
TRP.
407
try
ember

Level 3

Models and Services

Organizing the Data

The same data structures are being used across the app and should be centralized.

app/routes/orders.js

```
export default Ember.Route.extend({
  model() {
    return [
      { id: '1', name: 'Nate' },
      { id: '2', name: 'Gregg' }
    ];
  }
});
```

app/routes/order.js

```
export default Ember.Route.extend({
  model(params) {
    return [
      { id: '1', name: 'Nate' },
      { id: '2', name: 'Gregg' }
    ].findBy('id', params.order_id);
  }
});
```

We need product and order data throughout the app.
Duplicating it everywhere makes it difficult to keep in sync.

Generating a Service

Services are long-living objects (aka, “singletons”) that are available throughout your app.

```
ember generate service <service-name>
```

Console

```
$ ember generate service store
installing service
  create app/services/store.js
  …
```

Services are good for:
Centralized logging
User sessions
WebSocket management
Data repositories

We’ll store our data in a data repository service.

Defining a Service

Services are defined in app/services and extend Ember.Service.

The file name matches the service name ("store").

app/services/store.js

```
import Ember from 'ember';

export default Ember.Service.extend({
});
```

This was generated from ember generate.

Centralizing the Data

With a service in place, the shared data can be moved from the routes.

app/services/store.js

```
import Ember from 'ember';

export default Ember.Service.extend({
});
```

app/routes/orders.js

```
export default Ember.Route.extend({
  model() {
    return [
      { id: '1', name: 'Nate' },
      { id: '2', name: 'Gregg' }
    ];
  }
});
```

Centralizing the Data

With a service in place, the shared data can be moved from the routes.

app/services/store.js

```
import Ember from 'ember';

export default Ember.Service.extend({
  getOrders() {
    return [
      { id: '1', name: 'Nate' },
      { id: '2', name: 'Gregg' }
    ];
  }
});
```

app/routes/orders.js

```
export default Ember.Route.extend({
  model() {




  }
});
```

New function to return the data

Injecting the Service

Service objects are made available within another object using Ember.inject.service().

app/services/store.js

```
import Ember from 'ember';

export default Ember.Service.extend({
  getOrders() {
    return [
      { id: '1', name: 'Nate' },
      { id: '2', name: 'Gregg' }
    ];
  }
});
```

app/routes/orders.js

```
export default Ember.Route.extend({
  model() {
    const store = this.get('store');
    return store.getOrders();
  },

  store: Ember.inject.service('store')
});
```

The local name of the service

The name of the service to inject

After injection, the store service becomes available as the "store" property.

Injecting the Service

Service objects are made available within another object using Ember.inject.service().

app/services/store.js

```
import Ember from 'ember';

export default Ember.Service.extend({
  getOrders() {
    return [
      { id: '1', name: 'Nate' },
      { id: '2', name: 'Gregg' }
    ];
  }
});
```

app/routes/orders.js

```
export default Ember.Route.extend({
  model() {
    const store = this.get('store');
    return store.getOrders();
  },

  store: Ember.inject.service()
});
```

Because the service name matches the local property name, we can leave it off.

Centralize the Data Filtering

Now that the data is in the service, the service can be used to find and filter the app data.

app/services/store.js

```
import Ember from 'ember';

export default Ember.Service.extend({
  getOrders() {
    return [
      { id: '1', name: 'Nate' },
      { id: '2', name: 'Gregg' }
    ];
  }
});
```

app/routes/orders/order.js

```
export default Ember.Route.extend({
  model(params) {
    return [
      { id: '1', name: 'Nate' },
      { id: '2', name: 'Gregg' }
    ].findBy('id', params.order_id);
  }
});
```

Centralize the Data Filtering

Now that the data is in the service, the service can be used to find and filter the app data.

app/services/store.js

```
import Ember from 'ember';

export default Ember.Service.extend({
  getOrderById(id) {
    const orders = this.getOrders();
    return orders.findBy('id', id);
  },

  getOrders() { /* … */ }
});
```

app/routes/orders/order.js

```
export default Ember.Route.extend({
  model(params) {
    return [
      { id: '1', name: 'Nate' },
      { id: '2', name: 'Gregg' }
    ].findBy('id', params.order_id);
  }
});
```

Centralize the Data Filtering

Now that the data is in the service, the service can be used to find and filter the app data.

app/services/store.js

```
import Ember from 'ember';

export default Ember.Service.extend({
  getOrderById(id) {
    const orders = this.getOrders();
    return orders.findBy('id', id);
  },

  getOrders() { /* … */ }
});
```

app/routes/orders/order.js

```
export default Ember.Route.extend({
  model(params) {
    const id = params.order_id;
    const store = this.get('store');
    return store.getOrderById(id);
  },

  store: Ember.inject.service()
});
```

Everything Still Works!

