CS 169A Group Project

# CHIP 10.5: Agile Iterations

Welcome to CHIP 10.5!

The goal of this assignment is to experience a small-scale version of real agile software development as it is typically practiced in industry. In particular:

* You will work in small teams of 4-5 (rather than alone or just in pairs).
* Your team will work on an existing codebase (rather than creating a new app from scratch).
* You will use the same tools to track tasks/issues, do automated testing at checkin (continuous integration), measure code coverage and quality, and deploy to the cloud as agile software houses use.
* The unit of work will be the user story; the unit of measuring project time will be the iteration; your team’s planning will involve delivering one or more stories by the end of each iteration.

In this multi-part assignment you will do some prep work followed by two iterations.

**In Iteration 0 (prep work)** you will get the existing app running in each developer’s environment and set up the infrastructure for team development (repos, deployment, CI, and so on). The existing codebase is a nontrivial full-stack app called ActionMaps that lets users learn more about their elected and appointed government representatives, learn about political events in their local area, and aggregate, share and view news items in their local area.

**In Iteration 1,** all teams will add the same basic subset of features to the app, using the agile/XP workflow based on user stories, BDD/TDD, and lo-fi mockups.

**In Iteration 2,** each team will select a set of additional features to implement.

**NOTE.** It is very hard for us to prove that you followed BDD/TDD and the agile/XP workflow in developing the app. However, if you try to code the way you always have, we guarantee two bad things will happen. First, you will run into trouble when you try to coordinate the results of teamwork by merging code. Second, you will completely miss the experience of how software is developed in real agile teams, which is largely the point of the class, and you won’t be in a great position during interviews to talk about your views on agile. In working on this assignment, the process is as important as the end product.

## The App and the Project

A major difference between this project and earlier CHIPS is that there will be very little hand-holding. We’ll start you off with some legacy code, which will be a basic Rails app with a few models already implemented. The app uses an external API (Google Civic Information), JavaScript code to render the maps, and an asset pipeline (Webpacker) that manages all the front-end files.

Another important part: testing! Throughout this whole project, you’ll be expected to add tests where you see fit, leveraging both BDD with Cucumber and Capybara, as well as TDD with RSpec. When it comes to stubbing external APIs (a big portion of this project), we’ll give you some extra tips on how to do so.

Navigate to the next page to get started with Iteration 0 and setup!

Current layout: 2 Panels with treePanel 0: terminal command:

**2. Iter 0: Getting Started**

**2. Iter 0: Getting Started**

**Iteration 0/Getting Started: Individual Setup**

All but the last two steps of this setup process should be done by *each* member of the team to get the app running in their local environment and ready for development. The last two steps are only done once, on the team’s shared repo.

1. Because this project makes nontrivial use of JavaScript, ensure the correct versions of Node.js and the Yarn package manager for JavaScript are installed.
2. Like all nontrivial apps, this one depends on libraries–both Ruby libraries (“gems”) and JavaScript libraries (“packages”). You will use bundle to install Ruby dependencies as usual, and yarn to install JavaScript package dependencies
3. Create and seed the Rails database by running the initial migration.
4. Check the configuration of our provided *linters* —programs that check your source code for common errors and potential issues related to coding style. These are run when you add or modify code to the codebase and serve as an early warning that you need to fix stuff.
5. Setup credentials management. The maps API used by the app requires API keys, and these should be stored in a secure way that makes them available to each developer, to Heroku for deployment, and to the CI environment in the team’s shared repo.
6. Finally, in the team’s shared repo, setup continuous integration (CI) so that tests run each time you push code. Note that this is in addition to each developer regularly running tests as they develop. Particularly after merges of pull requests, CI provides one more layer of confidence that you haven’t broken something.

Ready? Here goes.

**GitHub**

**Initial Setup (Do now)**

Similarly to previous CHIPS, your team has been added to a su24-actionmap-<xx> repo where the <xx> is replaced by your team’s number. What’s different this time around is that instead of only one member being in charge of making PRs to the repo, you will all be responsible for maintaining the repo together. Start by cloning in the repo using:

git clone git@github.com:cs169/su24-chips-actionmap-<xx>.git actionmap

As with previous CHIPS, once you’ve gotten the repo connected to GitHub, you’ll need to setup your own (or your pair’s) branch on which you’ll be making edits and pushing to remotely. Run the following commands, replacing <GITHUB\_USERNAME> with (believe it or not) your GitHub username:

git checkout -b <GITHUB\_USERNAME>

git push -u origin <GITHUB\_USERNAME>

**Commit frequently (For future reference)**

You will now have a local branch to make your edits on that is connected to an identically named remote branch on the GitHub repo to which you should push frequently using the following commands:

git status # review your changes

git add [...] # you will need to include your own files!

git commit -m "your message here"

git push origin

**PR once you’ve finished your part (For future reference)**

Once you’ve finished a story or feature, it’s time to get your changes on the main branch of your GitHub repo through a Pull Request (PR). This can be done directly through the GitHub site by clicking the “X branches” button near the top of the repo page.

A screenshot of a phone

Description automatically generated

From here, find your branch and click New pull request.

