# Rapid Development of Web Applications for the Access Economy: A Tutorial in the Meteor Web Development Framework

Kyle King—ENGL393

July 10, 2015

## Purpose

The purpose of this paper is to encourage the use of the Meteor JS web development framework. In particular, to examine the potential application of this technology for sharing economies.

## Audience

My audience are front-end web developers and tech-company entrepreneurs with limited back-end or full stack web development experience.

Kyle King

501 Yarmouth Road
Towson, MD 21286

July 10, 2015

Mr. Allen W. Milton
Chief Genius, Atlas Bikes
592 Mongoose Hall
Wilmington, NC 20034

Dear Mr. Milton,

Enclosed is a tutorial for the development of real time web applications based on the Meteor development framework.

After our insightful discussion regarding the possibilities for the access economy, I was inspired to demonstrate the potential for the use of Meteor in an entrepreneurial setting. I hope you find the enclosed document insightful and I wish you the best of luck with your growing bikeshare startup.

After reviewing multiple frameworks and researching the access economy, it is clear that new web technologies should leverage the Meteor framework for its built-in data protocols, package system, and the general speed of which a prototype can be constructed. Meteor has the potential to save new companies tens of thousands of dollars that would otherwise be spent on hiring a team of developers to accomplish the same task.

Regards,
Kyle King

**Abstract**

The sharing economy enables consumers to directly exchange goods and services with other consumers for maximum resource utilization, economic growth, and reduced waste. The sharing, or access economy, relies heavily on technology to connect consumers. For the first time, small teams of developers can rapidly develop applications for sharing economy companies. Web development frameworks employ tested and proven code that can be used across a wide-range of applications for fast, cheap development. Frameworks address the technological issues of access-economy development. One particular framework, Meteor, is highly applicable to the sharing economy. To illustrate its use, I will outline the structure of a bike-sharing application for a potential company. After reading this tutorial, developers and entrepreneurs will be able to benefit from the improvements that the Meteor framework provides.

**Rapid Development of Web Applications for the Access Economy: A Tutorial in the Meteor Web Development Framework**

Companies like Uber, Ebay, and Airbnb define the next generation of business by enabling consumers to exchange goods and services with other consumers for maximum resource utilization, economic growth, and reduced waste (Fremstad, 2014). However, the sharing economy faces "profound legal, political and ethical questions" (Rauch & Schleicher, 2015), alongside difficult technological barriers (Sonders & Braun, 2014). The sharing or access economy heavily relies on technology to connect consumers with services. The technology is often in the form of a website or mobile application that requires a secure, multi-platform marketplace that can be rapidly deployed and updated with user feedback (Cardoso, et al., 2007). Developers must receive feedback from users for product validation, prioritization of features, and understanding use cases. Web development frameworks accelerate the process of user feedback and allow for great reductions in cost, while producing a better product. Web development frameworks offer unique and important benefits such as data handling, security, and multi-device capabilities (Meteor, n.d.). Frameworks address the technological issues of slow access-economy application development. One particular framework, Meteor, is highly applicable to the sharing economy. To illustrate the benefits from development with the Meteor platform, I will demonstrate the minimal code necessary to start a bike-sharing application for a potential company. After reading this tutorial, developers and entrepreneurs will be able to

benefit from the improvements that the Meteor framework provides.

**The Access Economy**

The access economy model can be applied to a wide variety of companies that "[offer] customers the benefits of disintermediation-tourist accommodations without hotel chains, local transportation without taxi companies, peer-to-peer lending without banks, retail storefronts with brokers and leases" (Denning, 2014). A wide of array of diverse companies can be considered part of the sharing economy such as Craigslist, a site offering direct peer-to-peer transactions ("Craigslist | About > Factsheet," n.d.), or Lyft, a hybrid company that connects available drivers with riders. Other companies improve resource distribution through their extensive open source software, such as the Open Food Foundation that connects farmers to local consumers. The foundation's software provides newfound consumer transparency, which motivates ethical farming practices (Gaziulusoy & Twomey, 2014). Companies such as Elance and oDesk promote employee independence by enabling users to contract their services to other users inside a professional marketplace (Owyang, et al., 2013). These examples of the sharing economy demonstrate the ways in which industries have revolutionized to meet the demands of a diverse and growing marketplace.

