

# WEB322 Assignment 6

## Submission Deadline:

Friday, July 21<sup>st</sup>, 2017 @ 11:59 PM

## Assessment Weight:

5% of your final course Grade

## Objective:

Work with a MongoDB data source on the server and practice working with data in views (.hbs templates)

## Specification:

For this assignment, we will be updating our "about.hbs" page with a comments section!

**NOTE:** If you are unable to start this assignment because Assignment 5 was incomplete - email your professor for a clean version of the Assignment 5 files to start from (effectively removing any custom CSS or text added to your solution). Remember, you must successfully complete ALL assignments to pass this course.

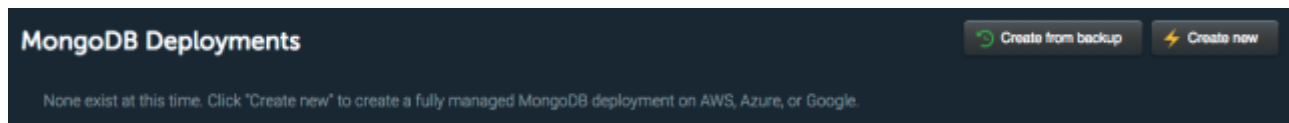
## Getting Started:

Before we get started, we must create a new account on [www.mlab.com](http://www.mlab.com) to host our new MongoDB database:

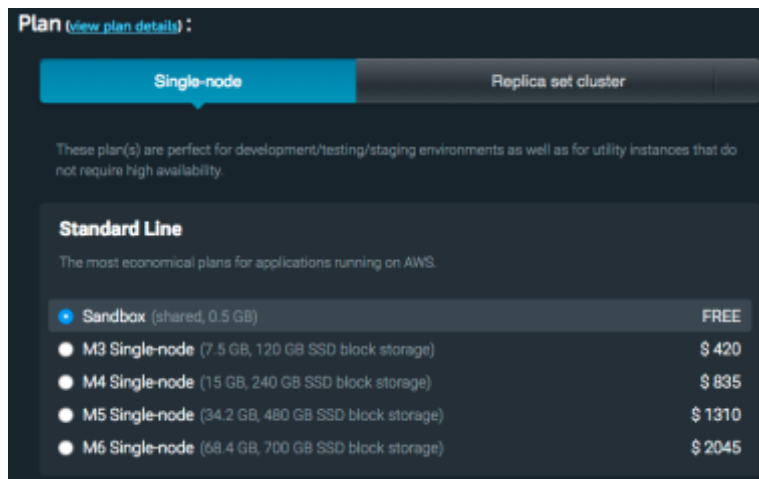
- Navigate to [www.mlab.com](http://www.mlab.com)
- Click the blue "SIGN UP" button on the top right of the page
- Fill out the form to enter your:
  - Account Name (use your My.Seneca user name - if it's already taken, choose something similar, so that it's easy to remember)
  - User Name (use your My.Seneca user name - if it's already taken, choose something similar, so that it's easy to remember - **you will need this user name to log in**)
  - Email (use your full Seneca email address)
  - Password (pick something you'll remember - **you will need this password to log in**)
- Enter your password one more time, accept the terms and click the blue "Create Account" button
- You should then receive (in your Seneca email) an email from mLab asking you to "Verify your mLab email address". Click the link in the email to verify your account

Now that we have created and verified our account, we must create a new "MongoDB Deployment"

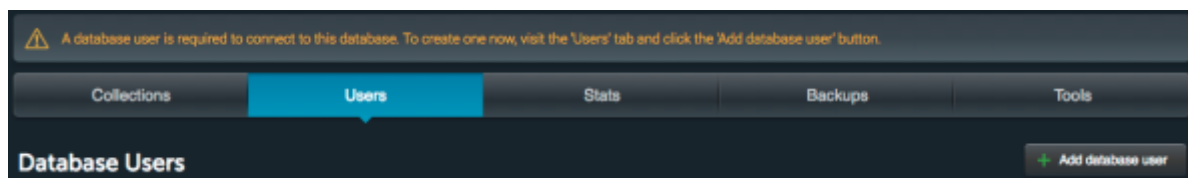
- Click the "Create new" button (indicated with a lightning bolt) next to the "MongoDB Deployments" section:



- On the next page, leave the "Cloud Provider" at the default ("Amazon web services")
- For "Plan" click the blue "Single Node" tab and choose "Sandbox" under "Standard Line" - this is the FREE option

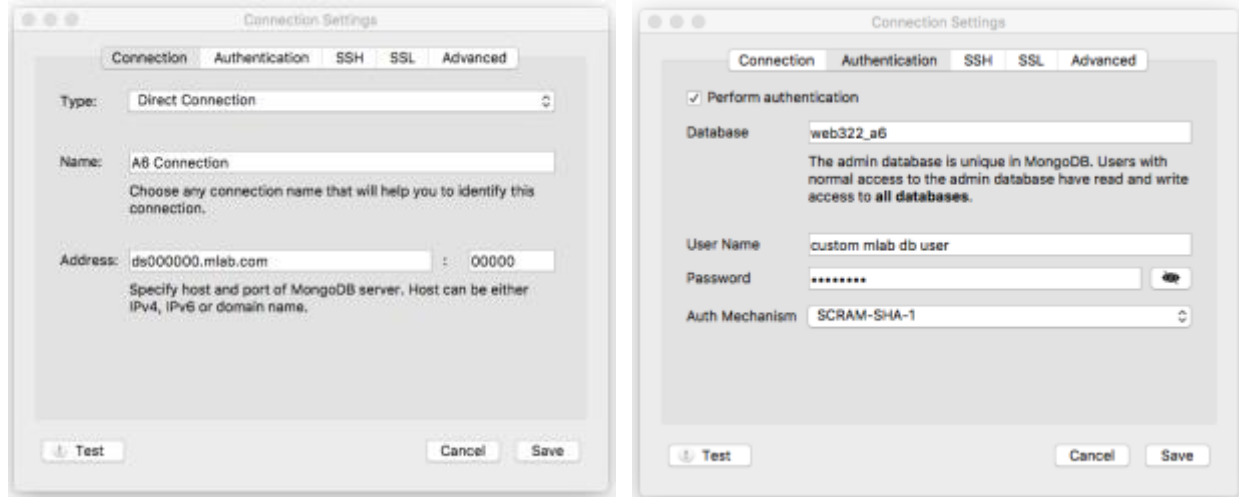


- Scroll down to the "Database Name" field and enter: "web322\_a6"
- You should see "Price: %0 / month" at the bottom of the page. Next to this, click the "Create new MongoDB deployment" button
- Once this is complete, you should be redirected to your "MongoDB Deployments" page and you should see a table with single deployment with a green checkmark next to it. Click this row to edit the details of the deployment
- This will take you to a page "Database: web322\_a6" where you can edit the details of the database.
- Click on the "Users" tab and click the button: "+ Add database user"



- This will open a modal window prompting you to create a user (user name / password). Use something that you will remember and click the blue "CREATE" button (make sure to leave "Make read-only" unchecked)
- Take note of the section: "To connect using a driver via the standard MongoDB URI" - it should look *something* like this: "mongodb://<dbuser>:<dbpassword>@ds000000.mlab.com:00000/web322\_a6"
- This is the connection string that we will be using to connect to our database (where <dbuser> is your newly created "Database User" and <dbpassword> is the corresponding password)

- Write down your **completed connection string** using your **<dbuser>** and **<dbpassword>** - we will be using it in our application
- **NOTE:** It is recommended at this point that you confirm your connection using Robo 3T (refer to the [Week 8 notes](#) under "Tools" & "The MongoDB Connection String Format" for instructions) - once connected, leave the window open to view your data & test your solution. **NOTE:** You will have to use the "Connection" and "Authentication" tabs with your new mLab database credentials to successfully connect:



## Getting Started - Adding a new "data-service" module

- Since we will be working with a new data source, we should keep our code modular and add a new module to manage all of our CRUD (Create, Read, Update, Delete) operations with our new MongoDB database:
- Create a new file at the root of the folder (beside the existing "data-service.js" file) and call it "data-service-comments.js"
- At the top of your "server.js" file, add the line:
  - **const dataServiceComments = require("../data-service-comments.js");**
- We will use "dataServiceComments" to execute the functions exported from within our new module "data-service-comments.js"