A screenshot of a black screen

Description automatically generated

Now make sure that the base branch is main and that the compare branch is your own. Then add a title and a description of the changes you’ve made as well as any other info that you’d like your team to know about your PR. Once you create the PR, let your teammates know so that they can check out the changes and approve the PR.

A screenshot of a computer

Description automatically generated

Protocols vary from team to team, but in general, it is good practice to have at least 1 or 2 teammates review your changes to ensure that no bugs creep through to the merge. Once the reviews are in, all comments addressed, and any potential merge conflicts are resolved, you can merge in your changes!

**Codio**

The Codio stack for this assignment comes with the following already installed, but if you’re not using Codio, you will need to install these yourself:

* nvm, which allows you to manage multiple active Node versions (just as rvm allows installing and managing multiple Ruby versions)
* Node.js 13
* yarn 1.22.4, the default Node package (library) manager for [Rails Webpacker](https://github.com/rails/webpacker)
* bundler 2.1.4
* libpq-dev 10.22-0ubuntu0.18.04.1, an apt package necessary for using PostgreSQL in the local (development) environment

**Install dependencies in your local environment**

Now you can install the gems, and as usual, for local development, we recommend you skip installing production dependencies:

cd actionmap

bundle install --without production

If not using Codio, this step may take a while since some of the gems include Ruby extensions written in C that must be compiled as they are installed.

In the package.json change the node and yarn versions on line 8 and 9 to match the following:

"node": "13.x",

"yarn": "1.19.x"

Next we install the JavaScript libraries used by Node. Node projects have a package.json file that is analogous to the Gemfile, and running yarn produces [yarn.lock](https://matrixdragon-vocalperson.codio.io/yarn.lock), analogous to Gemfile.lock. (JavaScript projects that don’t use yarn instead have a file package-lock.json. Be sure you only have one of these and not both in your project since they may diverge and lead to inconsistent environments for different developers. We have added package-lock.json to .gitignore to help avoid this.)

nvm install 13

nvm use 13 # You will need to run this in every terminal where you're using node.

yarn install

**Run migrations and seed the database**

The app is almost ready for launch. You need to run database migrations in [db/migrate](https://matrixdragon-vocalperson.codio.io/db/migrate) to prepare your local database to store and serve data, and add the seed data to it:

bundle exec rails db:migrate

bundle exec rails db:seed

(Note: what kind of data is in db:seed? Do you agree this is a good use of the seed file?)

**Launch the app!**

We’re ready to run the app!

Normally, we would run the app using

bundle exec rails server -b 0.0.0.0

However, in this case, a downside of this option is that JavaScript-related changes may not be visible right after you make them. Essentially, when you change Rails app files, Rails knows to “reload” any changed Ruby classes on the very next request to your app; but because we’re doing separate JavaScript package management with yarn, the same isn’t true for JavaScript, CSS, or other front-end files.

Instead, we recommend you have *two* terminals open when developing. In one terminal, execute:

bin/webpack-dev-server

This will launch a webpacker instance that live-reloads your browser as you edit the javascript code or CSS styles and makes the development process much faster.

After launching webpack-dev-server on one terminal, switch to the second terminal window and launch a Rails server from there as usual:

bundle exec rails server -b 0.0.0.0

**Using Linters**

A linter is a program that analyzes source code for common errors and conformance to a coding style. The original [lint program](https://en.wikipedia.org/wiki/Lint) was written in C and checked C source code; today the term “linter” has come to be generalized to any program that does this task for a particular language or framework.

We have installed three linters in this project. [rubocop](https://github.com/rubocop-hq/rubocop) checks Ruby code for common errors and stylistic problems; we provide a default configuration file [.rubocop.yml](https://matrixdragon-vocalperson.codio.io/.rubocop.yml) telling it what to check for. [eslint](https://eslint.org/) similarly uses [.eslintrc.js](https://matrixdragon-vocalperson.codio.io/.eslintrc.js) to check for common errors and stylistic problems in JavaScript. Finally, [haml-lint](https://github.com/sds/haml-lint) finds potential problems with [Haml](https://matrixdragon-vocalperson.codio.io/haml.info), a very compact markup language that generates HTML. This project uses html.haml files in app/views rather than .html.erb, because Haml is supported by more programming languages (including Java, JavaScript and PHP) and frameworks, whereas .erb is specific to Ruby. This makes it easier for non-Ruby developers to work with the codebase. [.haml-lint.yml](https://matrixdragon-vocalperson.codio.io/.haml-lint.yml) is its configuration file.

You can manually run these linters:

bundle exec rubocop

yarn run lint

bundle exec haml-lint

Remember that, if you open a new terminal, you may need to rerun nvm use 13 in the new terminal so that the correct node version is used.

If you add a -a option to rubocop, or run lint\_fix instead of lint, those linters will automatically fix common errors for you. haml-lint [can’t yet do this](https://github.com/sds/haml-lint/issues/217).

(For yarn run, note that in [package.json](https://matrixdragon-vocalperson.codio.io/package.json) you can see that lint and lint\_fix are entries in JSON path $.scripts.lint and $.scripts.lint\_fix that invoke our “vendored” install of eslint, the actual underlying program.)

