Meteor.js - A New Paradigm for Connected Client Apps

By Paul Project Manager/Producer/Developer, Japan

#meteor #framework evaluation #single page applications #nodejs #react

Cross-industry international technical & project management experience combined with a successful background in world-class sporting competition and music performance. Available for project management, consulting, and development projects.

Meteor.js is one of the most exciting new developments in the continually evolving technology landscape for full-stack web & hybrid app developers.

Sometimes, the incremental and never-ending changes of web tools and platforms build up over time and collide with user expectations. Multiplied by market forces they produce a radical shift in the technology approach to solving user problems. The last time this happened with the web was the late 90's transition from static websites to database driven ones. We are now in the early stages of another one of these paradigm shifts.

Until recently we were comfortable browsing the web through individual browser sessions, hitting the refresh button to get updates. Now we're mainly mobile. Internet of Things, ecommerce, multi-screen multi device deployment and other forces are creating new contexts and use cases that demand solutions that are robust, performant, easy to develop with and easy to deploy. We work collaboratively, producing and consuming in real time. We expect a great UX and a great UI as key part of it. Notifications and real time updates happen automatically. Twitter, Trello, Line...the list goes on.

Meteor.js refers to this new paradigm of hybrid apps as Connected Clients or Cloud Clients. Others call them Single Page Applications (or SPA's) with reactivity.

Developing these types of apps can be a complex undertaking, with many competing solutions available in both the native space and the hybrid HTML/JavaScript space.

In a lot of cases we have development frameworks designed for the old web simply being extended with bolt on functionality in order to do a new job. This results in lost productivity on a lot of levels.

