<https://www.sitepoint.com/creating-and-handling-forms-in-node-js/>

For a high-quality, in-depth introduction to Node.js, you can’t go past Canadian full-stack developer Wes Bos. Try his course [here](https://learnnode.com/friend/SITEPOINT), and use the code SITEPOINT to get 25% off and to help support SitePoint.

Web applications commonly have to handle forms in which the user can enter data that need to be processed. Node.js by itself does not provide any way to handle the work required to process the data of the form (like validations, for example). It just provides raw body data. This might get cumbersome for the app developer to manage, but there are third party modules which can help process the form data and provide the necessary ones to the application. Among these modules, there’s one called [formidable](https://www.npmjs.com/package/formidable); it deals with a Node.js module which can help you parse form data and use them in your application.

In this article we’re going to see how we can use the formidable module to process form fields and files in our Web application.

## Creating an HTTP Server With a Form in Node.js

Let’s start by creating a HTTP server using Node.js and displaying a form to the user.To do that, we need to create a file named server.js and add the following code:

var http = require('http');

var fs = require('fs');

var server = http.createServer(function (req, res) {

displayForm(res);

});

function displayForm(res) {

fs.readFile('form.html', function (err, data) {

res.writeHead(200, {

'Content-Type': 'text/html',

'Content-Length': data.length

});

res.write(data);

res.end();

});

}

server.listen(1185);

console.log("server listening on 1185");

We’ll also need to create a form.html file which will contain the HTML code to render the form as shown below:

<html>

<body>

<form action="" method="post" enctype="multipart/form-data">

<fieldset>

<label for="name">Name:</label>

<input type="text" id="name" name="name" placeholder="Enter your full name" />

<br />

<label for="email">Email:</label>

<input type="email" id="email" name="email" placeholder="Enter your email address" />

<br />

<label for="description">Description:</label>

<textarea id="description" name="description" placeholder="Enter a short description about yourself"></textarea>

<br />

<input type="submit" value="Create Profile" />

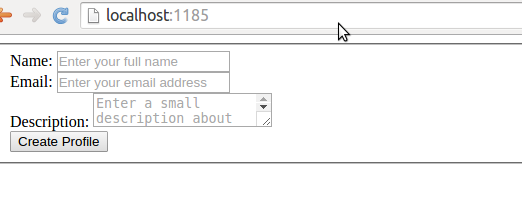
</fieldset>

</form>

</body>

</html>

In the above code we’ve created a HTTP server and a callback which calls the function displayForm. We pass the res object to it so that the response can be sent back to the client. In the function displayForm we create a callback to read the form.html file and return those data back to the client. The form.html file contains the HTML code to build the fields of a form which is meant to be displayed to the user. If we click on the “Create Profile” button, it’ll do a post request with the submitted form information to the same URL. Then, we’re starting the server at port 1185. After starting the server, if we go to localhost:1185 we should see the HTML form displayed as shown below.



## Adding the Dependencies on Formidable Module

Node.js by itself does not provide the code required to parse the form. This would require a lot of boilerplate code to be added in the application. The formidable module takes care of parsing the form of the request and return back the data from the submitted form. To use it in our project, we first have to create a package.jsonfile using the following command:

npm init

Once done, we should see the package.json file in the root folder as shown below:

{

"name": "forms-in-nodejs",

"version": "1.0.0",

"description": "This project demonstrates how to use forms in Node.js",

"main": "server.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"start": "node server.js"

},

"repository": {

"type": "git",

"url": "https://github.com/abbassoftware/forms-in-nodejs.git"

},

"author": "Abbas",

"license": "ISC",

"bugs": {

"url": "https://github.com/abbassoftware/forms-in-nodejs/issues"

},

"homepage": "https://github.com/abbassoftware/forms-in-nodejs"

}

At this point, we need to add the latest version of our module in the dependency using the following command.

npm install formidable@latest --save

After running this command, formidable should be added to our node\_modules directory.

