DEV PREVIEW

Step 3: Using data binding

Your first Polymer application

Edit on GitHub

▶ **Table of contents**

Step 3: Using data binding

One post is nice, but the app looks a little empty. In this step, you'll pull data from a web service and use Polymer's data binding to render it as a series of cards.

To get the data, you'll use the `<post-service>` element provided as part of the starter app. This element provides a very simple API for an imaginary social network. In this section, you'll use the `posts` property, which returns an array of `post` objects like this:

```
{
  "uid": 2,
  "text" : "Loving this Polymer thing.",
  "username" : "Rob",
  "avatar" : "../images/avatar-02.svg",
  "favorite": false
}
```

In this section you'll learn about:

- Using data binding.
- Publishing properties.

Edit post-list.html

Open the `post-list.html` file in your editor.

```
<link rel="import" href="../components/polymer/polymer.html">
<link rel="import" href="../post-service/post-service.html">
<link rel="import" href="post-card.html">

<polymer-element name="post-list" attributes="show">
  <template>
    <style>
    :host {
      display: block;
```

```
      width: 100%;
    }
    post-card {
      margin-bottom: 30px;
    }
    </style>

    <!-- add markup here -->
...
```

Key information

- The file already includes an import for the `<post-service>` element, so it's ready to use.
- The `attributes="show"` attribute creates a *published property* named `show`.

A *published property* is a property that can be configured in markup using an attribute, or connected to another property using two-way data binding. You'll use the `show` property in a later step.

Add a `<post-service>` element inside the element's `<template>`:

```
...
<post-service id="service" posts="{{posts}}">
</post-service>
...
```

Key information

- The `posts="{{posts}}"` attribute adds a two-way data binding between the `<post-service>` element and the `<post-list>` element.

The *data binding* links the service element's `posts` property to a local property (also called `posts` here). Any methods you define on your custom element can access the response as `this.posts`.

Render a dynamic list of cards.

Add the following `<div>` and `<template>` tag:

```
...
<post-service id="service" posts="{{posts}}">
</post-service>

<div layout vertical center>
```

```
  <template repeat="{{post in posts}}">
    <post-card>
      <img src="{{post.avatar}}" width="70" height="70">
      <h2>{{post.username}}</h2>
      <p>{{post.text}}</p>
    </post-card>
  </template>

</div>
...
```

Key information

- This new syntax `repeat="{{post in posts}}"`, tells the template to create a new instance for each item in the `posts` array.
- In each template instance, the individual bindings (such as `{{post.avatar}}`) are replaced by the corresponding values for that item.

Edit index.html

Import the `<post-list>` element into `index.html`.

Open `index.html` and add an import link for `post-list.html`. You can replace the existing link for `post-card`:

```
...
<link rel="import" href="../components/paper-tabs/paper-tabs.html"
<link rel="import" href="post-list.html">
...
```

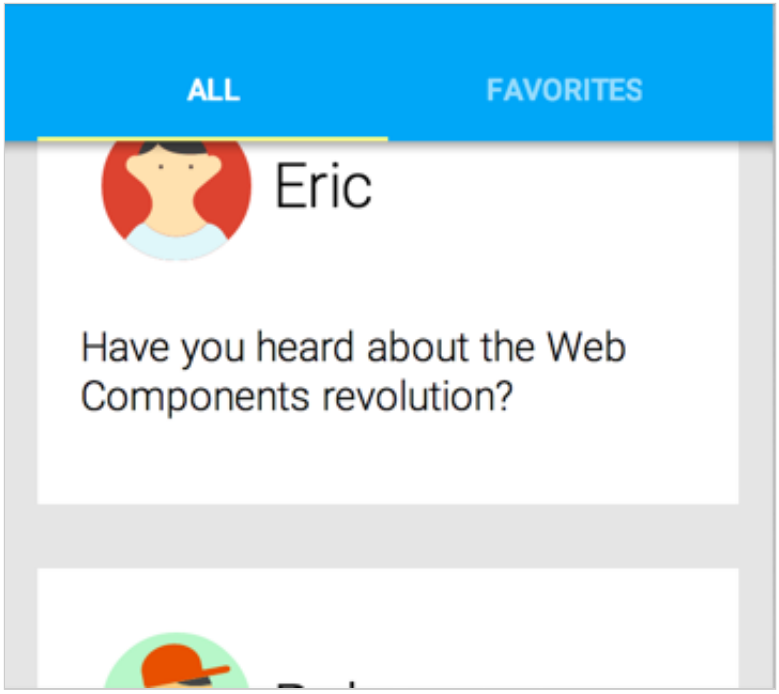
Use the `<post-list>` element.

Find the `<post-card>` element you added in the last step and replace it with a `<post-list>`:

```
...
<div class="container" layout vertical center>
  <post-list show="all"></post-list>
</div>
...
```

Test your work

Save the `index.html` file and reload the page in your browser. You should see a list of cards, something like this:



If you have any problems, check your work against the files in the `step-3` folder:

- `post-list.html`
- `index.html`

Explore: Open up `post-service.html` to see how the component works. Internally, it uses the `<core-ajax>` element to make HTTP requests.

← STEP 2: CREATING YOUR OWN ELEMENT → STEP 4: FINISHING TOUCHES

+POLYMER @POLYMER

/POLYMER FILE A BUG