These companies offer direct benefits to the user base, the economy, and the environment. Due to the higher level of resource utilizations, consumers benefit from an increased standard of living. For

example, transportation, often a large expense for employment, is optimized by SideCar, a carpool service that allows riders with varying start and end locations to carpool together (Andersson, et al., 2013). Services like Airbnb provide economic benefits to the local economy. Based on its own survey, Airbnb generated over $56 million in San Francisco alone from added tourism revenue (Airbnb, n.d.).

A new access economy company must overcome incredible odds to reach its full potential, since approximately 90% of startups fail (Sonders & Braun, 2014). For example, Spinlister started as a bikeshare business where users could rent their personal bikes to strangers for a fee. Seeking to grow their venture investments, the original founders sought to expand the service to every resource a user may own, such as sports equipment and outdoor tools. However, the founders struggled to recruit new users due to the sudden shift in vision and the lack of a mobile application. Consequently, the founders sold the company to one of their investors, Marcelo Loureiro. Immediately, Loureiro returned the company to bike rentals only and built a new mobile application, which accelerated user growth to “around 10,000 users and 2,000 bikes” (Gaziulusoy & Twomey, 2014; Schwartz, 2013). As seen in Spinlister and many other startups, a mobile application is critical for user growth and retention (Cardoso et al., 2007).

**The Underlying Technology**

Every developer faces similar challenges to address major

revisions, modify permissions, transmit data, and support multiple platforms. Traditionally, for an application to be available on multiple devices, the developer would create the same application in multiple coding languages. Each time an application was updated, a developer would repeat the changes across each platform and introduce platform-specific modifications (Cardoso et al., 2007). Since a company needs to support a number of devices, redundant codebases are often too expensive for startups to maintain.

To transcend these problems, future companies can leverage code written and tested by a large user base. A startup can reduce development time and cost by incorporating general code rather than custom-built code. There are several emerging web technology frameworks that simplify the incorporation and sharing of general code. Additionally, some frameworks can support multiple devices, store and exchange data, and ensure user security among other features.

A short list of some of the main frameworks in use include Ruby on Rails, MEAN.io (MongoDB, Ember, Angular, and Node), and Meteor. Ruby on rails is a complex framework that offers a high level of control over development. The software has been around for 7-8 years and has matured with a large number of blog posts, answered support questions, and well documented tutorials (Prochazka, 2007). MEAN.io is a newer framework that has risen in popularity and is around 5-6 years old. The framework is a collection of smaller frameworks that are added to a core Node.js application. Node supports and runs each component of the app

and is the most minimal part of a web application. On top of Node, the first component of MEAN is MongoDB, a highly flexible database for the application back-end. The second component, Express.js, is a framework to manipulate the user-interface. The third component is Angular.js which manipulates the web page content, such as text, links, or images. Along with the MEAN stack, Node offers a very extensive package framework and is not necessarily limited to a MEAN-only web application (Linnovate, 2015). However, Meteor is the newest framework in the group. Meteor is built on Node.js, but includes significantly more standard features. Meteor has its own package system and has quickly absorbed many of the Node packages. The software is straightforward to learn and uses JavaScript for every logic-based action (Strack, 2012).

These frameworks change the way development occurs. For multidisciplinary teams of designers, developers, and engineers, development is faster and more affordable. More time can be allocated to consider user feedback and make software improvements. While each framework has certain benefits and drawbacks, Meteor remains a forerunner for development. With a growing user community and an officially supported package management system, Meteor maintains optimal client to server communications and easy extensibility. The following sections will demonstrate the development of a stationless bikeshare application including a basic interface, data handling, and package inclusion.

**Configuring Meteor for a Hello World Example**

Meteor is installed onto a computer in a process similar to Node.js and other development frameworks. However, Meteor can be installed in one command and is significantly easier to install than comparable frameworks. For a Windows computer, the official installer is downloaded from https://www.meteor.com/install. Likewise, terminal is used to download Meteor on a Mac or Linux machine(Meteor Development Group, n.d.). The command is as follows:

```
curl https://install.Meteor.com/ | sh
```

Once downloaded, the first Meteor application can be created. Using Meteor's included build tools, enter, ```meteor create BikeShare``` into the command line. This command creates an application called BikeShare. To initiate the newly created application, navigate to the directory, ```cd BikeShare``` and run the application with ```meteor```.