We have included tests that will automatically run the linter checks in [spec/linters](https://matrixdragon-vocalperson.codio.io/spec/linters) and in the GitHub Actions workflow that runs on each push.

The next steps involve creating credentials and setting up CI and issues, which you should do as a team.

<https://docs.github.com/en/actions/security-guides/encrypted-secrets>

<https://github.com/commitizen/conventional-commit-types/blob/master/index.json>

Current layout: 2 Panels with treePanel 0: terminal command:

**4. Iter 1: Iteration Instructions**

**4. Iter 1: Iteration Instructions**

**Iteration 1: General Instructions**

The project will be completed in 2 iterations, following the Agile/XP methodology. Iteration 0 was the setup; the remaining 2 iterations are for doing the actual code and tests!

Both iterations will follow the same general structure:

1. Have a detailed *pre-iteration planning meeting* to discuss the work with the team and break it up into stories
2. Create an Issue corresponding to each story and assign developer(s)
3. Developer(s) use BDD and TDD to create tests and code for the story, doing Pull Requests when the feature is believed complete; others review the PRs and eventually decide to Merge and then close the Issue.
4. App is constantly pushed to Heroku to make sure everything still works in production
5. At end of iteration, the team as a whole submits a *retro* (retrospective survey), and each developer submits a confidential *peer evaluation survey* and *self-assessment survey*. The grading of the code and tests is based on your deployed Heroku app and the contents of the team repo.

The rest of this guide details each of the above steps; the next guide lays out the feature tasks for Iteration 1. **Please read this guide completely before going on.**

**Pre-iteration Planning**

For each iteration we’ll give you specs of what to do. You should think of the project spec as notes from a customer who knows *what* feature they want but not *how* to implement it. You should make lo-fi mockups if needed in order to create user stories from the spec.

Your team should have an iteration planning meeting (IPM) to discuss the stories. **A key goal of this meeting is to break down the stories into smaller ones until each story is something that can be finished by one person or a pair of people within about 1 day’s work.** In a full XP workflow, the IPM would include assigning points to stories; in our simplified version we’re essentially recommending you make all the stories 1-point stories.

A good rule of thumb is that if a developer or pair can generally describe in plain language what would have to be done to implement a story (e.g. “I need to add a route, a controller action, a view, and a few lines of logic in the model”), it’s more likely to be a 1-point story. If you’re not sure what needs to be done, you may need to get more specific, or break the story down into smaller chunks until each chunk is concrete enough that you can describe the tasks needed at a high but specific level.

**The planning step is extremely important.** If you can’t get to specifics at the time you’re assigning a 1-point story, chances are you won’t be able to get to specifics when you sit down to code either. This “translation” step has to happen at some point, so why not do it when the whole team is available to discuss and help?

You will report on how this meeting went using the checklist assignment found on Gradescope (and linked in the next page of this guide).

**Create Issues for the stories**

Each 1-point story should be entered as a GitHub Issue. Label each issue with “iter1” or “iter2” depending on which project iteration it is for.

When a developer or pair claims an issue to work on:

1. Add the label “started” to the issue, so others know the story is in progress
2. [Assign the issue](https://docs.github.com/en/issues/tracking-your-work-with-issues/assigning-issues-and-pull-requests-to-other-github-users) so everyone knows who is working on what.
3. Use branch-per-feature (book section 10.2) to manage your work: [create a branch that is linked to the corresponding issue](https://docs.github.com/en/issues/tracking-your-work-with-issues/creating-a-branch-for-an-issue) and do your work there.
4. Either create a new Cucumber feature file, or if appropriate, add a scenario in an existing file, for the issue you’re working on. You should write out all scenarios for that user story as we have done in class. Also write draft step definitions for all of the new steps that you are using, even if they are just placeholders for now. If you’re updating an existing feature, you can just edit the feature file(s) for that feature.
5. Use BDD to develop the feature, as in CHIPS 7.7.
6. When committing, specify the commit type (from [this list](https://github.com/commitizen/conventional-commit-types/blob/master/index.json)) at the beginning of the commit message with "[]". For example, a commit message will look like "[feat] A new feature that…(description here)".
7. Create a Pull Request to the master branch when a feature is finished. You can use rebase to resolve conflicts on your branch. **Remember** that a finished story means *both tests and code*: a commit or PR should never cause net test coverage to decrease. Comments & discussion as part of the PR form an integral part of keeping everyone on the team apprised of what’s happening.

**Setting Up a GitHub Project Board**

GitHub has a feature called "Project Boards", which work a lot like Pivotal Tracker and similar kanban style tools.

1. Navigate to your repo on GitHub and find the link which says "Projects".
2. Click the **dropdown** in the big green button, choosing "New Project". **Do not** click “Link a Project” directly (this will take ownership over the template!)
3. You should make a project from a *template*, in the column on the left. This includes all the automation to move stories to the right state as you open and close PRs. Select "[TEMPLATE] 10.5 Backlog Template" as the template to use.
4. Give your project board an appropriate name, like “Team CHIPS 10.5 Project”

**Standups**

**In every offering of this course we have taught,** the major student feedback on lessons learned from the group project was “I wish our team had communicated more frequently and more regularly.”

