Predix

UI Basics

Student Lab Guide

January 2015

GE Digital

Predix

Getting Started

This guide provides step-by-step instructions for lab exercises. Each lab corresponds to a topic covered in class and provides students with hands-on experience developing basic UI elements on the Predix platform.

Course Prerequisites:

- Install the most recent DevBox version
  - Download Virtual Box 4.3.12
  - Download and import DevBox into your Virtual Box

Start the VM in Oracle VirtualBox

- Start DevBox

> ***Tip:*** Make sure you are on the GE network (BlueSSO)

- Login with:
  Username: **predix**
  Password: **predix**

> ***Note:*** All lab exercises will be completed in your DevBox.

Set Your Environment

The UI Basics lab files are in the `/predix/PredixApps/training/UI` directory on your DevBox.

Lab 1: Using Angular JS

Learning Objectives

By the end of the lab, students will be able to:

- Add a link and route for the Patients page to the Predix Starter Pack
- Show data in a table

Lab Exercises

- *Adding a Route using Angular JS*, page 2
- *Creating a Controller*, page 7
- *Changing the View and Model*, page 9

Directions

Complete the exercises that follow.

The labs are designed to support novice and advanced users simultaneously.

Exercise 1: Adding a Route using Angular JS

Overview

In this exercise you will start up the predix seed web application and add a new page to the application, using the Predix Angular UI-router. (This is not the built-in router Angular JS ships with.) You will add a new Angular controller and use it to add data to the page.

Steps

1. Log into Cloud Foundry.

- Open the Terminal and log in
  - Double-click the Terminal icon on your desktop
  - Enter **`cf login -a https://api.system.aws-usw02-pr.ice.predix.io`**
  - Enter your email address (Your instructor will provide this)
  - Enter the password your instructor provides and press **Enter**

2. Unzip and install the Predix seed starter application.

- In the Terminal, change to the Downloads directory
  - Enter **`cd ~/Downloads`** and press **Enter**
- Unzip the predix-seed.zip file
  - Enter the following command:

```
unzip predix-seed.zip -d ~/PredixApps/training/UI/predix-seed-1.1.3
```

Note: The Predix seed application is now installed, along with its required npm and bower installations. Normally, you would run npm install and bower installs after saving the seed application files to your directory.

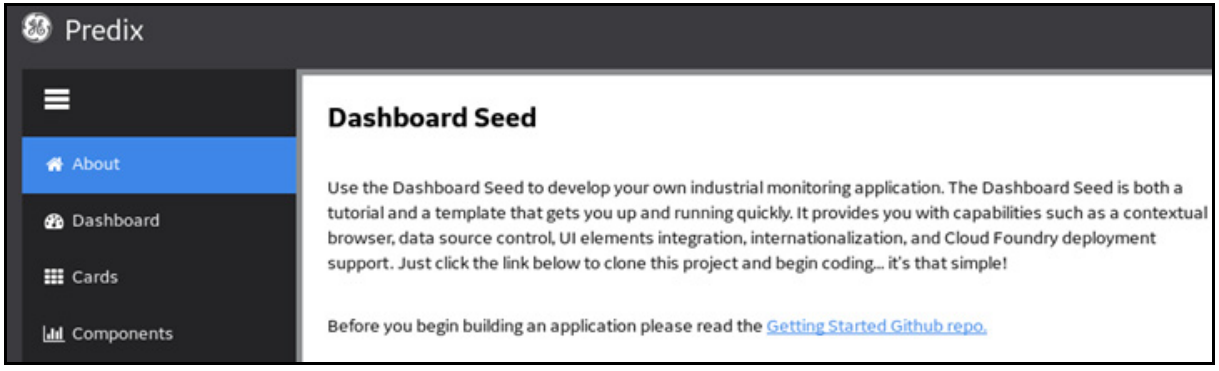
3. Test the application locally.

- In the Terminal, use the following command to navigate to the predix seed app:
  - **`cd ~/PredixApps/training/UI/predix-seed-1.1.3`**
- Run the following command to start the local web server
  - **`grunt serve`**

    The command line interface (CLI) responds with several lines, ending as follows.

```
Running "clean:build" (clean) task

Running "connect:livereload" (connect) task
Started connect web server on http://localhost:9000

Running "watch" task
Waiting...
```

The web browser opens and loads the predix starter web application



Note: Press <Ctrl> + <C> when you need to do something else in your Terminal. This stops the watch task of the grunt serve command.

4. Add a new navigation link (route).

Note: On the left side of the web page, there is an existing navigation component. You will add a new link called **Patients**.

The following code adds an Angular UI route which is used to associate a view and a controller with a URL. The `routes.js` file contains all of the routes for the application and you will add a new state in this file.

- In your text editor, navigate to this directory:
  `predix/PredixApps/training/UI/predix-seed-1.1.3`

Tip: The rest of the paths noted in the lab instructions assume you are starting in the `predix-seed-1.1.3` directory unless otherwise directed.

Note: For the lab file that provides code to copy and paste into your application, navigate to predix/PredixApps/training/UI. Open Lab1.txt in your text editor for the duration of the lab.