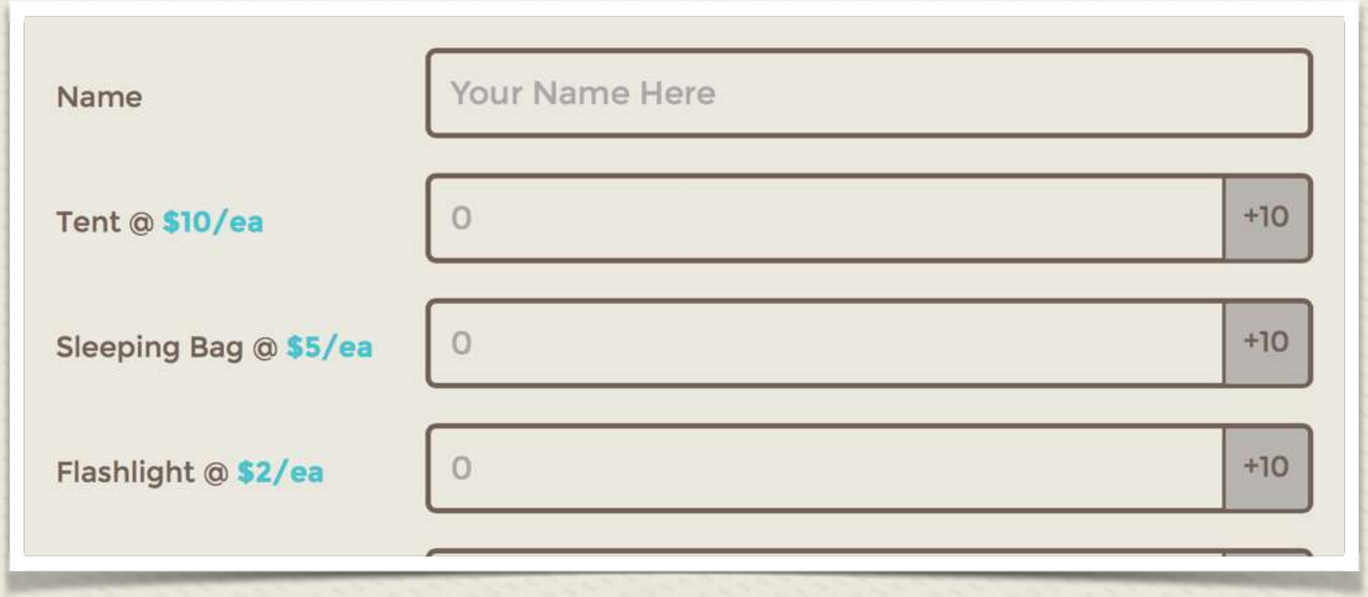
Filling Out the Data

Now that the data is centralized into the store service, we can replace the placeholder data.

Filling Out the Product Data

From the menu page, we see that products have four properties.

Product
1. title
2. price
3. description
4. imageurl



Filling Out the Order Data

From the orders page, we see that orders have two properties.

Product

1. title
2. price
3. description
4. imageurl

Order

1. name
2. line items

Filling Out the LineItem Data

From the orders page, we see that LineItems have two properties.

Product
1. title
2. price
3. description
4. imageurl

Order
1. name
2. line items

Line Item
1. product
2. quantity

Formalizing the Data

We've got three distinct types of objects. Let's start formalizing them using Ember.

Product
1. title
2. price
3. description
4. imageUrl

Order
1. name
2. line items

Line Item
1. product
2. quantity

We'll move away from plain, simple JavaScript placeholder objects and formalize them using Ember.

Introducing Models

Models represent the underlying (and sometimes persisted) data of the application.

Product
1. title
2. price
3. description
4. imageurl

Models are defined in app/models.

app/models/product.js

```
import Ember from 'ember';

export default Ember.Object.extend({
});
```

This defines the product model as a subclass of Ember.Object.

But why extend from Ember.Object?

