



TRP.

407

try ember

Level 4

Actions



Missing the Interaction

The app navigates and displays products and orders, but orders cannot yet be created.

The screenshot shows a web browser window at `localhost:4200/orders`. The page header includes a logo for 'WOODLAND WANDERER WHATCHAMACALLITS' and navigation links for 'MENU' and 'ORDER'. The main form contains a 'Name' field with the placeholder 'Your Name Here', followed by four rows of product entries. Each entry consists of a product name 'XX @ \$XX/ea', a quantity input field with the value '0', and a price '+10'. At the bottom of the form is a green 'PLACE ORDER' button. Handwritten annotations in teal include circles around the 'Your Name Here' field, the first product entry, and the 'PLACE ORDER' button. Yellow arrows point from the text 'We need something from which to populate the order form.' to the first product entry and the 'Name' field. Another teal circle is around the first product name 'XX @ \$XX/ea'. A red banner in the bottom right corner features a 'try ember' logo.

WOODLAND WANDERER
WHATCHAMACALLITS

MENU ORDER

Name Your Name Here

XX @ \$XX/ea 0 +10

XX @ \$XX/ea 0 +10

XX @ \$XX/ea 0 +10

XX @ \$XX/ea 0 +10

PLACE ORDER

We need something from which to populate the order form.

We'll start by initializing a new, empty order record.

This doesn't yet work.



Creating a New Order Record

To manage a new order's information, we need an empty order record to work with.

app/services/store.js

```
export default Ember.Service.extend({
  getOrderByid(id) { /* ... */ },
  getOrders() { /* ... */ },
  getProducts() { /* ... */ },

  newOrder() {
    return Order.create({
      items: products.map((product) => {
        return LineItem.create({
          product: product
        });
      })
    });
  }
});
```

*newOrder() returns a new
Order record with one
LineItem record per Product.*

*The "arrow function expression"
is an ES2015 feature. It's used
here as a shorthand for
function(product) { ... }.*



Using the New Order Record

The orders.index route is created and uses the new order record.

app/routes/orders/index.js

```
import Ember from 'ember';

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

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

The order form is on the orders/index template, so the orders.index route will use the new order record for its model.



Starting With the "Static" Form

The designed, static form is added with the addition of the `{{#each}}` expression and product info.

app/templates/orders/index.hbs

```
<form>
  <label for="name">Name</label>
  <input type="text" id="name">

  {{#each model.items as |item|}}
    <label>
      {{item.product.title}} @ ...
      <input type="number" min="0">
    </label>
  {{/each}}

  <input type="submit" type="Order">
</form>
```

This needs to be bound to the order name.

This needs to be bound to the `LineItem` quantity.

The browser will try to submit the form to the server, bypassing Ember.



Binding Properties With Input

Ember provides `{{input}}` helpers, which keep bound properties and input fields in sync.

app/templates/orders/index.hbs


```
<form>
  <label for="name">Name</label>
  {{input type="text" id="name" value=model.name}}

  {{#each model.items as |item|}}
    <label>
      {{item.product.title}} @ ...
      {{input type="number" min="0" value=item.quantity}}
    </label>
  {{/each}}