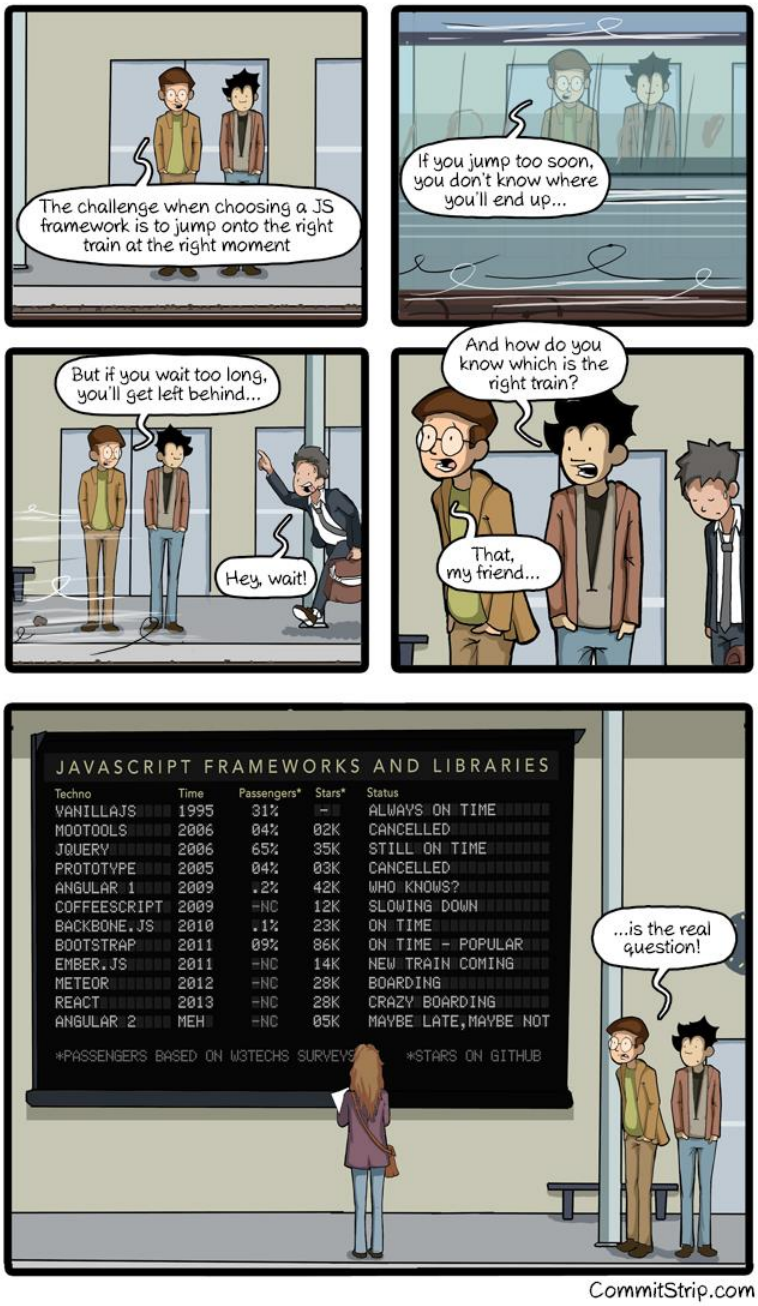


Figure How to choose the right JavaScript framework ©Commitstrip.com

New paradigms demand a new architecture. Enter Meteor.

But rather than just hop on the JavaScript crazy train and zip along for the ride maybe we should ask a few questions first? Let's dig into Meteor a bit more to see how we might answer them.

To determine if meteor is the right train for us we're going to evaluate it from 4 perspectives:

- Business
- Design
- Development
- Operations

We'll also dive down into some code to show a little of the internal structure and data flow.

Let's go!

Business Perspective

- Platform viability
- Product Requirements
- License
- Costs
- Platform Stability

Who is behind Meteor?

Meteor Development Group was founded in late 2011 by key developers with multiple successful products in their pedigree. Backed by some top VC firms including A16Z they most recently raised a $20 Million series B round in May 2015. That follows an $11.2 Million Series A round in mid-2012 led by A16Z.

It's actively developed by full-time employees with experience from Google, Facebook, and Asana.

How do they make money off such a well-funded but free and liberal MIT open source product? Their revenue model is Galaxy - a high-availability/large scale Meteor hosting and outsourced DevOps service recently launched for Enterprise. Even more recently they have announced the upcoming introduction of a low cost development tier. More on deployment options later, as this is a very important part of the development cycle.

What is Meteor.js?

Meteor.js, or Meteor for short, was created to integrate a collection of robust tools so you can build modern apps faster with less code.

- It's a pure 100% JavaScript full stack web-application framework.
- It uses NodeJS on the backend. As both the frontend and backend are 100% JavaScript it's also “Isomorphic”, which means the same scripts will run on client and server.
- The single codebase can be deployed to web or mobile platforms.
- It's MIT-licensed Free Open Source
- It has an open and extensible package based approach.
- It's easy to get started and you can be productive very quickly.

What Meteor is best used for

Single Page Applications (SPA's) or web applications that need:

- Rich client side interaction
- Unobtrusive asynchronous data updates
- Real time data visualization and interaction

What Meteor is not so good for

Like most technology you can usually do anything you want, but you may want to check first if there is a more appropriate solution for your use case. Use cases where it may not be the best solution include:

- When you're building a website not a web application
- Clientless API
- When you already have a server app and need to write a client
- Meteor out of the box uses MongoDB, though they are working on official support for other databases. Community solutions are also available for integrating other databases. You should check what your backend DB requirements are first.

Meteor Alternatives

Meteor is just one solution to developing hybrid apps for the real time web.

Direct alternatives are full stack reactive real-time development ready frameworks.

Some of these include:

- Derby
- Wakanda
- Phoenix
- ASP.NET 5 + .NET and SignalR

You should evaluate these alternatives for your own use case using the same methodology in this article. Each has their own pros and cons for your use case across the range of evaluation criteria. Choose wisely, rather than just choose it because it's the latest cool tech stack. Conversely don't discount a solution just because you don't currently know it.

Platform Stability

Stability in this context refers to business risk due to platform architecture changes. This is a common problem with JavaScript frameworks, which sometimes change rapidly under your feet and therefore may incur a significant technical refactoring load. Meteor has some risk in this regard due to recent announcements regarding a new direction in the UI Layer (ie: Blaze vs React vs Angular) plus some other issues. An important area worth paying attention to, especially in the Meteor forums.

Design Perspective

- UX
- Target Platforms
- Performance
- Scalability
- Security

User Experience (UX) covers a lot of ground, everything from look and feel to rendering performance to interaction snappiness. It's possible to optimize Meteor web apps to squeeze the best browser performance possible on mobile and desktop, but at some point you may need to deploy your app as a native client to deliver the best experience to match target user expectations. Meteor can also integrate many of the popular cross platform UI libraries to suit your application design. Tip: Check out Googles RAIL Performance Model for UX heuristics to guide you.

