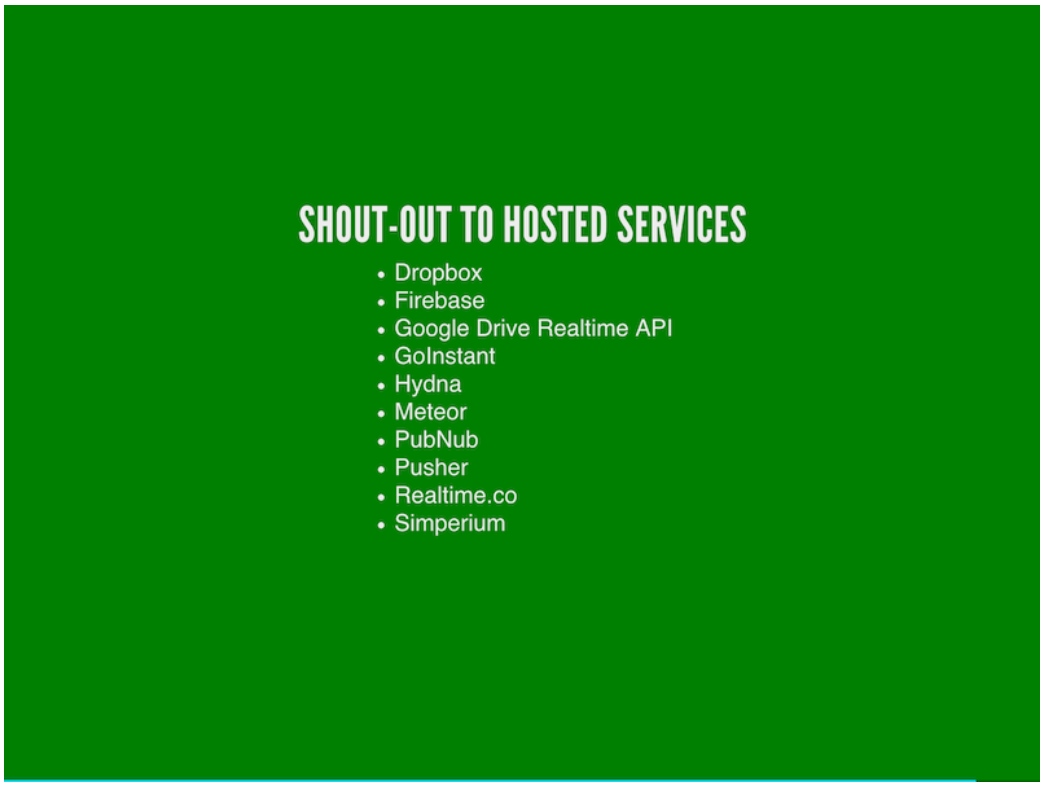
It's generally agreed that a loosely coupled architecture is going to be easier to change, maintain, test and scale. Your DB may even be message-queue-capable (e.g. Redis) so a loosely coupled integration may actually be very simple.

In this situation you can also scale-out your realtime server independently of your web server.



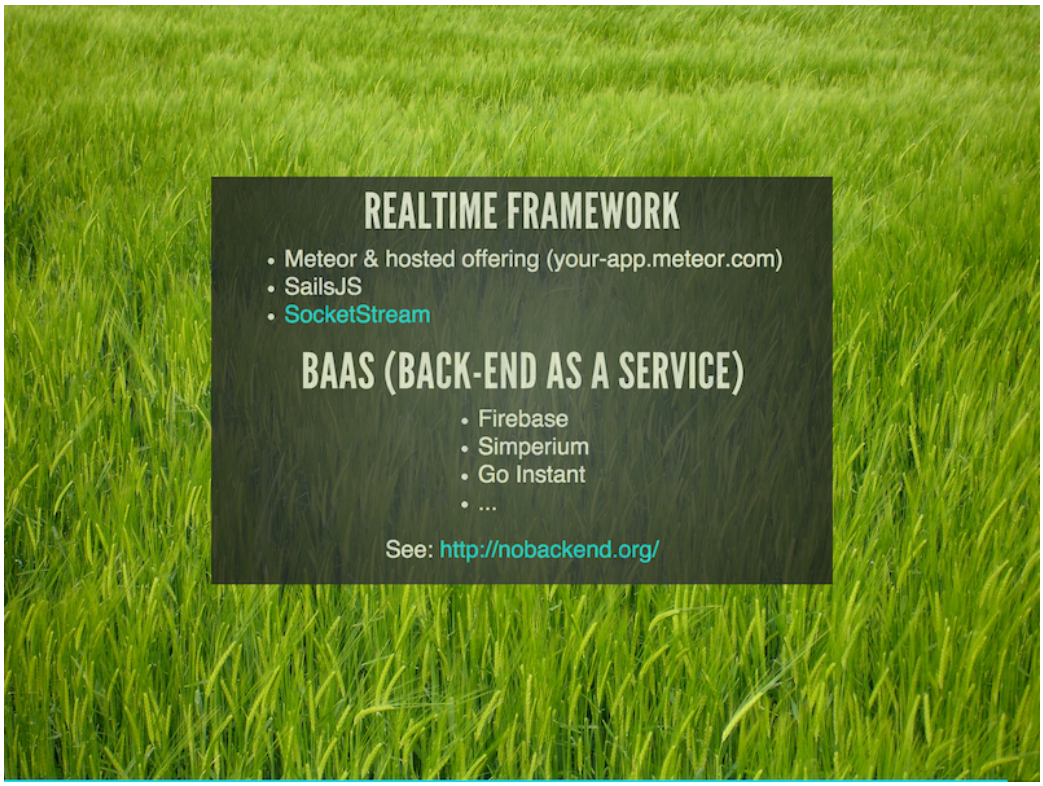
Hosted services offer the fastest and probably simplest integration option - it's one of their focuses, after all.

