

# 1. Intro

## CHIP 10.5: Agile Iterations

Welcome to CHIP 10.5!

Read the Ed post about this project to create your team's GitHub Classroom repository for the project.

Open a terminal window from Tools > Terminal, and clone your team's repo.

Then, navigate to the next page to get started with Part 1.

Next

## 2. Getting Started

# How to setup your local development environment

---

### 0. Pull Staff changes

Clone your team's GitHub Classroom repository, and `cd` into the directory. Then, run the following commands:

```
git remote add staff https://github.com/saasbook/hw-agile-iterations
git pull staff master --allow-unrelated-histories
git remote remove staff
```

### 1. Install Ruby

You need to install Ruby. We recommend you use `rvm` but you could also use `rbenv` to manage Ruby version.

These are environment management tools that help you switch between Ruby version easily across different projects.

Once you install one of these tools, you need to install the Ruby version listed in Gemfile.

First, run:

```
gpg --keyserver hkp://pool.sks-keyservers.net --recv-keys 409B6B1796
```

Then, the command to install `rvm` is:

```
curl -sSL https://get.rvm.io | bash -s stable --ruby
```

After installing rvm or rbenv, you need to check that it is loaded in you current terminal. **Refresh Codio** at this point for your `rvm` installation to take effect.

You can do so by checking the version of your specific ruby env manager. For example if you opt for rvm:

```
rvm -v
```

Note, if rvm is not loaded you might want to reload your shell profile using:

```
source ~/.bashrc
```

For example, if you are using rvm run the following commands.

```
rvm install 2.6.5  
rvm use 2.6.5
```

## 2. Install Node

Similarly, you need to install Node and we would recommend that you use nvm which allows you to manage multiple active Node.js versions.

The command to install `nvm` is:

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.37.0/install
```

After installing `nvm`, you need to check that it is loaded in your current terminal. **Refresh Codio** at this point for your `nvm` installation to take effect.

The specific version of node you need to install is listed in `package.json` under the JSON path `$.engines.node`.

Example use of `nvm` to install node version `12.13.1`:

```
nvm install 12.13.1
```

### 3. Install Yarn

Once Node is installed, you need to install yarn. Node has two popular javascript package managers: `npm` and `yarn`.

In this project, we will use `yarn` as it is the default for Rails Webpacker. The specific version of `yarn` you need to install is listed in `package.json` under the JSON path `$.engines.yarn`.

To install `yarn` run the following command, replacing `@1.22.4` with the version listed in `package.json`.

```
npm install -g yarn@1.22.4
```

Next, run the following command to ensure you have the correct version of `yarn` installed.

```
yarn -v
```

## 4. Install project dependencies

There are two sets of dependencies that you need to install: ruby dependencies and node dependencies.

First, to install ruby dependencies. RubyGems is a dependency management systems that allows developers to share and distribute code. We recommend you install your ruby dependencies within the project's `vendor/bundle` folder instead globally within you ruby installation.

To do this, run the following command:

```
bundle config set path vendor/bundle
```

The command above would imply that you will be required to run all ruby commands with the `bundle exec` prefix.

For example, to run a migration (which we will do shortly), you must execute the command `bundle exec rails db:migrate` instead of `rails db:migrate`.

Some may find this tedious, hence we suggest an option to alias the `bundle exec` prefix.

You could run `alias be="bundle exec"` on your terminal which will allow you to run `be rails db:migrate` instead of the more verbose `bundle exec rails db:migrate`.

You could add the alias to your shell profile eg. `~/.bashrc` to make it globally available.

For local development, we recommend you skip installing `production` dependencies by running the following command:

```
bundle config set without production
```

Next, run the following command to install dependencies listed in the Gemfile.

A Gemfile is a file that list all of a Ruby project's dependencies.

The Gemfile.lock pins the dependencies listed in the Gemfile to specific versions alongside their specific transitive dependencies.

Pinning dependencies helps different developers working on the same project have reproducible builds.

First, install our desired version of the `bundler` gem, as well as installing a library that will allow us to use Postgres to install correctly. Run the following commands:

```
gem install bundler:2.1.4  
sudo apt-get install -y libpq-dev
```

The following command will install all the requirements in the `Gemfile.lock`.

It may take a while since some of the dependencies are Ruby extensions written in C, eg. Nokogiri.

```
bundle install
```

After all the gems are installed, we now need to install the javascript requirements.

Node projects have a package.json file that serves the same purpose as Gemfile, but for Javascript dependencies. It lists all project dependencies. yarn.lock and `package-lock.json` serve the same role as `Gemfile.lock`, they pin dependencies.

Yarn uses `yarn.lock` whereas npm uses `package-lock.json`. This project uses `yarn.lock` since it uses yarn.

Projects that use npm have `package-lock.json` instead. It is highly advised that you only have one of these and not both in your project since they may diverge and lead to inconsistent environments for different developers. You will notice that `package-lock.json` is git-ignored in `.gitignore`.

To install javascript requirements, we will run the following command:

```
yarn install
```

## 5. Run migrations and seed the database

Our app is almost ready for launch. You need to run database migrations in `db/migrate` to prepare your localhost database to store and serve data. To run migrations, execute the following command:

```
bundle exec rails db:migrate
```

Note, you could use your alias `be rails db:migrate` if you opted to set it up.

Next, we need to seed our database with some data such as the list of states in the United States.

To do so, execute the following command:

```
bundle exec rails db:seed
```

## 6. Launch the app!

Finally, it is time to launch the rails application and begin hacking away. There are two ways to do this. The first option is that you could launch the application using the following bundle command.