Meteor uses Cordova to add mobile support. Through a combination of official Meteor packages and Atmosphere community packages you can access a variety of mobile features such as camera, microphone, GPS etc.

Performance with any app can be impacted by how you implement functionality. Performance needs to be considered on a variety of levels, from client side rendering, database accesses, through to server side scaling.

Does Meteor scale? While it's still a relative newcomer in the development framework stakes there are a few companies who have exceeded 100K users. You should pay particular attention to your concurrency requirements, and how best to optimize Meteor to satisfy these. Scaling is also intimately tied to your deployment so refer to the Operations perspective for more details. Also check the Case Studies and forums on the Meteor website for more info and community perspectives.

Security has to be front and center as part of any design, so it pays to spend some time checking out the security section in the official meteor guide to understand how to approach it. Suffice to say it's taken care of, but you need to be aware of what you are doing.

Development Perspective

- Learning curve
- Development tools
- Application Structure
- UI Design
- Reusability
- Deployment
- Testing and debugging
- Support
- Community

First let's step through some of Meteors key components

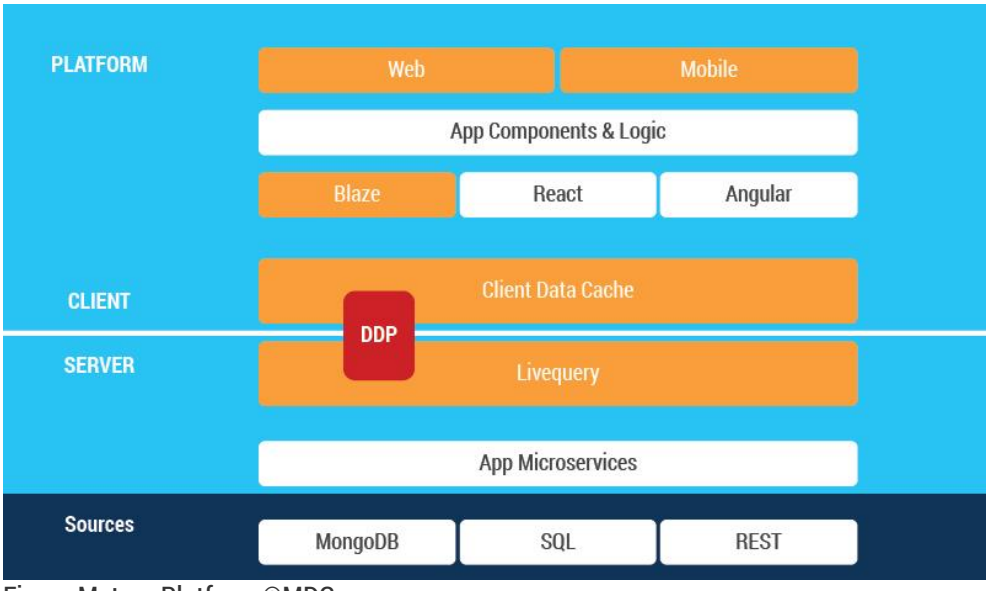


Figure Meteor Platform ©MDG

The meteor platform components are designed so you can focus on developing your app, not spend time dealing with the infrastructure issues which typically impose a huge amount of developer friction.

They include:

- A view layer, which by default is Meteor's Blaze template based approach. However, you can use any UI framework you want, such as Angular or React. Meteor recently posted on their forums about their future development plans for Blaze which is a must read (Hint: React...and Angular!).
- The client side data cache is called Minimongo, which is in-memory JavaScript database that acts like a MongoDB emulator.
- DDP, short for Distributed Data Protocol, is a websocket data interchange protocol. It's technically platform agnostic and could be used to connect non-meteor apps to meteor servers or even non-Meteor servers. Meteor only sends data over the network, and it's up the client to render it.
- Livequery transforms MongoDB into a real-time database.
- Microservices are an architectural pattern for breaking your app up into modules that run as separate services instead of the more common single app approach. The modules can communicate by API's or in Meteors case using service discovery and DDP. The advantages of this approach are for scalability, maintainability, and security. BulletproofMeteor is a great source for practical learning about this powerful approach to app architecture.
- Meteor uses MongoDB as its current main database. Galaxy does not include database support, so you need to either host your own (do you really want to do that?) or use a reliable 3rd party service like https://mongolab.com/ or https://compose.io/ Meteor has more database drivers under development (redis) and the community has also made efforts to integrate reliable production ready databases such as PostgreSQL, RethinkDB, and Neo4J.