## Installing "mongoose"

- Open the "integrated terminal" in Visual Studio Code and enter the command:
  - **npm install mongoose --save**
- At the top of your new "data-service-comments.js" file, add the lines:
  - **const mongoose = require('mongoose');**
  - **let Schema = mongoose.Schema;**

## data-service-comments.js - Creating Our Schema

- As we discovered in the Week 8 notes, in order to work with MongoDB records using mongoose, we must first create a "Schema" (**HINT**: See "Setting up a schema" in the [Week 8 Notes](#) for examples)
- Define a new "**contentSchema**" according to the following specification:

Property	Mongoose Schema Type												
authorName	String												
authorEmail	String												
subject	String												
commentText	String												
postedDate	Date												
replies	<div>[ { Property: Type, Property: Type } ]</div> <div>NOTE: this will be an array of <b>objects</b> that use the following specification:</div> <table><tr><th>Property</th><th>Mongoose Schema Type</th></tr><tr><td>comment_id</td><td>String</td></tr><tr><td>authorName</td><td>String</td></tr><tr><td>authorEmail</td><td>String</td></tr><tr><td>commentText</td><td>String</td></tr><tr><td>repliedDate</td><td>Date</td></tr></table>	Property	Mongoose Schema Type	comment_id	String	authorName	String	authorEmail	String	commentText	String	repliedDate	Date
Property	Mongoose Schema Type												
comment_id	String												
authorName	String												
authorEmail	String												
commentText	String												
repliedDate	Date												

- Once you have defined your "**contentSchema**" per the specification above, add the line:
  - let Comment; // to be defined on new connection (see initialize)**

## data-service-comments.js - Exported Functions

Each of the below functions are designed to work with the **Comment** Object (defined by **commentSchema**). Once again, since we have no way of knowing how long each function will take, **every one of the below functions must return a promise** that **passes the data** via it's "**resolve**" method (or if an error was encountered, passes an **error message** via it's "**reject**" method). When we access these methods from the server.js file, we will be assuming that they return a promise and will respond appropriately with **.then()** and **.catch()**.

### initialize()

- Much like the "initialize" function in our data-service module, we must ensure that we are able to connect to our MongoDB instance before we can start our application.
- We must also ensure that we create a new connection (using **createConnection()** instead of **connect()** - this will ensure that we use a connection local to our module) and initialize our "Comment" object, if successful

- Additionally, if our connection is successful, we must **resolve()** the returned promise without returning any data
- If our connection has an error, we must, **reject()** the returned promise with the provided error:
- To achieve this, **use the following code** for your new initialize function, where **connectionString** is your connection string from mLab as identified above:

```
module.exports.initialize = function () {
  return new Promise(function (resolve, reject) {
    let db = mongoose.createConnection("connectionString");

    db.on('error', (err)=>{
      reject(err); // reject the promise with the provided error
    });
    db.once('open', ()=>{
      Comment = db.model("comments", commentSchema);
      resolve();
    });
  });
};
```

#### addComment(data)

- This function will add a new Comment to the database by:
  - Setting the **data.postedDate** to the current date/time (using: Date.now())
  - Creating a "new" Comment with the "data" passed to the function (ie: let newComment = new Comment(data))
  - Saving the **newComment** using the **newComment.save(...)** method
  - If there was an error, **reject()** the returned promise with the message: "**There was an error saving the comment:** " and **include the error**
  - If there wasn't an error, **resolve()** the promise with the **\_id** value of the newly created comment (ie: **resolve(newComment.\_id);**)

#### getAllComments()

- This function will return all comments by:
  - Invoking **Comment.find()** with the **.sort({...})** option to sort by "postedDate" ascending (asc)
  - Making sure to use **.exec()** to return a promise
  - If the Comment.find() promise was rejected, **reject()** the returned promise and pass the error that was "caught" during the Comment.find() operation, ie, "err"
  - If the Comment.find() promise resolved successfully, **resolve()** the returned promise and pass all of the data returned by the Comment.find() promise

#### addReply(data)

- This function will add a new Reply to an existing Comment to the database by:
  - Setting the **data.repliedDate** to the current date/time (using: Date.now())