  <input type="submit" type="Order">
</form>
```

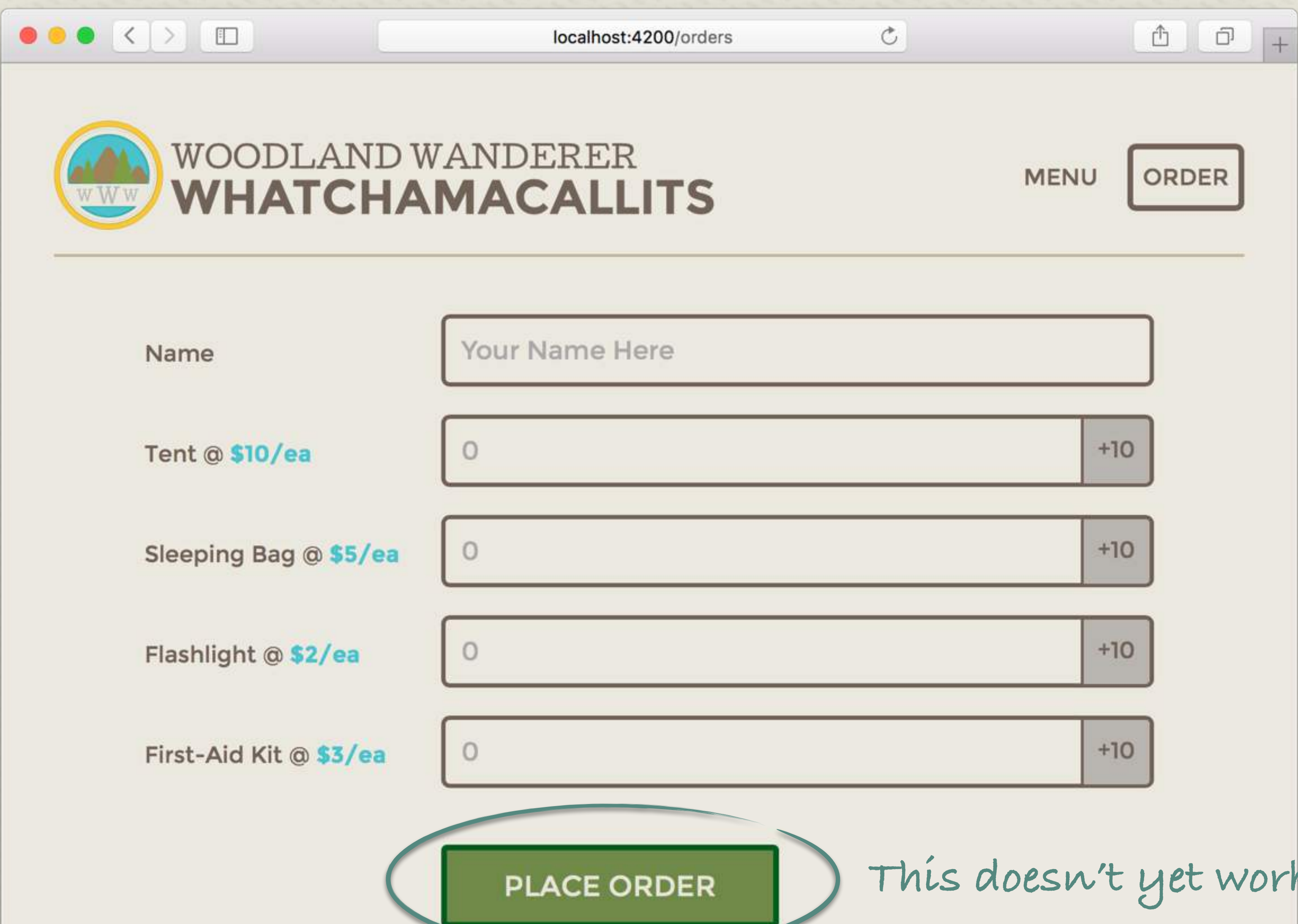
The `{{input}}` helper accepts the same properties as an input element.

No quotes around the property gives the helper direct access to the property for manipulation.



Handling the Submission Event

Somehow we need to intercept and handle the form submission event to create the order.



The screenshot shows a web browser window with the address bar displaying 'localhost:4200/orders'. The page features a logo for 'WOODLAND WANDERER WHATCHAMACALLITS' and a navigation bar with 'MENU' and 'ORDER' links. Below the navigation bar, there is a form with the following fields:

- Name: A text input field with the placeholder text 'Your Name Here'.
