Building Real-Time Web Applications with Meteor

Sophie Rust
Leiden University MSc student
sophie.rust@gmail.com

Jasper Schelling
Leiden University MSc student
jasper.schelling@gmail.com

Donna Schipper
Leiden University MSc student
donnaschipper@gmail.com

ABSTRACT
The Meteor framework provides a way to develop real-time web applications with just one programming language: JavaScript. While keeping in mind other conventional technology stacks as LAMP, the framework can be seen as a whole new paradigm because it makes use of server-side JavaScript. First, we shortly explore the history of web development frameworks in order to describe the context of Meteor. Second, the architecture and operating principles are discussed in order to explain the real-time aspects of Meteor. Strengths and weaknesses are explored and example applications will be discussed to show the power of Meteor. Finally a guide to build a Meteor application is given and core elements contributing to create real-time applications with Meteor, such as 'templates' and 'collections' are explained.

1. PURPOSE, CONTEXT AND HISTORY

Meteor is a set of interlocking web technologies that provides developers a modern method to create web-applications [20]. This paper discusses how Meteor facilitates the creation of real-time web applications. With real-time is meant, web-based applications where the interactions between client and server take place at such a speed that users experience direct feedback on their interactions with the application and other users. Modified data is directly shown without the user having to refresh the page or the system having to execute periodical checks [15].

In order to understand the characteristics of a platform like Meteor, general background knowledge about the way web developers combine different technologies, is desired. Web developers rely on a 'stack': a set of software technologies that provides users with the functionality of a web-based application [29]. In this set, the following technologies are commonly found: an operating system, providing the server with basic computer operations and networking, an http-server, which handles http-requests from web-browsers, database software to store data and a scripting language with an interpreter to write the code that is executed when the user interacts with the web-application [33].

While many other stacks exist in parallel, the 'LAMP stack' has become the most common set of server-side technologies since the late 1990s. This stack consists of four open-source technologies; Linux (operating system), Apache (web-server), MySQL (relational database) and PHP, Perl or Python (scripting languages). A reason for the adoption of these open source technologies seems partly its low cost of acquisition, but mainly that the usage of an open-source stack prevents what is called 'vendor lock-in', where an entity that uses closed source software becomes dependent on the commercial vendors of that closed source software [9].

A weaknesses of the LAMP-based approached is the broad knowledge web developers needs to have to be able to create a complete web application. Each layer of the stack has its own set of principles and constraints, complicating the interfacing between the different layers. The following milestones in the young history of web development indicate a transformation of this convention and in the end led to the foundation of the Meteor platform.

2008: Release of the Chrome browser by Google that included a new JavaScript interpreter (V8). It was significantly faster than its competitors. The speed of JavaScript interpreters in general increased in a period of competition between browsers [28]. Developers started to integrate JavaScript interpreters into projects and technologies commonly associated with the 'server-side' of web-application development.

2009: JavaScript based alternatives to technologies common in the LAMP stack emerged: a new type of web server (Node.JS/ IO.JS[1]), a new type of non-relational database (MongoDB[2]) and JavaScript as a server-side development language to create web applications [12].

2011: 'Skybreak' is announced [17]. Soon the project changes its name to Meteor [4]. The Meteor platform is built on top of the previously mentioned developments and coalesces technologies like Node.JS and MongoDB into a cohesive whole. With one programming language (JavaScript) on both the server and client side, and one data interchange format, JavaScript Object Notation (JSON), the breadth of skills required to develop web applications decreased. [12].

2012: The Meteor Group receives funding of Netscape creator Marc Andreessen. This allows future development of the project [27].

2014: Release of Meteor 1.0, the first public release [5].

2. OPERATING PRINCIPLES

In this part the basic operation of a Meteor project is discussed. Since to some extent, the architecture has already been discussed in order to explain the context and history of Meteor, we start with a description of what happens when a user makes a request to a web server that is running Meteor. Afterwards, we describe what interactions between the client and the server take place, to synchronize the data between both. In order to limit the scope of this article to Meteor's real-time characteristics, we do not discuss the development and deployment chain of a Meteor project.