- In your text editor, navigate to the `public/scripts` directory
  - Open the `routes.js` file
  - Add a new state to the file by pasting in the code provided at the end of the $StateProvider section just after `/components.html`, replacing braces and parenthesis before and after the patient state

  Ensure that your code reads exactly as below:

```
})
.state('components', {
    url: '/components',
    templateUrl: 'views/components.html'
})
.state('patients', {
    url: '/patients',
    templateUrl: '/views/patient/index.html',
    controller: 'PatientsCtrl'
});
```

  - Save the file (**<Ctrl + S>**)

5. Create a new link (tab) in the navigation and some data for display in the view.

- Create a new link
  - In your text editor, navigate to the `public/scripts` directory
  - Open the `app.js` file
  - Add a comma after `label: 'Data Control'}`
  - Add the code provided to the tabs array

Your code should read as below:

```
    //Global application object
    window.App = $rootScope.App = {
        version: '1.0',
        name: 'Predix Seed',
        session: {},
        tabs: [
            {icon: 'fa-home', state: 'about', label: 'About'},
            {icon: 'fa-tachometer', state: 'dashboard', label: 'Dashboard'},
            {icon: 'fa-th', state: 'cards', label: 'Cards', subitems: [
                {state: 'interactions', label: 'Interactions'},
                {state: 'dataControl', label: 'Data Control'},
                {state: 'patients', label: 'Patients'}
            ]},
            {icon: 'fa-bar-chart', state: 'components', label: 'Components'}
        ]
    };

}]);
```

- In the same file, store some dummy data for display in the view
  - Locate this line in the file:

    `predixApp.controller('MainCtrl', ['$scope', '$rootScope',function,...`
  - After this line, paste the code provided into the root scope

    Your code should read as follows:

```
predixApp.controller('MainCtrl', ['$scope', '$rootScope', function ($scope, $rootScope) {
    // we'll store patients in rootScope for now, so we can use them on multiple views lat
    $rootScope.patients = [
       { id: 1, firstName: 'Bob', lastName: 'Dylan' },
       { id: 2, firstName: 'Joe', lastName: 'Sammy' }
    ];
    //Global application object
```

  - Save the file

Exercise 2: Creating a Controller

Overview

In this exercise, you add the Patients controller to the application.

Steps

1. Create the new patients.js file for the controller.

- In your text editor, navigate to the `public/scripts/controllers` directory in the seed application
  - Create a new file called `patients.js`
  - Copy the content provided and paste it into the `patients.js` file

    Your code should read as follows:

```
'use strict';
define(['angular', 'sample-module'], function(angular, controllers) {
    // Controller definition
    controllers.controller('PatientsCtrl', ['$rootScope', '$scope', function($rootScope, $scope) {
        $scope.form = {};
        $scope.addPatient = function () {
            var patient = {
                id: $scope.patients.length + 1,
                firstName: $scope.form.firstName,
                lastName: $scope.form.lastName
            };
            $rootScope.patients.push(patient);
            $scope.form = {};
        };
    }]);
});
```

  - Save the file

2. Add a reference to the new controller.

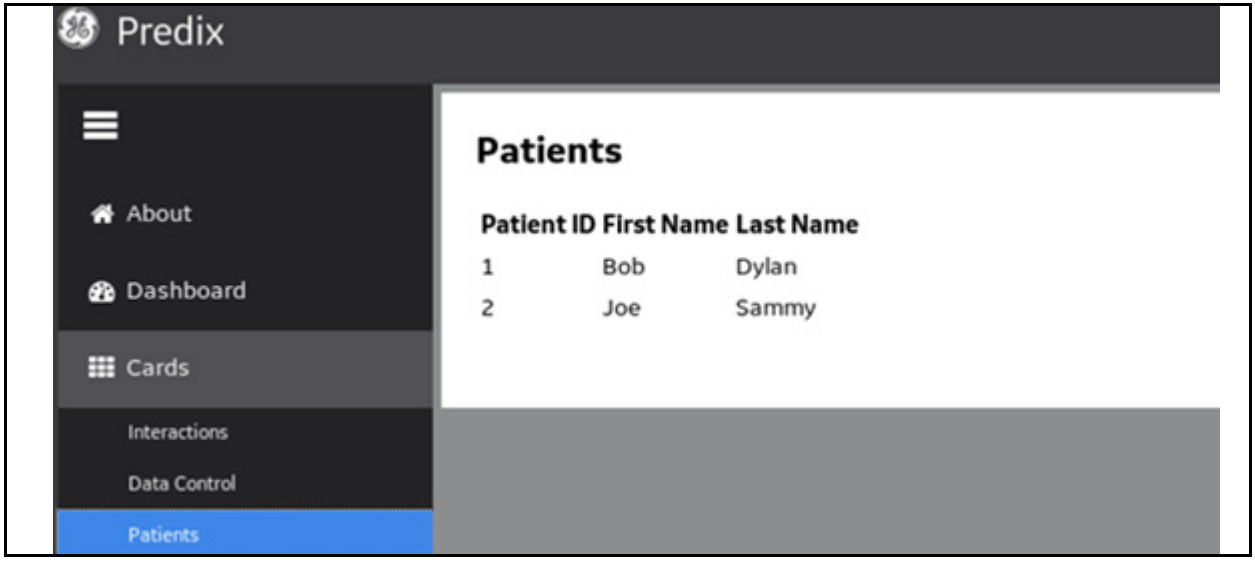
- In your text editor, navigate to the `/public/scripts/controllers` directory
  - Open the file `main.js` file
  - Paste the code provided into the file, replacing all code

  This reference ensures that the new controller is loaded into the browser.
  - Save the file

3. Create a new view to display a table of patient data.

- In your text editor, navigate to the `public/views` directory
  - Create a directory and name it `patient`
  - Change to the `patient` directory and create a file named `index.html`
  - Paste the code provided into the file
  - Save the file

    This view uses the Angular `ng-repeat` directive to iterate over the list of patients in scope.
  - Refresh the browser and the application appears with the new navigation link.



