Using Parse as the Backend to your Artwork Gallery Web App

MEA Mobile

Parse Credentials:

First we will need credentials for a Parse account at https://www.parse.com. You can provide these or we can set it up for you.

We will then setup an app for your Gallery inside the account for your convenience.

Intro:

This document is meant to assist someone just starting out with Parse understand how it is used as a back-end to support an Artwork Gallery Web Application. While a large amount of information will be presented here, it is important to note that it would be impossible to provide all information about Parse in this document. There will invariably be instances over the course of development where one must read the documentation Parse provides in order to understand a particular feature or solve a particular problem.

What this document does hope to accomplish is to give the reader a familiarity with Parse that will help them throughout the development process. Using this document and the linked materials as a reference, a robust application can be created in a very short period of time using Parse as a back-end.

What is Parse:

In a technical sense, Parse is a pre-made back-end solution for a generic web app.

In simpler terms, Parse is an application that takes much, if not all, of the server side (back-end) development that is to go into the creation of an application, and automates as much of it as possible. Parse greatly simplifies tasks such as setting up and viewing the content of a database, fetching content from a database, monitoring the requests that are sent to a database over given periods of time, and many others.

While outlining all the different use cases for Parse is beyond the scope of this document (links to advanced topics are present at the end of the doc) it is important that we go over some of the main features that you will be using to develop an Artwork Gallery Web Application.

Major Parse Features:

When developing a web app that will be used to display artwork from Parse there are two features that will prove invaluable to getting the job done. These features are the Parse dashboard (more specifically the data browser) and Parse' ability to host images online. Lets go over both of these features in a little more detail.

The Parse Dashboard:

To try and explain the purpose of the Parse Dashboard I'm going to compare it to the more common wordpress dashboard. If you have worked with wordpress in the past I hope this clears things up a bit for you. If you haven't worked with Wordpress don't worry, I'll try and make things as clear as I can regardless of past experiences.

In a broad (very very very broad) sense, the Parse Dashboard and Wordpress Dashboard seek to accomplish a similar goal. Much like the Wordpress Dashboard allows a user to easily manipulate the front-end content of their site (including paragraphs, posts, the css (through wordpress themes and the visual editor), Parse's Dashboard attempt to allow users to easily manipulate the back-end components of their application.

The Parse Dashboard has a number of tools you can use to accomplish most any back end task you can think of. It is divided into 5 major components…..

1) Analytics

Analytics is used in order to track the usage of your web application. It can answer questions such as "How many people are using my web application?" and "Are more people using my app now than have been in the past?". These aren't the only questions Analytics can answer. In general, Analytics is a fantastic tool for monitoring your web application in any way you see fit. Best part is that no difficult backend coding is required to track anything, all of it is done for you.

2) Data Browser

The data browser is by far the most important feature in the Parse Dashboard. If you have worked with databases before, it is easiest to see the Data Browser as Parse's equivalent to a program such as phpmyadmin and MySQLWorkbench (this is a broad comparison).

The Data Browser is organized into classes (not unlike classes found in a programing language such as java/php/ruby/etc.). By clicking on a class you can view each currently instantiated object of that class that is stored in Parse. A class will define a number of different properties and each object of a class will have a value for each property. It is worth noting that each object also has its own unique (to that class) objectId that is automatically generated upon its creation.

Another way of looking at Classes, Objects, and Properties is to put them in terms more comparable to that of a regular relational database. You can think of Classes as tables which house rows(Objects) and have columns(Properties). This way of viewing the Data Browser may be preferable to those who have more experience working with more traditional back-ends.

Using the Data Browser you can add and delete classes, add and delete objects of previously defined classes, and add/delete properties for a class (as well as the values Objects of each class hold for these values).
The majority of your interaction with Parse will be spent both looking at the data browser to see what is presently stored in Parse, and fetching objects that are stored (much like sql statements are used to fetch data from a database) using the Javascript SDK and Rest API (more on those later).

3) Cloud Code

Cloud code is an advanced feature that Parse provides and thus will only be lightly touched upon in this document.

In a normal Web Application, much of the code that powers the app is run on the device that is running the application. Since the number of devices that can run a web app can be very large, relying on code that runs on a users device can lead to highly variable performance. While a web app may run quickly and efficiently on a desktop, perhaps it is sluggish on mobile where local resources are scarcer and less powerful.