Our recommended workflow: every day (or as close to every day as possible), meet with your teammates as a group in a simple video call for just a few minutes. The meeting can follow this simple script, where each developer takes turns answering the questions:

* What have you worked on/finished since our last standup?
* What are you planning to work on today?
* Have you encountered any problems that others on the team could help with, or learned anything interesting that others might benefit from?

You don’t need to submit evidence of this meeting, but we *promise* your dev process will be *much smoother* if you actually take advantage of team knowledge with frequent-but-short meetings rather than trying to meet once for a long time during the iteration.

**End of Each Iteration: Deliverables**

1. Retrospective survey (1 per team): Comparing your planned iteration work with the stories actually Finished or Delivered, what could your team have done better? Suggestion: you can use the Postfacto tool to scaffold this meeting ([Usage guide](https://drive.google.com/drive/folders/18FCUo2wOI0hKJuPh7BuGyenZuFSPtK8A), [Tutorial Video](https://drive.google.com/drive/folders/18FCUo2wOI0hKJuPh7BuGyenZuFSPtK8A)).
2. Peer evaluation survey (each developer submits): We will ask you to evaluate the overall contributions of each team member during the iteration—exceeds expectations, meets expectations, somewhat/barely meet expectations, or fall short of expectations. (Consider a variety of factors: did this person communicate with rest of team effectively? Did they try to do their share of the work? Were they prepared to work on the project, that is, did they seem to have command of the material covered in the homework?) These surveys are confidential to instructors only. We use this information to help with project grading, so please be honest and fair!
3. Code and tests
   * Your GSI will expect to be able to interact with the features for Delivered stories and inspect code, tests, etc. for Finished stories. The goal is to be able to Deliver all the stories you planned, but the goal is not to punish your team for failing to do so but rather to help your team understand what could have gone better so that next iteration will go more smoothly.
   * Your code coverage is 55% or higher (for app/ folder). For Iteration 1 you will need >45% and by Iteration 2 you should have >55% overall coverage.

Your GSIs may apply a subjective component based on the quality of your stories, interaction within team, and so on.

**Next, continue with Iteration 1 specs and deliverables.**

Current layout: 2 Panels with tree

**5. Iteration 1: Spec and Deliverables**

**5. Iteration 1: Spec and Deliverables**

**Milestone 1: Implement a couple of new key features**

**App Overview**

ActionMap’s goal is to be an integrated, seamless, and shareable platform that makes it easier for voters to connect with the progressive community while at the same time enabling progressive organizations, candidates, and elected leaders to reach new activists. The idea behind the application is to allow the user to visualize the political environment within all levels of government while also providing a platform to contact and voice their opinions to the decision makers within politics. This happens through attending events in their community, and sharing news articles related to a particular candidate and sharing their opinion. By asking the user to provide a news article before giving the candidate a score on a particular issue, it adds a layer of reputability to the score. Users can discover candidates by clicking on a map location, or searching for their local address.

The web\_steps.rb file that was present in CHIPS 7.7 is also provided as starter code here, but you will probably want to create your own scenario steps rather than using these basic ones.

**Understanding the code base**

To render the maps, this project uses a combination of JavaScript and [Topojson](https://github.com/topojson/topojson-specification), a compact alternative to [GeoJSON](https://tools.ietf.org/html/rfc7946). These file formats are both used to represent simple geographical features.

You will find prebuilt Topojson files in assets/topojson, but we need the JavaScript d3.js library to render the maps as SVG images. This library is fetched using yarn from npmjs.com. In addition, we use the Webpacker gem (which wraps the Webpack front-end library manager) to compile JavaScript, SCSS, CSS and their dependencies. You will find the JavaScript that renders the browser in app/javascript/packs.

We also use JQuery to listen and react to DOM events eg. when a form value changes. Open the /events path on your browser and assess how the Filter By form works. What JavaScript file handles the changes in the form?

**Intended User Experience**

A user can search for candidates in one of two ways:

* **Entering their address into the search field** -OR-
* **Clicking on a state in the US Map**.
* When a state is clicked, it will redirect the user to that state’s map, on which the state’s counties will be selectable. When a county is selected, it will populate the table below with that county’s candidates.

Once on a particular place’s representatives list, a user can view any one representative’s associated news articles, add their own, and score it.

**Iteration 1, Part 1: Representatives - Fix Legacy Code**

The Representatives model is the first thing we’ll be working on as part of this app. It has some basic functionality.

Let’s take a look at what’s already there:

We have a representative resource with basic views, model, and controller. The model includes the representative name, Open Civic Data ID (OCD ID), office/title, and a has-many association to news\_items.

We also provide RepresentativesController and SearchController. The SearchController handles calling the Google Civic API. Below is the schema for representative model.

| **OCD ID** | **Representative Name** | **Representative Title** | **Created At** | **Updated At** |
| --- | --- | --- | --- | --- |
| For use with Google Civic API | For use with Google Civic API | For use with Google Civic API | Auto-generated | Auto-generated |

**TASK 1.1**: Refactoring Legacy Code

* There is currently an issue with the way the civic\_api\_to\_representative\_params method in the Representative model is implemented.
* Explore the code further to understand the functionality surrounding the method, and build and understanding as to what the error may be. **HINT:** What happens if the method is called for a given representative who *already exists* in the database?
* Write a characterization test to encapsulate your understanding, and modify the code to allow the test to pass. (Remember Red->Green->Refactor; your test should fail before you implement a fix).