Exercise 3: Changing the View and Model

Overview

In this exercise, you will add a form that allows the user to dynamically change the view and the model (data) on the web page. You will add **First Name** and **Last Name** input fields and an **Add Patient** button to allow users to add patient data to the table. Finally, you will add a search field to provide filtering.

Steps

1. Add **Name** entry fields and an **Add Patient** button.

- In your text editor, navigate to the `views/patient` directory
  - Open the `index.html` file
  - Enter the code provided after the <h2> tag

    Your code should read as follows:

```
<px-card>
    <article role="article">
        <h2 class="u-mt0 u-pt+">Patients</h2>
        <div class="flex">
    <form ng-submit="addPatient()">
      <label for="first-name">First Name: </label>
      <input type="text" id="first-name" placeholder="i.e. Joe" ng-model="form.firstName">
      <label for="last-name">Last Name: </label>
      <input type="text" id="last-name" placeholder="i.e. Smith" ng-model="form.lastName">
      <button type="submit">Add Patient</button>
    </form>
</div>
<br/>
        <div class="flex">
```

  - Save the file
  - Refresh your browser

Your web page should look similar to the one below:

- Add some patient First and Last Names to test your page
  The **Add Patient** button should submit the names to your list.

2. Add a search field to filter the patient data on the page.

- In your text editor, navigate to the `public/views/patient` directory
  - Open the `index.html` file
  - Add the code provided after the
 tag (step 2.1)

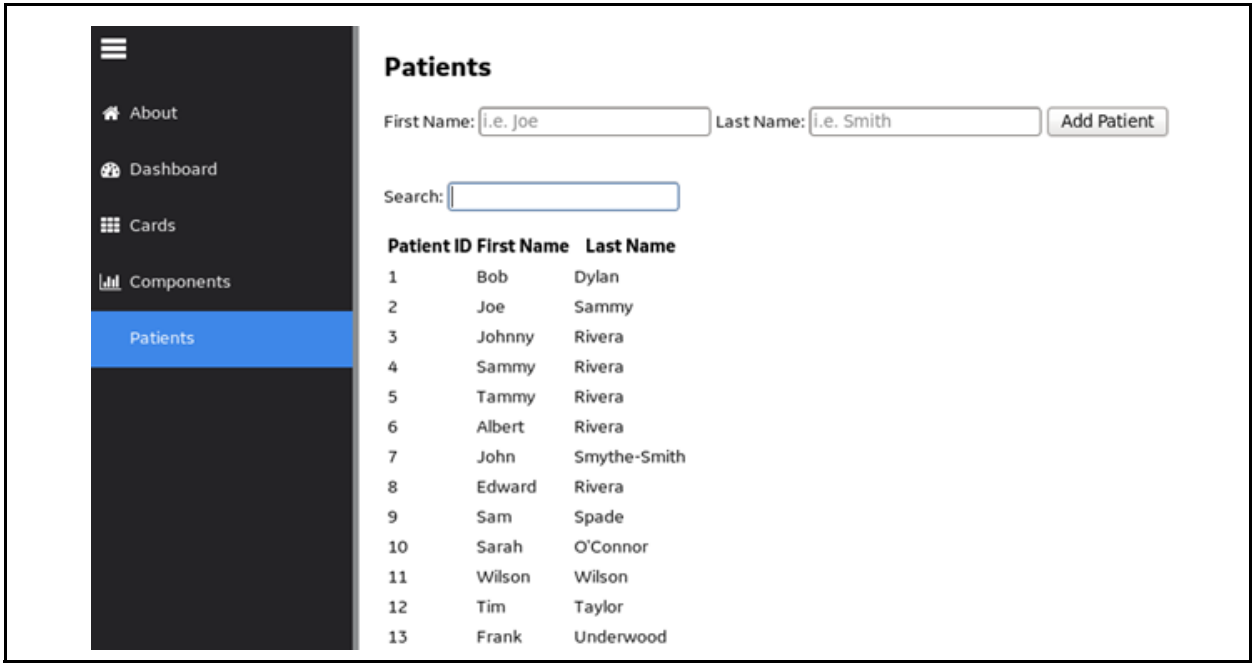
```
</div>
<br/>
<p>Search: <input ng-model="filterText"></p>
        <div class="flex">
```

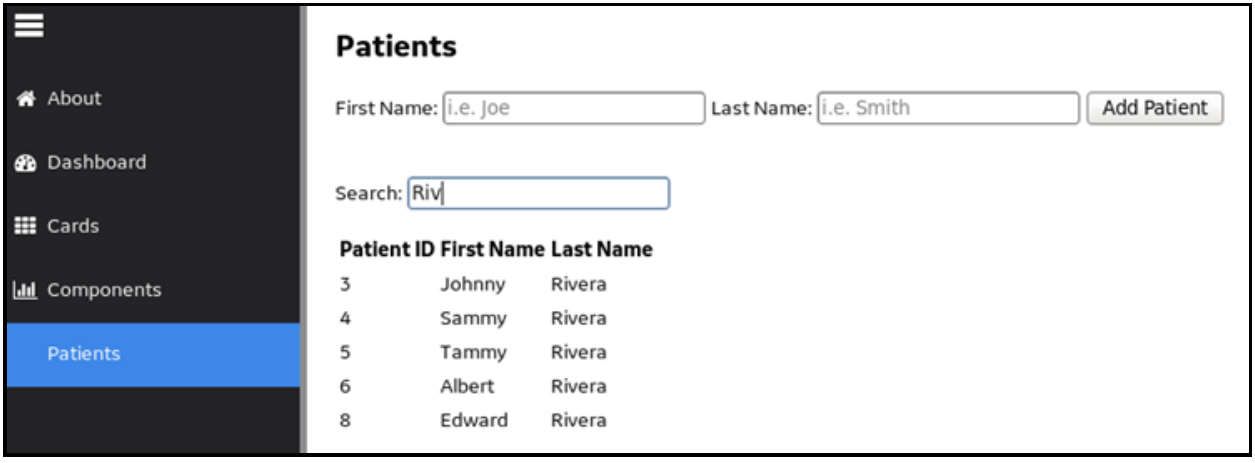
  - Find the following line in the same file:

```
<tr ng-repeat="patient in patients">
```