Parse solves this problem with Cloud Code. Cloud Code is (as its name suggests) code that runs online in the Cloud rather than locally on a users device. This allows a web application to have more consistent performance across a multitude of platforms.

Another advantage of utilizing Cloud Code is it exposes less of your source code to users, therefore protecting the inner workings of your Web App from those with malicious intent.

4) Push Notifications

Push Notifications are another advanced feature and their use will not be outlined in this document in detail.

Push notifications are used to send messages to your users. You have the option of sending these messages to all users or specific subsets of users. Use this feature to communicate with your user base easily.
5) Settings

The settings panel displays important information about an application. It houses a large amount of data but by far its most important content is the application keys. Application keys are crucial to your ability to connect to Parse in your code, you cannot use the Javascript SDK or the Rest API without them.

Parse File Hosting

In addition to its dashboard, Parse also offers a cloud hosting service to its users for files such as images. Unfortunately, as of this moment at least, files can only be posted to the cloud by utilizing the Rest API. There is no easy graphical tool for placing files in the Cloud as of yet.

While it is upsetting that uploads must be done in code, hosted files are still invaluable to using Parse. Classes can define properties of type "File", and, in an object of that class, the values for that object are stored as pointers to files stored in the cloud. This will come in handy later on.

Communicating with Parse In Code

While the Parse Dashboard is an excellent resource for viewing Parse and keeping track of what is in being stored in it, different tools must be used in order to communicate with Parse in the actual code of a website. To accomplish this task, there are two options, the Javascript SDK and the Rest API.

Javascript SDK

The Javascript SDK is an excellent tool for communicating with Parse. With it you can fetch objects, alter them, change them, and then send them back to Parse, updating them. You are not forced to retrieve objects one by one either. You can request collections of objects either through requesting a collection outright or issuing a query with parameters multiple objects meet.

Another great thing about the Javascript SDK is that its thoroughly documented; each function you can perform with it is outlined in the documentation and has an example provided to help understanding.

While the tutorial later in this document will give a basic understanding of how the Javascript SDK works, nothing will replace reading the documentation in terms of understanding the multitude of features the SDK implements.

Rest API

The Rest API allows the user to communicate with Parse with any coding language that can send an http request (which is the vast majority of them). While the Rest API may not be as convenient to use as the Javascript SDK in most situations, there are many things it can do that simply cannot be accomplished using any other tool (sometimes including the data browser).

The most important thing the Rest API can do that the Javascript SDK and Parse Dashboard can't is upload image files to Parse to be hosted on the Cloud. Remember, Javascript can send http requests so you can use the Rest API and the Javascript SDK simultaneously in the same document. This is useful because, in a given file, you can use the easier Javascript SDK for all the more basic functions it can accomplish, using the Rest API only when is necessary.

This is not to say the Rest API is hard to use, it is just harder than using the Javascript SDK. If you are more comfortable in a language other than Javascript there is nothing stopping you from using that language and communicating with Parse exclusively with the Rest API in your Web App.

Like the Javascript SDK, the Rest API is too large to realistically cover fully in this document. For a complete look at the Rest API, consult the Parse Documentation.

Putting it All Together (a short tutorial project):
*project code is available on github. See end of document

Now that we have all the textbook stuff out of the way its time to start writing some code!!!

In this short tutorial project, we will set up a simple Parse application that takes an image file and image name as input and sends this data to be stored in Parse. Fire up your local server (MAMP, WAMP, XAMPP, ect..), go to your usual htdocs folder for local projects and create a folder called "parseTutorial" (If you are unfamiliar with setting up a local development environment with MAMP/WAMP/XAMPP links to tutorials are provided at the end of the document).

Lets start by creating a basic html document with a text input field, file input field, and submit button (labels and breaks have also been added to give the page basic formatting). Call this document imageUploader.html (code is provided to help you follow along).

```
<!DOCTYPE html>
<html>
```

```
        <head>
        </head>
        <body>
                <label>Name: </label><input type='text' name='name'></image>
                <br/>
                <label>Image: </label></label><input type = 'file' name='image'
id='fileselect'></input>
                <br />
                <button id = 'submit'>submit</input>
        </body>
</html>
```

Next we have to connect our page to the Parse Javascript SDK. Luckily Parse distributes the Javascript necessary to utilize their SDK via a CDN (much in the same way jQuery is distributed) so you don't have to download the js files and put them in your project. You can simply reference the sdk in your header.

