CAFEBABE - Elect.io

Andrew McClusky, Shahar Dahan, Neal Merkl, Elaina Bliss, Alex Karle, Alex Cobian

Submission 6

4/19/16

Special Server Setup Procedure: Same as submission5, need email file

Before anything can be run, npm install must be run in both client and server, and a file must be created '/server/src/emailaccount.json' (See private piazza post for details!). Make sure you reset the DB for emailing to work.

List of HTTP requests and routes:

- Get '/weeks'
    - Gets all the "weeklyStates" which are the objects which store user votes, used in trends
    - Anybody can make this request but it doesn't really matter because the info gets displayed anyway.
    - Accesses the "weeklyState" collection in the db
- Get '/users/race/gender'
    - Gets all the race and gender of all the users , used in trends
    - Kind of broken, in that anyone can see the race and gender of other users.
    - Accesses the 'users' collection in the db
- Get '/events'
    - Gets a list of all events
    - Anybody is authorized to make this request
    - Accesses the 'events' collection in the db
- Get '/events/:page', :page specifies which events to retrieve
    - Gets 3 events, which are decided based on :page
    - Anybody is authorized to make this request
    - Accesses the 'events' collection in the db
- Get '/candidates/independent'
    - Gets a list of all Independent-Candidates
    - Anybody is authorized to make this request
    - Accesses the 'candidates' collection in the db
- Get '/users/:userid'
    - Gets user data for a specific user id
    - Must be authorized as this specific user of id :userid to make this request
    - Accesses the 'users' collection in the db
    - Elaina was responsible for this part to implement it into the user settings page
- Put '/users/:userid'
    - Sets the user data for a specific user id (parameter :userid)
    - Must be authorized as this specific user of id :userid to change any of the user's data

    - Uses a schema to check the body sent which is a full user json object like those in the db
    - Accesses and modifies the ‘users’ collection in the db
    - Elaina was responsible for this part so it could be used in the implementation of the user settings page
- Get '/users/:userid/fullName'
    - Gets the username of a user of id parameter :userid
    - Does not have authorization (anyone is authorized), as this information should be public in order for it to be used in the chat or other locations.
    - Accesses the ‘users’ collection in the db
- Get '/users/:userid/party'
    - Gets the user’s political affiliation for user of id :userid
    - Does not have authorization (anyone is authorized), as this information should be public in order for it to be used in the chat or other locations.
    - Accesses the ‘users’ collection in the db
- Get '/parties/:partyid'
    - Gets the party from a specific party id (parameter :partyid)
    - Anyone is authorized to make this request
    - Accesses the ‘parties’ collection in the db
    - Elaina is responsible for this part
- Get ‘/candidates/party/:partyid
    - Gets all candidates of the given party (an integer parameter :partyid)
    - Anyone is authorized to make this request, as the data is public
    - Accesses the ‘candidates’ collection in the db
    - Alex K was responsible for this part
- Get ‘/candidates’
    - Gets all candidates, ordered alphabetically.
    - Anyone is authorized to make this request, as the data is public
    - Accesses the ‘candidates’ collection in the db
    - Adapted by Andrew
- Get ‘/candidates/id/:candidateId’
    - Gets a single candidate, by their id number.
    - Anyone is authorized to make this request, as the data is public
    - Accesses the ‘candidates’ collection in the db
    - Adapted by Andrew

Individual Contributions:

Alex K: Vote feature, Updating Trends in Backend

- Get '/candidates/party/:partyid
- Get '/candidates/independent'
- I created a backend setInterval which accomplishes our goal of reading the db and creating a new weeklyState (a snapshot of how everyone voted at that time). In real life this would run every week, but for dev and grading purposes, it creates a new weeklyState every 20 seconds. Check it out! You need to refresh the page to see the new option. If there are too many, simply reset the db.

Alex C: Trends feature

- Get '/weeks'
- Get '/users/race/gender'

Andrew: Home feature

- Get '/candidates/id/:candidateId'
- Get '/candidates'

Elaina: Settings feature

- Get '/users/:userid'
- Put '/users/:userid'

Neal: Archive feature

- Get '/events'
- Get '/events/:page'
- Started work on fixing pagination in the archive

Shahar: Calendar feature

- Get '/users/:userid/fullName'
- Get '/parties/:partyid'
- Get '/users/:userid/:politicalAffiliation

Known Bugs:

- Archive page: the pager is currently hardcoded to flip through only 2 pages. Scrolling through pages does not render such that the old events are replaced by the new events. Checking the react developer tools shows that the page correctly retrieves relevant data (3 newer or older events), but the page doesn't display these events correctly. Work for this is contained in the 'pages' branch.