2.1 Request

Most major software execution takes place on the client side. When a user makes the first request to a web application, the server sends a complete package to the

[1] Event driven framework to build network applications. https://nodejs.org

[2] Open source document database. http://docs.mongodb.org/manual/

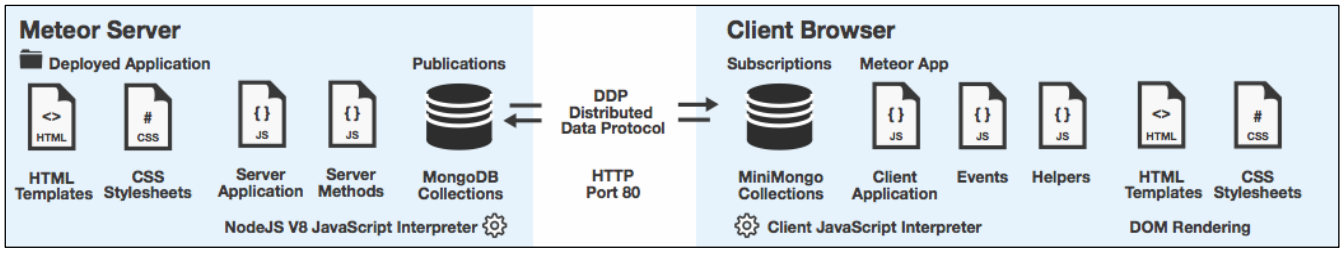

Figure 1. Overview of files an technologies taking part in client-server communication.

client: html templates, css stylesheets, JavaScript application code and the parts of the Meteor framework that enable the operation on the client, such as the in-memory JavaScript database, miniMongo [24] (see Figure 1). This set of technologies provides the basic operation of the application in the client's browser, as well as the low level functionality to enable the application to interact with the Meteor server in a real-time manner.

From the moment the client possess the initial state of the application, no html markup is sent anymore. This setup is chosen to facilitate one of the principles of Meteor: 'Data on the Wire'. This means that during operation, the client and server only exchange data: values that are inserted in the database by any of the clients or the server. As a developer, you define which parts of your html react to future changes in the data. When the data changes, its presentation is automatically updated [16]. This reduces the amount of data the application needs to send back-and forth, and as such is one of the principles that enables real-time operation of an application [15]. However, this principle on its own is not enough to remain responsive with a large amount of users.

2.2 Client and server interactions

To further enable the operation of a real-time web application, the Meteor framework makes use of a programming paradigm known as 'reactive programming'. Reactive programming is a specific method to create software in such a manner that, when changes occur in the underlying data, changes in the data get propagated to all parts of the program that make use of that data, similar to the way that changes in data get propagated within a digital spreadsheet: when a value of one cell is updated, all other cells using the value of that cell, update the outcome of their calculations directly. The system reacts when data has changed, instead of periodically asking if something has changed. Meteor presents the benefits of this style of programming as '*reactivity*' [21][25]. Much of the actual functioning of this reactive standard cannot be seen while using the Meteor framework. As such, the perception of a developer using it, is that reactivity is something that 'comes for free' and is not something he or she should be concerned with during the creation of an application. In the next part, the operation from the client to the server will be examined in more detail.

Meteor makes use of the principle 'latency compensation' to complete the experience of a real-time application. Besides the actual MongoDB database[3] that is running on the server, the Meteor server ships to each client a subset of its database: MiniMongo, a reduced version of the MongoDB system [24]. This reduced database is used to facilitate the apparent aspect of real-time operation of a Meteor application. Whenever the data of a connected client changes, it is first affected in the subset of the database that is part of the clients application. Simultaneously the server changes the data in the actual database [23]. In the background operation of the application, the client communicates with its server using the Distributed Data Protocol (DDP). This protocol keeps the databases in sync between central database on the server and every connected client [22].