Lets also add jQuery to our document head (this will come in handy when we need to make Rest API calls, but more on that later).

```
<!DOCTYPE html>
<html>
        <head>
                <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"></script>
                <script type="text/javascript"
src="http://www.parsecdn.com/js/parse-1.2.18.min.js"></script>
        </head>
        <body>
                <label>Name: </label><input type='text' name='name'></image>
                <br/>
                <label>Image: </label></label><input type = 'file' name='image'
id='fileselect'></input>
                <br />
                <button id = 'submit'>submit</input>
        </body>
</html>
```

Alright, now what we have is a document with two input fields (one for text and one for a file), a button labeled "submit", and all the setup needed to utilize the Javascript SDK and Rest API. Now we can start connecting to Parse.

Set up a script tag in your code so you can start entering javascript and use the Parse.initialize method (shown in the next code segment) in order to connect to your Parse App.

```
<!DOCTYPE html>
```

```
<html>
	<head>
		<script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"></script>
		<script type="text/javascript"
src="http://www.parsecdn.com/js/parse-1.2.18.min.js"></script>
	</head>
	<body>
		<label>Name: </label><input type='text' name='name'></image>
		<br/>
		<label>Image: </label></label><input type = 'file' name='image'
id='fileselect'></input>
		<br />
		<button id = 'submit'>submit</input>
	</body>
	<script>
		Parse.initialize("etB4ba0v9sBBlzDGQCFc8zj3QMCrjAWB5k8X1mrR",
"vxbiXarE7nIBXNMH5Af4jyl9KKVZLfjxfz1lZrT7");
	</script>
</html>
```

The Parse.initialize method takes two parameters(the long string of random characters). The first is your apps specific application Key and the second is your apps specific javascript key. Both of these values can be found by going into your Parse Dashboard and navigating to your apps “Settings” tab. When in “Settings” click on Application keys to view all the keys needed to connect to an app.

Now lets start sending data to Parse. Since we are connected to our app, we can now make objects of classes defined in our app. lets create a new object of class ArtPiece.

```
<!DOCTYPE html>
<html>
	<head>
		<script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"></script>
		<script type="text/javascript" src="http://www.parsecdn.com/js/parse-1.2.18.min.js"></script>
	</head>
	<body>
		<label>Name: </label><input type='text' name='name'></image>
		<br/>
		<label>Image: </label></label><input type = 'file' name='image' id='fileselect'></input>
		<br />
		<button id = 'submit'>submit</input>
	</body>
	<script>
		Parse.initialize("etB4ba0v9sBBlzDGQCFc8zj3QMCrjAWB5k8X1mrR",
"vxbiXarE7nIBXNMH5Af4jyl9KKVZLfjxfz1lZrT7");

		//action listener for submit button
		$('#submit').click(function() {
			//define ArtPiece Class
			var ArtPiece = Parse.Object.extend('ArtPiece');
			//create an object of Class ArtPiece
			var artPiece = new ArtPiece();
```

```
                });
        </script>
</html>
```

We now have a button that, when pressed, will create a new object of the ArtPiece Class. Our next step is to save this object to Parse so it can be viewed in the data browser and retrieved at a later date.

We can accomplish this goal with the save method. Note that the save method can be issued plainly or with a callback function. The callback function can designate separate blocks of code to run whether the save succeeds or fails (more on this later). Using save function callbacks can be very useful but is unnecessary for our purposes in this tutorial.

```
<!DOCTYPE html>
<html>
        <head>
                <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"></script>
                <script type="text/javascript"
src="http://www.parsecdn.com/js/parse-1.2.18.min.js"></script>
        </head>
        <body>
                <label>Name: </label><input type='text' name='name'></image>
                <br/>
                <label>Image: </label></label><input type = 'file' name='image'></input>
                <br />
                <button id = 'submit'>submit</input>
        </body>
        <script>
                Parse.initialize("etB4ba0v9sBBlzDGQCFc8zj3QMCrjAWB5k8X1mrR",
"vxbiXarE7nIBXNMH5Af4jyl9KKVZLfjxfz1lZrT7");

                //action listener for submit button
                $('#submit').click(function() {
                        //define ArtPiece Class
                        var ArtPiece = Parse.Object.extend('ArtPiece');
                        //create an object of Class ArtPiece
                        var artPiece = new ArtPiece();
                        //save the object
                        artPiece.save();
                });
        </script>
</html>
```

