Web Applications with Meteor

Anant Bhardwaj
anantb@csail.mit.edu
April 13, 2014

Contents

1 History of Web Applications 2
2 Meteor Web Framework 3
  2.1 Motivation for Meteor 3
  2.2 Meteor architecture 3
3 Writing Web Applications with Meteor 3
  3.1 Set up a new Meteor project 3
  3.2 Meteor Project Structure 4
  3.3 Models (Data) 4
  3.4 Templates (Views) 5
  3.5 Realtime/Reactive updates 6
  3.6 Caching 6
4 Conclusion 7
5 Exercises 7
Abstract
In this lecture, we will learn how to build web applications. We will discuss different approaches and frameworks in practice, and introduce Meteor. Meteor is a web framework for end-to-end JavaScript applications. You can use JavaScript on both the client and server. The framework is designed to make building web applications simple. It goes beyond traditional MVC frameworks by breaking the client-server separation and pushing the View completely to the client. This enables reactive programming and real-time synchronization (changes to the models on the server can be reflected live in the browser).

1 History of Web Applications

1. Initially: Static Web Pages
   (a) HTML + CSS
   (b) Javascript for interactive UI
   (c) Example: what you did for your problem sets

2. CGI: URLs Map to Executable Programs
   (a) When URL is referenced, program runs. Example: https://courses.csail.mit.edu/6.831/form/form.py?formkey=dHJpWDBsZThkYXdlVWpyQ013LwpMcXc6MQ
   (b) Program’s output is a web page, returned to the browser
   (c) Program quits after each request
   (d) Introduced the notion of stateless servers: each request independent, no state carried over from previous requests.
   (e) Perl/Python typically used for writing CGI programs

3. Server Side Scripting: PHP, JSP, ASP
   (a) HTML mixed with code. Example: https://github.com/abhardwaj/redprint/blob/master/web/ajax/examples.php
   (b) Language runtime system embedded directly into web server (for example: the apache web server has mod_php)
   (c) These languages provided reusable library packages for common web operations such as request handling (GET, POST), sessions, databases, etc.

4. Web Frameworks: MVC and ORM (Ruby on Rails, Django)
   (a) Model-View-Controller: An architecture for decomposing web applications:
      i. Model: manages the application data
      ii. View: generates the required HTML pages
      iii. Controller: a glue between the model and view which responds to incoming HTTP requests, fetches/modifies data in the models, and invokes view(s) to render results.
   (b) Object-Relational Mapping: Database tables are represented as class objects. It simplifies the use of databases.

5. Web Frameworks 2.0: Realtime, Reactive, All JavaScript (Meteor)
   (a) All JavaScript: Same code/language can be used on the server and the client both. It is possible to write entire web application in JavaScript.
   (b) No client-server separation: A client code can call any functions (like database calls) as if it is running on the server. Meteor pushes the View completely to the client side.
   (c) Realtime/Reactive updates: Views (pages on the client) can react to Model (database on the server) changes. Pages automatically update whenever data in the Model changes. No more client-server data synchronization (AJAX) required.

---

1I thank John Ousterhout and Phil Levis for letting me use materials from their Web Applications class at Stanford. I took this class in Fall 2010, and TAed it in Spring 2012.
2 Meteor Web Framework

Meteor is a full stack framework for end-to-end JavaScript applications. Now, an obvious question is: there are so many web frameworks out there – why Meteor? The main advantage of Meteor is that you don’t have to deal with client-server separation. In problem sets, you learned event-based programming using JavaScript where the idea was to implement handlers for events. What’s great about Meteor is that it allows you to write handlers for changes happening on the server (example: database model changes) the same way you write handlers for DOM events. Meteor magically abstracts all the client-server separation which allows your client code running inside a browser to listen to the events on the server. Meteor stack takes care of all the underlying complexities.