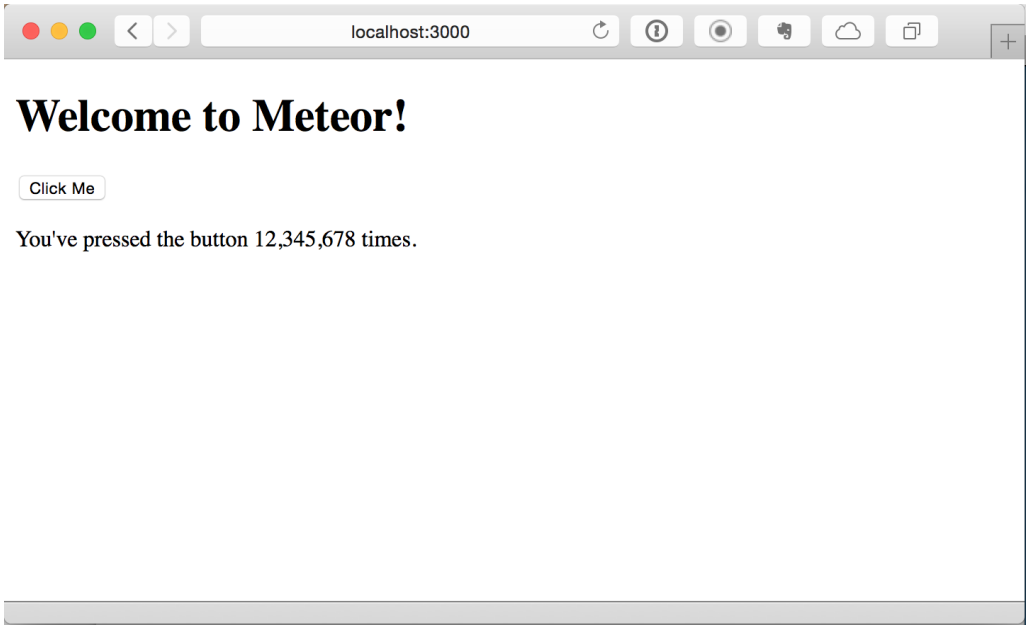



*Illustration 1: Meteor example welcome page*

After running these three commands, the browser will display the Meteor application. The application runs a “Hello World” example of the most basic code to prove proper initiation of the software. The Hello World example displays a simple user interface and reactively stores data as seen in illustration 1. Each time the button “Click Me” is clicked, the current count of button presses is saved and displayed instantly. To introduce the core meteor concepts of reactivity, data storage, and rapid prototyping, the initial code will be analyzed.

To develop a Meteor application, a text editor is required to view and edit application files. Although there are multiple text editor options, Atom and Sublime are recommended due to their simplicity and ease of use. Both text editors support a wide array of computer operating systems and offer extensive shortcuts and tools to simplify writing code (Atom, n.d.; Sublime, n.d.). Open the folder “BikeShare” in your selected text editor. Within the “BikeShare” folder, there will be a second folder and three files. The folder, “.meteor,” holds all of the Meteor core functionality and remains largely untouched throughout the development process. The files “BikeShare.css”, “BikeShare.HTML”, and “BikeShare.js” create the simple Hello World application shown in Illustration 1. The Cascading Style Sheets (CSS) file controls the page’s visual settings, such as color and font. The HTML file holds the text content of the page, such as headers, buttons, and paragraphs. The JavaScript (JS) file counts the number of button clicks and then stores the number in a special variable, called a session variable.

**Data in Meteor**

Meteor has several methods for storing information, namely through the use of a session variable. These variables are only available while the user is logged in and once the user logs out or leaves the app, the variable is reset. Session variables are very useful when working with complex views or short term data. In the bikeshare example, variables are used to store the number of button clicks since the window was last opened. Inside the BikeShare.js file, the session variable is created by the following code snippet.

```
// counter starts at 0
Session.setDefault('counter', 0);

Template.hello.events({
    'click button': function () {
        // increment the counter when button is click
        Session.set('counter', Session.get('counter')
    }
});
```

The core component of a Meteor page is a template that encases the HTML content within its designated section. Without the templates, the HTML content is rendered on every page until routing control is added. This template structure reduces the amount of HTML needed and triggers JS functions based on template events. To select which template to render in a given HTML block, the syntax, ```{{> hello}}``` is used (Meteor, n.d.). In the Hello World example code, this is used to include the

'hello' template in the body element.

```
Template.hello.helpers({
    counter: function () {
        return Session.get('counter');
    }
});
```

To render the data for the user, Meteor uses Spacebars, a tool for actively manipulating the HTML document. When given a value from a JS file with ```return <value here>```, Spacebars will render the JS value in HTML. In this case, the session variable counter is created in BikeShare.js and shown as a component of the HTML through this syntax: ```{{counter}}```. This looks very similar to the template inclusion syntax and serves a similar function. Anytime the session variable-counter is updated, the HTML content is automatically re-rendered and updated by Meteor. The Meteor development framework easily prototypes tools that reactively update and display user data.