After completing this step, you may want to purge and re-initialize your database (why?), both locally and on Heroku.

Locally:

bundle exec rake db:drop

bundle exec rake db:create

bundle exec rake db:migrate

bundle exec rake db:seed

On Heroku:

heroku addons:destroy heroku-postgresql -a <YOUR\_APP\_NAME>

heroku addons:create heroku-postgresql -a <YOUR\_APP\_NAME>

heroku run rails db:migrate

heroku run rails db:seed

**Iteration 1, Part 2: Representative’s Profile Page**

The current Representatives model only includes their name, OCD ID, and office. We want it to show a lot more, including contact address (street, city, state, zip), political party, and a photo. This info should be viewable on the representative’s profile page, which you should create as the views/representatives/show.html.erb (or .html.haml if you prefer) view, which you will create. This information will come from the Google Civic Information API. See the Representatives controller and model for a basic implementation of getting some fields.

The profile page should be linked from views/representatives/search.html.haml, as well as anywhere the representative name appears in views/news\_items.

Note that you’ll need a migration to store the new Representative information.

**TASK 1.2**: Create and add a profile page for a representative, as described above. You can use [Bootstrap](https://getbootstrap.com/) for basic styling, but the functionality is the most important thing.

* **Testing**: Refer to ESaaS Chapter 8 Section 4 (Stubbing the Internet) to add RSpec tests and/or Cucumber scenarios that increase coverage for this portion of the app.

**Iteration 1, Part 3: The Counties Map**

For this part, you should explore the interconnectivity between the various controllers, models, and views that allow the app to search the Google Civic Information API. Starting in views/representatives/index.html.haml, trace the search code all the way to views/representatives/search.html.haml. You may refer back to the previous page to see how the map works in JavaScript.

Understanding this will make the next task much easier.

The map is broken! You can click on a state, and it’ll show you a list of counties, but there’s no way to then click on a county to show its representatives. It’s up to you to figure out how to fix it.  
This doesn’t require a lot of code changes, but does require you to use a mix of your basic JavaScript knowledge and creativity. Figure out which URL you’ll need to modify so that a click on a county triggers the search#search route.

Ensure you are also writing proper tests in Cucumber for map actions.

**TASK 1.3**: Make the Map Functional!

**Iteration 1, Part 4: Improving Coverage**

**TASK 1.4**: Ensure test coverage exceed 45% for any files in the /app directory.

You can use simplecov locally to see your coverage. SimpleCov is already enabled in the project for RSpec and Cucumber tests, but you need to instruct SimpleCov to ignore the /lib folder, which contains library code that isn’t your responsibility to cover.

In features/support/env.rb and spec/rails\_helper.rb, change:

SimpleCov.start 'rails'

to:

SimpleCov.start 'rails' do

add\_filter 'lib'

end

Then run both your RSpec and Cucumber tests:

bundle exec rspec

bundle exec cucumber

Finally, open coverage/index.html and look for the percentage next to "All Files".

**Deliverables at end of Iteration 1**

Woo! You made it, hope you are enjoying the ride so far. The only things you will need to submit for this part are as follows:

**Project deliverables: one per team through Gradescope:**

These forms have been released and both due by Monday, 08/05. You can access them on Gradescope , which is linked on bCourses. You should add your group members to the submission on Gradescope.

* [Iteration Planning Meeting 1](https://www.gradescope.com/courses/797632/assignments/4676562)
* [CHIPS 10.5 Iteration 1 Group Submission](https://www.gradescope.com/courses/797632/assignments/4676703)

For the group submission, you’ll submit a short recorded screencast (which you can create by having a Zoom “meeting” and recording the screenshare) showing the functionality of your deployed app on Heroku. Submit either a direct video upload or a link we can access. We will also be checking your GitHub repository, code, and issues/project board. We care not just about your code being technically functional, but also code quality measures like nontrivial test coverage! There is **no autograder** for CHIPS 10.5; your course staff will be reading your code and looking at your deployed application.

**Per person, on Gradescope and TEAMMATES:**

* [10.5 CHIPS Iteration 1 Retrospective questionnaire](https://www.gradescope.com/courses/797632/assignments/4676618)
* 10.5 CHIPS Iteration 1 Peer review questionnaire (on TEAMMATES)
* 10.5 CHIPS Iteration 1 self-assessment (on TEAMMATES)

# TA Meeting Notes

Both iterations will follow the same general structure:

1. Have a detailed *pre-iteration planning meeting* to discuss the work with the team and break it up into stories
2. Create an Issue corresponding to each story and assign developer(s)
3. Developer(s) use BDD and TDD to create tests and code for the story, doing Pull Requests when the feature is believed complete; others review the PRs and eventually decide to Merge and then close the Issue.
4. App is constantly pushed to Heroku to make sure everything still works in production
5. At end of iteration, the team as a whole submits a *retro* (retrospective survey), and each developer submits a confidential *peer evaluation survey* and *self-assessment survey*. The grading of the code and tests is based on your deployed Heroku app and the contents of the team repo.

