1. Project Setup (5 min)

a. Create an Octane app.

```
ember new acm-octane
```

- Open the project in an IDE.
- Mention that most things happen in the `app` folder. Assure participants to not be overwhelmed. They'll learn what's inside `app` as they go.

b. Run the app.

```
cd acm-octane
ember s
```

- Visit `localhost:4200`.
- Edit `application.hbs` to show `Hello world!`. Keep `{{outlet}}` around.
- Point out that templates are HTML. We will make them reactive soon. (Wouldn't it be nice if the page says `Hello {customer}!`?)

c. Create a GitHub repo with the same name.

```
git remote add origin git@github.com:ijlee2/acm-octane.git
git push origin master
```

d. Deploy the project on Heroku.

```
heroku create --buildpack https://codon-
buildpacks.s3.amazonaws.com/buildpacks/heroku/emberjs.tgz
```

- Find the project on Heroku Dashboard. Enable auto-deploy.

2. Addons (5 min)

a. Install Mirage and Faker.

```
ember i ember-cli-mirage
npm i faker --save-dev
```

- Explain the benefits of Mirage. (Mocking, prototyping, and testing.)
- Mirage can now be used with other major JS frameworks.

b. Edit `app/config/environment.js` to run Mirage in production.

```
if (environment === 'production') {
    ENV['ember-cli-mirage'] = {
        enabled: true
    };
}
```

- If we forget this step, we will encounter "Unexpected token < in JSON at position 0" when we load data in the production app.

c. Install Sass and Ember CSS Modules.

```
ember i ember-cli-sass ember-css-modules ember-css-modules-
sass
npm i postcss-nested
```

- Explain the benefits of Sass. (Nesting, BEM, and variables for style guide.)
- Explain the benefits of localized styles. (Collaborate easily. Refactor styles with confidence.)

- Explain why postcss-nested is needed. (To use nesting and BEM.)
- Change `app.css` to `app.scss`.

d. Update `ember-cli-build.js`.

```
const EmberApp = require('ember-cli/lib/broccoli/ember-app');
const PostcssNested = require('postcss-nested');

module.exports = function(defaults) {
    let app = new EmberApp(defaults, {
        cssModules: {
            plugins: {
                before: [PostcssNested]
            }
        }
    });

    return app.toTree();
};
```

e. Restart the app.

- Verify that the participants' apps are still working.
- Push to GitHub.

3. Routes (10 min)

a. Motivate why we make routes, a.k.a. pages. (Each page can load and show different data to a user.)

b. Create three pages: Home, Products, and Cart.

```
ember g route index
ember g route products
ember g route checkout
```

- Briefly go over the files that are generated. Have a look at `app/router.js`.
- Add a header title `Home` to the `index` page, etc.
- Note that `Hello world!` appears in all pages. Ask participants how each page is also able to show something different.
- Remove `Hello world!` from `application.hbs`. Instead, add a navigation so that we can easily switch pages.

c. We want to display products when a user visits `/products`.

- Show that the `model` hook can return a static value for prototyping.
- Dial up complexity (string → object → array of objects) and show how to render the value: `{{@model}}` → `{{@model.name}}`, `{{@model.price}}` → `{{#each @model as |product|}} … {{/each}}`.
- Push to GitHub so that exercise code can be stashed if it doesn't work.

Exercise. On `/checkout`, display 3 items by their name and price.

- Encourage participants to start simple: show 1 item, then 2, then 3.

4. Models (5 min)

a. Explain why we make models. (They define what a user may want to see. They can also represent a contract between backend and frontend teams.)

b. Create `product` model.

```
ember g model product
```

c. Define the model's attributes.

```
import Model, { attr } from '@ember-data/model';


export default class ProductModel extends Model {
    @attr name;
    @attr price;
}
```

- Mention that `@` is how Ember Data knows `name` and `price` are attributes. Don't go into details about decorators.

