Yuko Uda
Brittany Davis
Christopher Zenner

Full Stack Web Developers

The Back Story

Once upon a time, there were three aspiring full stack web developers, who were tasked with coming up with an original project for their coding boot camp. Clueless, they racked their little geek brains.

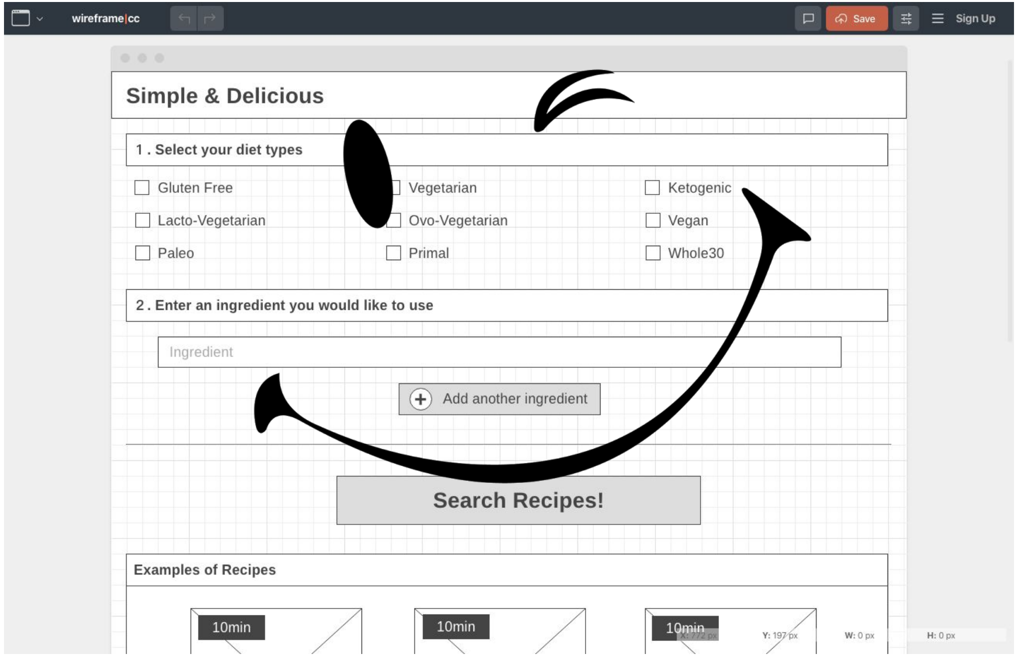
“What are you interested in?” Chris asked Yuko.

“Cooking, wine, and dogs!” Yuko replied.

And Brittany, well, she was hoping to God the other two would make up their minds and decide on a cooking app.

And thus Project 1 was born.

And then the wireframes started appearing out of nowhere ...



Simple & Delicious

1 . Select your diet types

☐ Gluten Free
☐ Lacto-Vegetarian
☐ Paleo
☐ Vegetarian
☐ Ovo-Vegetarian
☐ Primal
☐ Ketogenic
☐ Vegan
☐ Whole30

2 . Enter an ingredient you would like to use

Ingredient

⊕ Add another ingredient

Search Recipes!