```
bundle exec rails s -b 0.0.0.0
```

However, this option has one major downside. It takes long for any changes that you make to reflect on the browser since you will not have live reload.

The second option, is to have two terminals. In one terminal, run the following command:

```
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.

```
bundle exec rails s -b 0.0.0.0
```

Next, you should read about linters.

Next





### 3. Linters

A linter is a program that analyzes source code for common errors and conformance to a coding style.

When working in a team, it helps to have the repository conform to the same coding style.

For example, a specific team would like to have a specific template rendering system, say erb over haml.

These standards tend to make a codebase consistent and hence easier to work with.

Such decisions are made on a team-by-team basis and each team may have different preferences resulting in different styles being adopted.

Regardless of the final decision, teams tend to benefit overall from picking a standard or style and sticking by it.

You may have noticed that most of our HTML in app/views is rendered using HAML. Our team decided to use it as it is more cross-platform, that is, you can find HAML engines in more programming languages including Java, Javascript and PHP. This makes it easy for most developers to work with the codebase.

To enforce coding standards within this project, we have setup three different linter systems: rubocop, haml-lint and eslint for Ruby files, HAML files and Javascript files respectively. In additions, you will notice there are configuration files `.rubocop.yml`, `.haml-lint.yml` and `.eslintrc.js` that list our codified standards for each linter system.

Run the following command to run rubocop lint checks:

```
bundle exec rubocop
```

Run the following command to auto-correct some common rubocop lint errors:

```
bundle exec rubocop -a
```

Run the following command to execute haml-lint checks:

```
bundle exec haml-lint
```

Unfortunately, haml-lint does not yet have auto-correct functionality as of this writing.

See the following open github issue: Autocorrect feature issue.

Run the following command to execute eslint checks:

```
yarn run lint_fix
```

Notice that we run eslint using `yarn run` command. If you check package.json, you will notice that `lint` and `lint_fix` are entries in JSON path `$.scripts.lint` and `$.scripts.lint_fix` that invoke a vendored install of eslint.

We have included tests that will enforce the linter checks in spec/linters.

Sometimes, you may just want to have a file watcher that runs linter check and tests as you work on the project.

Guard is a tool that allows you to run file watchers that run automated tasks whenever a file or directory is modified.

The Guardfile specifies file watchers and tasks to run. To initiate guard, run the following command in a separate terminal:

```
bundle exec guard
```

Next, you should read about Credentials management, Heroku setup and CI Setup.

[Next](#)

# 4. Credentials Management, Heroku, and Travis CI

## Credentials Management

In past version of `rails`, credentials management has been done in a variety of ways.

There are gems that were used to push credentials and environment variables across environment (eg from localhost to heroku).

In this projects, we will be using built-in `config/credentials.yml.enc` in Rails 5.2. If you are working with legacy applications, you may encounter `Figaro`, a popular gem that is used to manage credentials.

Note, for all the following commands, make sure to `cd` into the `path/to/your/rails_app` (not into the `app` directory, but it's parent!).

1. The first step to set up your own encrypted credentials is to delete the current `config/credentials.yml.enc`

```
rm -rf config/credentials.yml.enc
```

2. Then we need to regenerate a new `config/master.key` and `config/credentials.yml.enc`.

```
EDITOR=vim bundle exec rails credentials:edit
```

Save and exit vim (using `escape` then `:wq` + `Enter`).

3. Do not edit the `credentials.yml.enc` for now but you will later add Google Oauth, Google Civic API and Github Oauth keys.

Just take note of the values in the file. It should look like this:

```
▪  
▪  
▪  
secret_key_base: some-long-string
```

4. Open a rails console with `bundle exec rails c`. Inside the console, you should be able to run the following command and see the `secret_key_base` from the file.

```
Rails.application.credentials[:secret_key_base]
```

`Rails.application.credentials.dig` reads the given key eg `'GOOGLE_CLIENT_ID'` from

```
config/credentials.yml.enc
```

 by decrypting it with 

```
config/master.key
```

.

You could specify environment specific groups as follows in `config/credentials.yml.enc`:

```
production:  
  GOOGLE_CLIENT_ID: xxxx  
  GOOGLE_CLIENT_SECRET: xxx  
  
development:  
  GOOGLE_CLIENT_ID: xxx  
  GOOGLE_CLIENT_SECRET: xxx
```

Then use the following syntax to read keys for specific environment:

```
Rails.application.credentials.dig(:production, :GOOGLE_CLIENT_I  
Rails.application.credentials.dig(:development, :GOOGLE_CLIENT_I
```

(This step did not require you to change any files. It's simply an introduction of how to use credentials in Rails 5.)

## Heroku

Now we would like to setup the app on Heroku ensuring that our credentials are available there.

1. Install Heroku CLI client on your machine using the following instructions.

The command you should use is:

```
curl https://cli-assets.heroku.com/install-ubuntu.sh | sh
```

2. Make sure your installation works by running the following command `heroku -v` to print the version of Heroku CLI available on your terminal.
3. Log in to Heroku CLI using `heroku login -i`.
4. Create a new heroku app using `heroku create`. You may provide your desired appname using `heroku create fancyapp`.  
This will allow you to access your app on `fancyapp.herokuapp.com`
5. You need to add nodejs `buildpack` to your heroku app for it to work.  
Do so using:

```
heroku buildpacks:add heroku/nodejs
heroku buildpacks:add heroku/ruby
```

To confirm that you have all the buildpacks needed, run the following command:

```
heroku buildpacks
```

You should see both ruby and nodejs in the output.

