FoodZen Documentation

Handy GitHub Commands

- Remove local changes
    - `git checkout .`
    - `git clean -f`

Angular Routes

URL	Template	Controller
/ingredients	app/ingredients/ingredients.html	IngredientController
/recipes	app/home/recipes.html	RecipeController
/grocery	app/grocery/grocery.html	GroceryController
/	app/ingredients/ingredients.html	IngredientController
/signup	app/auth/signup.html	AuthController
/signin	app/auth/signup.html	AuthController
/email	app/grocery/grocery.html	GroceryController
/map	app/map/map.html	MapController

Database Schemas

- Users
    - **user_id:** Number
    - **email:** String (unique and required)
    - **password:** String (hashed with bcrypt)
    - **salt:** String
- Ingredients
    - **email**: String (Foreign key from email in Users)
    - **ingredients**: Array
- Recipes
    - **id**: Number
    - **title**: String
    - **email:** String (Foreign key from email in Users)
    - **image:** String
- Groceries
    - **email:** String
    - **groceries:** Array
- (we don't actually use the user_recipe model)

What happens on the server for each route:

URL	HTTP Verb	Server
/api/users/signin	POST	1) User email and password are passed to server in request body 2) Server looks for user email in users collection 3) If not found, send back an error 4) If found, compare passwords 5) If no match, send error 6) If match, create a jwt token which encodes the user info and send the token back in the response
/api/users/signup	GET	1) User email and password are passed to server in request body 2) Server looks for user email in users collection 7) If found, send back an error (user already exists) 8) If not found, create a new user with the passed in email and password 9) Create a jwt token which encodes the user info and send the token back in the response
/api/ingredients	POST	1) ingredient sent in request body 2) server gets jwt token from header and decodes to get user 3) use user email to find that user's ingredients array in the Ingredients collection 4) add ingredient to array
/api/ingredients	DELETE	1) Same as a POST to api/ingredients, except ingredient is spliced from ingredients array
/api/ingredients	GET	1) Same as POST, but returns array of a user's current ingredients list
/api/recipes	GET	2) server gets jwt token from header and decodes to get user 3) use user email to find that user's ingredients array in the Ingredients collection 4) add ingredients array to query string and make api request
/api/users/recipes	GET	1) Get all a user's saved recipes

/api/users/recipes	POST	1) Save a recipe to user_recipe collection 2) AND save recipe to recipe collection so we can access it later
/api/email/	POST	1) Emails grocery list to recipient using **Mailgun**

APIs Used

- Mailgun
    - Server to manage emailing grocery list
- Google Maps
    - Google maps provides a free key for the google maps embedded api.
    - It provides functionalities such as search, select, and directions.
    - API call was made on the frontend controller.
- Spoonacular
    - Grab Recipe information based on current ingredients
    - Grab nutritional information for each ingredient

Deployment

Designate someone to be deployment admin. They should follow the ADMIN steps in this document first, then let non-admin teammates know when the droplet and git remote is set up. After that, non-admins should follow the "**Running the server on Digital Ocean"** instructions, so that anyone can redeploy, view server-side console logs in the terminal, etc (i.e. you don't need to rely on the admin for that).

ADMIN: Creating a Digital Ocean droplet

Set up a Digital Ocean droplet, following the instructions from the Shortly Deploy sprint:

> Create a new DigitalOcean droplet, using the MEAN on 14.04 image in the "One-click Apps" section of the images. You can choose the smallest size, in any datacenter, give it any hostname you like. and do not need to select any additional options. **Do be sure to create an SSH key for it.**

Instructions for creating an SSH key are further down this doc (more detailed than the instructions linked in the shortly deploy sprint).

Once you've set up the droplet, SSH in by running `ssh root@<your droplet IP address here>`. Clone the project into the home directory and follow steps 6 onward from "How to wipe the work-tree from the droplet and start over from a fresh clone," substituting in your own IP address and API key.

ADMIN: Set up a git remote for easy deployment

Follow these instructions - https://www.digitalocean.com/community/tutorials/how-to-set-up-automatic-deployment-with-git-with-a-vps

You can also reference the shortly deploy readme.

ADMIN: authorize your teammates' SSH keys

Reference: https://www.digitalocean.com/community/tutorials/how-to-create-ssh-keys-with-putty-to-connect-to-a-vps ("Save The Public Key On The Server" section)

This thing only needs to be done once. From inside the droplet:

1. `chmod 0700 ~/.ssh`
2. `chmod 0644 ~/.ssh/authorized_keys`

Each time you want to add a new authorized SSH key:

1. SSH into the droplet - `ssh root@<your ip address>`
2. `sudo vim ~/.ssh/authorized_keys`
3. Press 'i' to enter insert mode
4. Paste in the new SSH key. Include the entire thing, including the ssh-rsa bit at the beginning, the key itself, and the name@Names-Computer.local bit at the end
5. Press esc to get back in command mode
6. Type `:wq` and hit enter to save and close.