## Handling the Post of Text Fields

Once we’ve got all our dependencies set, we’re now ready to actually handle the form data. When the user clicks on “Create Profile”, we’ll get a post request that we’ll need to handle accordingly. To do that we need to update the code in our server.js file as shown below:

var http = require('http');

var fs = require('fs');

var formidable = require("formidable");

var util = require('util');

var server = http.createServer(function (req, res) {

if (req.method.toLowerCase() == 'get') {

displayForm(res);

} else if (req.method.toLowerCase() == 'post') {

processAllFieldsOfTheForm(req, res);

}

});

function displayForm(res) {

fs.readFile('form.html', function (err, data) {

res.writeHead(200, {

'Content-Type': 'text/html',

'Content-Length': data.length

});

res.write(data);

res.end();

});

}

function processAllFieldsOfTheForm(req, res) {

var form = new formidable.IncomingForm();

form.parse(req, function (err, fields, files) {

//Store the data from the fields in your data store.

//The data store could be a file or database or any other store based

//on your application.

res.writeHead(200, {

'content-type': 'text/plain'

});

res.write('received the data:\n\n');

res.end(util.inspect({

fields: fields,

files: files

}));

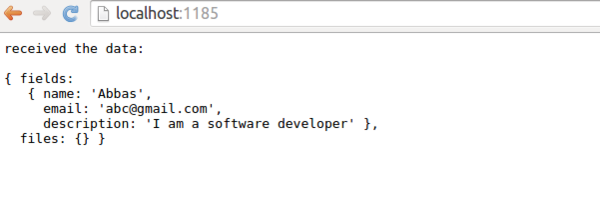
});

}

server.listen(1185);

console.log("server listening on 1185");

In the above code we’ve updated the callback on the HTTP server to call the processAllFieldsOfTheForm function in case we get a post request. In the previously cited function we create an object for the incoming form. On that object we pass the res and set a callback on it which will be called when the form is parsed and will return the parsed fields to us. In the above code we’re just returning the fields back as output to the client, but depending on the application we’re building, we can store the values in a data store or process it accordingly.   
Now, if we run our server and submit the form, we’ll have the result shown below:



If we have many fields to process, it might be better to get a callback on as soon as each field is parsed rather than the whole form being parsed. The formidable module gives us a set of events on which we can add some callbacks which are called for each field. To do that, first update our server. js code as shown below:

var http = require('http');

var fs = require('fs');

var formidable = require("formidable");

var util = require('util');

var server = http.createServer(function (req, res) {

if (req.method.toLowerCase() == 'get') {

displayForm(res);

} else if (req.method.toLowerCase() == 'post') {

//processAllFieldsOfTheForm(req, res);

processFormFieldsIndividual(req, res);

}

});

function displayForm(res) {

fs.readFile('form.html', function (err, data) {

res.writeHead(200, {

'Content-Type': 'text/html',

'Content-Length': data.length

});

res.write(data);

res.end();

});

}

function processAllFieldsOfTheForm(req, res) {

var form = new formidable.IncomingForm();

form.parse(req, function (err, fields, files) {

//Store the data from the fields in your data store.

//The data store could be a file or database or any other store based

//on your application.

res.writeHead(200, {

'content-type': 'text/plain'

});

res.write('received the data:\n\n');

res.end(util.inspect({

fields: fields,

files: files

}));

});

}

function processFormFieldsIndividual(req, res) {

//Store the data from the fields in your data store.

//The data store could be a file or database or any other store based

//on your application.

var fields = [];

var form = new formidable.IncomingForm();

form.on('field', function (field, value) {

console.log(field);

console.log(value);

fields[field] = value;

});

form.on('end', function () {

res.writeHead(200, {

'content-type': 'text/plain'

});

res.write('received the data:\n\n');

res.end(util.inspect({

fields: fields

}));

});

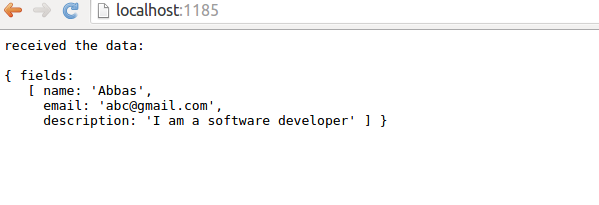
form.parse(req);