Session variables are only effective for short-term data storage. For long-term storage, Meteor comes equipped with Mongo DataBase, a back-end framework to store information. Mongo stores information in a relational format, similar to a binder with many documents inside it. Each document contains fields of information that can be accessed through a request query. As Mongo is standard with Meteor, an application can quickly store and share information. To demonstrate the use of MongoDB, the code snippet below is meant to replace the code inside BikeShare.js.

```
DailyBikeData = new Mongo.Collection('dailyBikeData')

if (Meteor.isServer) {
    // Create Bike GPS coordinates
    function randGPS() {
        // Set bounding constraints
        var bottomLng = -76.936569; var topLng = -76.
        var leftLat = 38.994052; var rightLat = 38.98
        var ConvFactor = 1000000; // To convert to ar

        // Create random coordinates
        function getRandomArbitrary(min, max) {
            return Math.random() * (max - min) + min;
        }
        var randCoordinates = {
            lat: (getRandomArbitrary(leftLat*ConvFact
                / ConvFactor),
            lng: (getRandomArbitrary(bottomLng*ConvFa
                / ConvFactor)
        };
        return randCoordinates;
    };

    // Insert data into the database
    if (DailyBikeData.find().count() === 0) {
        var i = 1;
        while (i <= 50) {
            var randCoordinates = randGPS();
            DailyBikeData.insert({
                Bike: i,
                Tag: Math.round(0.65 * Math.random())
                Positions: {
                    lat: randCoordinates.lat,
```

```
                    lng: randCoordinates.lng
                }
            });
            i++;
        }
        console.log('Created DailyBikeData data schem
    }
}
```

The code above first creates a collection of documents called DailyBikeData. On the server the database is instantiated with a for loop and a JavaScript object. The math at the beginning of the function allows the random GPS coordinates to be generated within a bounding box. This geographic box is especially useful in designing the application with respect to a geographically-relevant data set. When correctly run, a short string should be printed in the command line tool that says, 'Created DailyBikeData data schema.' This means that the data is now available on the client, but to see it, something needs to be built on the client.

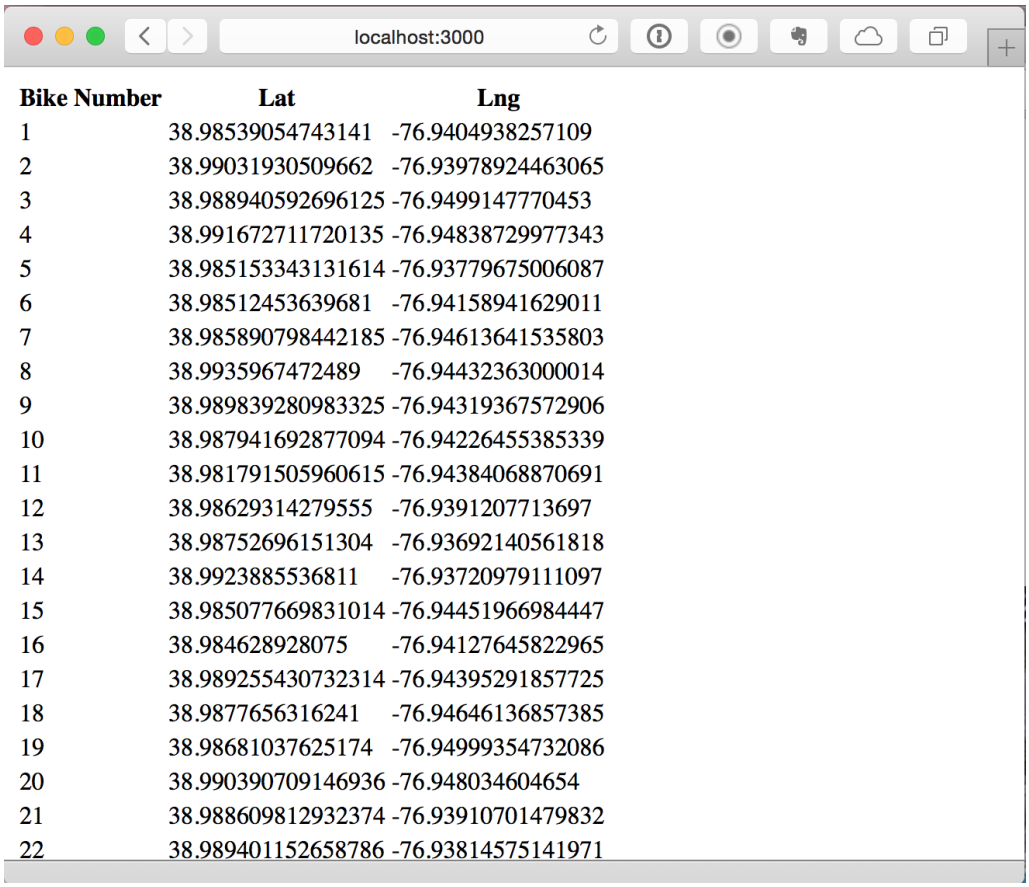
```
<body>
    <h1>Welcome to Meteor!</h1>

    {{> hello}}
    {{> table}}
</body>

<template name="table">
    <table>
        <thead>
        <tr>
```

```
            <th>Bike Number</th>
            <th>Lat</th>
            <th>Lng</th>
        </tr>
        </thead>

        <tbody>
            {{#each bikeInfo}}
        <tr>
                <td>{{Bike}}</td>
                <td>{{Positions.lat}}</td>
                <td>{{Positions.lng}}</td>
         </tr>
             {{/each}}
        </tbody>

    </table>
</template>
```