Now, make a push to Heroku using:

```
git add -A
git commit -m "<commit-message>"
git push heroku master
```

6. You should be able to access your application using your specific `your-heroku-1234.herokuapp.com` link.
7. Next you need to enable `PostgreSQL` on heroku for your app. Run the typical database instructions to execute migrations and seed the database.

```
heroku run rails db:migrate
heroku run rails db:seed
```

8. Run the following command to make `config/credentials.yml.enc` available on heroku:



```
heroku config:set RAILS_MASTER_KEY="$(< config/master.key)"
```

## 9. See Walkthrough Video #2 for Step-by-step Walkthrough of getting API Credentials

Now you need to update the `credentials.yml.enc` using `EDITOR=vim bundle exec rails credentials:edit`

for login with Google and Github to work. This is also the step where we'll get our API key for the Google Civic Information API.

Go to [console.developers.google.com](https://console.developers.google.com), create a new project, click on `Credentials`, then add new `OAuth 2.0 Client IDs`.

Copy the `GOOGLE_CLIENT_ID` and `GOOGLE_CLIENT_SECRET` and add them to your `credentials.yml.enc`. Make sure to set your callback url on the Google Developer Console using your herokuapp link eg. `https://your-heroku-1234.herokuapp.com/auth/google_oauth2/callback`.

Go to [github.com/settings](https://github.com/settings), click on `OAuth apps` and add a new `OAuth App`.

Copy the `GITHUB_CLIENT_ID` and `GITHUB_CLIENT_SECRET` and add them to your `credentials.yml.enc`. Make sure to set your callback url on the Google Developer Console using your herokuapp link eg. `https://your-heroku-1234.herokuapp.com/auth/github/callback`.

Use this guide to generate a Google Civic Info API key.

Copy the key and add it as `GOOGLE_API_KEY`

Your final `credentials.yml.enc` should look like:

```
# aws:  
#   access_key_id: 123
```

```
# secret_access_key: 345

# Used as the base secret for all MessageVerifiers in Rails, i
secret_key_base: some-key

GITHUB_CLIENT_ID: some-key
GITHUB_CLIENT_SECRET: some-key

GOOGLE_CLIENT_ID: some-key
GOOGLE_CLIENT_SECRET: some-key
GOOGLE_API_KEY: some-key
```

10. Go to this link to activate the Google Civic Information API for use with your project.
11. Commit and push your changes to Heroku. At this point, you should be able to go to your Heroku app, and make sure that you can log in with either GitHub or Google. You should also be able to go to the Representatives page and search for a location. Test out your app on your phone, see if it is responsive.

## Travis CI

1. Make sure you have an account with travis-ci.com.  
Create the account with the Github account that you use for Github Classroom.
2. Install Travis CI CLI client on your terminal using the following command:

```
gem install travis
```

3. Login into Travis CI using your Github Credentials on the terminal:

```
travis login --pro --auto
```

If you get a `Bad Credentials` error, visit the `Travis CI – Common Issues` page listed in the menu.

Afterwards, since we need to give Travis CI a means to clone our private repo to run CI builds, we need to generate an ssh-key for Travis to use. First identify your repo's org and name using:

```
git remote -v
```

If the above command prints out something similar to the following:

```
origin  git@github.com:[myorg]/[myrepo].git
```

Replace the `myorg` and `myrepo` below and run the command below.

```
travis sshkey --generate -r [myorg]/[myrepo]
```

(Your command should look something like this:)

```
travis sshkey --generate -r cs169/hw-agile-iterations-fa20-000
```

If you get a prompt that looks like `Store private key? [no]`, type `y`.

If you get a prompt that looks like `Path: |id_travis_rsa|`, type `y`.

4. Go to [travis-ci.com](https://travis-ci.com). Click the + to add a new repo to Travis CI. Search for your repo name, and click on it to add it to Travis.



My Repositories Running (0/0) +

5. Navigate to your project page on Travis, and click More Options > Trigger Build.
6. Now push your `config/master.key` to Travis CI using:

```
travis encrypt --pro RAILS_MASTER_KEY="$(cat config/master.key)"
```

If you see a message like `Detected repository as cs169/hw-agile-iterations-fa20-0909, is this correct?`, type `y`.

After that, if you see a message like `Overwrite the config file /home/codio/workspace/hw-agile-iterations-fa20-0909/.travis.yml with the content below?`, type `y`.

7. Then run the following command to ensure your `.travis.yml` file is valid.

```
travis lint
```

8. Add, commit and push to GitHub.

## Codecov

1. Head to [codecov.io](https://codecov.io) and Click [Students](#) in the navbar, then [Sign In with Github](#).
2. Open your Personal Settings on CodeCov by clicking on your profile image in the top right corner. Ensure that you are a **Student**, and there is a box similar to the one below. If you do not see a box like that, follow the steps in the [CodeCov – Common Issues](#) page listed in the menu.

Student Details	
Status	Since
Active	November 12th 2020

3. Visit your repo on Codecov via [codecov.io/gh/\[myorg\]/\[myrepo\]/settings](https://codecov.io/gh/[myorg]/[myrepo]/settings). If you see a warning that says to [Add Private Scope](#), go ahead and click that. If you see an [Activation Required](#) warning, follow the steps in the [CodeCov – Common Issues](#) page listed in the menu.