Running the server on Digital Ocean

To access and push to your admin's DO droplet, do this stuff once:

1. Set up SSH keys (https://www.digitalocean.com/community/tutorials/how-to-use-ssh-keys-with-digitalocean-droplets). From your terminal:
    a. `cd ~`
    b. `ssh-keygen -t rsa`
    c. hit enter to accept the default file path
    d. make and remember a passphrase if you want to (you don't have to). hit enter.
    e. It should say something like: "Your identification has been saved in /Users/yourname/.ssh/id_rsa

    f. Copy paste that file path, and cd to the .ssh folder, should be something like: `cd /Users/yourname/.ssh`
    g. `cat id_rsa.pub`
    h. it should print your ssh key, starting with "ssh-rsa" and ending with something like "yourname@Your-Macbook.local"
    i. copy-paste your ssh key. This is your public key and can be made available to team members.
    j. Your admin (or someone else who's already authorized) will add your SSH key as an authorized key to the droplet, and let you know when you should be able to SSH in.

2. Once you've been given the thumbs up, trying SSH-ing in to see if it works. From your terminal:
    a. `ssh root@<your ip address>` to SSH into the droplet. Answer "yes" to the question.
    b. you should now be able to explore the droplet machine. You can expect to see a ~/GigglingGoiters directory with our project in it, and a /var/repo/site.git directory that lets us do our `git push live master` thing. Keep in mind we all have access to this machine, so changes may happen that you didn't do.

3. In your local repo, set a "live" remote:
    a. `git remote add live ssh://root@<your ip address>/var/repo/site.git`

<u>Do this stuff each time you want to redeploy to the droplet</u>:

1. In your local repo, `git status` to make sure you are on your master branch.
2. `git pull --rebase upstream master` to get latest from GigglingGoiters/GigglingGoiters.
3. `git push live master`
4. Go to http://<your ip address>:3000/ to see your new changes.

Troubleshooting:

- If the server seems to be down, SSH in and do `node server/server.js` inside foodZen.
- If the server is running, but doesn't seem to have changed, ping the group to kill the server if they have it running (exiting the terminal tab should do the trick, or ctrl-c). Then SSH in and restart the server.
- If it's not letting you `git push live master` at all, **carefully** double-check that you are on your master branch, and that your local `git log` matches your canonical github org's commit history. Then do `git push live master --force`.

Sidenote: You don't need to worry about this, but in case of confusion... if you were to ssh into root@<your ip address> and cd into the work-tree (`cd ~/GigglingGoiters`), you would see "unstaged changes" when you git status. That's because the actual git versioning is taking place in the git-dir, i.e. /var/repo/site.git, and the commits received there are checked into the

work-tree. If you cd'd to /var/repo/site.git and did `git log` there, THAT one should match the git log of your local that you did `git push live master` on.

How to wipe the work-tree from the droplet and start over from a fresh clone
Note: git push live master should still work after you do this

1. In your terminal, `ssh root@<your ip address>` to SSH into the droplet. Answer "yes" to the question.
2. `cd ~`
3. You should see GigglingGoiters in the home directory when you `ls`
4. `rm -rf GigglingGoiters`
5. `git clone https://github.com/GigglingGoiters/GigglingGoiters.git GigglingGoiters`
6. `cd GigglingGoiters/server/env`
7. `cp env.example.js env.js`
8. `vim env.js`
9. Press 'i' to enter insert mode
10. Paste in a valid API key with a comment of when the key expires
11. Press esc to get back in command mode
12. Type `:wq` and hit enter to save and close.
13. `cat env.js` to double check it
14. `cd ../..` back to foodZen folder
15. `npm install`
16. `sudo bower install --allow-root`
17. `node server/server.js`
18. In your browser, navigate to http://<your ip address>:3000/ to see the deployed app running

Setting up a cool domain name (e.g. mydomain.com) for your digital ocean droplet

Buy the name you want at namecheap.com or similar.

This is basically what you need to do:
https://www.digitalocean.com/community/tutorials/how-to-point-to-digitalocean-nameservers-from-common-domain-registrars

The namecheap UI has changed since the above was written, so this is helpful for finding where in the new UI you would do the above:
https://www.namecheap.com/support/knowledgebase/article.aspx/767/10/how-can-i-change-the-nameservers-for-my-domain

The first link points here for the next step:
https://www.digitalocean.com/community/tutorials/how-to-set-up-a-host-name-with-digitalocean

Important notes about ports:
The above steps basically turn your domain name into a mirror of your droplet ip address. You could think of it as telling your DNS “when I navigate to mydomain.com, I want you to redirect me to whatever’s showing at this IP address.”

If your server is running on port 3000 or similar, you will need to go to mydomain.com:3000 to see your app, just like you had to add :3000 to your ip address. Obviously, we’d rather not have to type :3000 after mydomain.com.

The way around this is to have our server run on port 80, the standard web port. When navigating to an IP address without specifying a port, a web server will default to serving up whatever’s running at port 80.

HOWEVER, if you were to hard-code to port 80 in server.js and try to run it locally, it will not work. Port 80 is a special reserved port and cannot be used for development. Instead, in server.js, we did:
var port = process.env.PORT || 3000;

Then inside the digital ocean droplet ONLY, we create a PORT environment variable and set it to 80. This way, when we run server.js locally, it runs on 3000, but when we run server.js on our droplet, it runs on 80.

How to set the PORT environment variable (you might have to do this periodically):

SSH into the droplet. From anywhere, run:
`echo $PORT`

This will show what the PORT environment variable is currently set to. If there is no PORT env variable, it will show up blank. (It was blank when we did it.)

To set PORT to 80, run this:
`export PORT=80`

Now when you do `echo $PORT` you should see “80”, and when you run server.js, you should see “listening on port 80.” When you navigate to your IP or to mydomain.com, you should see your app running.