# Project Progress

# You will need to run this in every terminal where you're using node.

**nvm use 13**

The app is almost ready for launch. You need to run database migrations in [db/migrate](https://matrixdragon-vocalperson.codio.io/db/migrate) to prepare your local database to store and serve data, and add the seed data to it:

**bundle exec rails db:migrate**

**bundle exec rails db:seed**

Initial commit: ff8bf3494c9baf9d

However, in this case, a downside of this option is that JavaScript-related changes may not be visible right after you make them. Essentially, when you change Rails app files, Rails knows to “reload” any changed Ruby classes on the very next request to your app; but because we’re doing separate JavaScript package management with yarn, the same isn’t true for JavaScript, CSS, or other front-end files.

Instead, we recommend you have *two* terminals open when developing. In one terminal, execute:

**bin/webpack-dev-server**

This will launch a webpacker instance that live-reloads your browser as you edit the javascript code or CSS styles and makes the development process much faster.

After launching webpack-dev-server on one terminal, switch to the second terminal window and launch a Rails server from there as usual:

**bundle exec rails server -b 0.0.0.0**

You can manually run these linters:

**bundle exec rubocop**

**yarn run lint**

**bundle exec haml-lint**

And I press 'California, CA'

And I press 'San Francisco County, CA'

Then I should see the following representatives: Joseph R. Biden, Kamala D. Harr

is, Alex Padilla, LaPhonza R. Butler, Gavin

Newsom, Eleni Kounalakis, Ricardo Lara, Shir

ley N. Weber, Malia M. Cohen

c13cce79868618a995db8c5913d03e02

c13cce79868618a995db8c5913d03e02 (mine)

7887788211c0dda1e69be0c842b0221b (Bruce’s; the correct one)

https://matrixdragon-vocalperson-3000.codio.io/state/CA

<https://matrixdragon-vocalperson-3000.codio.io/state/CA/county/045>

Mine:

--OWOqSmaGcCA3M4Rb--46QhekXwNfm25B22B30tjg==

His:

--YnIIv+C61mvruYsc--RQnh0QJvnD95MMm+7mlE3g==

# aws:

# access key\_id: 123

# secret\_access\_key: 345

production:

GOOGLE\_CLIENT\_ID: 883547480099-82tkk8u3jf3ibgtrgrerct8ftc8lge6m.apps.googleusercontent.com

GOOGLE\_CLIENT\_SECRET: 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

GOOGLE\_API\_KEY: AIzaSyDpckMcRirMG7D3QxwrDgjcXUVjZoUOPt0

GITHUB\_CLIENT\_ID: Ov23likjxE3EyBhU0ix0

GITHUB CLIENT SECRET: 4f77d0ebadf23elc452fc35ebceba5eclaecc574

development:

GOOGLE\_CLIENT\_ID: 883547480099-q7l4jcb30odsnn59out5nb09piqau34u.apps.googleusercontent.com

GOOGLE\_CLIENT\_SECRET: GOCSPX-UNgrY9ljDielnxDZZAEhfwf\_ikXf

GOOGLE\_API\_KEY: AIzaSyDT7nDH3Sd\_wTQDJV32EiLNexzWTvslMOQ

GITHUB\_CLIENT\_ID: Ov23likjxE3EyBhU0ix0

GITHUB\_CLIENT\_SECRET: 569a4a97ebccae54b4b08d6aa95cf4475e7cf59

# Used as the base secret for all MessageVerifiers in Rails, including the one protecting cookies.

secret\_key\_base: 462499de7a46e79fc82c5aab5660586a8a6bdc3cdd7aa6al8be0ab77eaca5494704fb140f8d667565ec3lc29432c8995d20b7b96023470264ec83e07215f3bdc

Should have only the following

Then I should see only the following representatives: Joseph R. Biden, Kamala D. Harris, Alex Padilla, Laphonza R. Butler, Gavin Newsom, Eleni Kounalakis, Tony Thurmond, Ricardo Lara, Rob Bonta, Malia M. Cohen, Shirley N. Weber, Fiona Ma, Carol A. Corrigan, Goodwin H. Liu, Joshua P. Groban, Kelli Evans, Leondra R. Kruger, Martin J. Jenkins, Patricia Guerrero, London Breed, Joaquín Torres, Brooke Jenkins, Paul M. Miyamoto, Manohar Raju, José Cisneros, David Chiu

Mayor of San Francisco

San Francisco County Sheriff

Scenario: click through the map to find the county representatives

And I press 'California, CA'

And I press 'San Francisco County, CA'

Then I should see the following representatives: Joseph R. Biden, Kamala D. Harris, Alex Padilla, LaPhonza R. Butler, Gavin Newsom, Eleni Kounalakis, Ricardo Lara, Shirley N. Weber, Malia M. Cohen

Scenario: use the Representatives search bar to find representatives

And I press 'Representatives'

And I fill in 'Enter a location: ' with 'San Francisco'

Then I should see the following representatives: Joseph R. Biden, Kamala D. Harris, Alex Padilla, LaPhonza R. Butler, Gavin Newsom, Eleni Kounalakis, Ricardo Lara, Shirley N. Weber, Malia M. Cohen