Identify the [Repository Upload Token](#) and copy it.

4. Add this to your repo on Travis CI dashboard. In the project's Travis page, click [More Options](#) then [Settings](#). Add the token to the [Environment Variables](#) section by pasting the Repository Upload Token you copied from CodeCov into the [Value](#) field, typing [CODECOV\\_TOKEN](#) in the [Name](#) field, and clicking [Add](#). Turn on [Display Value in Build Log](#).
5. Add and commit your changes on Codio, then push to Github. Head to [travis-ci.com](https://travis-ci.com)

to try **Trigger Build** and test the CI pipeline. You should see a success message like the one below in your Travis Build log.

```

32
33
34
35
36
37
38
39 ==> Travis CI detected
40 ==> Appending file network
41 ==> Gzipping contents
42 ==> Uploading reports
43 url: https://codecov.io
44 query: token=secret&flags&package=ruby-0.2.3&service=travis&branch=master&pull_request=false&job=440458558&slug=cs169%2Fhw-agile-iterations-fa20-0909&build=2.1&commit=53518b1abe9bc5265fc901fd52d53cab2cc0e747&env=2.6.5
45 -> Pinging Codecov
46 https://codecov.io/upload/v4?token=secret&flags&package=ruby-0.2.3&service=travis&branch=master&pull_request=false&job=440458558&slug=cs169%2Fhw-agile-iterations-fa20-0909&build=2.1&commit=53518b1abe9bc5265fc901fd52d53cab2cc0e747&env=2.6.5
47 -> Uploading to
48 https://storage.googleapis.com/codecov/v4/raw/2020-11-15/48630035308CB31F30C56F4EFB0F9D67/53518b1abe9bc5265fc901fd52d53cab2cc0e747/9b21c160-3909-4ad4-8616-d59d60df52a2.txt?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=G00G1EQX60ZVJGHKK3633AAFGLBUC00ATRACRQRF6HMSMLYUP6EAD6XSWAAYK2F20201115%2FUS%2F3%2Faws4_request&X-Amz-Date=20201115T221249Z&X-Amz-Expires=10&X-Amz-SignedHeaders=host&X-Amz-Signature=dc22656a35c2374cb0a50798dc40c74a7e716da1affb9f6fd479d322dd679a76
49 View reports at https://codecov.io/github/cs169/hw-agile-iterations-fa20-0909/commit/53518b1abe9bc5265fc901fd52d53cab2cc0e747
50 The command "bundle exec cucumber" exited with 0.
51 store build cache
52

```

- Now replace the Travis and CodeCov badges in README.md. For Travis, if you click on the **build: ....** black and green badge next to your repo page on Travis, you will get option to copy the status image (in the **Result** box). Head over to your **README.md** file in your project repo, and modify the existing Travis badge to be yours instead.



For Codecov, follow these instructions to find your badge, and replace the existing badge in **README.md** just as you did for Travis.

Next, let's set up Pivotal Tracker.

Next

# 5. Pivotal Tracker and Auditors

## Tool Setup

---

### 1. Set up Pivotal Tracker

For this project, you'll be using Pivotal Tracker to track user stories, estimate story points, etc. First, all team members should create Pivotal Tracker accounts (which is free); then one member creates a new project, and add your teammates as owners.

Your Pivotal Tracker project should be titled **Actionmap-Fa20-GroupXX**, with the **XX** filled in with your group number (found on the team assignment spreadsheet).

Configure your Pivotal Tracker project settings as follows:

- Start Iterations On: Monday
- Project Start Date: 2020-11-09
- Iteration Length: 2 weeks
- Point Scale: Power of 2 (0, 1, 2, 4, 8)
- Initial Velocity: 10
- Velocity Strategy: Average of 1 iteration
- Number of Done Iterations to Show: 4
- **Uncheck** Plan Current Iteration Automatically
- Follow this tutorial to set up GitHub integration for the Pivotal Tracker project. Note that:
  - The setting is in your Pivotal Tracker project instead of GitHub repository.

- You should connect the project with the forked repository instead of the golden repository.
- Only the owner of your project can create this integration.

**Important: when you create any branches on your GitHub repository later on as you work on the project, they will need to be linked to a specific story on Pivotal Tracker. You will need to prefix any branches you create on your GitHub repo with the story ID.**

For example, if your story ID is `123123`, any branch you create that relates to that story should be named `123123-some-branch-name`. You can read more about this in the **Using the GitHub integration: Branches** section of this page.

## 2. Add Auditor Accounts

- Add `srujayk` and `governifyauditor` as **Owners** of your Pivotal Tracker project.
- Add `srujay@berkeley.edu` and `governify.auditor@gmail.com` as **Owners** of your Heroku project.

Next



## 6. Understanding the Codebase

To render the maps, this project uses a combination of Javascript and Topojson.

Topojson is a compact alternative to GeoJSON.

These file formats are both used to represent simple geographical features.

Inside the `lib/tasks` folder you will note that there are rake tasks that fetch the GeoJSON files from `census.gov` and convert them to Topojson. You do not need to run the rake tasks as this project ships with pre-fetched and pre-built Topojson files.

You will find the Topojson files in `assets/topojson`. They are served via the `assets pipeline` using by `Sprockets`.

On the browser, we need Javascript in addition to Topojson files to render maps as SVG.

To do so, we use `d3.js` library.

This library is fetched using `yarn` from `npmjs.com`. NPM is to Javascript as RubyGems is to Ruby (or Maven is to Java).