Examples of Recipes

10min

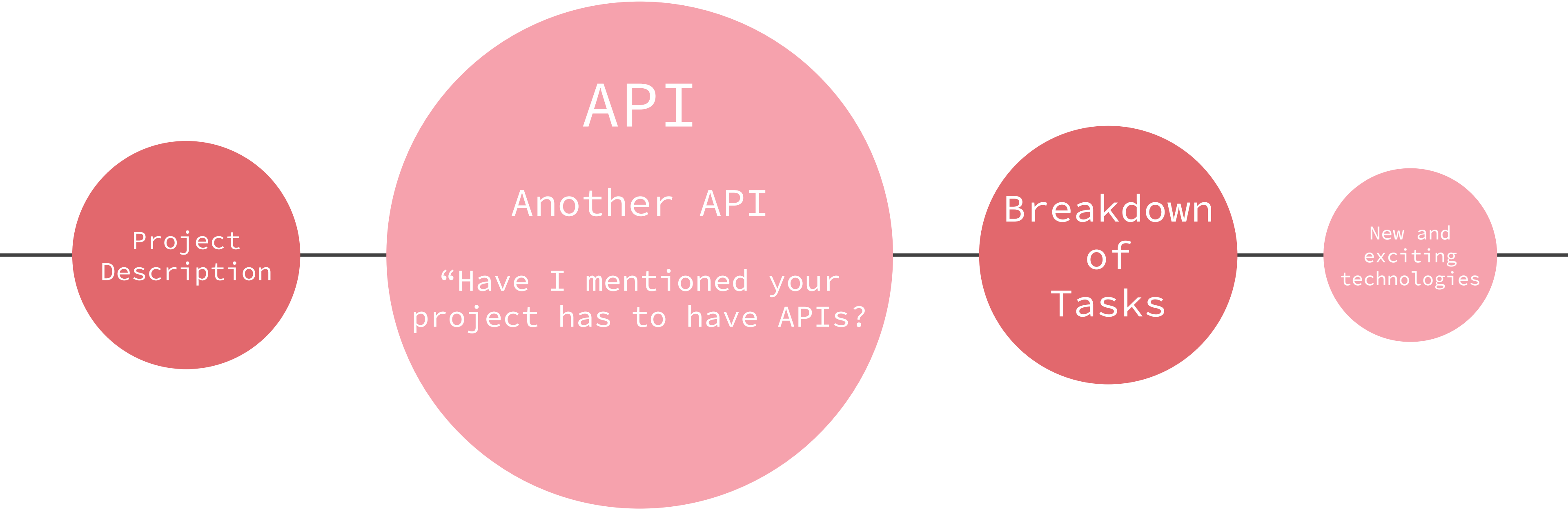
10min

10min

X: 772 px Y: 197 px W: 0 px H: 0 px

Meanwhile …

Their instructor had some ideas of his own. “Not so fast, my young developers!” Diarmuid exclaimed, “That little project of yours has to have a few things:”



Project Description

API

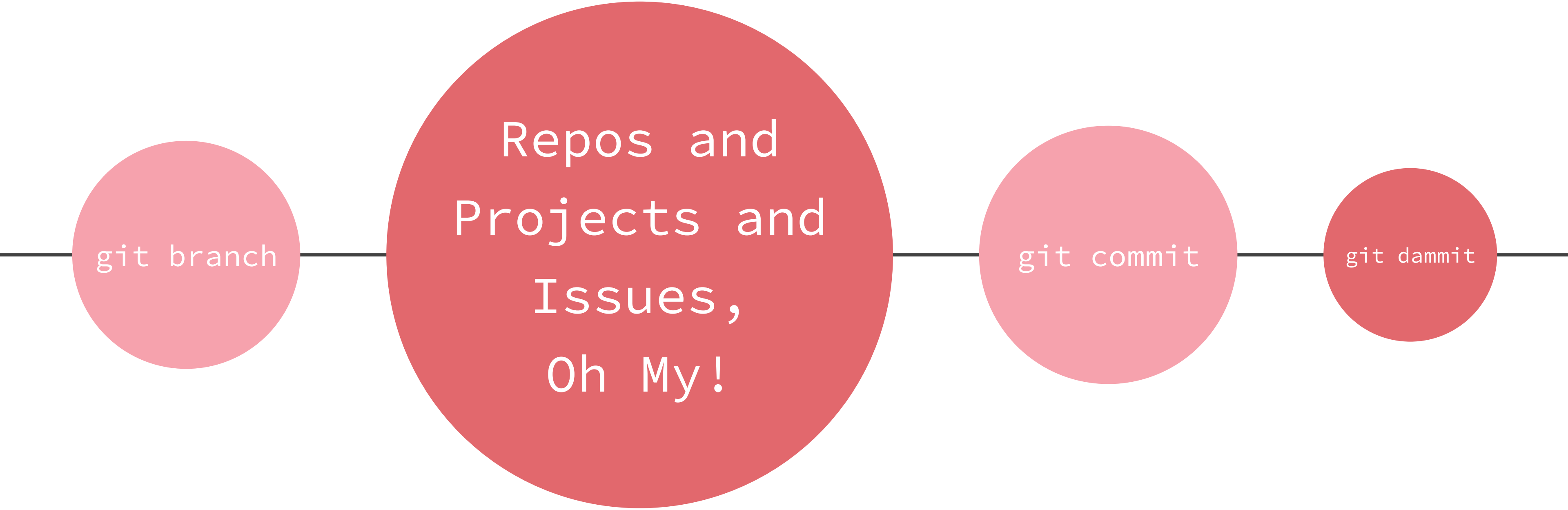
Another API

“Have I mentioned your project has to have APIs?

