


Hey y'all, just a quick summary of lab.

Environment Variables

1. When working in your c9 environments, please run the following terminal command:

```
# A similar thing has been done in heroku already. Link up with steeve to see  
export DBURL=mongodb://localhost/<workspace name>/<path to local db>
```



2. Moving forward, when we connect to a db using mongoose, we'll use this line in index.js:

```
mongoose.connect(process.env.DBURL, {useNewUrlParser: true});
```

This allows us to switch to using our local database when on c9, and the production database when the app is deployed, no mess, no fuss.

Github

Tasks

- Tasks have been assigned for the majority of the usability requirements. Please don't feel bound to only this/that task/role for the rest of eternity just because it's assigned to you right now.
- The tasking is there as more of a guideline as to what needs to be done, and who the [loop will be closed with](#) as far as status.
- That being said, if you wanna get exp with a certain aspect of the app, feel free to implement it yourself as well. Of course the assignee is the one whose implementation will probably be in prod, but feel free to co-op with them if they'd like/want/need a second hand.
- Of course, if everyone does the task assigned to them and gets it done quickly, we can ensure that we are making steady progress. That way, we can more quickly wrap up the

compulsory requirements and have time to explore more fun things like new features/design, etc.

Issues and Projects View

- Issues view is pretty garbage tbh.
- Under Projects/Contact Manager, you can see a nice little Kanban-style board for what everybody's working on.

Git Flow

In general, the workflow should be something like this:

1. Create a new branch for the issue/feature.
2. Develop new feature.
3. Push your new branch to origin.
4. Open a Pull Request (PR).
5. Have someone review your PR.
6. If everything's cool:
 - They'll approve your PR
 - You merge your PR with "Create a merge commit"
 - Delete your branch (if you're done with it).

Otherwise: Make suggested changes, do 4-6 again.

7. Close the issue.

- It may be convenient to have your counterpart review your PR, since they might be depending on it.

App Walkthrough (Routes View)

Accessible by default (User's not logged in):

| PAGE | GET | POST | REDIRECT | REDIRECT |
|--------|---------|---------|----------|----------|
| Signup | /signup | /signup | /login | /signup |
| Login | /login | /login | /home | /login |

Accessible only when User logged in:

| PAGE | GET | POST | REDIRECT | REDIRECT |
|----------------|----------------|----------------|----------------|----------------|
| Home | /home | x | x | /login |
| Add Contact | /addcontact | /addcontact | /home | /home |
| Search Contact | /searchcontact | /searchcontact | /searchresults | /searchresults |
| Delete Contact | /deletecontact | /deletecontact | /home | /home |

App Walkthrough (Task View)

Recall from [init_tasking.md](#)

LEGEND

auth

backend

bug

database

development

duplicate

enhancement

frontend

good first issue

help wanted

invalid

presentation

question

RESTful

wontfix

■ User must be able to sign up.

1. User shown sign-up form (Wyatt)
2. After user enters info, form is submitted to '/signup' (Wyatt)
3. Backend pulls info from request body of form. (Nyasha)
4. User added to DB. (Nyasha), (Rolando)
5. User redirected to homepage login. (Nyasha)
6. ~~User's info dynamically loaded into homepage.~~ User goes to log in. (Nyasha)

■ User must be able to be logged in.

1. User at splash page. (Abe)
2. User shown login form (Affner)
3. After user enters info, form is submitted to '/login' (Affner)
4. Backend pulls username/email + password from request body of form. (Affner) , (Rolando)
5. Username + pass searched for in DB. (Rolando)
6. If success:
 1. User re-directed to homepage. (Affner)
 2. User's info dynamically loaded into homepage. (Affner)

else:

1. Error message displayed, e.g, "user/pass not found" (Affner)
2. User redirected to sign-in again. (Affner)

■ User must be able to add Contacts.

1. User shown form for Contact info. (Anthony) , (Rolando)
2. After user enters contact info, a form is submitted. (Anthony)
3. Backend pulls Contact info from request body of form. (Wittel)

4. If success:

1. Contact added to DB (Wittel)
2. Contact pushed to matching User's Contact array. (Wittel)
3. User redirected to homepage. (Wittel)
4. User's info dynamically loaded from DB. (Wittel)

else if Contact already in DB:

1. User alerted that Contact is already in DB
 - (which params to check? Phone? B-day?) (Wittel)

else:

1. User shown error message. (Wittel)
2. User shown Add Contact form again. (Wittel)

■ User must be able to search for Contacts. (AJAX?)

1. User enters search criteria. (Wyatt)
2. DB lookup. (Stephen), (Rolando)
 - Lookup by {firstName, phone, etc}
3. If Contact found, load 'individual' view for contact. (Stephen)
 - ~~could be just pop-up~~
 - could be separate page. (Anthony)
 - ~~could just reload homepage with only that Contact displayed in it.~~

■ User must be able to delete Contacts.

1. User in Contact's 'individual' view. (Abe)
2. User presses delete contact button. (Anthony)
3. (Optional) confirmation alert shown (Anthony)

4. If sure:

1. Remove Contact from User's Contact array. (Stephen)
2. Lookup Contact in DB. (Stephen), (Rolando)
3. Remove specified contact from DB. (Stephen)

Else:

4. Do nothing.

REST APIs

- Just putting it on your radar that we are required to have an API for our app, and that it must be RESTful.

Backend Peeps: Merge Conflicts

- Anyone working on a backend task should be aware of the (likely) merge conflicts you'll face.
 - Basically, since you'll all be modifying index.js, you will often need to manually discern which changes to keep when.

- Circumvent this by pulling from ~~main~~ **master** often.

- If you run into a conflict when pulling, i.e.

```
$ git pull
```

Cannot pull with rebase: You have unstaged changes. Please commit or stash them. :

```
# Store your changes for safekeeping.
```

```
git stash save "description of the changes I made"
```

```
# Pull from master to stay up to date.
```

```
git pull origin master
```

```
# Reapply your changes after getting updated.
```

```
git stash apply
```

```
# push your (now merged) changes to your branch
```

```
git push origin <your branch name>
```

Frontend Peeps: AJAX

- For now most of our frontend will be
 - displaying(GET) forms,
 - submitting(POST) forms,
 - and redirecting forms.
- Once we get meet all usability requirements, we will want to see if we can AJAX-ify some of those frontend elements.
- Be thinking about which elements would actually *need* a reload and which can be "brought to front" instead. The ones that don't necessarily need a page reload will be prime candidates for using AJAX (another development requirement from Leinecker).

Peeps

- If you have a frontend task, see if you can coordinate with whoever has the backend counterpart of your task, and vice-versa.
 - For example, the person doing the frontend for Add Contact (Anthony), would sync with the person doing the backend for Add Contact (Wittel)
 - This isn't a hard and fast rule, it's really so there's synchronicity between the two halves, such that the backend knows what to expect from the frontend, and the frontend knows what to send to /get from the backend