A key concept of Meteor is "reactive programming" - *"This means that you can write your code in a simple imperative style, and the result will be automatically recalculated whenever data changes that your code depends on." Ref:* http://docs.meteor.com/#/full/reactivity

Learning Meteor is fairly easy, with lots of examples available. If you want to do something beyond a small app it is advisable to spend some time building your knowledge base. Check the resources section at the end of this article for some key links.

Code development tools can be anything that supports JavaScript, such the freely available Atom or Sublime Text, or one of the more fully featured commercial tools. You can also use Visual Studio Community edition. The Meteor command line tool lets you create projects, edit, integrate packages, build, and deploy apps.

Application structure is an important topic. Meteor is not opinionated like Ruby on Rails, so it's advisable to study some best practices in setting up your projects. Meteor apps consist of a variety of scripts and assets that run on the client side and server side. It's essential to impose order on your project development practices for maintenance purposes. This is even more important in a team environment so team members can coordinate without friction. Start with the official meteor docs first, then try some example apps to see how different people approach it. Our example app coming up also demonstrates an approach to this, as do some of the community resources listed at the end of this article.

UI design is getting easier for graphics challenged developers on a budget. You can leverage libraries such as Bootstrap, AngularJS, or React. Third party theme sites like http://wrapbootstrap.com/ also provide a valuable alternative to jumpstart your meteor project with high quality UI components.

Server Side Rendering (SSR) is also possible to do with Meteor, if your use case demands that.

Reusability of components on both client and server side is relatively easy with Meteor, due to the pure JavaScript nature of development. You will need to think about design a little more if you do intend to reuse components across projects.

Local deployment is very easy. Meteor auto compiles after any save and refreshes on the fly making development workflow a lot easier and rhythmic. Server deployment we'll cover later as it is more involved.

There's a variety of debugging methods you can use, both client side and server side, and also some great community debug tools like the Constellation Dev Console.

Atmospherejs.com is a full-stack package system. It's similar to NPM, and you can also integrate NPM packages if needed. The Meteor community is very active, with over 6000 packages on Atmosphere. Many projects also exist on GitHub.

Check the resource section at the end for official support links. The community is very active, and also vocal about both the strengths and weaknesses of Meteor so dig into it to find out more if you need to.

How does Meteor work in practice?

Meteor methods are created both client side and server side. The client side methods go into a JavaScript file paired with the HTML template the methods are fired from. The server methods go into a JavaScript file in the server directory.

When you click a button or generate some other UI event it can be bound to a corresponding meteor method, client side or server side. The data being updated (or inserted) is sent via DDP and at the same time updates minimongo on the client.

The reason it updates minimongo is so the UI can be updated via the DOM straight away. This is called optimistic UI and improves the user experience with the appearance of snappy responsiveness. Meanwhile the server is processing the data, and results can be fed back to the client to confirm the interaction, roll back the optimistic UI update if needed and/or display errors.

Let's see some code!

Let's walkthrough some code demonstrating some of the above concepts. We're going to use a freely available boilerplate project from GitHub called Meteor-skeleton.

This project demonstrates a few things, including organizing your project. It's a great way to jumpstart a new Meteor project quickly. You can check it out here: http://meteor-skeleton.meteor.com/