- Updating the comment whose `_id` value matches `data.comment_id` by using the `Comment.update( ... )` method
- Making sure to use `{ $addToSet: { replies: data } }` (instead of `$set`) to ensure that the "data" is added to the "replies" array for the corresponding comment
- Making sure to use `.exec()` to return a promise
- If the `Comment.update()` promise was rejected, **reject()** the returned promise and pass the error that was "caught" during the `Comment.find()` operation, ie, "err"
- If the `Comment.update()` promise resolved successfully, **resolve()** the returned promise without returning any data

## Testing our new data-service-comments module

Now that we have completed our data-service-comments module, it is a good idea to test it before moving on. In your `server.js` file, comment out the block of code surrounding `app.listen(...)` (ie, `dataService.initialize.then()`, etc.) and instead use the following "test" code:

```
dataServiceComments.initialize()
  .then(() => {
    dataServiceComments.addComment({
      authorName: "Comment 1 Author",
      authorEmail: "comment1@mail.com",
      subject: "Comment 1",
      commentText: "Comment Text 1"
    }).then((id) => {
      dataServiceComments.addReply({
        comment_id: id,
        authorName: "Reply 1 Author",
        authorEmail: "reply1@mail.com",
        commentText: "Reply Text 1"
      }).then(dataServiceComments.getAllComments)
        .then((data) => {
          console.log("comment: " + data[data.length - 1]);
          process.exit();
        });
    });
  }).catch((err) => {
    console.log("Error: " + err);
    process.exit();
  });
```

If you run this code using "node server.js", instead of your server starting up, you should instead get something like the following in your console:

```
comment: { _id: 59499c405d5c76e7d0f64729,
  authorName: 'Comment 1 Author',
  authorEmail: 'comment1@mail.com',
  subject: 'Comment 1',
```

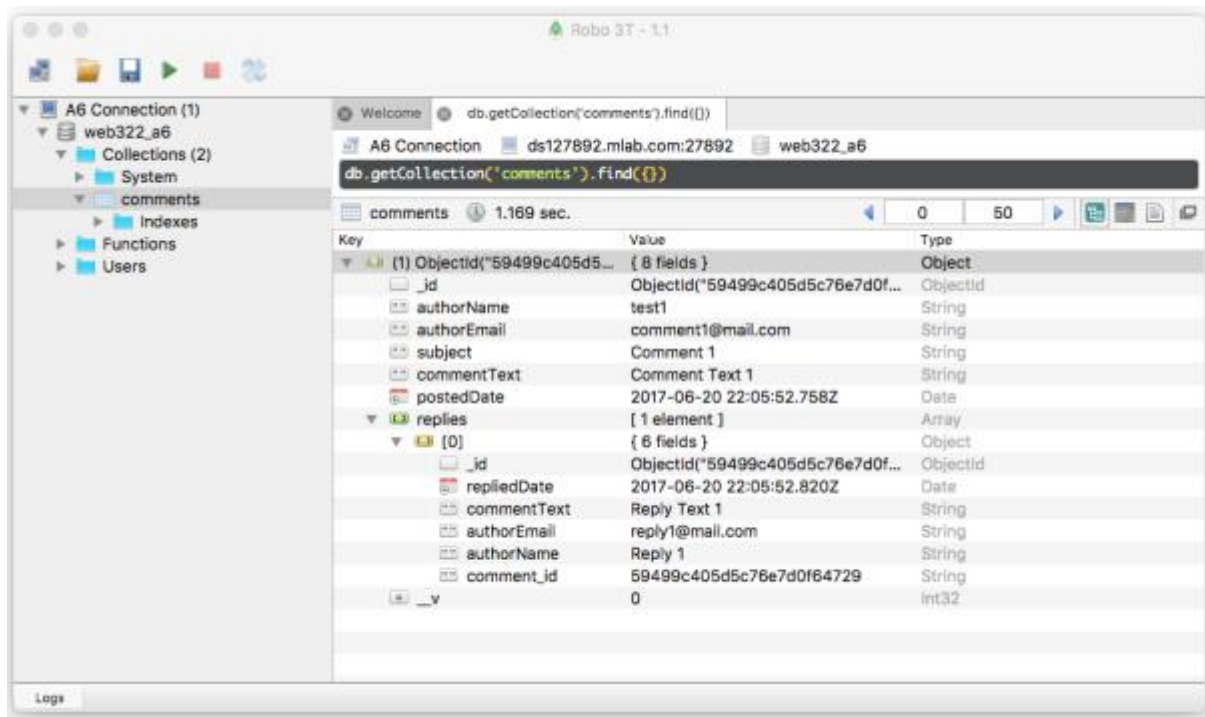
```

commentText: 'Comment Text 1',
postedDate: 2017-06-20T22:05:52.758Z,
__v: 0,
replies:
[ { _id: 59499c405d5c76e7d0f6472a,
  repliedDate: 2017-06-20T22:05:52.820Z,
  commentText: 'Reply Text 1',
  authorEmail: 'reply1@mail.com',
  authorName: 'Reply 1 Author',
  comment_id: '59499c405d5c76e7d0f64729' } ] }