2.1 Motivation for Meteor

In traditional web frameworks, when user visits a URL on a browser, the URL is sent to the server, the server invokes the corresponding controller handler – the controller handler fetches/updates the models (if required), generates a response (typically HTML), and sends it back to the browser. For every URL, there is a round trip visit to the server.

For some applications, server round trips can be very costly. The network latency, the overhead of recreating the DOM tree, flushing the browser context, reloading JavaScript, CSS, etc. on each update can affect the user experience. Applications typically use AJAX to get the updated application state (usually a very small data) from the server and update the view (page) in the browser by dynamically updating the DOM. This delivers a much better user experience because the client doesn’t need to reload the page for updating the state. You can imagine how bad the user experience would be if Facebook reloaded the page whenever a new update needs to be displayed on your newsfeed. Facebook uses AJAX to periodically poll the server to check whenever a new update is to be displayed on the newsfeed – whenever a new feed is available, it dynamically inserts the new feed to the page by updating the DOM.

2.2 Meteor architecture

Meteor encourages the architecture where nearly all application states are handled by the client, in JavaScript. Traditionally the client side was totally separated in the MVC architecture. As we saw with the Facebook newsfeed example – applications have to workaround by using AJAX. Meteor solves it by letting views (on the browser) to reactively update as the data in the models (on the server) is updated. Pushing the view completely to the client side eliminates the need of synchronizing client side models with the server side models. Traditional MVC frameworks required applications to use AJAX for synchronizing client side models with the server side models.

3 Writing Web Applications with Meteor

In this section, we will discuss how to write a web application using the Meteor framework.

3.1 Set up a new Meteor project

1. Install Meteor
   
   $ curl https://install.meteor.com | /bin/sh

2. Pick a name for the new app. Let’s call it my_uid_app
   
   $ meteor create my_uid_app

3. Run it locally
   
   $ cd my_uid_app
   $ meteor
   $ => Started proxy.

I highly recommend looking at the sample demo applications and documentations available on the Meteor website.
3.2 Meteor Project Structure

When you create a new project using `meteor create`, Meteor puts the following 3 files in the project directory by default.

- **my_uid_app.css**: This file is for putting the style definitions. By default it’s empty.
- **my_uid_app.html**: This file is a View/Template file. This code is responsible for generating the HTML. We discuss **Templating** in detail in the following section.
- **my_uid_app.js**: The JS file is for writing the handlers (the program logic) and the Model definitions. The JS file has the following structure:

  ```javascript
  if (Meteor.isClient) {
    // code to run on client
  }

  if (Meteor.isServer) {
    Meteor.startup(function () {
      // code to run on server at startup
    });
  }
  ```

3.3 Models (Data)

Model is an abstraction for a persistent data storage. Meteor ships with MongoDB as the default database.

1. Declare all the collections. The declaration should be outside `if (Meteor.isClient) { }` and `if (Meteor.isServer) { }` so that both the client and the server have access to it.

   ```javascript
   // declare collections
   Students = new Meteor.Collection("students");
   Staff = new Meteor.Collection("staff");
   
   if (Meteor.isClient) {
     // code to run on client
   }

   if (Meteor.isServer) {
     Meteor.startup(function () {
       // code to run on server at startup
     });
   }
   ```

2. Let’s put some default data

   ```javascript
   // on server side: add the staff names if the database is empty.
   if (Meteor.isServer) {
     Meteor.startup(function () {
       if (Staff.find().count() === 0) {
         Staff.insert({name: "Rob Miller", type: "Instructor"});
         Staff.insert({name: "David Karger", type: "Instructor"});
       }
     });
   }
   ```
Staff.insert({name: "Daniel Jackson", type: "Instructor"]);
Staff.insert({name: "Phil Guo", type: "Instructor"});