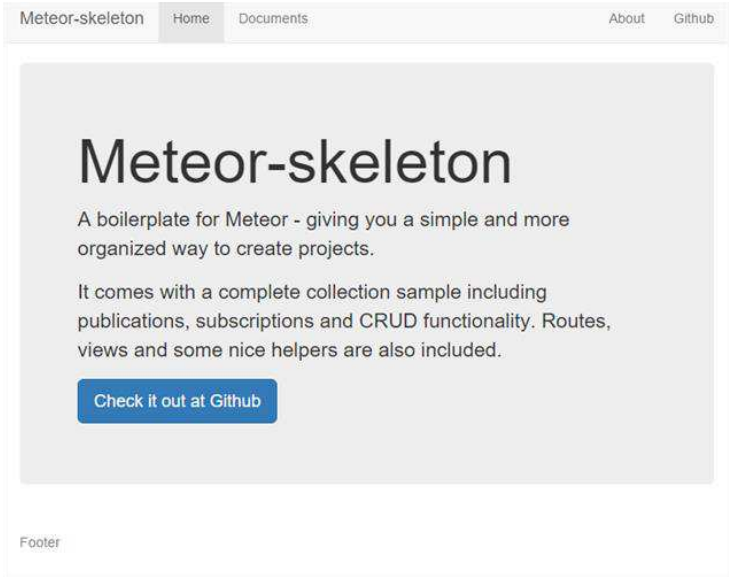


Figure Meteor-Skeleton

The application structure, which we spoke about previously, looks like the following at a high level:

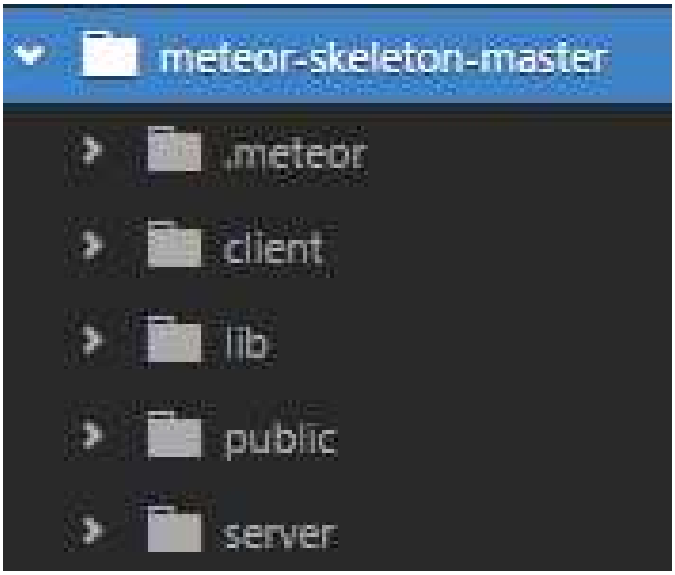


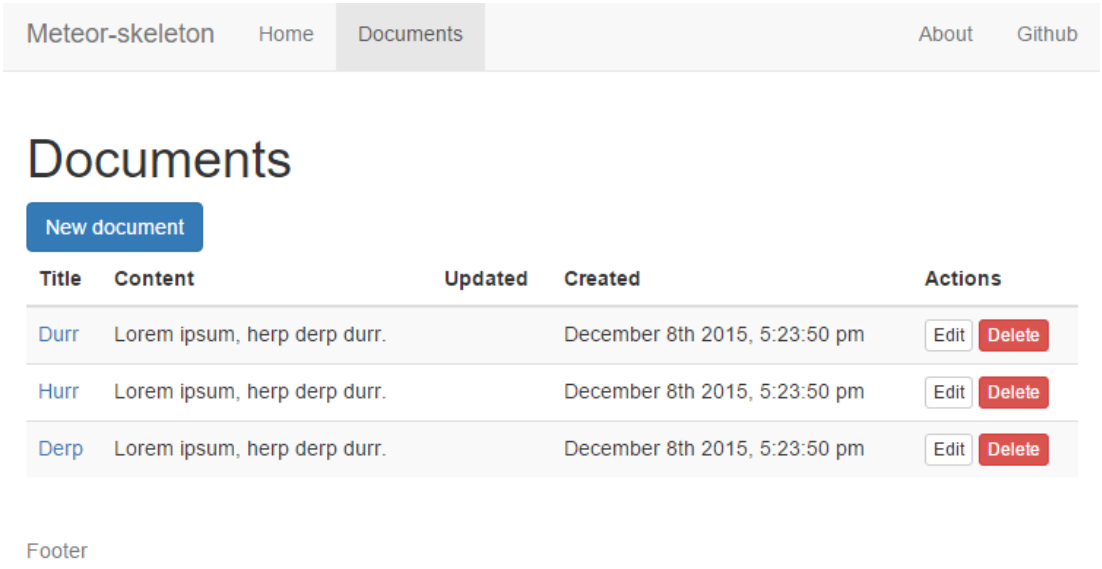
Figure 5 High Level Application Structure

Here's what the structure means:

- .meteor folder contains all the meteor framework code, which we can safely ignore.
- Client folder contains client related code
- Lib contains code common to client and server
- Public contains files directly served to the client
- Server is for server side code only

Within these folders you can then partition the code into more folder levels based on functionality.