To test that you have performed all the above steps in this tutorial correctly simply load up your project in the web browser of your choice and click the submit button. Now go to the Parse data browser (located in the Parse Dashboard) and look at the ArtPiece Class. A new object should be present in it with a value of ‘undefined’ stored in each of its columns.

If the code you have written succeeds in putting a new object into the Class then congratulations!!! You have just saved your first object to the Parse data browser. Now that you are able to save objects the next logical step is to save values for the various attributes of the Parse object so that the Parse object will hold more that just an objectId in the data browser. Confused? Don't worry it's far simpler than it sounds.

Lets start by using the text input field to update the name of the new ArtPiece we are creating. Simply place the value of the text input into a variable for convenience then use Parse's set() method to set the artPiece's "piece_name" equal to that variable. Save the variable after this and you will be able to see the object in the data browser with the name specified in the text input stored in the piece_name column.

*code blocks will now only show javascript present between <script> tags to save space. Full document will be shown at the end of the tutorial.

```
<script>
       Parse.initialize("etB4ba0v9sBBlzDGQCFc8zj3QMCrjAWB5k8X1mrR",
"vxbiXarE7nIBXNMH5Af4jyl9KKVZLfjxfz1lZrT7");

       //action listener for submit button
       $('#submit').click(function() {

              //define ArtPiece Class
              var ArtPiece = Parse.Object.extend('ArtPiece');
              //create an object of Class ArtPiece
              var artPiece = new ArtPiece();

              //grab value entered by user for name
              var name = $('[name="name"]').val();
              //set the artPiece's piece_name to name
              artPiece.set('piece_name', name);

              //save the object
              artPiece.save();
       });
</script>
```

Now that you are able to set basic properties of object to user input values before uploading that object to Parse we can focus on a more complicated problem, uploading images to Parse and associating these images with objects using the Rest API.

As of right now, there is no function in the Javascript SDK that will allow us to easily upload images to Parse and associate them with an object. Our only option to accomplish this is to use the Rest API, which is used by sending HTTP requests formatted in manners specified in the Parse documentation.

In this tutorial, we will use jQuery's ajax() function to send the HTTP request necessary to upload the file. Then associate the file in its stored position in the cloud with our Parse object. While it definitely isn't an easy process to grasp initially, I'll try to go through it step by step to make it as clear as possible.

Please refrain from writing code until you have fully read over the steps that must be done for an image to be successfully uploaded and associated.

The first step is to set the file selected by the user equal to a variable. This way we have an easy way to reference the selected file when posting the image to Parse. We do this by creating a variable that will store the file and, when the state of the file input element changes, storing the contents of the file input to the variable.

```
//variable that will store selected image file
var file;

//grab the file input tag with javascript
var fileselect = document.getElementById('fileselect');

//When the state of the file input button
//is changed, the selected file is set equal
//to the file var for later use
fileselect.onchange = function(e) {
  var files = e.target.files || e.dataTransfer.files;
  // Our file var now holds the selected file
  file = files[0];

       //action listener for submit button
       $('#submit').click(function() {
              //additional code here…
       });
}
```

Next up is getting the serverUrl ready as well as storing the keys necessary to make the HTTP request in variables so we can use them conveniently.

The serverUrl is important because it is the location which the image will be stored in the Parse Cloud. The base of the Url that stays the same no matter how many requests you make is "https://api.parse.com/1/files/". This however is not the complete serverUrl you must provide. In addition to this you must provide the name of the file at the end of the Url in order to give it a unique location in the cloud.