With a hosted service you get natural loose coupling and you can integrate via simple calls to their APIs. Or by pushing messages onto a queue where integration with the service is performed via one or more dedicated processes.



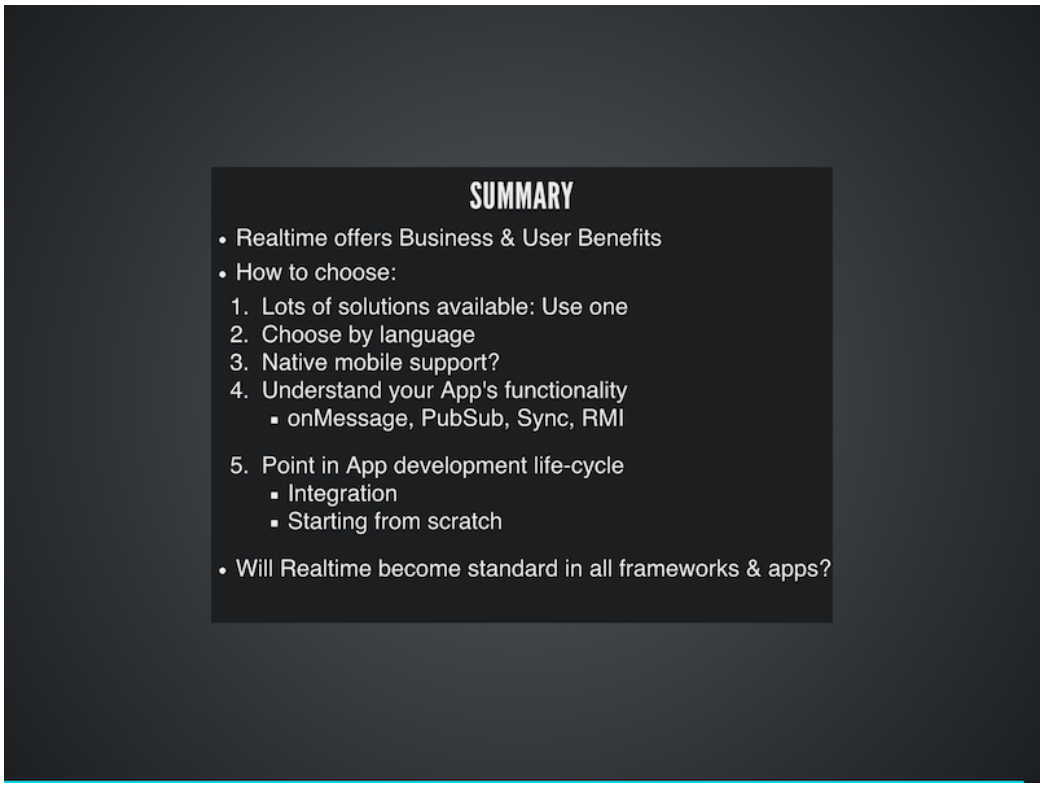
There are an increasing number of realtime hosted services available. Those leading the way are Firebase, Pusher and PubNub.

If you don't presently have an application and are just kicking off your application development you have the full playing field of realtime web solutions to choose from. That includes all the solutions I've already discussed but also some new breeds of framework.