```

You can also verify that your code is working using Robo 3T:

- Right-click on the "web322\_a6" database in the left sidebar and choose "refresh"
- Expand "Collections"
- Right-click on "Comments" and chose "View Documents"
- Expand the data in the right pane (there should be at least 1 document)



## Starting the Server

Now that we know our "dataServiceComments" module is working, remove the "test" code (above) and uncomment (restore) the code surrounding `app.listen()`. We must make a small change to the "startup procedure" for our server (ie: the methods that are invoked before `app.listen()`). Since our app is now going to be relying on the new "dataServiceComments" module, we must modify the following code:

```
dataService.initialize()
  .then(()=> {
    app.listen(HTTP_PORT, onHttpStart);
  })
  .catch(()=> {
    console.log("unable to start dataService");
  });
```

This code needs to include a call to **dataServiceComments.initialize()** as well before starting the server, so include it somewhere in the promise chain above so that it gets executed before `.then(()=> { app.listen(HTTP_PORT, onHttpStart); })`.

**Recall:** functions that return promises can be chained using the pattern:

```
function()
  .then(function)
  .then(()=>{
    // ...
  })
  .catch(()=>{
    //...
  });
```

## Adding new "Post" Routes

Since we will be allowing users to add comments on our "About" Page; we need to add a couple of new "POST" routes to our `server.js` file that use our new **dataServiceComments** module to persist the data:

### `/about/addComment`

- This **POST** route will invoke the **addComment(data)** method of our **dataServiceComments** module and pass the POST data (`req.body`). If the `addComment(data)` method resolved successfully, **redirect** the user to the `/about` route. If the `addComment(data)` method was rejected, output the **error to the console** and **redirect** the user to the `/about` route

### `/about/addReply`

- This **POST** route is almost identical to `/about/addComment`, only it will invoke the **addReply(data)** method of our **dataServiceComments** module and pass the POST data (`req.body`). If the `addReply(data)` method resolved successfully, **redirect** the user to the `/about` route. If the `addReply(data)` method was rejected, output the **error to the console** and **redirect** the user to the `/about` route



## Updating our GET "/about" Route

Since our "About" page will now contain "Comments" functionality, it only makes sense to pass all of the "Comments" data to the "about.hbs" view:

- In the GET route "/about", invoke the **getAllComments()** method of our **dataServiceComments** module. If the **getAllComments()** method resolved successfully, pass the data returned from the promise into the **res.render** method for the "about" view, ie: **res.render("about", {data: dataFromPromise});**
- If the **getAllComments()** method was rejected, simply render the "about" view as before without passing it any data, ie: **res.render("about");**

## Updating our "about.hbs" view

With all of the back-end logic in place, the only thing left is to update our "about.hbs" view to include some form elements and blocks (<div>...</div>) to allow the user to enter new comments/replies and read existing comments/replies.