We construct the url by taking the string that is the name of the file to be uploaded and removing the whitespace from it (using regular expressions, regular expressions can perform look for specific strings and perform functions on them. In this case the expression seeks out all whitespace and deletes it). We then append this string to the end of "https://api.parse.com/1/files/" and set that all equal to a variable called serverUrl. The serverUrl's final prepared form can be expressed by the following psuedocode equation:

PsuedoCode: Not real javascript
server Url = “https://api.parse.com/1/files/” + removeWhiteSpace(file.name)

Next we store the application Id and rest key in variables so we can easily reference them later (both keys can be found in the setting portion of the Parse Dashboard). Place the code for these variables towards the beginning of the “Submit” buttons actionlistener. Just after the creation of the ArtPiece object.

```
$('#submit').click(function() {
        //define ArtPiece Class
        var ArtPiece = Parse.Object.extend('ArtPiece');
        //create an object of Class ArtPiece
        var artPiece = new ArtPiece();

    //whitespace is cut out of file's name (if present) using
    //regular expressions. File name then used to construct the
    //url the file will be stored at on the Cloud
    var subDirect = (file.name).replace(/\s+/g, '');
    var serverUrl = 'https://api.parse.com/1/files/' + subDirect;

    //application Id and Rest Key stored in vars for convinient
    //later use (both found in Parse settings)
    var appID = "etB4ba0v9sBBlzDGQCFc8zj3QMCrjAWB5k8X1mrR";
    var restKey = "tszvoEvi2pNycFv6BVq4EbqCSA4HlcFkCpGm9VN1";

    //additional code goes here...
});
```

Alright, almost there. We have taken care of all the necessary set up, now all there is left to do is use the jQuery ajax() method to send out the HTTP request and associate the uploaded file with the Parse object.

Call the $.ajax() method, within its body will be the contents of the HTTP request as well as the success and error blocks. Specify POST as the HTTP request type the type (the two most commonly used HTTP request types are GET and POST).

Next set the headers. The headers must include the application id, the rest key, and the type of the file being uploaded.

Next specify the url as the serverUrl we previously prepared. Set the data equal to the file variable that contains the file to be uploaded. Set processData to false and set contentType to false. Finally we move on to defining what will happen in our success and error blocks.

Within the success block we will associate the newly uploaded image with our object in Parse with the set method (note the set method takes a different parameter when associating a file. Use the code provided as a reference). We will then set the piece_name to the user entered name and save the object to Parse, just as we did before.For good measure, we will include an error message in the fail block to help us with debugging later if necessary (fear not the code provided works.just trying to promote good practice is all).

```
$.ajax({
        type: "POST",
        beforeSend: function(request){
                request.setRequestHeader("X-Parse-Application-Id", appID);
         request.setRequestHeader("X-Parse-REST-API-Key", restKey);
         request.setRequestHeader("Content-Type", file.type);
        },
        url: serverUrl,
        data: file,
        processData: false,
        contentType: false,
        success: function(data) {

                //The uploaded image is assosiated with the
                //Parse object here
                artPiece.set('image', {
                        name: data.url.substring(data.url.lastIndexOf('/') + 1),
                        url: data.url,
                        __type: "File"
                });

                //grab value entered by user for name
                var name = $('[name="name"]').val();
                //set the artPiece's piece_name to name
                artPiece.set('piece_name', name);
                //save the object
                artPiece.save();
        },
        error: function() {
                console.log("Error: The image was not uploaded to Parse Properly.");
        }
});
```

Alright, now that you have all the pieces you need to make this application work its time to put everything together. Make sure your code is organized in a similar manner to the code below. The code that sets the file variable equal to the user selected file must be BEFORE the submit buttons actionlistener. Also the code that associates the uploaded file with the object, sets the object's piece_name, and saves the object to Parse must be placed in the success block of the ajax method.

Try running the project and uploading an image. Now check the data browser. If you look in the image field of your newly uploaded object in the data browser and see a reference to your image stored in the Parse cloud then congratulations!!! You have completed this tutorial and taken your first step into the world of Parse.

If your application doesn't work, check your code against the code below. If the code below is hard to read (as space is limited) I've made the project available on github. Simply clone the project to your local machine and view the source code. Compare it to the code you wrote and learn from whatever mistype you made. Do this and you'll have the project up and running in no time!!!