Breakdown of Tasks

New and exciting technologies

“And don’t forget about all that gooey Git goodness!”



git branch

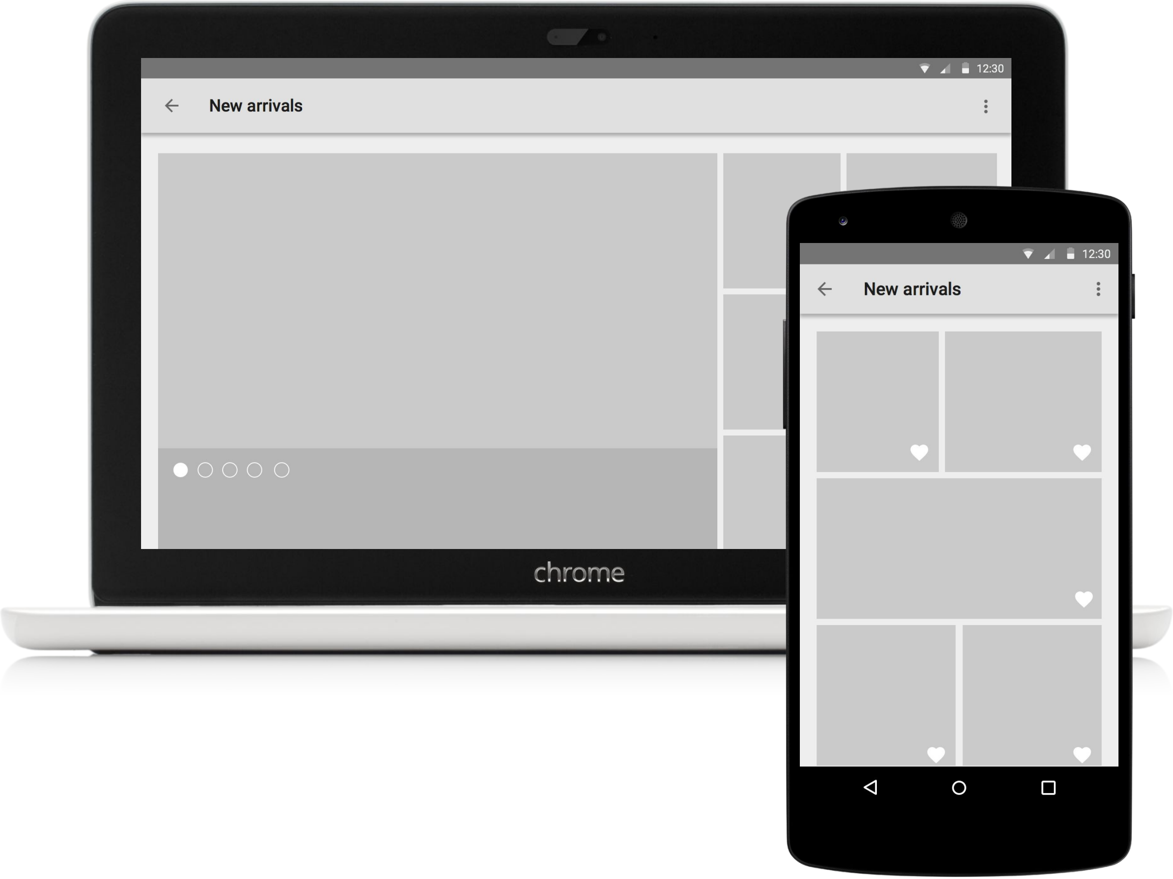
Repos and Projects and Issues, Oh My!

git commit

git dammit

OOPS!

And so the three wayfaring newbie developers went back to the drawing board to retrofit a problem their app would address, a target audience for their app, and just a smattering of essential user stories. Oh, and have I mentioned a primary goal?



Project Description

We will build a recipe app that allows the user to select a diet type (*e.g.*, vegetarian, Kosher, gluten-free, etc.) and optionally specify a list of ingredients. The app will then use an API call to one/more recipe services to pull back recipes conforming to the specified parameters--that is, diet type and ingredients. The results will be displayed in an aesthetically pleasing manner using a carousel. When the user selects a particular recipe from the carousel, the recipe will load full-page and, at the bottom, display a form field that will allow the user to input the number of people to serve. The app will then calculate the ingredient portions accordingly. Finally, the app will use local storage so the user can persist one/more favorite recipes.

Target Audience

Who is your target audience?

Target audience comprises amateur chefs, busy professionals, working moms, etc.

Key Problem

What is the key problem this product will address?

a. Helping people find simple but delicious recipes (Cooking time is important here.)
b. Accommodating special diets and/or dietary restrictions.
c. Making adjustments of serving sizes easier.

Primary Goal

What is the primary goal of this product?

To make cooking easier for amateur chefs, busy professionals, working moms, etc.

Essential User Stories

Identify and prioritize essential user stories.