3. STRENGTHS AND WEAKNESSES

The major strength of using the Meteor framework is its shallow learning curve. Developers do not need to have a broad knowledge of multiple technologies anymore, because a lot of the interactions that usually cause the major problems in a development process, are already covered by Meteor [7]. Just having a thorough understanding of JavaScript, HTML and CSS gives developers the possibility to produce large scale projects. The Meteor framework has a big community and the Meteor organization offers a lot of free resources that can be used to learn Meteor. Another advantage of working with Meteor is that web applications are real time by default. Updating an application made with Meteor, can be done without temporarily being offline. Finally Meteor offers a free hosting service that allows developers to deploy their application in a minute with only one command [2].

The apparent strengths of the Meteor platform are quite closely intertwined with its weaknesses. Developing an application with Meteor for the first time may satisfy quickly, but for developers with little experience, this can be a pitfall, as it is easy to ignore what is really happening in the background. We feel it is important to take into account how the underlying technologies are accessed by Meteor, and how an application uses Meteors intermediate layers and functionalities. More experienced developers can also experience the easiness of Meteor as a drawback. It may be hard to get used to the frameworks as a new paradigm and it also could feel like there is 'magic happening'. When it is desired to host the app on a private or third-party server, it should match the Meteor server requirements. The server hosting of Meteor can be slow and the application is only running when at least one user is online [2].

While other JavaScript web-development frameworks exist, they do not completely cover the scope of web application development as it is covered by the Meteor platform [8].

[3] At the time of writing Meteor only supports the MongoDB database, however support for other storage systems such as Redis (open source data structure server. http://redis.io) , are in development.

We will briefly discuss some alternative technologies and note where they overlap with functionality that is provided by Meteor. A popular combination of technologies that offers similar functionality to Meteor is the combination of ExpressJS (server-side) and AngularJS (client-side) on top of Node.JS and MongoDB [34]. This architecture is different from Meteor. ExpressJS allows for the creation of REST-ful endpoints[4] on a Node.JS based server while AngularJS takes care of interfacing with these endpoints on the client side, providing an application interface that updates in real-time in a similar manner as Meteor. Another framework that promises similar functionality, is Derby. Though Derby's uptake by the web-development community has been rather poor.

There are situations in which another framework is preferred, such as when one wants to develop a web*site*, instead of an application, which Meteor is not meant for. Meteor would be ill suited, if one is developing a server-application with a RESTful interface, since Meteor presumes a client-server architecture. A developer would be better served in that situation by the combination of Node, MongoDB and ExpressJS [8].

4. TYPICAL APPLICATIONS

Since Meteor is still in its early days, not a lot of large scale projects can be found yet. However, some interesting applications have been built. We found two typical applications that Meteor is used for: *online collaboration* and *real-time data streams.*

4.1 Online collaboration

With the first application, *online collaboration*, is meant that the platform is appropriate to develop tools that enables multiple users to work together on one project via their own devices. Users can directly see each others actions with a minimum delay and without having to refresh the page [20].

An example of this application is 'Pintask'. It can be used by project teams to manage tasks [26]. These tasks, written on digital cards can be divided into custom subcategories, assigned to team members and during the project be structured according to their current status. Team members know where everyone is currently working on and the real-time aspect avoids people trying to execute the same actions or overwriting each others changes. Another example of a collaborative web tool is 'Madeye', a web editor that enables developers to write code together simultaneous in the same document while (video) chatting simultaneously in google hangouts. It can be seen as a combination of a chatroom and a text editor [18].

4.2 Real-time data streams

Showing real-time data streams as visualizations can be a valuable journalism tool besides the more conventional ways of providing news information [35].

Livebus is one of those real-time data visualizations [14]. On a map, it shows the current location of all the vehicles part of Honolulu's Bus System. It uses Meteor combined with the JavaScript library Data Driven Documents (D3, specific developed to create data visualizations) and the Google General Transit Feed Specification (GFTS) of TheBus [16][11].

