Live site:
http://52.38.127.124:3011/

The site is a simple list maker and brief obituary viewer of notable people who died in June of 2016. The data that can be saved is the person's name, the date they died, their age at the time that they died, their gender, and whether or not they outlived the life expectancy of the average American of their sex.

Testing

Data validity tests:

1) Input: Save entry with **no data**. Expected result: Entry is not saved, alert box indicating that the entry needs a name is displayed. No other alert boxes.

2) Input: Save entry with **no name**, but with other data. Expected result: Entry is not saved, alert box indicating that the entry needs a name is displayed. No other alert boxes..

3) Input: **Name, age, date**. Expected result: Entry saved, new entry is displayed on page.

4) Input: **Name, but no age**. Expected result: Confirmation box that informs the user that they did not choose an age and allows them to choose whether to continue anyway or cancel is displayed.

5) Test: **"OK"** in alert box mentioned in test case 3. Expected result: Entry saved.

6) Test: **"Cancel"** in alert box mentioned in test case 3. Expected result: Entry not saved.

7) Input: **Name and age, but no date**. Expected result: Confirmation box that informs the user that they did not choose an date and allows them to choose whether to continue anyway or cancel is displayed.

8) Input: **Name with no age and no date.** Expected result: Confirmation box that informs the user that they did not include an age is displayed and allowing them to choose to save anyway or cancel is displayed.

9) Test: Hit "**Cancel**" on save-without-age confirmation box mentioned in test 8. Expected result: Entry not saved, no additional dialogs.

10) Test: Hit "**OK**" on confirmation box mentioned in test 8. Expected result: Confirmation box informing the user that they did not choose a date and allowing them to save anyway or cancel is displayed.

11) Test: Type letters into "**date**" field. Expected result: No keyboard input.

12) Test: Type letters into "**age**" field. Expected result: No keyboard input.

13) Test: Type numbers into "**age**" field. Expected result: Numbers entered.

General integration tests:

14) Test: Edit entry, save values. Expected result: Entry has same id as it did prior to edit (id is displayed in the left-most column in table).

15) Test: Click triangle next to “date” field. Expected result: Current month drop-down, current day has square around it.

16) Test: Hit edit, do not change any values on edit page, hit save. Expected result: Entry unchanged to prior to edit.

17) Test: Assign gender as female for entry, save and click “view” button next to entry in table. Expected result: User is referred to as “she” on view page.

18) Test: Assign gender as male for entry, save and click “view” button next to entry in table. Expected result: User is referred to as “he” on view page.

19) Test: Do not check box indicated that the person exceeded the average lifespan, save and click “view” button next to entry in table. Expected result: View page includes message about how the person was taken to soon.

20) Test: Check box indicated that the person exceeded the average lifespan, save and click “view” button next to entry in table. Expected result: View page includes message about how the person lived a long life.

21) Test: Save entry with name, age, DoD, male, and checked box. Hit “edit” next to entry. Expected result: Entry page displays correct name in name field, correct age in age field, correct DoD in the date box, the male radio dial checked and the female dial unchecked, and the checkbox checked.

22) Save entry with blank age, no date, female, and unchecked box. Hit “edit” next to entry. Expected result: Edit page displays blank age, “female” radio dial checked and male unchecked, and checkbox unchecked.
23) Test: Enter name of person who is already listed. Expected result: Alert box indicating that the person is already in the database, entry not saved.

Test Results:

At the time of writing, 21 of the tests passed and there were two failures: Number 3 and number 22.

Number 3, the save data test (mostly passed but one aspect has failed):
My website adds a new entry with an ajax call from a client-side script to a server side script, which sends the box entries to the server to be saved, and then the server sends back aJSON of the database, which the client turns into a tree. For most of the time I spent working on this, my script would instantly add the entry when it was saved. Now, it does not display new entries until the page has reloaded. I guess something went wrong down the line that has interfered with the client script’s ability to either receive the updated JSON data properly, or to rebuild the table after it does.

Number 22, the duplicated entry test: I attempted to control for duplicate entries on the “name” field. To do this, I assigned the name of each person as the id of the <td> element that lists the person’s name, and then I attempted to avoid adding entries that were already listed in the tree. I tried using

```
if(document.getElementById(name))
/* Do not add entry */
```

where I had pre-loaded the variable *name* with the name that was attempting to be added. For reasons I cannot figure out, duplicate entries were added anyway. Then I tried

```
if(document.getElementById(name).length > 0)
    /* Do not add entry */
```

and javascript gave me the error “cannot evaluate length of null.” I inspected the “name” td elements and they were id’d as they were supposed to be (by their name), so I don’t know what the deal is.

Discuss how you used the idea of templating to display data.

In this project I used the handlebars template, which I learned in CS290. I have a server-side script which acts as the controller and controls the routes, and it sends data in as elements of a “container” in the edit page and view pages (which are in handlebars), where I display them with {{ data }.

Yesterday, before deciding to use my current set-up (which was based on my final CS290 assignment) I made another app that runs on Gradle with Velocity Template Engine. I was able to hook it up to an Amazon SQL database and run it from my home computer, however, I was not able to get the app running with Gradle running on a server.

Discuss if you had it to do over again, what changes you might make.

Today I opened up a Google Cloud Engine account and got the $300 free credit you can get when you sign up now, and I used a little of it to start a Compute Engine. I did the tutorial, and that showed me how to make a very cool To-Do application with MangoDB and node. Within five minutes I had the to-do list app live on my server, just following the tutorial. The live app looked just like this: http://todomvc.com/examples/emberjs/

I think I’m might try to use that for my future project if it’s not too complicated, though I had some trouble messing around with it. It seemed like it would be a lot faster though if i could figure it out, because I didn’t have to start and stop node to view changes: as soon as you save a file the changes to the site are saved. That would be a big time saver, because I spent a lot of time navigating between files in ssh this time.

I also found that the makers of the project had made that todo list using many different MVC’s, and they share them all to help people make an informed decision.

http://todomvc.com/

I downloaded all of the MVC's into one folder and am not sure which one looks the best to use. I'm scoping them out now and still considering options.