Use this above snippet in place of the BikeShare.HTML file. Once the HTML document is updated, the JavaScript file needs the short helper function to populate the HTML table. The combined code changes should load in your browser as shown in Illustration 2.

```
if (Meteor.isClient) {
    // Create Spacebars helper to load bike data intc
    Template.table.helpers({
        bikeInfo: function () {
            return DailyBikeData.find();
        }
    });
```

```
}
```




| Bike Number | Lat | Lng |
|---|---|---|
| 1 | 38.98539054743141 | -76.9404938257109 |
| 2 | 38.99031930509662 | -76.93978924463065 |
| 3 | 38.988940592696125 | -76.9499147770453 |
| 4 | 38.991672711720135 | -76.94838729977343 |
| 5 | 38.985153343131614 | -76.93779675006087 |
| 6 | 38.98512453639681 | -76.94158941629011 |
| 7 | 38.985890798442185 | -76.94613641535803 |
| 8 | 38.9935967472489 | -76.94432363000014 |
| 9 | 38.989839280983325 | -76.94319367572906 |
| 10 | 38.987941692877094 | -76.94226455385339 |
| 11 | 38.981791505960615 | -76.94384068870691 |
| 12 | 38.98629314279555 | -76.9391207713697 |
| 13 | 38.98752696151304 | -76.93692140561818 |
| 14 | 38.9923885536811 | -76.93720979111097 |
| 15 | 38.985077669831014 | -76.94451966984447 |
| 16 | 38.984628928075 | -76.94127645822965 |
| 17 | 38.989255430732314 | -76.94395291857725 |
| 18 | 38.9877656316241 | -76.94646136857385 |
| 19 | 38.98681037625174 | -76.94999354732086 |
| 20 | 38.990390709146936 | -76.948034604654 |
| 21 | 38.988609812932374 | -76.93910701479832 |
| 22 | 38.989401152658786 | -76.93814575141971 |

*Illustration 2: Table of database values*

**Data Visualization**

To visualize the data in the browser window, the JavaScript helper function fetches all available bike data from the server and makes it available to the HTML file. Through the intuitive Spacebars syntax, Meteor iteratively fills each field of the table with the fields of the document object. In this case, the table renders the array of bike positions and information in a long list of locations in longitude and latitude format.