5. SURPRISING APPLICATIONS

Besides typical applications that provides user with real-time collaboration or real-time data streams, we found some applications that go beyond the intended use of Meteor to develop real-time applications.

The smart coffee machine *SpinnCoffee* is currently being developed. It comes with a smartphone application and besides brewing coffee, also takes care of ordering new coffee beans when the machine is running out [30]. This is an interesting example because it shows Meteor is also used outside the general use of sending and receiving textual data in real-time. The system communicates with the physical environment by reading sensors.

ThingStudio, a free tool for hobbyist developers, helps to create applications that control devices across the so called 'Internet of Things'[5] [32]. It is currently easy and affordable to create networked systems with technologies like the Arduino circuit board[6] or Raspberry Pi[7]. Creating a user friendly interface that lets you controls these systems was difficult [6]. ThingStudio makes it easy to add interface elements to your applications that in real time are able to communicate with your system. Another advantage is that communication is handled locally, so no data is sent or can be sent to any third party [13] .

6. GETTING STARTED

This is a guide to install Meteor, build a first Meteor application and experience how the reactive part is handled. Basic knowledge of html, css and JavaScript is required, just as familiarity on how to navigate and execute basic actions in command line. We presume usage of a Linux or OSX based computer. Consult the Meteor website for more information about installing on Windows. The given code is based on several examples of the book *Discover Meteor.*

6.1 Installing Meteor

1. Open the terminal to Install Meteor and type: `curl https://install.Meteor.com | sh`

Now Meteor has been installed in your main directory. To see the Meteor files, you should enable your hidden files. Meteor can be seen as an in-between layer, on which every application is based on.

6.2 Creating a Meteor application

In this part you are going to make a simple chat application.

2. Navigate to a desired directory and create a Meteor chat application in there by giving the following command: `Meteor create chat`

This instructs the terminal to use Meteor and you tell Meteor you would like to create a new application named 'chat'. Meteor then created a subdirectory that contains a html, css and JavaScript file. These files can contain all of

[4] With REST, other entities besides browsers, such as mobile phones can communicate with a server over HTTP.

[5] Internet Of things: a term used for a network of computers and physical objects that are connected to the internet.

[6] Open-source electronics platform for interactive projects. http://www.arduino.cc

[7] Mini-computer that is able to interact with the outside world. Http://www.raspberrypi.org

your code. For a more structured overview or to have more control over which code runs on the server, the client or both of them, code can be structured in special directories like 'client' and 'server'. In the directory, a hidden Meteor file is automatically placed that is only there to serve the Meteor package to your application.

3. Navigate to the chat directory and run the application by typing: `cd chat Meteor`

Navigate to http://localhost:3000/ in your browser to run the application. The result shows the default content of the Meteor application and looks like figure 2.

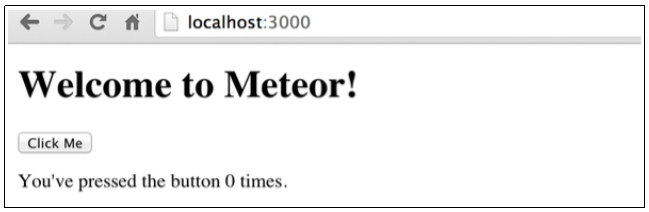

Figure 2. Screenshot of a just created Meteor application.

4. Open a code editor (for example Brackets or Sublime Text 2[8]) navigate to your created directory and open the chat.html, chat.js and chat.css files.

The html and css files are automatically running on the client side. Within the JavaScript file, two boolean variables determine if certain blocks of code run on the client or the server side [2]. See line 1: `( if (Meteor.isClient) { )` and line 19 `( if (Meteor.isServer) { )` in the JavaScript file.