- Tent @ \$10/ea: A quantity input field with a value of 0 and a '+10' button.
- Sleeping Bag @ \$5/ea: A quantity input field with a value of 0 and a '+10' button.
- Flashlight @ \$2/ea: A quantity input field with a value of 0 and a '+10' button.
- First-Aid Kit @ \$3/ea: A quantity input field with a value of 0 and a '+10' button.

At the bottom of the form, there is a green button labeled 'PLACE ORDER'.

How do we
intercept the
submission?

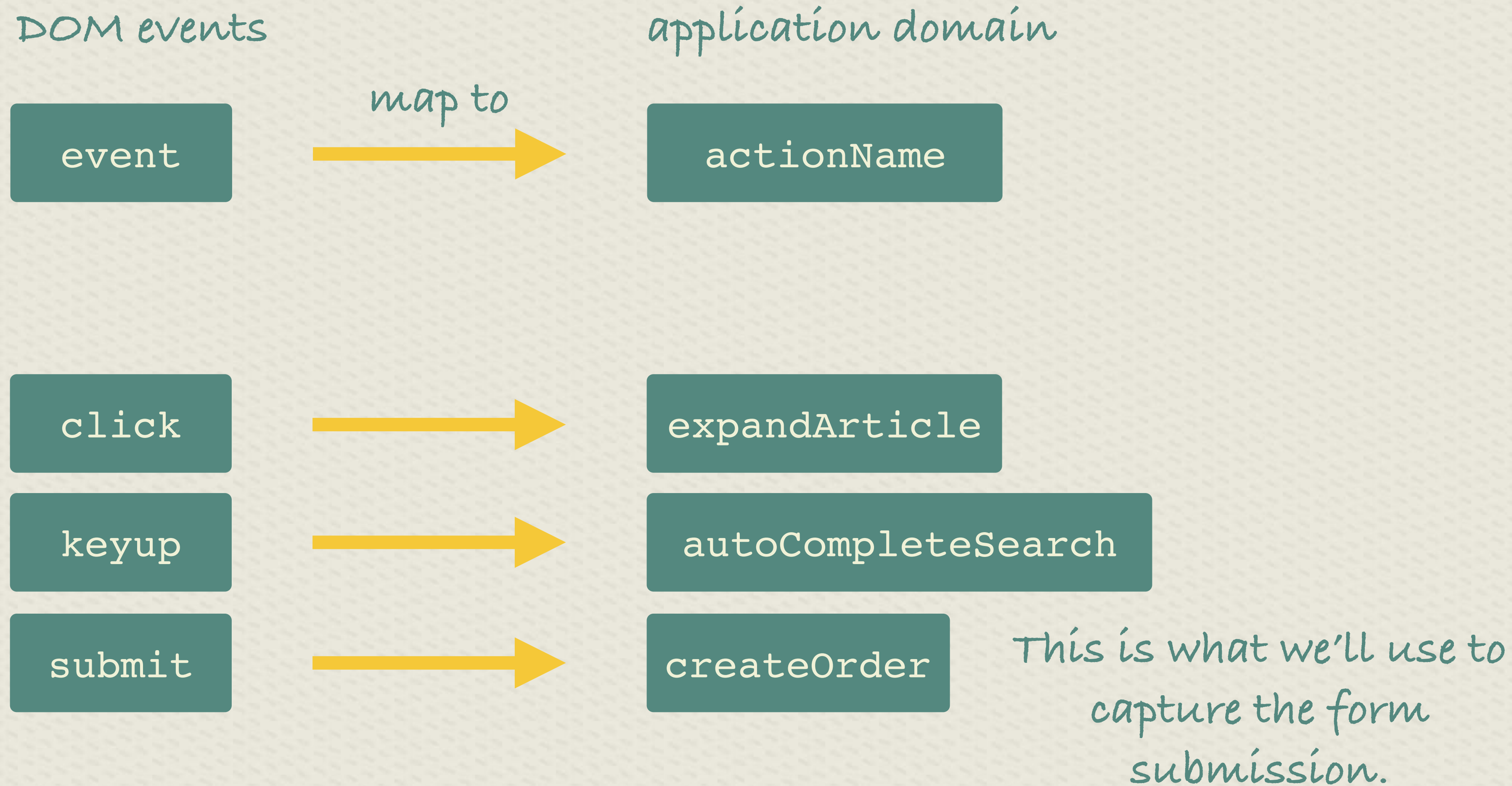
This doesn't yet work.



Introducing Actions



Actions map generic DOM events to specific application activities and functions.



Intercepting the Submit Event

How do we intercept the FORM's submit event using an Ember action?

app/templates/orders/index.hbs