  - Replace the line with the code provided (step 2.2)
  - Save the file
  - Refresh the browser
  - Navigate to the **Patients** page and test the search/filter functionality

The application allows you to add multiple patient names and filter on them.





Lab 2: Styling with Polymer

Learning Objectives

By the end of the lab, students will be able to:

- Customize the px-theme component to style the application
- Customize a reusable Predix web component
- Use the customized component in your application

Lab Exercises

- *Styling Your Application*, page 13
- *Creating a View to Display a Web Component*, page 17
- *Creating a Web Component*, page 20

Directions

Complete the exercises that follow.

Note: For the lab file that provides code to copy and paste into your application, navigate to predix/PredixApps/training/UI. Open Lab2.txt in your text editor for the duration of the lab.

Exercise 1: Styling Your Application

Overview

In the last lab, you created a Patient Input Form without any styling. In this lab, you'll use Predix styles to give the form a nice look and feel.

Normally, you use the GitHub repository to copy Predix projects (web parts, components, elements) to your laptop. We have already staged the px-theme, px-library-design, px-forms-design, and the generator-px-comp projects on your DevBox. All of the projects' dependencies have also been installed using npm install and bower install. To see the actual steps for this, see the **Installing Px Theme Components from GitHub** section in the Appendix at the end of this guide.

Steps.

1. Generate the CSS files and add the px-forms-design component.

- From the `predix/PredixApps/training/UI/px-theme` directory in the Terminal, run the `grunt` command

  The Sass pre-processor generates the CSS files, including the px-forms-design component.

Tip: Grunt is a command line tool that runs tasks for JavaScript. Here it assures that the Sass files generate the appropriate CSS. In the next step, the `grunt watch` command ensures that CSS files are updated as changes are made to .scss files. This allows you to immediately see your code changes in the web application.

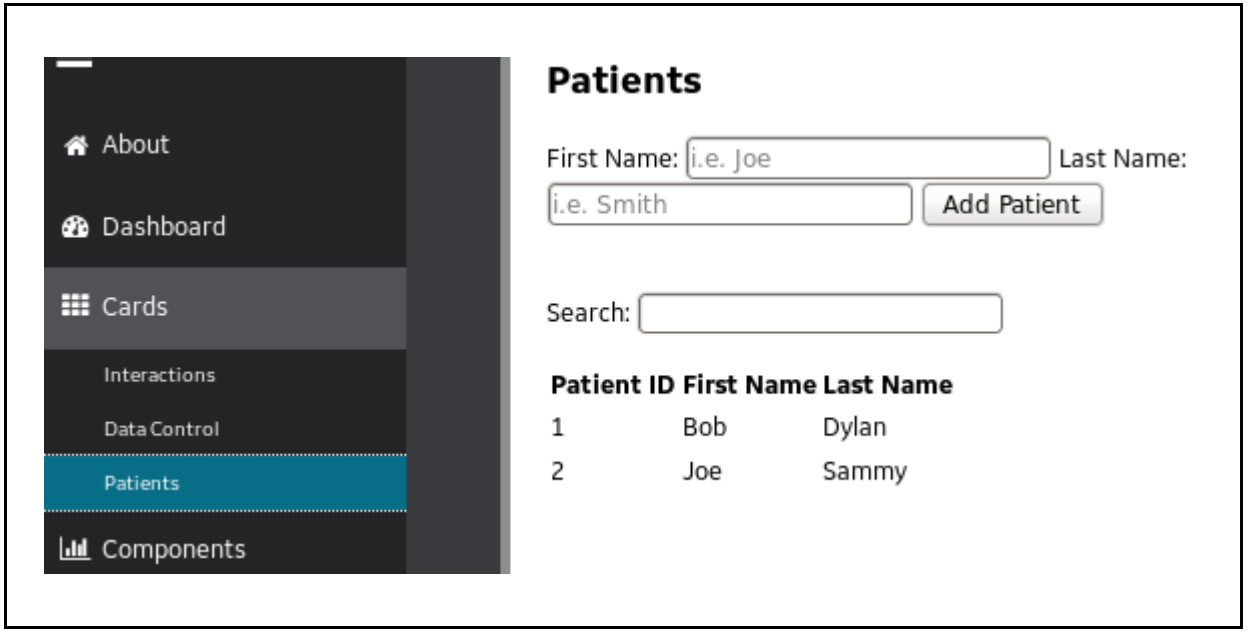
- Run the `grunt watch` command from the `UI/px-theme/sass` directory
- In your text editor, navigate to the `px-theme/sass` directory
- Open the `px-page-theme.scss` file
- Insert the code provided just above the `//App` line so your code appears as follows:

```
@import "px-theme.scss";
@import "px-forms-design/_base.forms.scss";
// App
html {
  position: relative;
```

- Run the following commands in the Terminal:
  - From the `/predix/PredixApps/training/UI/px-theme` directory, run the **`bower link`** command
  - From the `predix-seed-1.1.3` directory, run **`bower link px-theme`**
    These commands link your px-theme project to the predix-seed-1.1.3 project directly.
  - Reload the "Patients" page in your browser

Tip: You may need to run the `grunt serve` command in the `predix-seed-1.1.3` directory again.