Staff.insert({name: "Anant Bhardwaj", type: "TA"]);
Staff.insert({name: "Tami Forrester", type: "TA"]);
Staff.insert({name: "Connie Huang", type: "TA"]);
Staff.insert({name: "Marcus Lowe", type: "TA"]);
Staff.insert({name: "Phillip Mercer", type: "TA"]);
Staff.insert({name: "Phu Nguyen", type: "TA"]);
Staff.insert({name: "Beneah Wekesa", type: "TA"]);
Staff.insert({name: "Nahom Workie", type: "TA"]);
Staff.insert({name: "Carolyn Zhang", type: "TA"]);

3.4 Templates (Views)

Templating is the most common way to generate dynamic HTML; available in virtually all Web development frameworks. Meteor has an embedded templating language called Spacebars [Spa14], inspired by Handlebars [Han14].

To show all the instructors and TAs in the UID class, a template might look like the following:

- my_uid_app.html

```html
<body>
  <h1>6.813 Staff</h1>
  {{> staff}}
</body>

<template name="staff">
  <h2>Instructors</h2>
  <ul>
    {{#each instructors}}
    <li>{{name}}</li>
    {{/each}}
  </ul>

  <h2>TAs</h2>
  <ul>
    {{#each tas}}
    <li>{{name}}</li>
    {{/each}}
  </ul>
</template>
```

- my_uid_app.js

```javascript
if (Meteor.isClient) {
  // all staff records
  Template.staff.all = function () {
    return Staff.find(); // all staff
  };

  // all instructors records
  Template.staff.instructors = function () {
    return Staff.find({type: "Instructor"}); // all instructors
  };
}```
// all TA records
Template.staff.tas = function () {
    return Staff.find({type: "TA"}); // all TAs
};

The template is expanded by executing code snippets, substituting the results into the document. You should be able to see all the instructors and the TAs when you open the app in a browser (Figure 1). Armed with this knowledge, you can now build a leaderboard app. See Exercise 2.

![List of all the instructors and the TAs for the UID class](image)

**Figure 1: List of all the instructors and the TAs for the UID class**

### 3.5 Realtime/Reactive updates

Meteor Template functions run as a reactive computation. Meteor uses a live page update technology called Blaze [Bla14], which renders the HTML reactively. This means that whenever data in the model changes (for example: a new TA gets added), it will automatically update the HTML by tracking all the places staff template is used.

### 3.6 Caching

Caching helps improve performance by eliminating the server round-trip in many cases. It is typically used by production applications. If you are building a simple prototype for demo, you don’t need to worry about caching.

In Meteor, although the client and server both can make database queries, only the server has direct access to the database. Meteor client includes an in-memory database which can cache the records published by the server. The server publishes database records as a document. A client can subscribe to many documents. Once subscribed, the client uses its cache as a fast local database, saving the costly round trip to the server. The publish function runs each time a new client subscribes.

- Server: publishes records as document

    // server: publish staff records
    Meteor.publish("staff", function () {

return Staff.find(); // all staff
});

- Client: subscribes to the published documents

    // client: subscribes to staff
    Meteor.subscribe("staff");

    // client: makes local queries
    Staff.find({type: "TA"}).fetch();

4 Conclusion

This completes a brief tour of Meteor. Meteor has many more functionality than what is covered in this note. This note covers only the basic components required to get started with Meteor. Reading Meteor documentation (http://docs.meteor.com/) is highly recommended.

5 Exercises


2. Leaderboard: Watch the video (http://crowdy.csail.mit.edu/play/39/) and look at the example code ($ meteor create --example leaderboard). Add a button that resets everyone’s score to zero. Implement a way to add and remove scientists from the leaderboard.

3. TODO List: [Optional] Watch the video (http://crowdy.csail.mit.edu/play/40/) and try to understand the example code ($ meteor create --example todos). Now take this data: http://confer.csail.mit.edu/static/conf/chi2014/data/papers.json, a list of papers. Write an app that displays all the paper titles and allows user to add tags to each paper. Allow filtering on tags.

References