In addition, we use Webpack to compile Javascript, SCSS, CSS and their dependencies.

You will find the Javascript that renders the browser in `app/javascript/packs`.

For those of you interested in an extra challenge (there will be no extra credit though),

figure out how to download shapefiles for Congressional Districts from `census.gov` and create a page that renders the map for Congressional Districts. You may

find [bl.ocks.org](https://bl.ocks.org) useful.

We also use JQuery to listen and react to DOM events eg. when a form value changes.

Open the `localhost:3000/events` path on your browser and assess how the `Filter By` form works.

What Javascript file handles the changes in the form?

Most fully-fledged applications will need a substantial amount of Javascript to manage state.

While JQuery would suffice for this project, you may later encounter frameworks like React, Vue and Angular that are more suited for managing state on a page. For medium-sized Rails applications in particular, StimulusJS is a great option. For your own practice with Javascript, you may, for example, re-write the pre-existing Javascript code using StimulusJS (there will be no extra credit for this though).

Next, read the Iteration Instructions in Checkpoint 2.

Next

## 7. Iteration Instructions

### General Iteration Instructions

---

The project will be completed in 2 iterations, following the Agile methodology. Checkpoint 1 was intended for you to learn more about your project and work through all setup logistics, and Checkpoints 2 & 3 are when you will actually create code and tests.

#### Pre-iteration Planning

##### Before Starting to Code:

1. You should think of the project spec as notes from a customer (note that they are not as descriptive as previous CHIPS in terms of what you have to do functionality wise. This is to simulate the experience of working with a customer). You should make lo-fi mockups if needed, and convert all of the spec to user stories in Pivotal Tracker.
2. Your team should have a pre-iteration planning meeting to prioritize and vote on assigning points to the stories, subdividing them down as needed; as discussed in lecture & section, each story must be  $\leq 3$  points (1 point stories are just fine).

##### Requirements prior to starting to code:

1. User stories you plan to implement this iteration are in Pivotal Tracker, with points and in priority order in the Backlog. Label each story with “iter1”, “iter2”, etc. using Tracker’s label feature. For stories that required lo-fi mockups, attach the relevant files to each Tracker story.

2. Create one or more Cucumber feature files for each user story that you plan to implement. You should write out all scenarios for that user story as we have done in class. Also write 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.

## Once you've planned out the iteration in the IPM, it's time to code!

Use the following standards to keep track of what's going on in each iteration.

- **Git & GitHub**

1. When committing, specify the commit type (from this list) at the beginning of the commit message with "[ ]". For example, a commit message will look like "[feat] A new feature".

*Note: ("A new feature" is a really bad commit message. This is just to illustrate the format. Make your commit messages much more informative)*

2. Use branch-per-feature to manage your work. When you create any branches on your GitHub repository later on as you work on the project, they will need to be linked to a specific story on Pivotal Tracker. You will need to prefix any branches you create on your GitHub repo with the story ID.

For example, if your story ID is `123123`, any branch you create that relates to that story should be named `123123-some-branch-name`. You can read more about this in the **Using the GitHub integration: Branches** section of this page.

3. Create a Pull Request to the master branch when a feature is finished. You can use rebase to resolve conflicts on your branch. You are encouraged to squash commits into a major commit. **Remember that comments & discussion** as part of the PR form an integral part of keeping everyone on the team apprised of what's happening.

- **Pivotal Tracker** | The story states should be used as follows:

1. **Started**: a developer or pair has claimed a story and is actively working on it, in its own branch and following the commit guidelines above.
2. **Finished**: all tests and code for the story are complete, and a Pull Request has been opened to merge the story. Depending on the PR discussion, the story may move back from Finished to Started; if only minor changes are needed, the story will just stay in Finished until a PR for it is finally merged.
3. **Delivered**: A PR is merged and the story's code is deployed to Heroku for the customer to try.
4. **Accepted**: the customer has indicated (by email, or during a f2f meeting) that the story meets their original needs.

- **Daily Standups**

As part of the agile development process, you should be completing daily standups. Especially as the whole world is remote at the moment, interfacing with your teammates regularly is critical for a functioning team.

Our recommended workflow: every day (or as close to every day as possible), meet with your teammates as a group in a simple Zoom or Google Hangouts call. Have one person be the driver to fill out this standup Google form to keep a record of your meeting. This form should be submitted by ONE team member each time you have a standup.

## End of 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?
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 85% or higher (Coverage badge should be

part of your repo's README)

- Grade will be scaled down accordingly

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

## Tools To Help You Get Started

Here are some tools to help you with your Iteration Planning Meetings and Retrospectives.

### Planning Poker (for Iteration Planning Meetings):

- Usage guide
- Tutorial Video

**IF PLANNING POKER IS DOWN:** (e.g. if you are unable to log in or if stories aren't showing up): Watch the tutorial video above to get an idea of how planning poker is used. The same style of meeting can be conducted informally over a simple Zoom or Google Hangouts call, using the chat functionality for voting. Do not wait for Planning Poker to be fixed to conduct your IPM.

### Postfacto (for Retrospective Meetings):

- Usage guide
- Tutorial Video

Next, read the spec for Checkpoint 2.

Next



## 8. Checkpoint 2 Spec

### Checkpoint 2 Spec

---

#### App Overview

ActionMap seeks to provide 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.

One of the big emphasis on this project 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, an external API being used (namely the Google Civic Information API), Javascript code for the map, and an asset pipeline (Webpacker).

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. CHIPS 7.7 & 8.9, as well as Quiz 4 are great resources to look at when deciding what is testable, and with which technique. 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.