4. Close the css file, since that one handles markup.
5. To create the chat application, replace all the content of the html and JavaScript files by respectively the html code and the JavaScript on page 6 in the attachment of this paper or download the code via github[9]. After saving these, the application is running at localhost:3000. If you open multiple browser windows, you can see the application updates everywhere directly when you submit a text in one of the windows.

6.3 Showing the data in html in real-time

We will explain how this real time aspect is achieved by analyzing the most essential parts of the code.

Look for the <form> tag in the html code and and inspect its content (line 11-15). Now open the JavaScript file. First, a database named 'chats' is created (called 'collections' in Meteor) [2]. line 5: `chats = new Mongo.Collection ("chats");`

When a user submits the values of the html form by clicking 'submit', they are inserted in the chats collection. The text values named 'text' and 'name' are first assigned to the variables '`nameField`' and '`textField`' on line 25 and 26. After that, we insert a new object in the database by assigning the content of those variables to an appropriate datapoint name (in this case 'name' and 'text') on line 29-33. Now the collection contains an object with the values the user submitted.

To show the content of our collection to the user, we make use of templates in the html file. Templates are small pieces of html containing dynamic content that can be altered with JavaScript [2]. On line 30-33, a template called 'chat' is created that can contains the 'name' and 'text' data from our collection. We iterate over all the objects returned by the chats function in the JavaScript file on line 15-18 with `{{#each chats}}`. For every object we find, we display the template 'chat' with the content of the current object by calling the template with `{{> chat}}`. Important here is that this function is executed every time something changed in the collection. Hereby, our application updates in real-time.

A copy of the collection is running on each client. The real Mongo database runs on the server side, so if we would like to change the data, it happens there. In case the 'delete' button is pressed on the client side, the function `removeAllchats` is called on the server side (line 41-44). And in the part if (`Meteor.isServer`) all chat objects are removed from our collection (line 58).

We have seen that client- and server-side behaviour of JavaScript, together with the main database on the server and the sub database on each client, enable applications to update in real time. To see more tutorials and information, we advise to consult the Meteor website[10] and Atmosphere.js[11] for an overview of all additional packages.

7. FINAL THOUGHTS

Meteor is an accessible open source framework which is relatively easy to learn and use. The examination for this paper showed us, that Meteor provides extensive documentation, that is easy to read and to understand. A large community supports Meteor and besides the Meteor Group itself, other parties offer a multitude of resources. The Meteor hosting makes it easy to quickly share applications. There are a lot of ready made functionalities in so called 'packages' [1] that can be seen as building blocks used to quickly create intended applications.

This web-technology may be ideal for projects with a real-time aspect, however we would like to note once more that the steps taken by the Meteor Group to make Meteor accessible, hide specific aspects of the operation of the Meteor framework. While this lowers the barrier of entry to newcomers in web application development, we feel that, with the functionality that Meteor provides behind the scenes developing applications with Meteor can seem a like magic.

We think it is a strong framework, but also believe Meteor is not in all cases appropriate as replacement for other sets of technologies at the moment. Furthermore, with the project only being at its 1.0 state as of november 2014, we still need to see large scale projects realized with Meteor to determine its performance.

[8] Brackets text editor: http://brackets.io. Sublime Text 2: http://www.sublimetext.com.

[9] Github location of code example in section 6.3: https://github.com/Jasper2-0/WebTechnologies