- To begin, add the following small addition to your **site.css** file (this will make the comment replies look a little bit cleaner):

```
.well .well{
  background-color: white;
}
```

- Next, open your **layout.hbs** file and add the following 2 lines in the <head>...</head> element to incorporate the jQuery components of the Bootstrap framework:

```
<script src="https://code.jquery.com/jquery-2.2.4.min.js" integrity="sha256-
BbhdIvQf/xTY9gja0Dq3HiwQF8LaCRTXxZKRutelT44=" crossorigin="anonymous"></script>
```

```
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-
Tc5lQib027qvyjSMfHjOMaLkfuWVxZxUPnCIA7l2mCWNIpG9mGCD8wGNlCPD7Txa"
crossorigin="anonymous"></script>
```

- Finally, open your "about.hbs" view and add the following (*lengthy*) HTML code to the very bottom of the page:

```
<div class="row">
  <div class="col-md-12">
    <h3>Leave a Comment</h3>
    <div class="well">
      <div class="row">
        <div class="col-md-12">
          COMMENT FORM
        </div>
      </div>
    </div>
  </div>
</div>
```

```
{{#if data}}
<div class="row">
```

```

<div class="col-md-12">
  <h3>All Comments</h3>
  <hr />
</div>
</div>
{{/if}}

{{#each data}}
<div class="row">
  <div class="col-md-12">
    <div class="well">
      <div class="row">
        <div class="col-md-12">
          COMMENT[{{@index}}]<br /><br />
        </div>
      </div>
      {{#each replies}}
      <div class="row">
        <div class="col-md-12">
          <div class="well">REPLY[{{@index}}]</div>
        </div>
      </div>
      {{/each}}
    </div>
    <div class="col-md-12">
      <button class="btn btn-primary btn-sm pull-right" data-toggle="collapse" data-target="#reply-
{{_id}}" aria-expanded="false">Reply</button>
    </div>
  </div>
</div>

<div class="collapse" id="reply-{{_id}}">
  <div class="well">
    <div class="row">
      <div class="col-md-12">
        REPLY FORM
      </div>
    </div>
  </div>
</div>
</div>
{{/each}}

```

- Save your changes and run your server using "node server.js"

- Navigate to your **/about** route and you should see the following new elements on the bottom of your page:

#### Leave a Comment

COMMENT FORM

#### All Comments

COMMENT[0]

REPLY[0]

Reply

- The "COMMENT FORM" block will always remain on the page, however the "All Comments" section will only be displayed if there are one or more comments passed to the view. The reason that you see "COMMENT[0]" and "REPLY[0]" is because our database contains **one** comment with **one** reply (from when we were first testing our new **dataServiceComments** module) and they were rendered using the `{{#each}} ... {{/each}}` helper.
- If you went ahead and added more comments during the testing phase, you will see more comments, ie:

#### Leave a Comment

COMMENT FORM

#### All Comments

COMMENT[0]

REPLY[0]

Reply

COMMENT[1]

REPLY[0]

Reply

- You will also notice a "Reply" button. If you try clicking it, you will see a "REPLY FORM" section appear underneath the corresponding comment. Here, we are leveraging Bootstrap's [collapse functionality](#). This makes things a little nicer for the user; rather than displaying a "REPLY FORM" container underneath every comment, we hide it until it's necessary.

## Updating our "about.hbs" view - forms & data

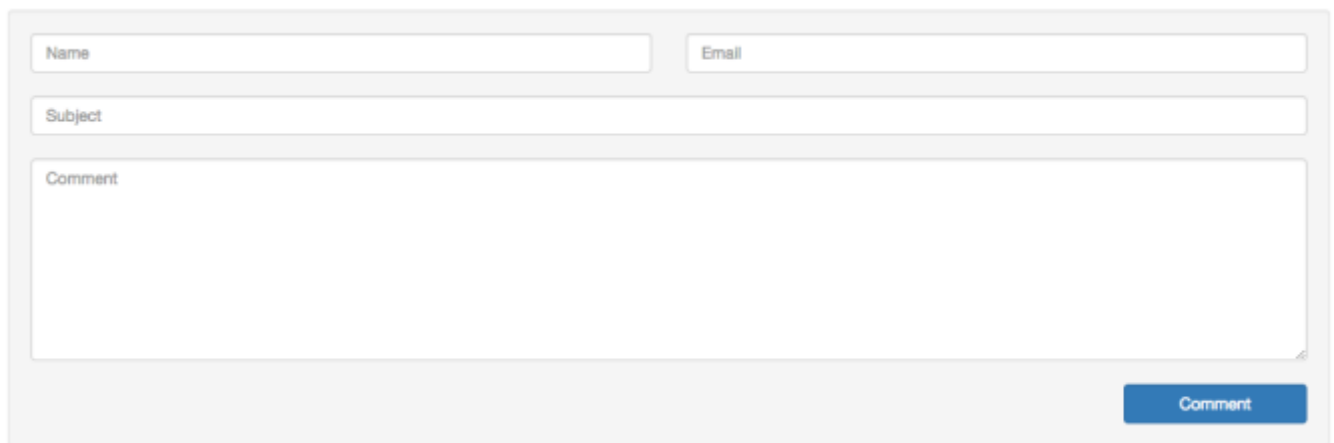
We now have the main containers in place to start creating our comment / reply forms and data output:

## "COMMENT FORM"

- The Comment <form> block must submit to **"/about/addComment"** using **POST** and contain the following fields:

form control type	name	attributes
text	authorName	placeholder="Name"
email	authorEmail	placeholder="Email"
text	subject	placeholder="Subject"
textarea	commentText	placeholder="Comment"
button type="submit"		( not an attribute, but must display the text <b>"Comment"</b> )

- Once complete, it should resemble the following image (you can see the HTML used to generate the form on the "Sample Solution" - [here](#) (all classes used are included in Bootstrap)



## "COMMENT[ ]" Text

- The text in the COMMENT[] area should be the actual text values of the comment itself, followed by the information for the user who submitted the comment. The following format should be used:

Comment Text 1

Posted on: Tue Jun 20 2017 18:05:52 GMT-0400 (EDT) by [Comment 1 Author](#)

- NOTE: The **"name"** ie: "Comment 1 Author" must be a "mailto" link using the author's email address
- Additionally, the space above the comment "well" (<div class="well">), must be the **Subject** of the comment (ie "Comment 1"):

### Comment 1

Comment Text 1

Posted on: Tue Jun 20 2017 18:05:52 GMT-0400 (EDT) by [Comment 1 Author](#)

## "REPLY[ ]" Text

- The text in the REPLY[] area should be the actual text values of the reply itself, followed by the information for the user who submitted the reply. The following format should be used:

Reply Text 1

Posted on: Tue Jun 20 2017 18:05:52 GMT-0400 (EDT) by [Reply 1 Author](#)

- NOTE: The "**name**" ie: "Reply 1 Author" must be a "mailto" link using the author's email address

## "REPLY FORM"

- The Reply <form> block must submit to **"/about/addReply"** using **POST** and contain the following fields:

form control type	name	attributes
text	authorName	placeholder="Name"
email	authorEmail	placeholder="Email"
textarea	commentText	placeholder="Comment"
hidden	comment_id	(not an attribute, but this value must be set to the <b>_id</b> of the current comment)
button type="submit"		( not an attribute, but must display the text <b>"Reply"</b> )

- Once complete, it should resemble the following image (you can see the HTML used to generate the form on the "Sample Solution" - [here](#) (all classes used are included in Bootstrap)

Name

Email

Comment

Reply

## Sample Solution

To see a completed version of this app running, visit: <https://fathomless-plains-77023.herokuapp.com/>

## Assignment Submission:

- Add the following declaration at the top of your server.js file:

```
/******  
* WEB322 – Assignment 06  
* I declare that this assignment is my own work in accordance with Seneca Academic Policy. No part of this  
* assignment has been copied manually or electronically from any other source (including web sites) or  
* distributed to other students.  
*  
* Name: _____ Student ID: _____ Date: _____  
*  
* Online (Heroku) Link: _____  
*  
*****/
```

- Publish your application on Heroku & test to ensure correctness
- Compress your web322-app folder and Submit your file to My.Seneca under **Assignments -> Assignment 6**

## Important Note:

- If the assignment will not run (using "node server.js") due to an error, the assignment will receive a **grade of zero (0)**.
- **NO LATE SUBMISSIONS** for assignments. Late assignment submissions will not be accepted and will receive a **grade of zero (0)**.
- After the end (11:59PM) of the due date, the assignment submission link on My.Seneca will no longer be available.