For now, add `gem 'cucumber-rails-training-wheels'` under `gem 'cucumber-rails', require: false` on line 54 in your `Gemfile`. Then run `bundle install`.

Once installed, run `rails generate cucumber_rails_training_wheels:install`. This will generate some training wheels similar to those in CHIPS 8.9 to speed up Cucumber test writing.

## Intended User Flow:

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.

## Part 1: Representatives

### Representatives

The Representatives model is the first thing we'll be working on as part of this app. It has some basic functionality; however, there's much to be desired.

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

We have a `representative` MVC structure already laid out. Take a look in `app/views/representatives`, `app/models`, and `app/controllers`.

A basic `representative` database model. This includes the representative name, OCD (Open Civic Data) ID, and office/title.

An association with `news_items`. As you can see, a `representative` has\_many `news_items`.

`RepresentativesController` and `SearchController`. The `SearchController` handles calling the Google Civic API.

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

This is the preliminary database model for our App.

### TASK 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.
- 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 run rails db:drop
heroku run rails db:create
heroku run rails db:migrate
heroku run rails db:seed
```

## **TASK 2:** Creating and Adding to the Representatives Profile Page

- All the information we have about our representatives are their name, OCD ID, and office! We want it to show a lot more, including address (street, city, state, zip), political party, and a photo.
  - The `representative` profile page hasn't been created (will need to be at `views/representatives/show.html.haml`). You'll need to create this.
  - The profile page should be linked from `views/representatives/search.html.haml`, as well as anywhere the representative name appears in `views/news_items`.

- This information will come from the Google Civic Information API. See the Representatives controller and model for a basic implementation of getting some fields.
- This will involve adding to the existing migration for the `representatives` model, so you can store the new information.
- You'll also need to make the representative's Profile page look good! Don't spend too much time styling, but you do have access to the full powers of Bootstrap! It should look usable when users come to your site. Be creative!
- **Testing:** Refer to ESaaS Chapter 8 Section 4 (Stubbing the Internet), and this unreleased CHIPS assignment, which serves as a walkthrough for how to mock out web requests. Use these two resources to add RSpec tests that increase coverage for this portion of the app.

## Part 2: 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`. Also, read understand the code to see how the map works in JavaScript.

Understanding this will make the next task much easier.

### TASK 3: Making the Map Functional

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 it's 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.

### Part 3: Improving Coverage

**TASK 4:** Use this opportunity to increase code coverage for your app. Use CodeCov to monitor your per-file code coverage.

As noted in the iteration instructions, test coverage should exceed 85% for any files in the `/app` directory you are asked to modify or create in the tasks above. The coverage for `/app` overall should be greater than or equal to 70%.

Next

## 9. Checkpoint 3 Spec

### Checkpoint 3 Spec

---

In the following parts, ensure you are writing tests and keeping coverage for the `app` directory `>=70%`, and for any files you modify `>=85%`.

#### Part 4: News Articles + Issues + Ratings

The Representatives table is currently associated with many news articles. What we'd like you to do in this section is add a new `"Issue"` column to News Articles to make it more specific, and a new model `"Ratings"` to get a particular user's opinion of a candidate's view on an issue, based on a news article.

#### TASK 5: Set up the News Article Rating infrastructure

- For a news article, add an 'Issue' column (similar to adding a Director column to Movies in CHIPS 8.9) to News Articles to relate an article with a particular issue.
  - This field should be populated by a dropdown on the 'Create News Article' page, to associate an issue with a particular article. It should be drawn from the following list:  
`["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"]`
- You should also add a new migration for the `Ratings` model, which should be associated with News Articles in the following way: a `news_item` should `have_many ratings`, and a `user` should

`have_many` `ratings`.

- A single `rating` associated with a news article should be populated by a dropdown on the 'Create News Article' page, to associate a rating with a particular article. A rating should be on a scale of 1-5.

## Part 5: Your Own External API

**TASK 6:** This part of the project comes with a choice: whether you'd like to augment the Representatives portion of the app, or the News Articles portion. In either case, you'll be selecting an external API to use.

Secrets management will play an important role here (similar to how we set up the Google Civic Information API), whichever option you end up choosing, as will stubbing out and testing external APIs. Refer to the earlier Representatives section for a resource on stubbing out the internet.

**Option 1: ProPublica Campaign Finance API:** Your first option for this project is to add additional details about campaign finances for US Presidents, Senators, and Congressional Representatives.

ProPublica Campaign Finance API is the link to the API (specifically, to the "Get Top 20 Candidates in a Specific Financial Category" which we'll be using).

The deliverable:

- add a new model, controller, and set of views for the `Campaign Finance` information we'll be accessing. Similar to the `Representatives` search page, we'll want to add a new search page to the navbar called `Campaign Finances`.
- the `Campaign Finances` search page should have 2 dropdowns, one for



each of the parameters ( `cycle` and `category` ) to the ProPublica API, as well as a `Search` button. Ensure you restrict the available options in the dropdowns to what the API allows.

- Once a user selects a specific category and election cycle, allow them to click a `Search` button to show a table with the top 20 candidates in that category that election cycle.

**Option 2: News API:** Your second option for this task is to use the News API to pre-populate a list of news articles that the user can choose from to rate for a particular representative.

News API is the link to the API.

- Currently, the new news article page ( `/representatives/[id]/my_news_item/new` ) has fields for Title, Link, Description, Representative, Issue, and Rating. Split this into two pages, where a user can first select the Representative and Issue they desire, and click `Search`.
- This should take them to a second page, where there is a list of the top 5 articles returned from the News API, and they can select one via **radio button**. This list of articles comes from making a query to the News API, and getting the top 5 titles, URLs, and descriptions.
- A user should be able to select a radio button for their desired article, rate it, and hit the `Save` button to add it to the database.
- See below for a lo-fi mockup. Text in `blue` is user input, and in `red` are notes.

### Edit News Item

Representative

Issue



### Edit News Item

Representative: Gavin Newsom ← should be a hyperlink to rep's profile page

Issue: Healthcare ← should be plaintext, not a dropdown

Select an article:

top 5 articles

☐ Title: ... ← plaintext

☐ Link: ... ← hyperlink that opens in new tab

☐ Description: ... ← plaintext

☐ Title: ...

☐ Link: ...

☐ Description: ...

Rating

Next

# 10. Travis CI - Common Issues

## Travis CI Errors

If, when trying to run `travis login --pro --auto`, you get an error that looks like the following:

```
Bad credentials. The API can't be accessed using username/password a
```

Go to <http://github.com/settings/tokens>. Generate a new token, name it `travis-ci` and select **all** of the scopes underneath.

Hit Generate Token. **SAVE** this token somewhere safe (in your notes, email it to yourself, etc). You will not be able to see it again. Wherever you see `<GITHUB_TOKEN>` below, replace it with this token.

Run `travis login --pro --github-token <GITHUB_TOKEN>` in Codio. Follow the rest of step 3 in the original travis instructions (pasted below for your convenience).

---

Afterwards, since we need to give Travis CI a means to clone our private repo to run CI builds, we need to generate an ssh-key for Travis to use. First identify your repo's org and name using:

```
git remote -v
```

If the above command prints out something similar to the following:

```
origin git@github.com:[myorg]/[myrepo].git
```

Replace the `[myorg]` and `[myrepo]` below and run the command below.

```
travis sshkey --generate -r [myorg]/[myrepo]
```

(Your command should look something like this:)

```
travis sshkey --generate -r cs169/hw-agile-iterations-fa20-000
```

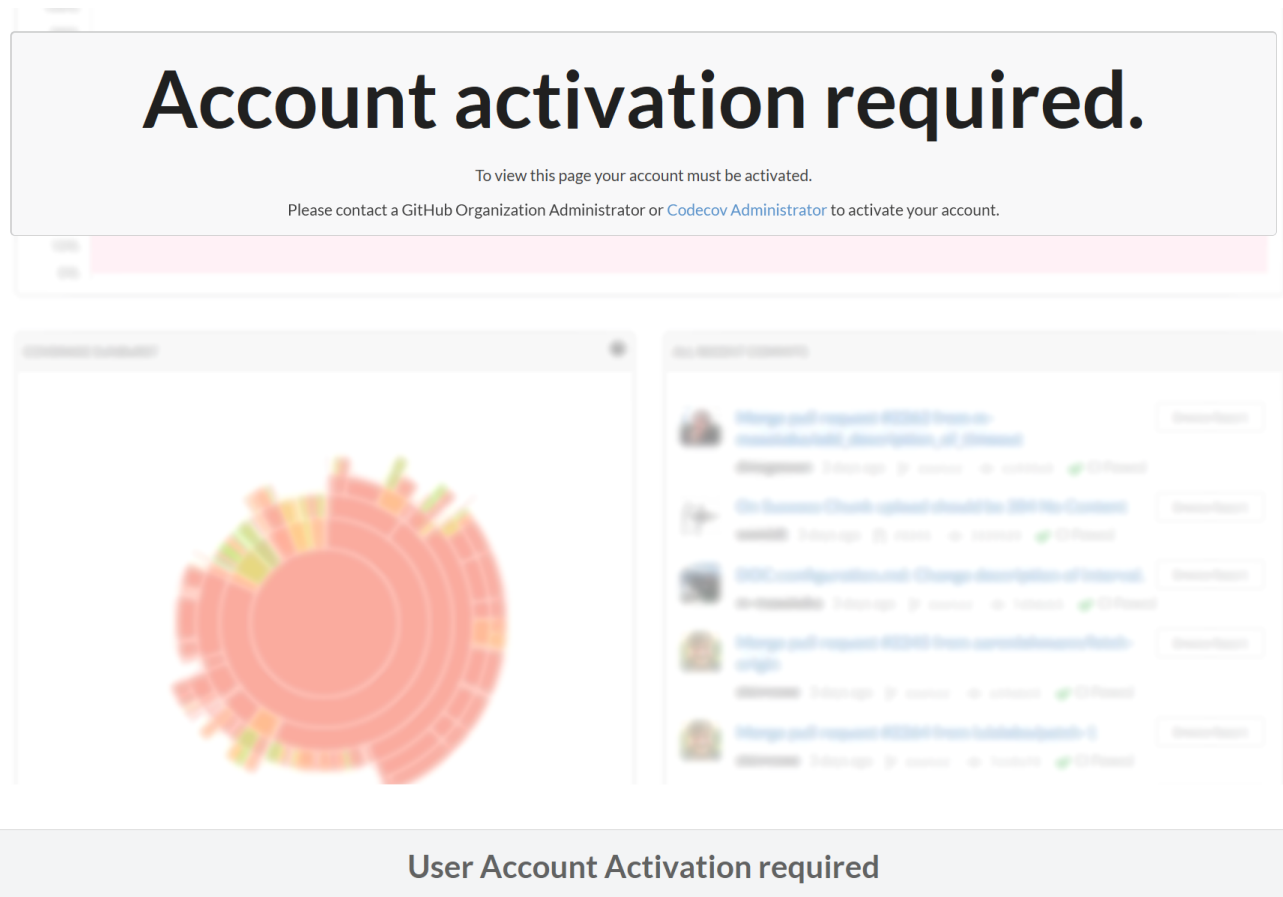
You may receive another prompt for GitHub login when you run the `travis ssh` step above. If you do, enter your GitHub username as usual, and enter `<GITHUB_TOKEN>` as your password. If you see a step saying `Generating RSA key.`, you have succeeded and can move on to Travis Step 4 in the original instructions.

Next

# 11. CodeCov - Common Issues

## CodeCov Errors

If you see an error like the one below:



First, ensure that you are a student. If you are, in your CodeCov Personal Settings, you should see a box similar to

Student Details	
Status	Since
Active	November 12th 2020

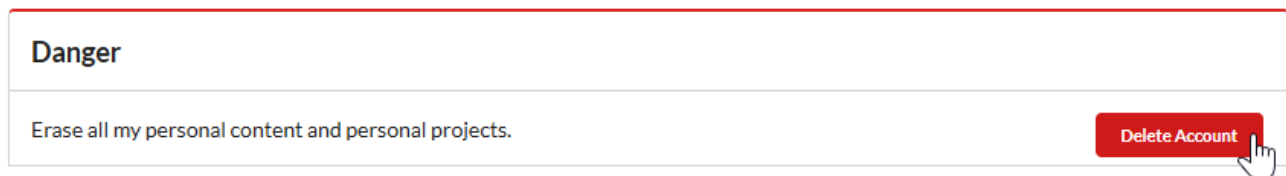
If you do, go to <https://codecov.io/account/gh/cs169/users>, find your name in the list of all users below, and Click the green [Activate](#) button next to your username. If

you see your name and it is not already active and the Activate button is grayed out, check if you have a blue Student tag in your row in the table. If you don't, you aren't a student, so follow the steps below. If you can click Activate, once you have done so, navigate back to your project page and you should no longer see the above error.

### If you are not a student:

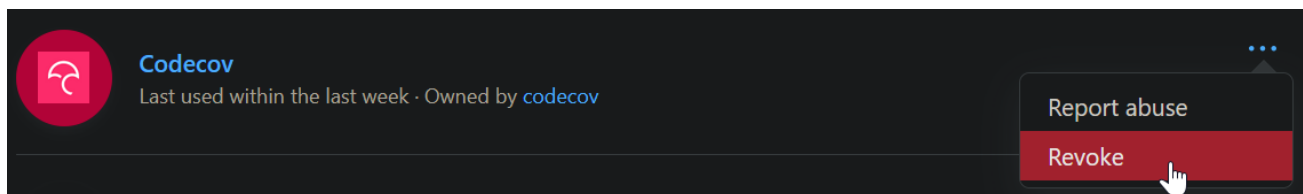
First, [make sure your GitHub account is under the education plan]  