}

server.listen(1185);

console.log("server listening on 1185");

In the above code, when we get a post request, we call the function processFormFieldsIndividual (instead of processAllFieldsOfTheForm) which creates an incoming form object and adds a callback on field and end events. The field event is called as soon as each field is parsed. In the callback we’re just storing the fields, but we can do any processing we like, based on our application here. In the end event we are just returning all the fields to the client.   
If we now run our server and submit the form, we should see the output as follows:



## Handling the Upload of Images

Once we’ve learned how to handle text fields. let’s check how we can handle files upload using formidable. In case we’re fine to get a callback after all the fields are passed, in that case the files are also passed as a parameter to the callback. If we want to be notified on each file upload and know how process of file upload is going on, instead, we need to use the callback on each file. Firstly, let’s change the HTML file as shown below in order to add an option to upload a profile picture:

<html>

<body>

<form action="" method="post" enctype="multipart/form-data">

<fieldset>

<label for="name">Name:</label>

<input type="text" id="name" name="name" placeholder="Enter your full name" />

<br />

<label for="email">Email:</label>

<input type="email" id="email" name="email" placeholder="Enter your email address" />

<br />

<label for="description">Description:</label>

<textarea id="description" name="description" placeholder="Enter a short description about yourself"></textarea>

<br />

<label for="profilepic">Profile Pic:</label>

<input type="file" id="profilepic" name="profilepic" />

<br />

<input type="submit" value="Create Profile" />

</fieldset>

</form>

</body>

</html>

At this point, let’s modify the processFormFieldsIndividual function to handle files as shown below:

function processFormFieldsIndividual(req, res) {

//Store the data from the fields in your data store.

//The data store could be a file or database or any other store based

//on your application.

var fields = [];

var form = new formidable.IncomingForm();

//Call back when each field in the form is parsed.

form.on('field', function (field, value) {

console.log(field);

console.log(value);

fields[field] = value;

});

//Call back when each file in the form is parsed.

form.on('file', function (name, file) {

console.log(name);

console.log(file);

fields[name] = file;

//Storing the files meta in fields array.

//Depending on the application you can process it accordingly.

});

//Call back for file upload progress.

form.on('progress', function (bytesReceived, bytesExpected) {

var progress = {

type: 'progress',

bytesReceived: bytesReceived,

bytesExpected: bytesExpected

};

console.log(progress);

//Logging the progress on console.

//Depending on your application you can either send the progress to client

//for some visual feedback or perform some other operation.

});

//Call back at the end of the form.

form.on('end', function () {

res.writeHead(200, {

'content-type': 'text/plain'

});

res.write('received the data:\n\n');

res.end(util.inspect({

fields: fields

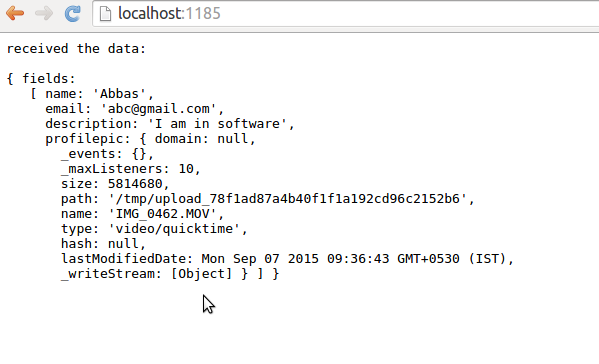
}));

});

form.parse(req);

}

In the above code we’ve registered the callbacks on the file event which will be fired when a file is uploaded. In this we just add the it to the fields array which will be sent back to the client when the form parsing is over. We also get the progress callback, in which we’re just displaying the progress on console. Now, if we fill the form and click on “Create Profile”, we’ll see these results, as shown below:





## Conclusions

We can say that some basic forms are present in all kinds of Web applications. Processing forms in Node.js applications would require a lot of handwritten boilerplate code. However, we’ve seen that third party modules like “formidable” make it easy for the application to parse forms and just concentrate on the application logic.   
Have you ever tried it (or similar modules) in your application? What is your experience? Let’s start a discussion!