If we click on the "Documents" tab we'll see the following screen:



Let's check out what the html template for this looks like:

```
<template name="documentsIndex">
 <div class="row">
  <div class="col-md-12">
   <h1>Documents</h1>
   <a class="btn btn-primary" href="{{pathFor 'documentNew'}}" role="button">New document</a>
   {{#if Template.subscriptionsReady}}
    {{#if documents}}
     <table class="table table-striped">
      <thead>
      <tr>
       <th>Title</th>
       <th>Content</th>
       <th>Updated</th>
       <th>Created</th>
       <th>Actions</th>
      </tr>
      </thead>
      <tbody>
      {{#each documents}}
       <tr>
        <td><a href="{{pathFor 'documentShow' documentId=_id}}">{{title}}</a></td>
        <td>{{content}}</td>
        <td>{{showTimeAgo updatedAt}}</td>
        <td>{{showPrettyTimestamp createdAt}}</td>
        <td>
         <a class="btn btn-xs btn-default" href="{{pathFor 'documentEdit' documentId=_id}}"
role="button">Edit</a>
          <a class="btn btn-xs btn-danger delete-document" href="#" role="button">Delete</a>
        </td>
       </tr>
      {{/each}}
      </tbody>
     </table>
    {{else}}
     <p>No documents yet.</p>
    {{/if}}
   {{else}}
    {{> loading}}
   {{/if}}
  </div>
 </div>
</template>
```

The associated JavaScript code for this template is:

```
Template.documentsIndex.onCreated(function() {
 this.subscribe('documents');
});
Template.documentsIndex.onRendered(function() {
});

Template.documentsIndex.onDestroyed(function() {
});
```

```
Template.documentsIndex.helpers({
 documents: function () {
  return Documents.find({}, {sort: {createdAt: -1}});
 }
});

Template.documentsIndex.events ({
 'click .delete-document': function(e) {
  e.preventDefault();
  var item = this;

if (confirm("Are you sure?")) {
    Documents.remove(item._id);
    console.log("Document deleted")
  }
 }
});
```

We can see the page template is populated from a collection of documents. This documents collection is published on the server

```
Meteor.publish('documents', function() {
return Documents.find();

});
```

and then subscribed to on the client:

```
Template.documentsIndex.onCreated(function() {
this.subscribe('documents'); });
```

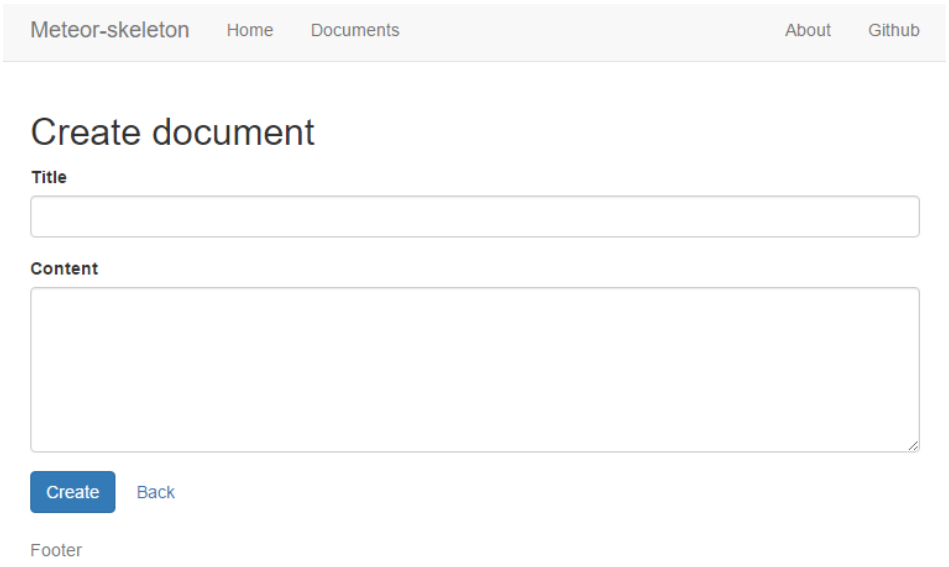
We can delete a document by clicking on the delete button which calls the client side meteor method:

'click .delete-document': function(e) and removes the selected document from the collection.