([https://github.blog/2019-07-30-how-to-get-the-github-student-developer-pack-without-a-student-id/#:~:text=Visit%20GitHub%20Education%20and%20click,select%20\(or%20add%20it\).](https://github.blog/2019-07-30-how-to-get-the-github-student-developer-pack-without-a-student-id/#:~:text=Visit%20GitHub%20Education%20and%20click,select%20(or%20add%20it).))  
If it isn't, follow those instructions to add your berkeley.edu email to your account, and you will be able to get approved automatically.

Once you have done that, head over to CodeCov > Personal Settings, and delete your CodeCov account.




Then, clear CodeCov's browser storage data (Instructions for Safari, Chrome, and FireFox). Follow the instructions for clearing cookies from a specific site, so that it does not erase all website data.

Go to <https://github.com/settings/applications> and revoke Codecov's access.




Then, Go to <https://codecov.io/students> and click the blue Sign in with GitHub

button.



**Codecov** is part of the  
**GitHub Student Developer Pack**

The [GitHub Student Developer Pack](#) gives students free access to the best developer tools in one place so they can learn by doing. As part of the program, students get a free license for Codecov.

 Sign in with GitHub

Now, check your CodeCov account's Personal Settings, and you should be a student. Go back to the top of this document, and follow the instructions to activate your account in the cs169 organization on CodeCov.

Next



## 12. Clarifications

---

### Tuesday, November 16th

2. (11:50pm) There was a slight change in the Travis CI command used to push your master key to Travis.

Rather than using this OLD command:

```
travis encrypt --pro RAILS_MASTER_KEY="cat config/master.key" -
```

Use THIS NEW command:

```
travis encrypt --pro RAILS_MASTER_KEY="$(cat config/master.key)"
```

This has been updated in the instructions.

---

### Monday, November 16th

1. (3:30pm) If you pushed to git before we released updated instructions, you may have many extraneous files that should not be in your remote repo. This is because the `.gitignore` was not included until you pulled from the staff repository in Step 0. **After** you have pulled staff changes (in `Getting Started/Pull Staff Changes`), you should see a `.gitignore` file in your repo. Once you have confirmed that you have this file, run

```
git rm -r --cached .
```

You will see that all of the files in your repo will be unstaged for commit. After that, run

```
git add .  
git commit -m "Add .gitignore."  
git push
```

[Mark as Completed](#)[Back to dashboard](#)