Your Patients page appears as below:



2. Add classes to the HTML.

- In your text editor, navigate to the `predix-seed-1.1.3/public/views/patient` directory
  - Open the `index.html` file, and replace all of the HTML from and including the <form> tags with the code provided (Step 2.1)

  Your code should appear as follows:

```
        <div class="flex">
<form ng-submit="addPatient()">
    <ol class=list-bare>
      <li class=form-field>
        <label for="first-name">First Name: </label>
        <input type="text" id="first-name" placeholder="i.e. Joe" ng-model="f
      </li>
      <li class=form-field>
        <label for="last-name">Last Name: </label>
        <input type="text" id="last-name" placeholder="i.e. Smith" ng-model="
      </li>
    </ol>
    <input class="btn btn--primary" type="submit" value="Add Patient">
  </form>
</div>
<br/>
```

  If you reload the browser now, you will not see the changes because the `input--small` and `btn--primary` classes are not included by default. Px only includes classes you want to use.

- To include these classes, in the `UI/px-theme/sass` directory, open the `px-page-theme.scss` file.
  - Add the code provided in the //Objects section on the line after `"$inuit-enable-btn--bare" (Step 2.2)`
  - Add the code provided in the same section on the line before `@import "px-forms-design/_base.forms.scss";"` (Step 2.3) and save

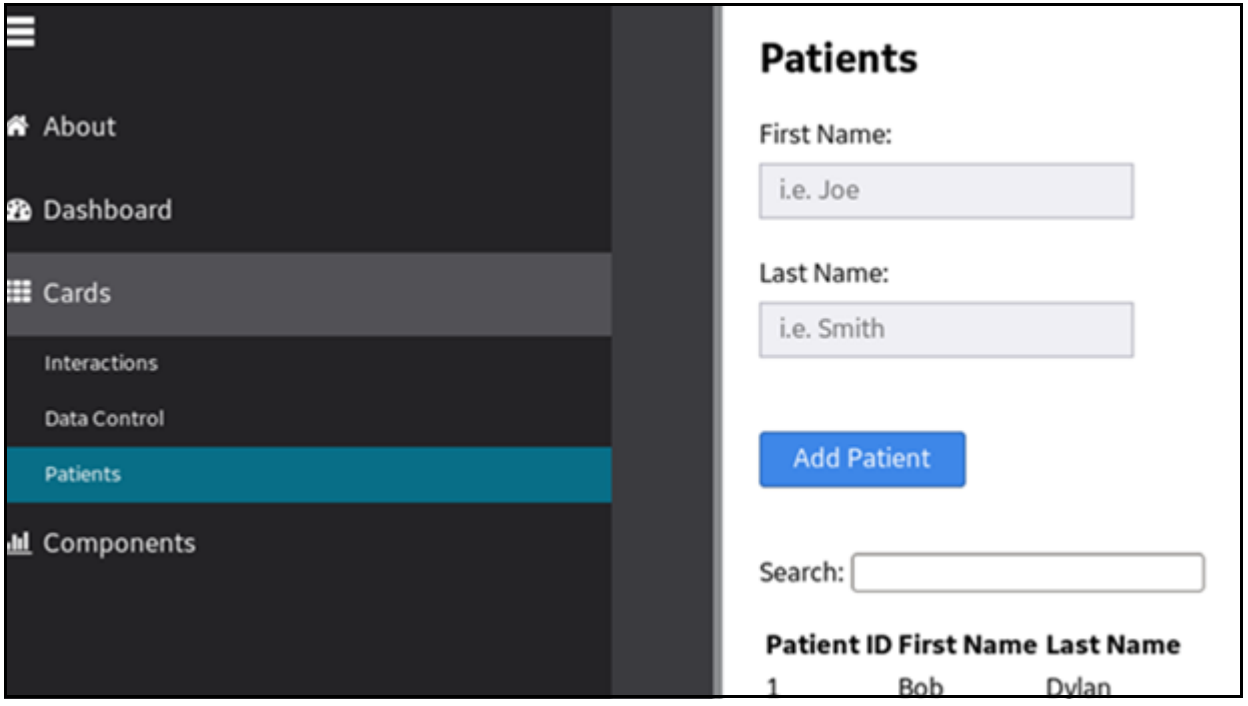
Note: Be sure to place the code exactly as instructed in the .scss file because line order is critical.

Your code should appear as follows:

```
// Objects
$inuit-enable-btn--bare      : true;
$inuit-enable-btn--primary : true;

@import "px-buttons-design/_objects.buttons.scss";

$inuit-enable-layout--small : true;
$inuit-enable-layout--flush : true;
$inuit-enable-layout--full  : true;

@import "px-layout-design/_objects.layout.scss";

@import "px-theme.scss";
$inuit-enable-input--small : true;
@import "px-forms-design/_base.forms.scss";
// App
```

- Reload the page in your browser

  Your Patients page should have a new blue button and nicely formatted, small text fields.



Exercise 2: Creating a View to Display a Web Component

Overview

In this exercise, you will create a view in order to display a hospital web component. The web component displays the number of hospitals within the network. In order to create the view, you create a new state for the Angular UI router to use. You will also create a new link to be able to navigate to the view. Finally, you will add a controller with some dummy data to display in the view.