- As a user, I want to find recipes that conform to my specific diet/dietary restrictions.
- As a user, I want to be able to modify recipe portions to feed my specific number of people.
- As a user, I want to find recipes that are simple (*i.e.*, six ingredients or less), delicious, and 30 minutes to cook or less.

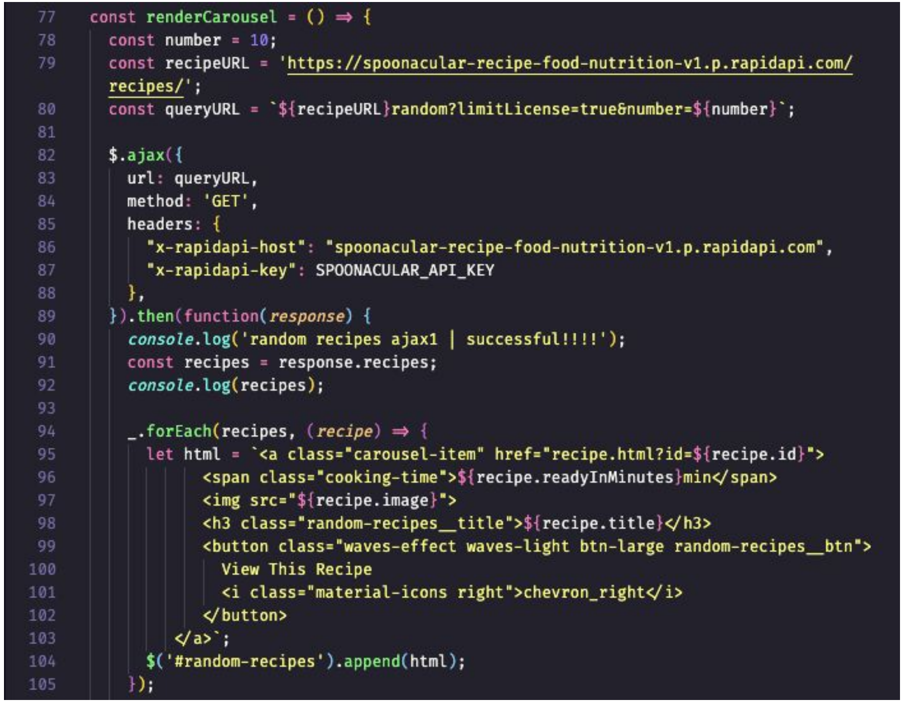
Back to our regularly scheduled programming (literally) ...

```
$(document).ready(() => {
```

After two long weeks of planning, coding, and pulling their hair out … coding … and pulling their hair out some more, the trio of FSWDs—with copious help from their instructors—finally created an MVP!

```
return MVP;
```

```
});
```



```
const renderCarousel = () => {
  const number = 10;
  const recipeURL = 'https://spoonacular-recipe-food-nutrition-v1.p.rapidapi.com/recipes/';
  const queryURL = `${recipeURL}random?limitLicense=true&number=${number}`;

  $.ajax({
    url: queryURL,
    method: 'GET',
    headers: {
      "x-rapidapi-host": "spoonacular-recipe-food-nutrition-v1.p.rapidapi.com",
      "x-rapidapi-key": SPOONACULAR_API_KEY
    },
  }).then(function(response) {
    console.log('random recipes ajax1 | successful!!!!');
    const recipes = response.recipes;
    console.log(recipes);

    _.forEach(recipes, (recipe) => {
      let html = `<a class="carousel-item" href="recipe.html?id=${recipe.id}">
            <span class="cooking-time">${recipe.readyInMinutes}min</span>
            <img src="${recipe.image}">
            <h3 class="random-recipes__title">${recipe.title}</h3>
            <button class="waves-effect waves-light btn-large random-recipes__btn">
              View This Recipe
              <i class="material-icons right">chevron_right</i>
            </button>
          </a>`;
      $('#random-recipes').append(html);
    });
```

CookQuik

Simple and delicious recipes for the modern busy household

The Autopsy

Of course, no good story would be complete without an autopsy …

(Wait … WHAT?! 😳)

Don’t worry, boys and girls, no one died for this autopsy! (They just felt like they did after staying up all night the night before the project was due.)

So what did our three protagonists manage to disect?

Lessons Learned

- Don’t code before you plan!
- Limit your scope. (And then limit it again.)
- Read the API documentation *thoroughly*!

Oh, and ...

- Bjorn knows WAY more about webpack (and API calls) than he’s willing to concede!

Technologies Implemented

- ✓ webpack
- ✓ Materialize
- ✓ Sass
- ✓ Spoonacular API
- ✓ YouTube API
- ✓ Lodash
- ✓ query-string
- ✓ Math.js
- ✓ Naps.io

THE END