Shahar Dahan’s Honors Feature: Chat

- Get '/chat'
    - Gets a list of all the possible chats.
    - Anyone can make this request.
- Get '/chat/:chatId'
    - Gets a single chat and shows messages appropriately
    - Anyone can make this request as of right now, perhaps in the future you’ll have to be signed in.
- Post '/chat/:chatId/messages/'
    - Posts a message to the current chat.
    - The user posting must be authorized as the post’s author.
    - There is a schema to make sure that the right information is put into the message.

These were updated from submission 5 to include the use of MongoDB instead of the mock database.

An outstanding bug is that there is a delay between when the page originally loads and when it translates messages / names to their actual Strings. This makes the chat page load in a funky manner.

Alex Karle's Honors Feature: Email Settings

Special Server Setup Procedure: **Same as Submission 5!**
Follow instructions for regular project (make sure you read the private piazza post for stuff regarding the email address!)

HTTP requests and routes **Now updated to use Mongo!**:

- Get '/users/:userid/emailsettings'
    - Gets the user's email settings
    - Replaces getEmailSettings() in mock db
    - Only the user with :userid is authorized to get this data (used code similar to facebook).
    - Accesses the 'users' collection in the db
- Put '/users/:userid/emailsettings/:candid'
    - Adds a candidate of id :candid to the user's email subscription list (the user of id :userid)
    - Replaces subscribe() in mock db
    - Only the user with :userid is authorized to modify this data (used code similar to facebook).
    - Accesses and modifies the 'users' collection in the db by adding the candidate of id :candid to the user's email settings
    - Emails the user about the subscription
        - Note: the email who it sends it to is NOT the email from the db (although in real life it would be). The email it sends to is the devRecipient address in the created server/source/
- Delete '/users/:userid/emailsettings/:candid'
    - Removes a candidate to the user's email subscription list (the user of id :userid)
    - Replaces unsubscribe() in mock db
    - Only the user with :userid is authorized to modify this data (used code similar to facebook).
    - Accesses and modifies the 'users' collection in the db by removing the candidate of id :candid from the user's email settings
    - Emails the user about the unsubscription
        - Note: the email who it sends it to is NOT the email from the db (although in real life it would be). The email it sends to is the devRecipient address in the created server/source/

Notes about emailing **Same as Submission 5!**:

Emails get sent for 3 reasons:

1. Candidate subscription: This is simple, in the http request I send an email back. Right now this sends to the developer (because I don't want to send to fake address). It is a quick change to send it to a real user's email property of their json object in the db.
2. Candidate Unsubscription: Same as above
3. Upcoming events: This one is a little more complex. Basically the server has an iterator that runs every 30secs right now (so I don't overload it) and checks if there is an event that is coming up within 2 weeks time that users havent been notified about (events now store a boolean whether or not an email has been sent out). In the real deployed version, this 30 seconds would be 24hrs, as it would only check once a day, but for now I needed a number reasonable enough that I (and you) could see the results. Note that you will have to reset the DB so that the boolean flips to false and then wait 30 secs for the email to send, but it will send. I have code in place that actually runs through every user and matches their subscription with the affiliated candidate of the upcoming event (and this works!), and I left it there for you to check out (just uncomment the console log!). It works, and in the deployed version with actual users with real emails all I would change is that in the mailOptions I would take the emails from this array of emails I made, instead of sending to only the devRecipient (you)

Please check piazza for a private post about how to configure the email stuff! It is needed for all server things now (since an error will occur if its not there). Note that every time you refresh the DB it will email you; if this gets old / tedious / annoying, just set the devRecipient to the email we are sending from.

NOTE / Possible Bug:
It recently came to my attention that one of my teammates was unable to send emails because google rejected their login attempt (saying there have been too many login attempts). I am unsure how to get around this, and hope that the login works for you (it has not yet failed for me).