```
<form>
  <!-- ... -->
  <button type="submit">Order</button>
</form>
```

Forms can be submitted through a submit button, ENTER in an input, or a JavaScript `form.submit()` call.



Mapping an Action

Actions are mapped in templates using the `{{action}}` helper, defined on the element to watch.

```
{{action "actionName" on="eventName"}}
```

The event to trigger on defaults to "click".

app/templates/orders/index.hbs

```
<form>
  <!-- ... -->
  <button type="submit">Order</button>
</form>
```



Mapping an Action

Actions are mapped in templates using the `{{action}}` helper, defined on the element to watch.

```
{{action "actionName" on="eventName"}}
```

app/templates/orders/index.hbs

```
<form {{action "createOrder" model on="submit"}}  
  <!-- ... -->  
  <button type="submit">Order</button>  
</form>
```

The action triggers
when the form emits a
"submit" event.

This fires the
"createOrder" action.

Any extra parameters are passed to
the triggered action as parameters.

By default, actions prevent the browser default activity (`preventDefault`).



Handling Actions

Action handlers are functions defined in an “actions” block on the route or its parents.

```
{{action "createOrder" model on="submit"}}
```

app/routes/orders/index.js

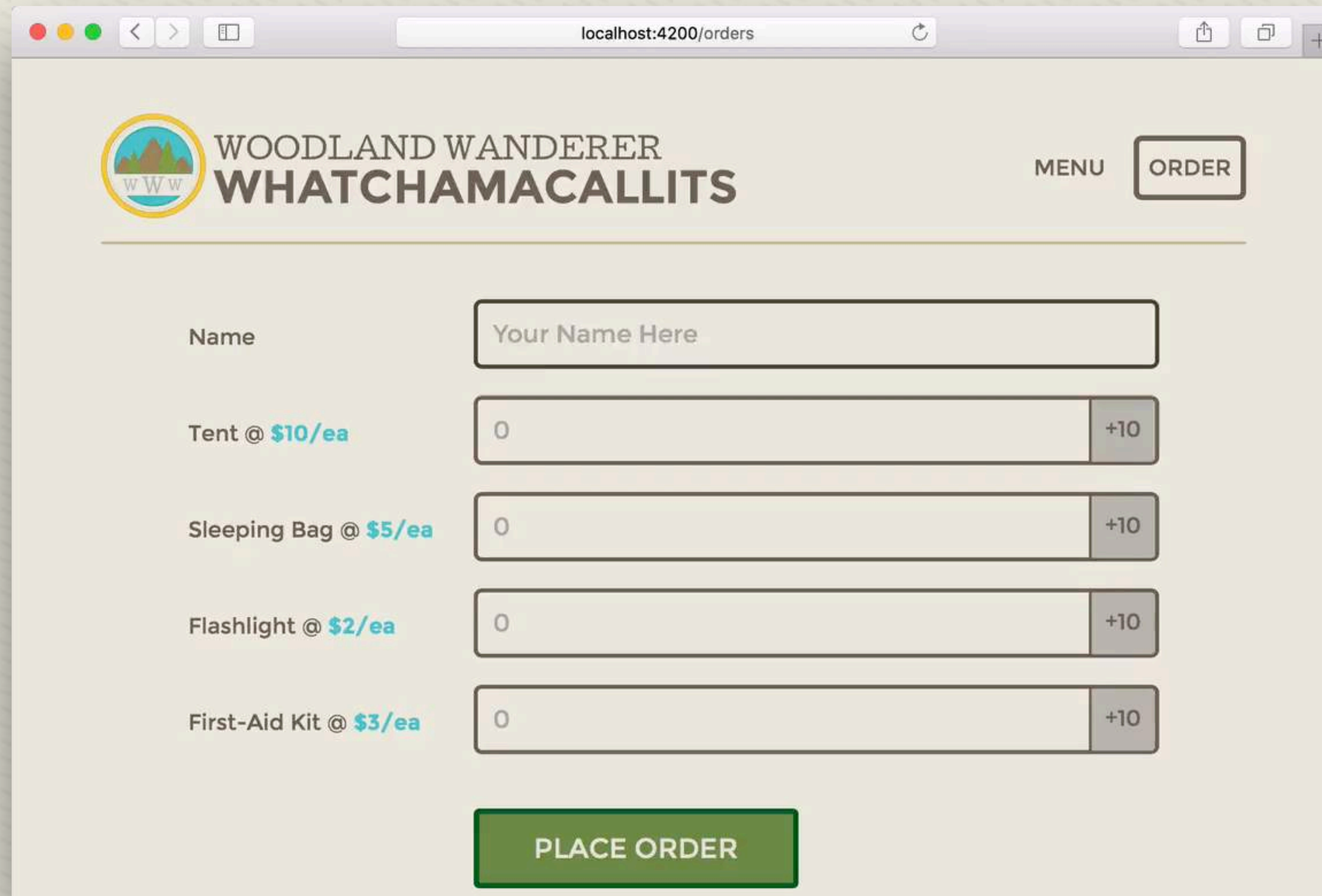
```
export default Ember.Route.extend({
  actions: {
    createOrder(order) {
      const name = order.get('name');
      alert(name + ' order saved!');
    }
  },
  model() { return this.get('store').newOrder(); }
});
```

1. The new order record populates the template as the model.
2. The triggered action passes the new order record to the action.



Alerting on the Action

The form submission is intercepted by the action and handled by the route's action handler.



The screenshot shows a web browser window with the address bar displaying 'localhost:4200/orders'. The page features a logo for 'WOODLAND WANDERER WHATCHAMACALLITS' and navigation links for 'MENU' and 'ORDER'. The form includes a 'Name' field with the placeholder 'Your Name Here', and four items for selection: 'Tent @ \$10/ea', 'Sleeping Bag @ \$5/ea', 'Flashlight @ \$2/ea', and 'First-Aid Kit @ \$3/ea'. Each item has a quantity input field set to '0' and a '+10' button. A green 'PLACE ORDER' button is at the bottom.

Item	Price	Quantity
Tent	\$10/ea	0
Sleeping Bag	\$5/ea	0
Flashlight	\$2/ea	0
First-Aid Kit	\$3/ea	0

But we need to show the order receipt, not an alert!

Transitioning to a New Route

Routes have a `transitionTo` function used to navigate to other routes.

app/routes/orders/index.js