Extending from Ember.Object

Ember.Object is the base of many of Ember's objects (including Ember.Route, for example).

app/models/product.js

```
import Ember from 'ember';

export default Ember.Object.extend({
});
```

Product
1. title
2. price
3. description
4. imageUrl

Ember.Object provides:

1. A consistent interface for creating and destroying records
2. Object lifecycle events and hooks
3. Properties and property observation functionality

This is how templates are updated when properties change.

Interacting With Models

Properties are read and set using get() and set().

app/models/product.js

```
import Ember from 'ember';

export default Ember.Object.extend({
});
```

Ember.Object provides create(). Record properties may optionally be passed in at creation.

```
var product = Product.create({
  title: 'Sleeping Bag'
});

product.get('title') //=> 'Sleeping Bag'
product.set('title', 'Matches')
product.get('title') //=> 'Matches'
```

Creating the Remaining Models

For now, all three models will be identical, empty Ember.Object Models.

This will change later in the course.

app/models/product.js

```
import Ember from 'ember';

export default Ember.Object.extend({
});
```

app/models/line-item.js

```
import Ember from 'ember';

export default Ember.Object.extend({
});
```

app/models/order.js

```
import Ember from 'ember';

export default Ember.Object.extend({
});
```

Using the Ember.Object Models

With basic product, order, and LineItem models defined, let's use them in the store service.

app/services/store.js

```
import Ember from 'ember';

export default Ember.Service.extend({
  getOrderById(id) { /* … */ },
  getOrders() { /* … */ }
});
```

Importing the Models With Relative Paths

The new models must be imported into the store to make them available for use.

app/services/store.js

```
import Ember from 'ember';
import LineItem from '../models/line-item';
import Order from '../models/order';
import Product from '../models/product';

export default Ember.Service.extend({
  getOrderById(id) { /* … */ },
  getOrders() { /* … */ }
});
```

```
app/
├── models/
│   ├── line-item.js
│   ├── order.js
│   └── product.js
└── services/
    └── store.js
```

import can be used with relative file path references.
These change when the importing file moves, however.

Importing the Models With Project Paths

The import statement may instead be used with app name-based paths.

app/services/store.js

```
import Ember from 'ember';
import LineItem from 'woodland/models/line-item';
import Order from 'woodland/models/order';
import Product from 'woodland/models/product';

export default Ember.Service.extend({
  getOrderById(id) { /* … */ },
  getOrders() { /* … */ }
});
```

```
app/
├── models/
│   ├── line-item.js
│   ├── order.js
│   └── product.js
└── services/
    └── store.js
```

```
$ ember new woodland
```

"woodland" was the app name we defined with ember new, back in Level 1.

Defining the Product Records

With the product model available, let's create the available product records for the app.

Product
1. title
2. price
3. description
4. imageUrl

app/services/store.js

```
import Ember from 'ember';
import LineItem from 'woodland/models/line-item';
import Order from 'woodland/models/order';
import Product from 'woodland/models/product';

export default Ember.Service.extend({
  getOrderById(id) { /* … */ },
  getOrders() { /* … */ }
});
```

Defining the Product Records

Product.create() is used to create all four records and hold them in an array.

Product
1. title
2. price
3. description
4. imageurl

getProducts() returns the product record array to the caller.

app/services/store.js

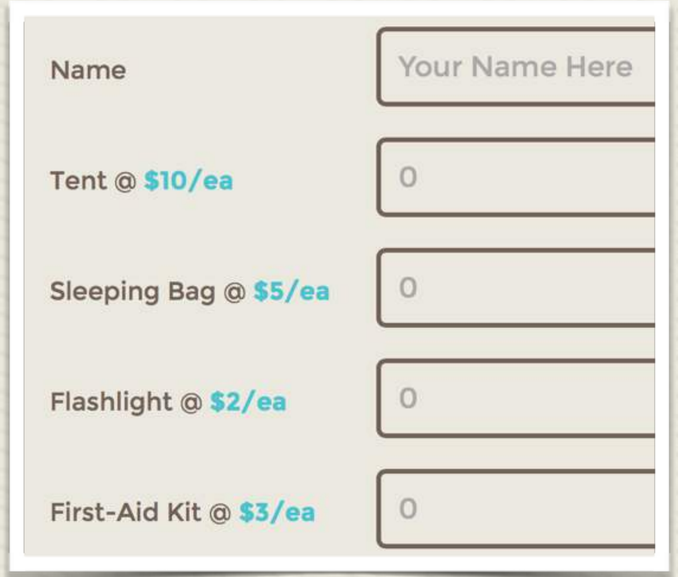
```
import Product from 'woodland/models/product';

const products = [
  Product.create({title: 'Tent', price: 10, descript…}),
  Product.create({title: 'Sleeping…', price: 5, desc…}),
  Product.create({title: 'Flashlig…', price: 2, desc…}),
  Product.create({title: 'First-Ai…', price: 3, desc…})
];

export default Ember.Service.extend({
  getOrderById(id) { /* … */ },
  getOrders() { /* … */ },
  getProducts() { return products; }
});
```