How do we add a new document? Let's go back to the template html and look at:

```
"btn btn-primary" href="{{pathFor 'documentNew'}}" role="button">New document
```

href="{{pathFor 'documentNew'}} redirects to template documentNew:



And the HTML:

```
<template name="documentNew">
<h2>Create document</h2>

{{#autoForm collection="Documents" id="documentForm" type="insert"}}
  <fieldset>
   {{> afQuickField name='title'}} {{> afQuickField name='content' rows=6}}
  </fieldset>
  <button type="submit" class="btn btn-primary">Create</button>
  <a class="btn btn-link" role="button" href="{{pathFor 'documentsIndex'}}">Back</a>
 {{/autoForm}}

</template>
```

What's interesting about this is how the Create button behavior is handled this time. Instead of calling a meteor method it is automatically inserting a document on form submit. How is it doing this? It's using a package from Atmosphere called aldeed:autoform which has a rather neat automatic insert and update capability, along with reactive validation. A very handy time saver! Meteor magic!

So there we have a basic demonstration of how meteor works in practice.

It's all 100% pure JavaScript so don't forget to also check the blog on Most Common JavaScript Mistakes of All Time to ensure you're avoiding common coding pitfalls.

Operations Perspective

- Cost
- Reliability
- Scalability
- Ease of Use
- Support
- Analytics
- DevOps

Deploying to a remote server is where things get interesting.

Galaxy , Meteor Development Group's high-availability/large scale Meteor hosting and outsourced DevOps, will service high cost Enterprise while also providing a low cost developer tier. More details are coming on this so stay tuned.

It will not include MongoDB hosting so you will still need to arrange this at a 3rd party service like MongoLab or Compose. Galaxy does have a tie up with Compose (now an IBM company) so it's an easy pathway to using them if they are your preferred service.

It's also possible to deploy Meteor to any of the big hosting services like Heroku, Digital Ocean, Modulus, Azure, AWS, or BlueMix. They need to be evaluated on their own merits and that is unfortunately outside the scope of this article.

Some services provide their scripts or you can use a custom script such as meteor-up to deploy to these services.

Doing the full deployment and management of your app to servers is also a time sink requiring at times quite specialist knowledge, especially when you move into scaling your app.

Therefore, some things to pay particular attention to are ease of deployment, ease of scaling, and ease of monitoring.

Some newer 3rd party services have recently entered this space with innovative one-click deployments from Git to Docker deployment with easily scalable control. The better ones also provide HTTPS out of the box, in addition to MongoDB along with Oplog tailing which is essential to Meteor and MongoDB reactivity. These are worth evaluating if you value your development time over server admin time:

- https://scalingo.com
- http://nodechef.com/

Performance Monitoring

It's also essential to monitor your apps operational performance. Kadira.io is a must have performance monitoring service that's easy to implement. It also has a free and low cost independent developer tier in addition to supporting enterprise.

Meteor Resources

Official

- Platform Components Guide: https://www.meteor.com/projects
- Meteor Guide: http://guide.meteor.com/
- Example Projects: https://www.meteor.com/tutorials/blaze/creating-an-app
- Documentation: http://docs.meteor.com/#/full/
- Forums: https://forums.meteor.com/

Community

- How to build production scale apps: https://bulletproofmeteor.com/
- Great tutorial/solutions guide: https://themeteorchef.com/
- Stack Overflow: http://stackoverflow.com/questions/tagged/meteor
- GitHub: http://github.com/meteor/meteor
- https://meteor-js.zeef.com/benjamin.clark.for.impresarios

Author Bio

Paul is an independent consultant based in Tokyo. He has 20+ years of international cross-industry development and project management experience working on a variety of projects from corporate IT to eLearning simulations to educational apps and games. He's also a former world champion in sport parachuting, and regularly performs music online as the blues infused slide guitar/harmonica cyborg "Komuso Tokugawa".