```
export default Ember.Route.extend({
  actions: {
    createOrder(order) {
      this.get('store').saveOrder(order);
      this.transitionTo('orders.order', order);
    }
  },
  /* ... */
});
```

Target Route name.

Optional model parameters.

This is similar to the Template's `link-to` helper.



“Saving” the Order in the Store

Here, new orders are “saved” by giving them an ID and adding them to the orders collection.

app/services/store.js

```
export default Ember.Service.extend({
  getOrderByid(id) { /* ... */ },
  getOrders() { /* ... */ },
  getProducts() { /* ... */ },
  newOrder() { /* ... */ },

  saveOrder(order) {
    order.set('id', 9999);
    orders.pushObject(order);
  }
});
```

Setting an ID to pretend
that the order was saved.

pushObject comes from Ember. It's like
push, but triggers value-changed events.



Transitioning to the Receipt

The order is submitted and saved, and a receipt is now correctly displayed.

WOODLAND WANDERER
WHATCHAMACALLITS

MENU ORDER

Name

Tent @ \$10/ea +10

Sleeping Bag @ \$5/ea +10

Flashlight @ \$2/ea +10

First-Aid Kit @ \$3/ea +10

PLACE ORDER

The pushObject added the new order to the orders list!

Ordering in Bulk

To finish the order form, the design calls for a button to increment item quantities by 10.

app/templates/orders/index.hbs

```
<!-- ... -->
{{#each model.items as |item|}}
  <label>
    {{item.product.title}} @
    ${{item.product.price}}/ea
    {{input type="number" min="0" value=item.quantity}}
    <button {{action "addToItemQuantity" item 10}}>+10</button>
  </label>
{{/each}}
<!-- ... -->
```

This `{{action}}` is using the default `on="click"` trigger.

Passing two arguments to the action to make this action more reusable: the item and the amount to increment.



Incrementing Property Values

Ember.Object provides `incrementProperty` and `decrementProperty` to quickly change numerics.

app/routes/orders/index.js

```
export default Ember.Route.extend({
  actions: {
    addToItemQuantity(lineItem, amount) {
      lineItem.incrementProperty('quantity', amount);
    },

    createOrder(order) { /* ... */ }
  },

  model() { /* ... */ }
});
```

lineItem and amount values came from the {{action}} arguments.

incrementProperty increases the property value by the amount given, or 1 by default.



Ordering Complete

All of the functionality for adding a new order is now in the system.

WOODLAND WANDERER
WHATCHAMACALLITS

MENU ORDER

Name

Tent @ \$10/ea +10

Sleeping Bag @ \$5/ea +10

Flashlight @ \$2/ea +10

First-Aid Kit @ \$3/ea +10

PLACE ORDER

Level 4

Actions