These solutions come with realtime functionality baked-in as a core feature. The framework getting the most publicity right now - and the one with a large amount of funding - is Meteor (not to be confused with the original Meteor Perl solution). Meteor offers a full stack framework built on Node.js that lets you write all your functionality in JavaScript, share code between the front-end and backend, offers data synchronisation, RMI style functionality and also has built in JavaScript templating. They also offer application hosting.

Other solutions worth looking into are Firebase (that also integrations well as a traditional hosted-service), SailsJS, SocketStream and a number of solutions listed as a no-backend solution.



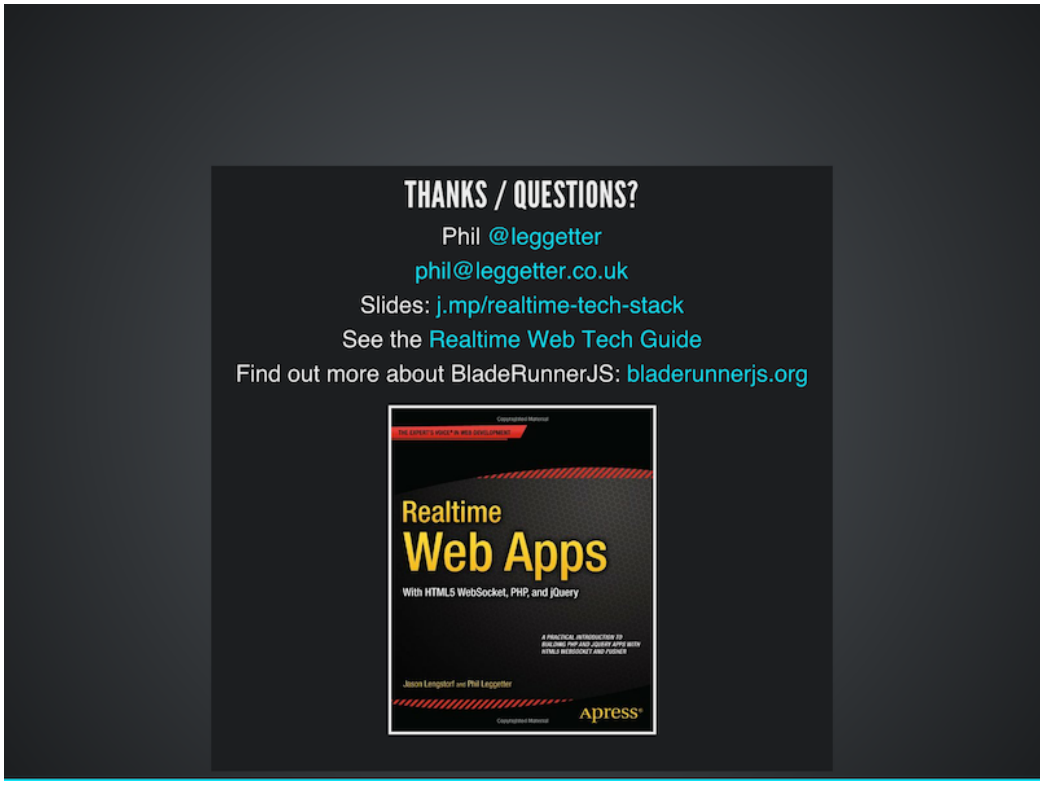
In this three-part series, based on my "Choosing your Realtime Web App Tech Stack" FOWA talk, I've covered a lot of information:

I've discussed why realtime functionality offers both business benefit in the form of increased capability and perception, and the user benefits through an improved service, faster access to information and richer functionality.

I've then gone into the type of functionality that realtime web solutions offer; simple messaging, PubSub, RMI and data synchronisation.

Finally, in this post I've covered how to go about choosing the right solution for you. Choose a good existing solution and consider choosing based on the language it was implemented in. Consider native mobile support as a requirement, and understand how your application will function so you can choose a solution offering the right kind of functionality. And I've covered how the stage of development that you are at should also be considered as there are some exciting new options available to choose from.

After over 10 years working with realtime web technologies I'm still very excited by realtime web technologies - I hope it shows. Realtime functionality is becoming more important and beneficial; the arrival of a number of realtime-focused frameworks demonstrates that. Other existing frameworks are adding this functionality and **maybe it won't be long until realtime will be a standard part of most application frameworks and applications**.



I'm Phil Leggetter, I maintain the realtime web tech guide, I've written a book on building Realtime Web Apps with a PHP stack (because it's not as easy as it should be) and I'm presently working on open sourcing a developer toolkit for building complex front-end web apps called BladeRunnerJS.

GET IN TOUCH

Social

Email

phil@leggetter.co.uk