Final imageUploader.html

```
<!DOCTYPE html>
<html>
	<head>
		<script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"></script>
		<script type="text/javascript" src="http://www.parsecdn.com/js/parse-1.2.18.min.js"></script>
	</head>
	<body>
		<label>Name: </label><input type='text' name='name'></image>
		<br/>
		<label>Image: </label></label><input type = 'file' name='image' id = 'fileselect'></input>
		<br />
		<button id = 'submit'>submit</input>
	</body>
<script>
	Parse.initialize("etB4ba0v9sBBlzDGQCFc8zj3QMCrjAWB5k8X1mrR",
"vxbiXarE7nIBXNMH5Af4jyl9KKVZLfjxfz1lZrT7");

	//variable that will store selected image
	var file;

	//grab the file input tag with javascript
	var fileselect = document.getElementById('fileselect');

	//When the state of the file input button
	//is changed, the selected file is set equal
	//to the file var for later use
	fileselect.onchange = function(e) {
    var files = e.target.files || e.dataTransfer.files;
    // Our file var now holds the selected file
    file = files[0];

		//action listener for submit button
		$('#submit').click(function() {
			//define ArtPiece Class
			var ArtPiece = Parse.Object.extend('ArtPiece');
			//create an object of Class ArtPiece
			var artPiece = new ArtPiece();

            //whitespace is cut out of file's name (if present) using
            //regular expressions. File name then used to construct the
            //url the file will be stored at on the Cloud
            var subDirect = (file.name).replace(/\s+/g, '');
            var serverUrl = 'https://api.parse.com/1/files/' + subDirect;

            //application Id and Rest Key stored in vars for convinient
            //later use (both found in Parse settings)
            var appID = "etB4ba0v9sBBlzDGQCFc8zj3QMCrjAWB5k8X1mrR";
            var restKey = "tszvoEvi2pNycFv6BVq4EbqCSA4HlcFkCpGm9VN1";

            //HTTP request sent using the jQuery ajax method. If the
            //request is successful the code in the success block saves
            //new artPiece object to Parse. If an error occurs an error
```

```
                //message is displayed in an alert box
                $.ajax({
                       type: "POST",
                       beforeSend: function(request){
                              request.setRequestHeader("X-Parse-Application-Id", appID);
                        request.setRequestHeader("X-Parse-REST-API-Key", restKey);
                        request.setRequestHeader("Content-Type", file.type);
                       },
                       url: serverUrl,
                       data: file,
                       processData: false,
                       contentType: false,
                       success: function(data) {

                              //The uploaded image is assosiated with the
                              //Parse object here
                              artPiece.set('image', {
                                     name: data.url.substring(data.url.lastIndexOf('/') +
1),

                                     url: data.url,
                                     __type: "File"
                              });

                              //grab value entered by user for name
                                     var name = $('[name="name"]').val();
                                     //set the artPiece's piece_name to name
                                     artPiece.set('piece_name', name);
                                     //save the object
                                     artPiece.save();
                       },
                       error: function() {
                              console.log("Error: The image was not uploaded to Parse
Properly.");
                       }
                });
               });
       }
</script>
</html>
```

Cloning the Project from github:

you can clone the working project from github by opening up the terminal (or command line on windows) and typing in the command:

(Project is being moved to bitBucket. Appropriate command to clone the project will be inserted after move is complete)

*Note that this project does not contain the application Id, javascript key, and rest key in its code. You will have to look up your apps various keys in the Parse dashboard and put them into the source code before the project will work. This has been done because the project is publicly available and it would be wise to hide your keys as much as is practically possible.

Conclusion:

After completing this tutorial you should have a much deeper understanding of Parse and how it can be used as a robust back-end to your web application. However, as stated before, there are a multitude of ways that Parse can be used to assist you in development that are not covered in this document.

For this reason it is best practice to, whenever a new issue is encountered, consult the Parse documentation. It contains very clear and concise explanations for most anything you will find yourself doing with Parse.

Hopefully this document proved helpful in getting you started with Parse. As you continue to learn more about Parse you are bound to discover more and more innovative ways to use it to make you app more robust, engaging, and fun to develop. Keep learning more about Parse and the back-end of your future projects will be done quicker than you ever thought possible.

Best of luck

Additional Resources:

Parse Documentation: https://www.parse.com/docs/

MAMP/WAMP tutorial: https://www.youtube.com/watch?v=6qOcdvgYT7E

jQuery $.ajax() Documentation: http://api.jquery.com/jquery.ajax/