Steps

1. Add a new state.

- In your text editor, navigate to the `predix-seed-1.1.3/public/scripts` directory
  - Open the `routes.js` file
  - Press **Enter** after `'PatientsCtrl'})`
  - Add the state information for this step from the code provided and align the text (using spaces) underneath the prior state information

    Your code should read as below:

```
})
.state('patients', {
    url: '/patients',
    templateUrl: '/views/patient/index.html',
    controller: 'PatientsCtrl'
})
.state('hospitals', {
    url: '/hospitals',
    templateUrl: 'views/hospital/index.html',
    controller: 'HospitalsCtrl'
});
```

- Press `<ctrl> + <s>` to save the file

2. Create a new Hospital link for the view.

- In your text editor, navigate to the `public/scripts` folder
  - Open the `apps.js` file
  - Add the given state and label information to the tabs array and align the text (using spaces) underneath the prior state information
  - Add a comma after `'Patients'}`

  Your code should read as below:

```
    name: 'Predix Seed',
    session: {},
    tabs: [
        {icon: 'fa-home', state: 'about', label: 'About'},
        {icon: 'fa-tachometer', state: 'dashboard', label: 'Dashboard'},
        {icon: 'fa-th', state: 'cards', label: 'Cards', subitems: [
            {state: 'interactions', label: 'Interactions'},
            {state: 'dataControl', label: 'Data Control'},
            {state: 'patients', label: 'Patients'},
            {state: 'hospitals', label: 'Hospitals'}
        ]},
        {icon: 'fa-bar-chart', state: 'components', label: 'Components'}
    ]
```

- Press `<ctrl> + <s>` to save the file

3. Add the controller for the view.

- In your text editor, navigate to the `public/scripts/controllers` directory
  - On the **File** menu, click **New** to create a new file
  - Add the given code to the file to create the controller

  The file includes some dummy hospital data.
  - Save the file as `hospitals.js`

4. Add a reference to the new controller.

- In your text editor, navigate to the `public/scripts/controllers` directory
  - Open the `main.js` file
  - Replace all of the contents of the file with the given code.
  - Press `<ctrl> + <s>` to save the file

5. Create the view to display the hospital web component.

- In your text editor, navigate to the `public/views` directory
  - Create a folder called `hospital`
  - Navigate to `public/views/hospital`
- Create a file
  - Enter the code provided
  - Save the file in the hospital folder you just created as `index.html`

6. Test your application locally.

- Open your browser, refresh your page and navigate to the **Hospitals** link.
  You should see a tab with the text, "*There are 2 hospitals within this network.*"

Exercise 3: Creating a Web Component

Overview

In this exercise, you will customize a Predix component and connect it to the seed application. We have staged the component on the DevBox for you.

Normally, you would download and install the component and its dependencies from GitHub and then generate the component using a Predix component Yeoman generator. This component renders a simple HTML table that displays hospital data. The Yeoman generator allows developers to specify how to build their web application. It uses the yo scaffolding tool from Yeoman as well as a package manager like bower or npm, and a build tool like Grunt.

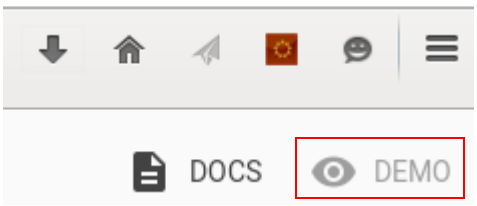
To see the actual steps for this, see the **Creating a Web Component** section in the Appendix at the end of this guide.

Steps

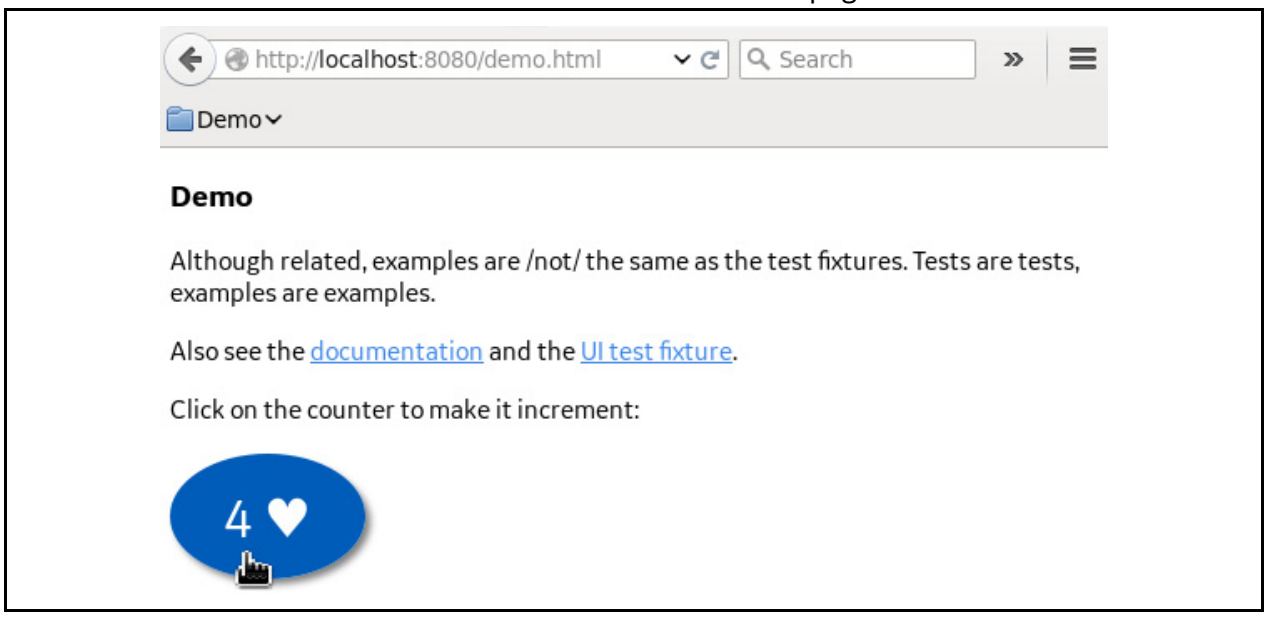
1. Test the web component locally.