[10] Meteor Tutorial and further reading: https://www.meteor.com/try/12

[11] Resource for Meteor packages: https://atmospherejs.com

REFERENCES

1. Atmosphere.js. Explore Meteor packages. Atmosphere.js. https://atmospherejs.com
2. Coleman, T. and Greiff, S. Discover Meteor - Building Real-time JavaScript Web Apps. (2015). Section: deploying on your own server.
3. D3. About D3. http://d3js.org. Retrieved: June 8 2015.
4. DeBergalis, M. Skybreak Is Now Meteor. Meteor (January 2012). Retrieved: June 8 2015. http://info.Meteor.com/blog/skybreak-is-now-Meteor.
5. DeBergalis, M. Meteor 1.0. Meteor (October 2014). Retrieved: June 8 2015. http://info.Meteor.com/blog/Meteor-1-0.
6. Finley, K. The Internet of Anything: How to Build Mobile Apps for Your DIY Connected Gadgets. Wired (2015). Retrieved: June 9 2015. http://www.wired.com/2015/04/thingstudio/
7. Fischer, D. Why Meteor. Dan Dascalescu's Wiki (2015). Retrieved: June 10 2015. http://wiki.dandascalescu.com/essays/why_meteor
8. Fischer, D. Meteor vs the MEAN stack. Dan Dascalescu's Wiki (2015). Retrieved: June 10 2015. http://wiki.dandascalescu.com/essays/meteor_js_vs_the_mean_stack
9. Geipel, M.M. Dynamics of communities and code in open source software. Eidgenössische Technische Hochschule (ETH Zürich) 18480 (2009).
10. Google. Google docs. https://www.google.com/docs/about/.
11. Google. GFTS. https://developers.google.com/transit/gtfs-real-time/.
12. Gravelle, R. Introduction to Server-side JavaScript. Webreference. http://www.webreference.com/programming/JavaScript/rg37/index.html. Retrieved: June 8 2015.
13. Karliner, M. Talk of Michael Karliner about Meteor and the Internet of Things during Devshop London. Meteor (January 2015). Retrieved: June 9 2015. https://www.youtube.com/watch?v=MdeaFLrvqtc
14. Kyle. Livebus. Pas de chocolat, LLC. http://www.pasdechocolat.com/2013/07/20/livebus-with-Meteor-and-d3/
15. Maestre, S. What Is This Real-time Web You Speak of? VentureBeat (October 2013). Retrieved: June 8 2015. http://venturebeat.com/2013/10/24/what-is-this-real-time-web-you-speak-of/
16. Manricks, G. Instant Meteor JavaScript Framework Starter: Enjoy Creating a Multi-page Site, Using the Exciting New Meteor Framework! Section: So, What Is Meteor? Packtpub (August 2013).
17. Martin, N. Skybreak. GitHub (January 2012). Retrieved: June 8 2015. https://github.com/skybreak/skybreak.
18. Madeye. https://madeye.io.
19. Meteor. File Structure. Meteor Documentation. Retrieved: June 10 2015. http://docs.meteor.com/#/basic/underscore.
20. Meteor. The Meteor Mission. Meteor. Retrieved: June 8 2015. https://www.Meteor.com/about
21. Meteor. Transparent Reactive Programming. Meteor Manual. Retrieved: June 8 2015. https://manual.Meteor.com/.
22. Meteor. DDP. Meteor. Retrieved: June 10 2015. https://www.meteor.com/ddp
23. Meteor. Latency Compensation. Meteor. Retrieved: June 10 2015. https://www.meteor.com/full-stack-db-drivers
24. Meteor. Minidatabases. Meteor. Retrieved: June 8 2015. https://www.meteor.com/mini-databases
25. Meteor. Session. Meteor Documentation. Retrieved: June 8 2015. http://docs.Meteor.com/#/basic/Session-get
26. Pintask. https://pintask.me.
27. Schmidt, G. Meteor's New $11.2 Million Development Budget. Meteor (July 2012). Retrieved: June 8 2015. http://info.Meteor.com/blog/Meteors-new-112-million-development-budget.
28. Shankland, S. Need for Speed Spurs Opera JavaScript Overhaul. CNET (February 2009). Retrieved: June 8 2015. http://www.cnet.com/news/need-for-speed-spurs-opera-JavaScript-overhaul/.
29. Solution Stack. Wikipedia. Wikimedia Foundation (May 2015). Retrieved: June 8 2015. http://en.wikipedia.org/wiki/Solution_stack
30. Spin Coffee. About Spin Coffee. Spin Coffee (2014). http://spinn.coffee/#smartcoffee.
31. Thijssen, J. LAMP-stack? Forget It! It's a LAMPGMVN MCSTRAH-stack Now…. A day in the Life of (October 2011). Retrieved: June 8 2015. https://www.adayinthelifeof.nl/2011/10/26/lamp-stack-forget-it-its-a-lampgmvnmcstrah-stack-now/
32. ThingStudio. Real Time User Interfaces for the Internet of Things. ThingStudio (June 2015). Retrieved: June 8 2015. http://www.thingstud.io/.
33. Tomaszewski, B. Geographic Information Systems (GIS) for Disaster Management. Taylor & Francis Group, CRC Press (2014), 105-107.
34. Mean. Io. The friendly and fun JavaScript full stack for your next web application. Mean.io. http://mean.io/#!/
35. Yau, N. Visualize This. The Flowing Data Guide to Design, Visualization, and Statistics. (2011).