Defining the Order Records

Order.create() is used to create order records for the listings in an array.



Order
1. name
2. line items

Line Item
1. product
2. quantity

app/services/store.js

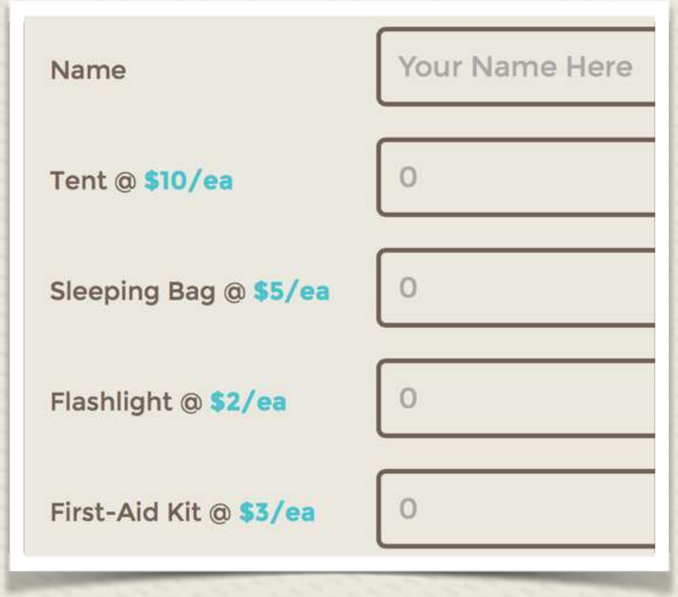
Some content is hidden for brevity.

```
import LineItem from 'woodland/models/line-item';
import Order from 'woodland/models/order';

const orders = [
 Order.create({ id: '1234', name: 'Blaise Blobfish',
  items: [
   LineItem.create({product: products[0], quantity: 1}),
   LineItem.create({product: products[1], quantity: 1}),
   LineItem.create({product: products[2], quantity: 0}),
   LineItem.create({product: products[3], quantity: 0})
  ]
 }),
 …
];
```

Defining the Order Records

The store service is updated to source its order data from the orders array.



Order
1. name
2. line items

Line Item
1. product
2. quantity

app/services/store.js

Some content is hidden for brevity.

```
import LineItem from 'woodland/models/line-item';
import Order from 'woodland/models/order';
import Product from 'woodland/models/product';

const products = […];
const orders = […];

export default Ember.Service.extend({
  getOrderById(id) { return orders.findBy('id', id); },
  getOrders() { return orders; },
  getProducts() { return products; }
});
```

Adding the Design Assets

The static HTML structure goes into the appropriate templates.

app/templates/index.hbs

```
<div class="card">
  <h1>Order Today!</h1>
  <p class="card-content">Our online store helps…</p>
  {{#link-to "orders"}}Order Today!{{/link-to}}
</div>

<div class="grid group">
  {{#each model as |product|}}
    <div class="product-media">
      <img src="{{product.imageUrl}}" />
    </div>
    <div class="product-content">
      <h2>{{product.title}}: <b>${{product.price}}</b>
```

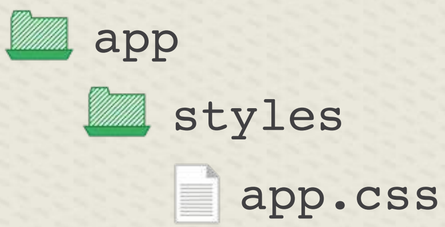
Adding the Design Assets

The design's images go into a new app/public/assets/images directory that we create.

```
app
    public
        assets
            images
                bag.svg
                light.svg
                kit.svg
                tent.svg
    styles
        app.css
```