With this short code, bike positions are randomly generated and

then stored in a database. In a published application, changes made on one computer would be visible to every user. However, displaying geographical locations in a table format is ineffective. For the data to be displayed in a meaningful way, the bike positions should be visible on a map that considers the user's current location. Leaflet, an open-source mapping software similar to Google Maps, is a readily available package that makes this process possible. Leaflet can be added by entering ```meteor add bevanhunt:leaflet``` in the command line. After a short download, the entire Leaflet library will be available. To select where in the application the map will be rendered and with what information. To see the maps, there are a few changes that need to be made to create the Leaflet map following their API.

```
<body>
    <h1>Welcome to Meteor!</h1>
    {{> map}}
    {{> table}}
</body>

<template name="map">
    <div id="BikeMap" class="map-style"></div>
</template>


.map-style {
    position: absolute;
    width: 100%;
    height: 500px;
    z-index: 1;
```

```
}


if (Meteor.isClient) {
    Template.map.rendered = function() {
        // Create the Leaflet Map
        L.Icon.Default.imagePath = '/packages/mrt_lea
        if (Meteor.isClient) {
            var map = new L.Map('BikeMap', {
                center: new L.LatLng(38.987701, -76.9
            });
            var url = 'http://{s}.{base}.maps.cit.api
                'maptile/2.1/maptile/{mapID}/hybrid.d
                '256/png8?app_id={app_id}&app_code={a
            var HERE_hybridDayMobile = L.tileLayer(ur
                attribution: 'Map © 1987-2014 '+
                    'HERE',
                subdomains: '1234',
                mapID: 'newest',
                app_id: 'JIX0epTdHneK1hQlqfkr',
                app_code: 'PchnUPPBcZ5VAuHmovac8g',
                base: 'aerial'
            }).addTo(map);
        }

        // Receive data from server and display on ma
        Meteor.autorun(function() {
            var bikesData = DailyBikeData.find().fetc
            bikesData.forEach(function(bike) {
                var marker = L.marker([
                        bike.Positions.lat,
                        bike.Positions.lng
                    ]).addTo(map);
            });
```

```
        });

        // Zoom to user location
        map.locate({ setView: true })
    };
}
```

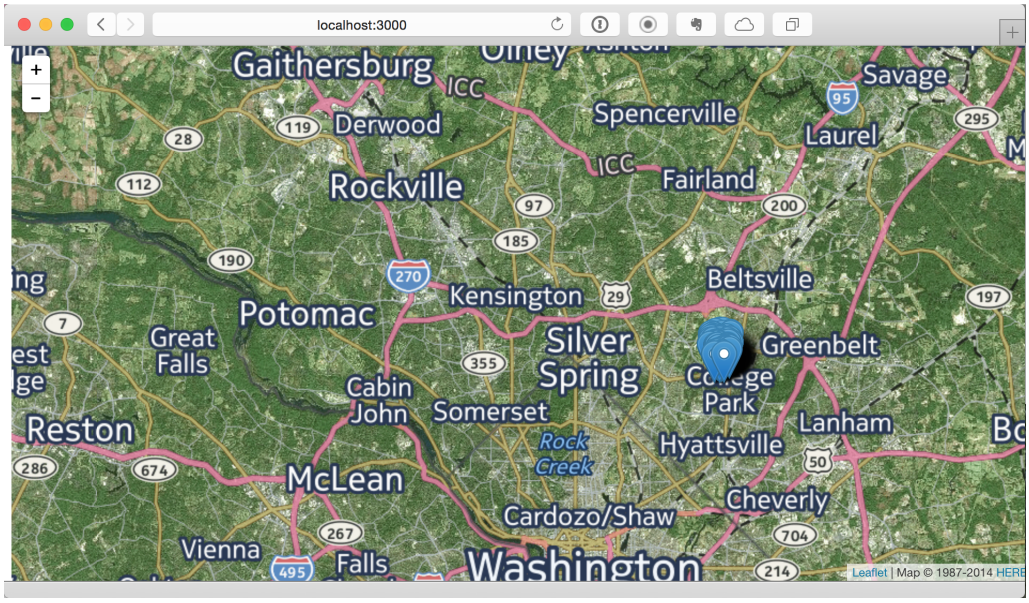



*Illustration 3: Bike positions plotted on a Leaflet map*

The first file creates a template called 'map' that contains an element with the tag, 'BikeMap'. This tells Leaflet where to initiate the map. In addition, the short CSS declarations determine the size of the map and prevent the creation of an invisible map with a height of zero pixels.. The JS file determines the map's configuration, such as map visual style, user credentials, markers, and other information. Through the Leaflet API, the bike-friendly map from HERE maps is displayed in the user window. The map should appear as shown in Illustration 3. Once the browser retrieves all available bike locations with, ```var bikesData = DailyBikeData.find().fetch();```, the data points are displayed as markers

on the Leaflet map from within the for loop. Using Leaflet's simple API, the user location can then be shown by changing the map's center position. With a small tweak to the client code, only bikes tagged “Available” will be shown in the map. For example, if```var bikesData = DailyBikeData.find().fetch();``` is replaced with, ```var bikesData = DailyBikeData.find({tag: 'Available'}).fetch();```, only available bikes will be returned.