ATTACHMENT 1: HTML AND JAVASCRIPT CODE OF A CHAT APPLICATION

JavaScript code to be used in section 6. Getting Started.

Replace the full content of the html file you created by the following code:

```
<!-- THIS IS THE HTML FILE, EVERYTHING HERE RUNS ON THE CLIENT SIDE -->
<head>
 <title>chatbox</title>
</head>
<!-- main body of the app. -- >
<body>
 <h1>chat it!</h1>

 <!-- <form> = html tag to make a form -->
 <form class="chatform">
   <input type="text" name="name" placeholder="Name"/>
   <input type="text" name="text" placeholder="What do you have to say?"/>
   <button type="submit">Submit</button>
 </form>

<!-- with each chat from the database render the chat-data with the chat template  -->
 {{#each chats}} <!-- her we iterate over the items returned by the chats function and display inside the block for each one -->
   {{> chat}} <!-- So here is the chat template in the html -->
 {{/each}}

 <button class="delete">Delete all</button>
</body>

<!-- template tags are specific tags that meteor uses to know where to put data -->
<!--This is a template for each chat, we can use it in our html and do that on line 19 -->
<template name="chat">
 <h2>{{name}}</h2>
 <p>{{text}}</p>
</template>
```

Javasript code to be used in section 6. Getting Started.

Replace the full content of the JavaScript file you created by the following code:

```
//THIS IS THE JAVASCRIPT FILE
// Make a new meteor collection that contains all chats.
chats = new Mongo.Collection("chats");

/*
*   CLIENT SIDE
*/
if (Meteor.isClient) {
```

```
 /*
 * This is a helper function for the 'body' template, see chatbox.html.
 */
 Template.body.helpers({   //With a helper, we can display data inside the template
   chats: function () {   //So here we define a helper called 'chats' for the template
     return chats.find({}, {sort: {createdAt: -1}});  //return all the chats we can find (and sort them)
   }
});

// This part handles the event of the body template

 Template.body.events({
   'submit .chatform': function (event) {   // When the chatform receives a submit event..
     var nameField = event.target.name.value;     // Put the value of the name input into a variable
     var textField = event.target.text.value;     // Put the value of the textinput into a variable

     // Insert a new OBJECT into the database, with name, text, and created at properties.
     chats.insert({
       name: nameField,
       text: textField,
       createdAt: new Date()         // current time
     });
     event.target.text.value = "";   // Clear form
     return false;                    // Prevent default form submit
   }
         // When the user clicks the delete button, remove all chats in the database on the server.
   'click .delete': function (event) {
     Meteor.call('removeAllchats')  // Now we call a function on the server side!!
   }
 });
}

/*
*   SERVER SIDE
*/
if (Meteor.isServer) {
 Meteor.startup(function () {        // Code to run on server at startup
// At server startup remove all previously created chats.
   return Meteor.methods({
     removeAllchats: function() {
 return chats.remove({});       // Empty out the chats collection
     }
   });
 });
}
```