Adding the Design Assets

The design's CSS go into the app/styles/app.css that was generated earlier by Ember CLI.



app/styles/app.css

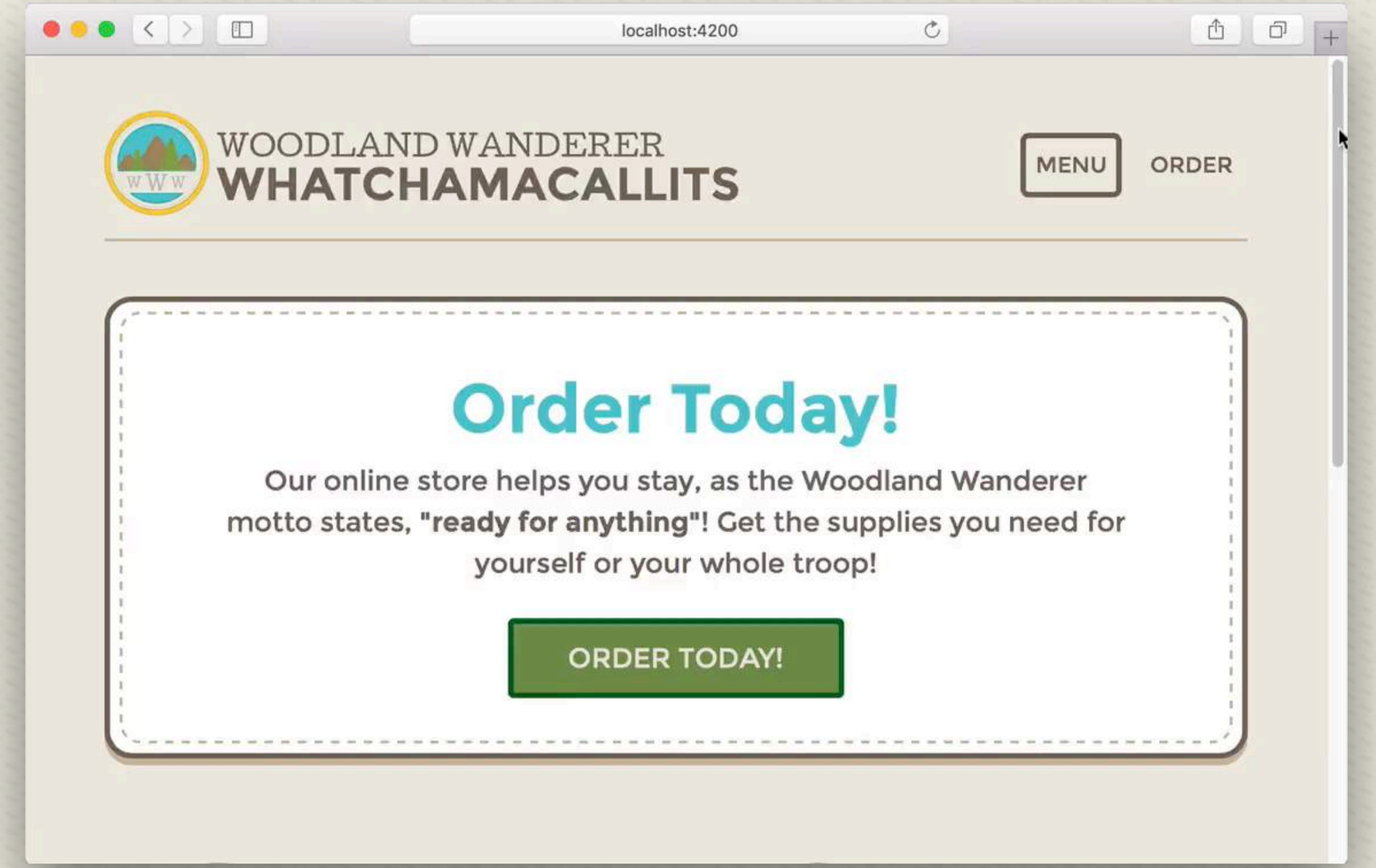
```
html {
  background-color: #ebe9df;
  color: #726157;
  font-family: 'Montserrat', sans-serif;
  font-size: 16px;
  line-height: 1.5;
}

body {
  min-height: 100%;
}

/* … */
```

The Ember CLI-generated index.html already includes app.css.

Displaying the design



Level 3

Models and Services