Exercise. Add 4 more attributes.

- What to add:
  - `description` - description of the product
  - `imageUrl` - URL to a photo of the product
  - `rating` - number used for a 5-star rating system
  - `seller` - name of the entity that sells the product

d. Update the model hook for `products`.

```
model() {

    return this.store.findAll('product');

}
```

- Open the Console. Show that we get an error when we visit `/products`.
- Explain what the error message means.
- Push to GitHub.

5. Mirage (10 min)

a. Mock the endpoint `GET /products`.

```
export default function() {
    this.get('/products');
}
```

- Point out how adding the endpoint fixes the error in Step 4d.
- We don't see any products when we visit `/products`, however.

b. We can manually create data. This is useful when we want some data to always show up in a dev or test environment.

```
export default function(server) {
    server.create('product', {
        name: 'Vanilla Ice Cream Cake',
        description: 'Made with organic herbs',
        price: 40,
        rating: 4.5,
        seller: 'Amy\'s'
    });
}
```

- We should now see 1 product when we visit `/products`.
- Open Ember Inspector. Show how to find and live-edit a product.

Exercise. Create 1 more product using `server.create`.

- Push to GitHub.

c. A factory can create many data. Use Faker and the provided helpers.

```
ember g mirage-factory product
```

- Before we define the factory, we can call `server.createList` to add 30 more products. We see placeholders on the products page.
- Helper file: https://bit.ly/3b35uUw
- What to return:
  - baseModel - generateBaseModel()
  - name - `${faker.commerce.productMaterial()} ${this.baseModel.name}`
  - price - randomInteger(1, 100)
  - description - faker.lorem.sentences()
  - imageUrl - this.baseModel.imageUrl
  - rating - 0.5 * randomInteger(1, 10)
  - seller - faker.company.companyName(0)

d. Currently, we only see name and price in `/products`. Show the image too.

```
<div>
    <img
        alt=""
        role="presentation"
        src={{product.imageUrl}}
    >
    <div>{{product.name}}</div>
    <div>${{product.price}}</div>
</div>
```

- Push to GitHub.

6. Components (5 min)

a. Explain why we make components. (We can easily see what's happening in a template. We can reuse code in different places and test our app better.)

b. Create a component called `<ProductCard>`.

```
ember g component product-card --gc --pod
```

- Explain why we use `gc` and `pod` flags. (When prototyping, we may not know yet if we need a backing class. Pods help us organize our project more.)

c. Move the code from Step 5d to the template.

- Replace all instances of `product` with `@product`.
- Explain that `@` means the value was passed from outside.
- Call `<ProductCard>` in the `products` route template.
- Confirm that the page looks exactly the same.

d. Add a local property for illustration.

```
export default class ProductCardComponent extends Component {
    isAmazing = (this.args.product.rating === 5);
}
```

- Edit the template. Show a 4px-gold border if the product is amazing.
- Explain the benefits of the syntax `@` and `this`. (We know where things come from. We can guess which file to look next.)
- Push to GitHub.

7. Styling (10 min)

a. Let’s use Sass and Ember CSS Modules to spruce up our app. Explain where stylesheets for a route template and a component template go.

```
touch app/styles/application.scss
touch app/styles/products.scss
touch app/components/product-card/styles.scss
```

b. Show how to use `local-class` using `<ProductCard>` as example.

```
<div local-class="container">
    <img
        alt=""
        local-class="image"
        role="presentation"
        src={{@product.imageUrl}}
    >
    <div local-class="information">
        <div local-class="name">
            {{@product.name}}
        </div>
        <div local-class="price">
            ${{@product.price}}
        </div>
    </div>
</div>
```

Exercise. Style the application and products routes.

- What to style:
  - Navigation in `application.hbs`
  - List of products in `products.hbs`
- Show the finished app for design ideas.
- Encourage participants to pursue their style. Do advise them to keep it simple (e.g. no mobile resolution, minimal color set, no shadow, no animation), since we are at the prototype stage and have limited time for workshop.
- For a custom font, use Google Fonts. Show how to add the `<link>` tag in `app/index.html`.
- Push to GitHub.