- In the Terminal, change directories and start the local test by running these commands
  - **`cd ~/predix/PredixApps/training/UI/hospital-info`**
  - **`grunt firstrun`**

    The `grunt firstrun` command processes the Sass into CSS and starts a local web server to test the component by itself.
  - Click the **DEMO** link in the upper right hand corder to see the web component

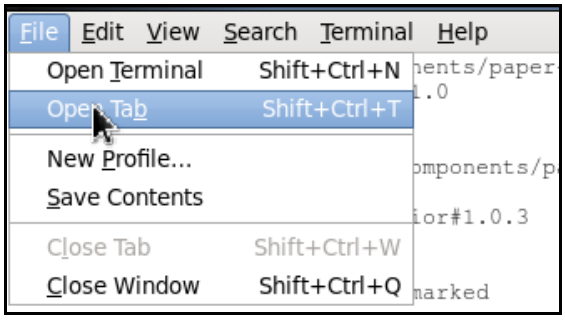


- Click the number to increment it and test the demo page



2. Customize the generated component.

- Open a new Terminal window by selecting **Open Tab** from the **File** menu in the Terminal



- Run the command: **`grunt watch`**
  This tells the grunt service to look for changes in the project. It automatically processes the Sass code (.scss files) into CSS as you make changes.
- In your text editor, navigate to the /`training/UI/hospital-info/sass/`directory

- Open the `hospital-info-sketch.scss` file
  - Replace the Component section with the code provided (Step 2.1)
- Navigate to the `/UI/hospital-info` directory
  - Open the `hospital-info.html` file
  - Replace the `<dom-module>` section with the code provided (Step 2.2)
- In the same file, replace the `<script>` section with the code provided (Step 2.3)
  Your file should read as follows:

```
-->
<dom-module id="hospital-info">
    <link rel="import" type="css" href="css/hospital-info.css"/>
    <template>
        <div class="flex">
            <h4>Hospital Details</h4>
        </div>
        <div>
            <table class="table hospital-table">
                <tr><th class="text--right">Hospital Name :</th><td><span>
{{hospitalDetails.name}}</span></td></tr>
                <tr><th class="text--right">Hospital Address :</th><td><span>
{{hospitalDetails.address}}</span></td></tr>
                <tr><th class="text--right">Email :</th><td><span>
{{hospitalDetails.email}}</span></td></tr>
                <tr><th class="text--right">Phone :</th><td><span>
{{hospitalDetails.phone}}</span></td></tr>
            </table>
        </div>
    </template>
</dom-module>

<script>
    Polymer({

    is: 'hospital-info',


    properties: {
        hospitalDetails: {
            type: Object
        }
    }

});
```

3. Connect the hospital-info component to the seed app.

- In the Terminal, run these commands:
  - (from the hospital-info directory) **bower link**
  - **cd ~/PredixApps/training/UI/predix-seed-1.1.3**
  - **bower link hospital-info**
- In your text editor, navigate to the `predix-seed-1.1.3/public` directory
  - Open the `index.html` file
  - Add the code provided at the end of the `<card.html>` section as follows (Step 3.1)

```
card.html"/>
    <link rel="import" href="bower_components/px-sample-cards/hide-card.html"/>
    <link rel="import" href="bower_components/px-sample-cards/toggle-card.html"/>
    <link rel="import" href="bower_components/px-sample-cards/card-to-card.html"/>
    <link rel="import" href="bower_components/px-sample-cards/description-card.html"/>
    <link rel="import" href="bower_components/px-sample-cards/fetch-data-card.html"/>
    <link rel="import" href="bower_components/px-sample-cards/temperature-card.html"/>
    <link rel="import" href="bower_components/hospital-info/hospital-info.html"/>

</head>
```

  - Modify the `<h1>` tag, after the `svg` tag, replace the word “Predix” with a name for your app, such as “My Healthcare System”