Then I should see only the following representatives: Joseph R. Biden, Kamala D. Harris, Dan Sullivan, Lisa Murkowski, Mary Sattler Peltola, Mike Dunleavy, Nancy Dahlstrom, Dario Borghesan, Jennifer Stuart Henderson, Jude Pate, Peter J. Maassen, Susan M. Carney

Kusilvak Census Area, AK

Modoc County, CA

Delaware

Then I should see only the following representatives: Joseph R. Biden, Kamala D. Harris, Christopher A. Coons, Thomas R. Carper, Lisa Blunt Rochester, John Carney, Bethany Hall-Long, Trinidad Navarro, Kathy Jennings, Lydia E. York, Colleen C. Davis

New Castle County, DE

Matthew Meyer, Scott Phillips, Ciro Poppiti III, Michael E. Kozikowski, Sr., Lisa Darrah, Karen Hartley-Nagle

Kent County, DE

Brenda Wootten, Norman R. Barlow, Harold K. Brode, Eugenia Thornton, Terry L. Pepper

Sussex County, DE

Norman A. "Jay" Jones, Jr., Alexandra Reed Baker, Greg Fuller, Robert Lee

Riverside County County, California

Given ## representatives ## exist:

And I am on the Actionmap homepage

Then I should be able to find my county representatives by clicking through the map or the Representatives search bar

Scenario: click through the map to find the state representatives

And I press 'Representatives'

And I fill in 'Enter a location: ' with 'Delaware'

And I press 'Search'

Then I should see only the following representatives: Joseph R. Biden, Kamala D. Harris, Christopher A. Coons, Thomas R. Carper, Lisa Blunt Rochester, John Carney, Bethany Hall-Long, Trinidad Navarro, Kathy Jennings, Lydia E. York, Colleen C. Davis

Scenario: click through the map to find the county representatives

And I press 'Representatives'

And I fill in 'Enter a location: ' with 'New Castle County, DE'

And I press 'Search'

Then I should see only the following representatives: Joseph R. Biden, Kamala D. Harris, Christopher A. Coons, Thomas R. Carper, Lisa Blunt Rochester, John Carney, Bethany Hall-Long, Trinidad Navarro, Kathy Jennings, Lydia E. York, Colleen C. Davis, Matthew Meyer, Scott Phillips, Ciro Poppiti III, Michael E. Kozikowski, Sr., Lisa Darrah, Karen Hartley-Naglew

===

Redoing credentials

# aws:

# access\_key\_id: 123

# secret\_access\_key: 345

# Used as the base secret for all MessageVerifiers in Rails, including the one protecting cookies.

secret\_key\_base: 7587a80ee08984904a7ba3d02cd313eb9e40d507a41e22a65842dc2db86ec93226608e461c2f0851fbd2a3df373fcc7da636f06c4d3301dfa93bec9ec3f5e419

Master Key: 21190b2d7fc587a0b121cdf7c1d9fd64

Appname su2402

A screen shot of a computer program

Description automatically generated

* Representatives; index.html

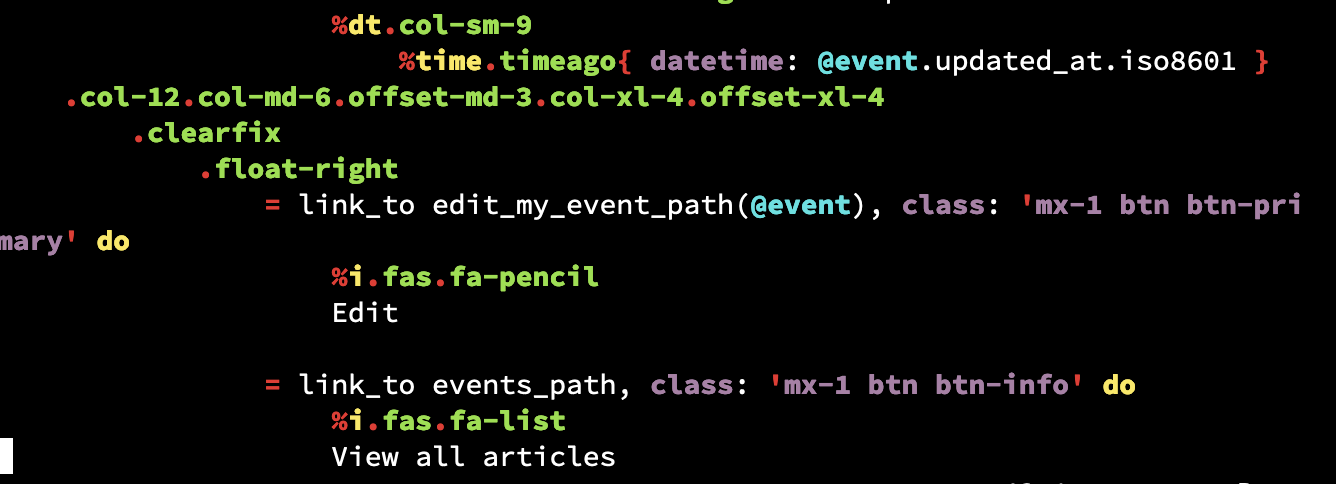
A screen shot of a computer program

Description automatically generated

* Search.html

A screen shot of a computer program

Description automatically generated



* Events show.html

A computer screen shot of text

Description automatically generated

A computer screen shot of text

Description automatically generated

A computer screen shot of a program

Description automatically generated

* Events index.html

# aws:

# access\_key\_id: 123

# secret\_access\_key: 345

production:

GOOGLE\_CLIENT\_ID: 883547480099-82tkk8u3jf3ibgtrgrerct8ftc8lge6m.apps.googleusercontent.com

GOOGLE\_CLIENT\_SECRET: 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

GOOGLE\_API\_KEY: AIzaSyDpckMcRirMG7D3QxwrDgjcXUVjZoUOPt0

GITHUB\_CLIENT\_ID: Ov23likjxE3EyBhU0ix0

GITHUB CLIENT SECRET: 4f77d0ebadf23elc452fc35ebceba5eclaecc574

development:

GOOGLE\_CLIENT\_ID: 883547480099-q7l4jcb30odsnn59out5nb09piqau34u.apps.googleusercontent.com

GOOGLE\_CLIENT\_SECRET: GOCSPX-UNgrY9ljDielnxDZZAEhfwf\_ikXf

GOOGLE\_API\_KEY: AIzaSyDT7nDH3Sd\_wTQDJV32EiLNexzWTvslMOQ

GITHUB\_CLIENT\_ID: Ov23likjxE3EyBhU0ix0

GITHUB\_CLIENT\_SECRET: 569a4a97ebccae54b4b08d6aa95cf4475e7cf59

# Used as the base secret for all MessageVerifiers in Rails, including the one protecting cookies.

secret\_key\_base: fc2bf2212f54ee52aa485447a79d1c02218aa501989c8906107603f5ec6a69fb624f36411a4b83f389afff5c8832e1d1cbc6d59fe595b1d3abaf6fe3482c4eea

Master Key: a9d421bbdb9dede5b22e245396349907

Initial commit: ff8bf3494c9baf9d

However, in this case, a downside of this option is that JavaScript-related changes may not be visible right after you make them. Essentially, when you change Rails app files, Rails knows to “reload” any changed Ruby classes on the very next request to your app; but because we’re doing separate JavaScript package management with yarn, the same isn’t true for JavaScript, CSS, or other front-end files.

Instead, we recommend you have *two* terminals open when developing. In one terminal, execute:

**bin/webpack-dev-server**

This will launch a webpacker instance that live-reloads your browser as you edit the javascript code or CSS styles and makes the development process much faster.

After launching webpack-dev-server on one terminal, switch to the second terminal window and launch a Rails server from there as usual:

**bundle exec rails server -b 0.0.0.0**

Credentials

--mE34mnuoVH/9mFqF--oSnf6gFazYg/lEBe8ze16g==

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generatedUndo

A computer screen shot of a computer code

Description automatically generated

Add migration for table (issue)

Redo database

bin/rails generate migration AddIssueToNews\_Items issue:string

bundle exec rake db:drop

bundle exec rake db:create

bundle exec rake db:migrate

bundle exec rake db:seed

heroku addons:destroy heroku-postgresql -a <YOUR\_APP\_NAME>

heroku addons:create heroku-postgresql -a <YOUR\_APP\_NAME>

heroku run rails db:migrate

heroku run rails db:seed

HRKU-395ea03d-855c-43fc-8d62-e8a465d66932

heroku config:set RAILS\_MASTER\_KEY=`cat config/master.key`

first, fix the key issue (or note to; on GitHub and Heroku)

get the double web page working

A screenshot of a web page

Description automatically generated

.form-group.row

= link\_to search\_my\_news\_item\_path(@representative, @news\_item), class: 'mx-1 btn btn-primary' do

%i.fas.fa-list

Search

—

.form-group.row

= link\_to search\_my\_news\_item\_path(@representative, @news\_item), class: 'mx-1 btn btn-primary' do

%i.fas.fa-list

Search

—

.col-12.col-md-6.offset-md-3.col-xl-4.offset-xl-4

= form\_tag(controller: :my\_news\_item, action: :search, method: :get) do

.form-group.row

.col-sm-2

= label :news\_item, :representative\_id, class: 'col-form-label'

.col-sm-10

= select :news\_item, :representative\_id, @representatives\_list, { include\_blank: true },

class: 'form-control'

.form-group.row

.col-sm-2

= label :news\_item, :issue, 'Issue', class: 'col-form-label'

.col-sm-10

= select :news\_item, :issue, ["Free Speech", "Immigration", "Terrorism", "Social Security and Medicare", "Abortion", "Student Loans", "Gun Control", "Unemployment", "Climate Change", "Homelessness", "Racism", "Tax Reform", "Net Neutrality", "Religious Freedom", "Border Security", "Minimum Wage", "Equal Pay"], { include\_blank: true },

class: 'form-control'

.form-group.row

=submit\_tag "Search", class: "form\_submit"

News API Key: 9e9026fefdd44b9fa3552d224e2b9483