8. Routes Revisited (5 min)

a. When the user clicks on a product card, we will redirect them to the "details" page. Explain why a details page is useful. (In the "parent" page, we can load partial data fast. The user isn't overwhelmed by seeing all information at once.)

b. Create a new route.

```
ember g route products/product
```

c. Edit the router map to allow dynamic URLs.

```
Router.map(function() {
    this.route('products', function() {
        this.route('product', { path: '/:id' });
    });

    this.route('checkout');
});
```

d. Add links to the products page.

```
<LinkTo
    @route="products.product"
    @model={{product.id}}
>
    <ProductCard
        @product={{product}}
    />
</LinkTo>
```

- Click on a card to show that the URL changes.

e. We want to load data for a particular product. Update the model hook for `products.product`.

```
model(params) {
    return this.store.findRecord('product', params.id);
}
```

- Open the Console. Show that we get an error when we visit `/products/1`.
- Explain how to use the error message to build the endpoint in Mirage.

f. Mock the endpoint `GET /products/:id`.

```
export default function() {
    this.get('/products');
    this.get('/products/:id');
}
```

- Again, point out that mocking the endpoint fixes the error.
- We don't see any details when we visit `/products/1`, however.
- Push to GitHub.

9. Components Revisited (5 min)

a. Update the route template.

```
<div>
    <h2>{{@model.name}}</h2>

    <ProductDetails
        @product={{@model}}
    />
</div>
```

- Open the Console. Show that we get an error for a different reason this time: We don't have the `<ProductDetails>` component yet.

b. Create a component called `<ProductDetails>`.

```
ember g component product-details --gc --pod
```

- Display the product's image, description, price, rating, and seller.
- Keep styling basic to save time.

c. Add an action for illustration. Make a button called `Add to Cart` and assign an on-click event.

```
@action addProductToCart() {
    const { id, name } = this.args.product;

    alert(`Added ${name} to cart! (ID: ${id})`);
}
```

- The component is supposed to show the details of a product. However, since the button allows adding the product to the cart, the component knows more than it should about our application (what goes on outside the component). It's also hard to reuse the component in different pages.
- One way to fix this problem is to use Data Down, Actions Up.

d. Explain the benefits of Data Down, Actions Up. (If only 1 thing is responsible for updating data, we can predict what value the data will have and why with more confidence.)

- Briefly mention that the `products.product` route, which was responsible for loading the data and passing it down to the `<ProductDetails>` component, could take care of adding the product to the cart.
- Don't go into details about how, since it would involve creating a controller and using `{{yield}}`. For now, ask participants to treat everything in `this.args` as read-only.
- Push to GitHub.
- Check the deployed website one last time. Enjoy!

10. Miscellaneous (30 min)

Don’t worry if you didn’t finish the workshop in 1 hour. Give yourself a pat for making this far and leading the workshop!

If you have time and participants want to learn more, here are a few topics to choose from:

a. Use QUnit DOM, Ember Test Selectors, and Mirage in tests.

- Write application tests for visiting `/products` and `/products/1`.

b. Use Tailwind for prototyping styles.