This simple demonstration rapidly prototypes a bikeshare application that can store inventory locations and sort through the data to display only available bikes on a Leaflet map. The code included in this tutorial can be extended with the addition of Meteor packages, as demonstrated with the leaflet package. Recommended packages such as twbs:bootstrap and useraccounts:bootstrap would aid in the development of visual components and login functionality respectively. With the Meteor web development framework, a developer can rapidly and cost-effectively develop large applications.

**Works Cited**


Airbnb. (n.d.). Airbnb Economic Impact. Retrieved June 18, 2015, from http://blog.airbnb.com/economic-impact-airbnb/

Andersson, M., Hjalmarsson, A., & Avital, M. (2013). Peer-to-Peer Service Sharing Platforms: Driving Share and Share Alike on a Mass-Scale.

Atom. (n.d.). Atom. Retrieved June 24, 2015, from https://atom.io/

Bootstrap. (2015). Bootstrap · The world's most popular mobile-first and responsive front-end framework. Retrieved June 21, 2015, from http://getbootstrap.com/

Cardoso, J., Hepp, M., & Lytras, M. D. (2007). The Semantic Web : Real World Applications from Industry. New York: Springer.

Coleman, T., & Greif, S. (2014). Discover Meteor. Retrieved June 21, 2015, from https://book.discovermeteor.com/

Craigslist | About > Factsheet. (n.d.). Retrieved June 22, 2015, from https://www.craigslist.org/about/factsheet

Denning, S. (2014). An economy of access is opening for business: five strategies for success. *Strategy & Leadership*, *42*(4), 14–21. doi:10.1108/SL-05-2014-0037

Firdaus, T. (2013). *Responsive Web Design by Example Beginner's Guide*.

Packt Publishing Ltd.

Fremstad, A. (2014). Gains from Sharing: Sticky Norms, Endogenous Preferences, and the Economics of Shareable Goods.

Gaziulusoy, A. I., & Twomey, P. (2014). *Emerging Approaches in Business Model Innovation Relevant to Sustainability and Low-carbon Transitions*.

Jain, N. (2014). Review of Different Responsive CSS Front-End Frameworks. *Journal of Global Research in Computer Science*, *5*(11), 5.

Linnovate. (2015). MEAN - Full-Stack JavaScript Using MongoDB, Express, AngularJS, and Node.js. Retrieved June 19, 2015, from http://mean.io/#!/

Meteor. (n.d.). Documentation - Meteor. Retrieved June 21, 2015, from http://docs.meteor.com/#/full/

Meteor Development Group. (n.d.). Try Meteor. Retrieved June 24, 2015, from https://www.meteor.com/install

Owyang, J., Tran, C., & Silva, C. (2013). The Collaborative Economy. *Altimeter Research*. Retrieved June 18, 2015, from http://www.collaboriamo.org/

Prochazka, A. (2007). Ruby on Rails: Up and Running > Ruby on Rails: Up and Running : Safari Books Online. doi:9780321533890

Rauch, D. E., & Schleicher, D. (2015). Like Uber, But for Local Governmental Policy: The Future of Local Regulation of the "Sharing Economy." *SSRN Electronic Journal*. doi:10.2139/ssrn.2549919

Schwartz, A. (2013). The Return Of Spinlister: How To Revive A Dead Sharing Economy Startup. *Co.Exist*. Retrieved June 17, 2015, from http://www.fastcoexist.com/1682784/the-return-of-spinlister-how-to-revive-a-dead-sharing-economy-startup

Sonders, D., & Braun, L. (2014). Validation in the Wild. *Design Management Review*, *25*(3), 20–26. doi:10.1111/drev.10290

Strack, I. (2012). *Getting Started with Meteor JavaScript Framework*. Birmingham: Packt Publishing.

Sublime. (n.d.). Sublime Text: The text editor you'll fall in love with. Retrieved June 24, 2015, from http://www.sublimetext.com/