1. Requirement Critiques:
   1. Project 1: One thing that stood out to me was the part where it says, “It is guaranteed that the time between two readings from the same sensor will be an exact multiple of the sensor’s period.” In the real world, I feel like nothing is guaranteed and this might be an unusual assumption. Another thing I would have to say is that flexibility might be an issue. The index.js file is programmed specifically for the sensor api. For an assignment this is more than enough but for the real world it might not be as flexible as it should be (in case requirements change). Another thing that might be unusual is having the timestamped values all the same. In the real world, timestamped values might be different.
   2. Project 2: One thing that immediately pops out to mind is the lack of some sort of object relational mapping structure. In the real world, if requirements were to change (let’s say we do not want to use mongo dB anymore, we want to use PostgreSQL) it would require us to change the whole file by rewriting it in its entirety. This isn’t very flexible in the real world since requirements are constantly changing. Another issue with this is the function newSensors(mongoDbUrl). This function expects a specific URL format. Project 1’s issues are also valid for project 2 (regarding the sensor assumptions).
   3. Project 3: Projects 1 and 2 reasons are valid for project 3. One assumption that is wrong in the real world is compatibility with other machines. Here it is assumed to be used within school technology, in the real world tis would not be the case. Also, logging error messages to standard error is not something you would do in the real world, or at least not the only thing you would do.
2. You could reference a function so the source data would be a wrapper around the actual data.
3. var data = {
4. name: "form1",
5. fields: function() {
6. /\* iterate through source data \*/
7. return fields;
8. }
9. }

We can ten do {{#fields}}

{{/fields}}

Templates look something like this:

<Entities>

{{#form}}{{#fields}}

  <Entity><Name>{{name}}</Name><ID>{{id}}</ID></Entity>

{{/fields}}{{/form}}

</Entities>

<https://newcome.wordpress.com/2010/08/18/transforming-json-to-xml-using-mustache/>

1. On some older e-commerce web sites, the button which starts some kind of irrevocable action like placing an order has cautionary text saying something like *Please click this button only once*.
   1. HATEOAS is the term
   2. Use hyperlink differentiation
2. Criteria on the following:
   1. **SelectBoxes:** A select box takes less space, so if space is an issue use Select boxes. This is good to use when the user does not need to necessarily see all of the information at once (either a default value or scroll to choose your option). For example, Imagine a “Select your state” and it auto defaults to NY, imagine every state shown at you at once on the screen, this would be really ugly, so we use a select box to choose your particular state.
   2. **Radio Button**s: You use this when choices are mutually exclusive. For examples “Is cs444 a fun class?” Yes [ ] or No [ ], The answer is obvious (Yes it is a fun class)
   3. **Checkboxes:** This is used when there are a list of options and the user may select multiple options (zero, one, or more). Each box is independent. Ex: What outcomes did you get from cs444? Learned JavaScript [ ]. Learned MongoDb [ ]. Learned server implementation [ ]

A lot of these urls will be of some format of http:://HOST//some http method.

To allow listings of different types you can use a get method with some sort of id

For authentication https should be used.

To allow a customer to start building a pizza, a post could be used with the same url as above. Can be over HTTP. Cache frequently.

Adding addition topings can be a put or patch. Similar url as before.

Having a cutomer remove a topping with delete and some id/toppingId. Similar url as before.

To remove entire pizza also delete with an id. Similar url as before

To obtain details you can use the get with an id. Similar url as before

To view an image also with a get with id. Similar as before

Dispatch the pizza, probably HTTPS and post. Similar url as before

1. One way to still have compression and avoid the problem is to explicitly including a GZIP encoding filter in the web.xml,
2. 1. The answer to this question depends. The default behavior of HTTP/1.0 is to open separate TCP connection for each TTP request/response pair. In HTTP/1.1, pipelining was introduced. The underlying TCP connection can be partially controlled using the Connection header. HTTP/2 uses multiplexes messages over a single connection.
   2. True. The url (subset of uri) specifies were an identified resource is available and the mechanism of retrieving it.
   3. False. If I understand correctly, it is asking if file must be loaded in head? Obtrusive means JavaScript code within an HTML file. If files need to be downloaded externally and it needs additional JavaScript files then it should not be before the body tag.
   4. False. It is generally advised to put these files separately. It is necessary for readability and maintainability. While it is possible to mesh them together for a small site, you generally should not.
   5. False. Caching will store files downloaded by browser. You can set up when to fetch newer files but you should be able to count the number of times a page is loaded via JavaScript.