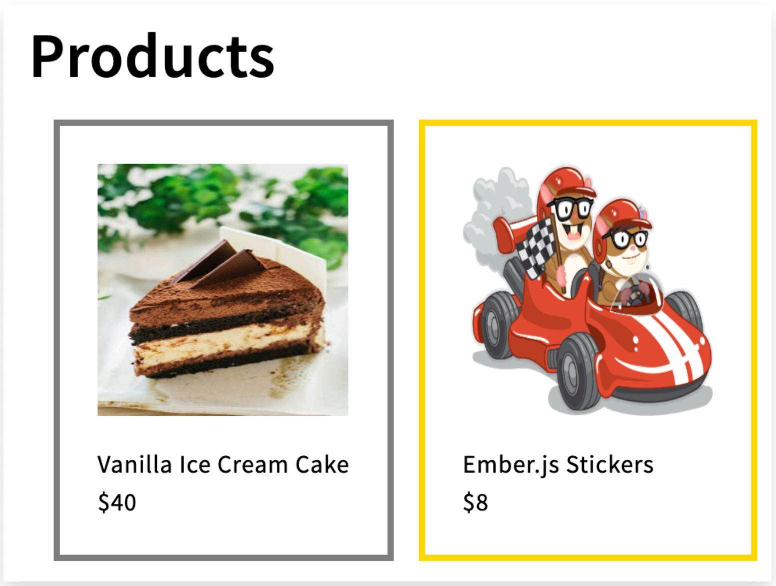
- Make a new Ember app.
- Walk participants through Chris Masters’ installation guide. https://github.com/chrism/emberjs-tailwind-purgecss#very-quick-start

c. Discuss tracked properties and element modifiers.

- Refer to Ember Animated and Lights Out apps for examples. https://crunchingnumbers.live/2019/12/23/rewriting-apps-in-ember-octane/

d. Deploy app on GitHub Pages.

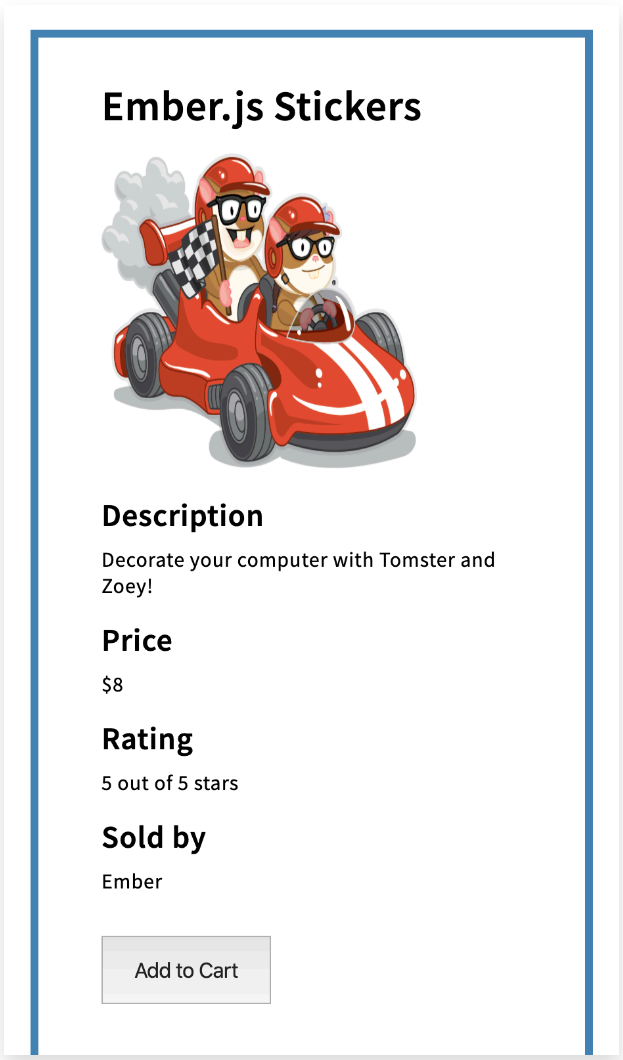
- Make a new Ember app.
- Create a new GitHub repo, connect it to the app, and make a branch called `gh-pages`.
- Walk participants through the ember-cli-deploy-git readme.



Products

Vanilla Ice Cream Cake
$40

Ember.js Stickers
$8



Ember.js Stickers

Description

Decorate your computer with Tomster and Zoey!

Price

$8

Rating

5 out of 5 stars

Sold by

Ember