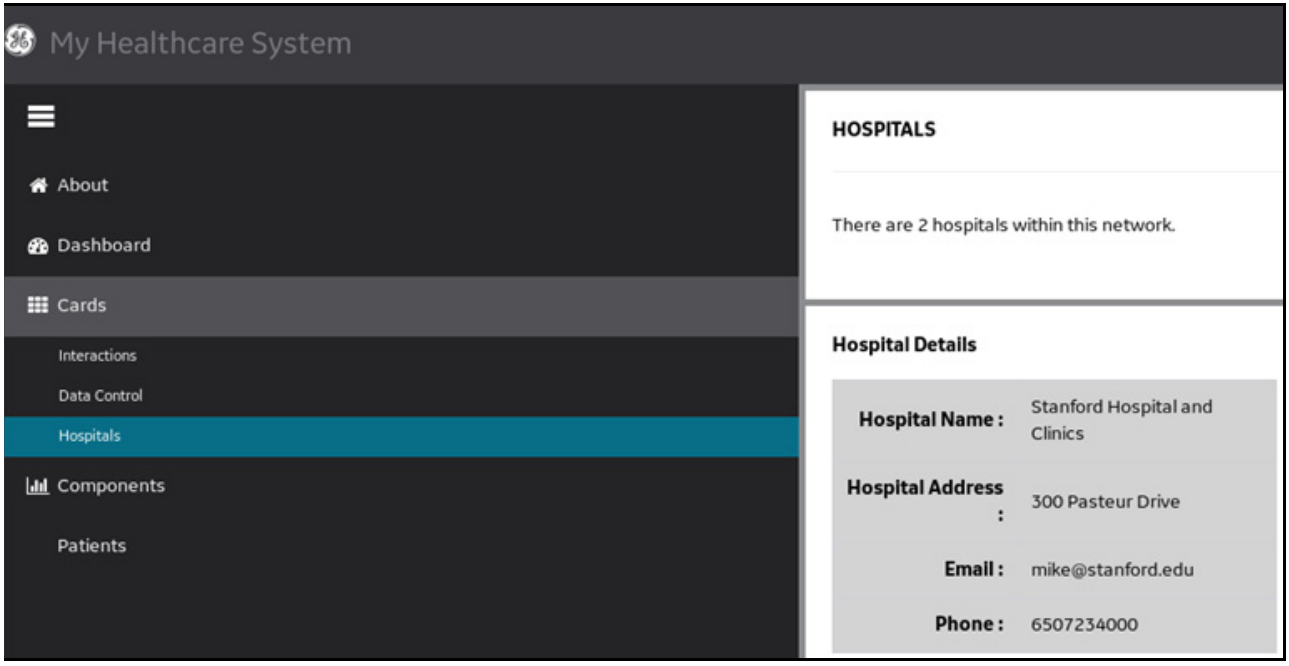
```
407.7-264.7 407.9zM-262.4 391.5c0-2.3 2.3-6.6 3.6-6.
391.5zM-249.4 390.5c0-2.8 1.9-5.6 3-5.1C-245.2 386-2
svg>
        My Healthcare System
      </h1>
    </div>
  </header>
  <div class="viewport">
    <div class="layout layout--full layout--flush">
```

  - Save the file

- In your text editor, navigate to the `predix-seed-1.1.3/public/views/hospital` directory
  - Open the `index.html` file
  - Add the provided code to the bottom of the file (Step 3.2)and save the file

```
<px-card header-text="Hospitals">
    <article role="article">
        <div class="flex">
            <p>There are {{hospitals.length}} hospitals within this network.</p>
        </div>
    </article>
</px-card>
<px-card>
    <hospital-info hospital-details="{{hospitalDetails}}"></hospital-info>
</px-card>
```

  - Run **`grunt serve`** from the predix seed directory to see the new component:



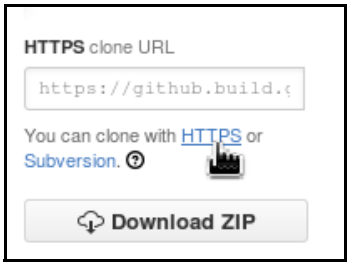
This component can be used anywhere in your application. You could use it in a different application by installing it into that application using `bower install`. In future, there will be a Predix web component catalog for sharing your work with other teams.

Appendix

Installing Px Theme Components from GitHub

Note: The following list is for your information only. It indicates the steps that you would normally take to set up your px-theme project. These were completed for you in order to provide the Px elements required for the lab exercises.

- Connect to the px-theme repository on GitHub
  You need a GitHub account as well as a Predix.io account to do this.
- On the right-hand side of the web page, click the HTTPS link under the **HTTPS** clone URL



- Copy the link in the box. You will use this in the next command in the Terminal window
- In the Terminal, change to directory into which the project should install
  - **`cd ~/predix/PredixApps/UI`**
  - Clone the px-theme library:

    `git clone https://github.build.ge.com/PXc/px-theme.git`

  In the Terminal, you should receive confirmation that this has completed correctly.
- In the Terminal, run the following commands to set up your px-theme project:
  - **`cd px-theme`**
  - **`npm install`**
  - **`bower install`**

The `npm` and `bower install` commands install dependencies for the application in the `px-theme` directory.

Tip: Look at the repository in GitHub for the PXd library to find styles or elements that can be used in your applications. You'll use some styles from the px-forms-design component

- In your browser, open the px-forms-design library in GitHub
- Copy the "bower install" line from the Readme information under the "Installation" heading
  - In the Terminal, run this command:

```
bower install --save https://github.build.ge.com/PXd/px-forms-design.git
```

This installs the px-forms-design module and its dependencies.

Creating a Web Component

- To Download the Yeoman generator for the Predix component, in the Terminal, run the following commands:

```
git clone https://github.build.ge.com/PX/generator-px-comp
cd generator-px-comp
npm link
```

The `git clone` command clones the generator from GitHub. The `npm link` command installs the `generator-px-comp`.

- To create a directory and run the Predix component generator, in the Terminal, run the following commands:

```
mkdir hospital-info
cd hospital-info
yo px-comp
```

The `yo px-comp` command runs the generator, which creates the new component. During this creation, you will be prompted for a component name, mix-ins, and PXd Sass modules. Answer the questions as follows:

  - ? What is the component's name, must have a "-", e.g. 'px-thing'?

    Enter **`my-table`** and press **Enter**

- ?Optional: Local paths to mix-ins the component uses, comma-separated (e.g. '../px-my-mixin,../px-my-other-mixin')

  Press **Enter**
- ?Which of these common Pxd Sass modules does your component need? (You can add more later in bower.jason)

  Use your arrow keys to scroll down to **Tables** and press the Space Bar to select **Tables**
- Press **Enter**

```
[predix@localhost hospital-info]$ yo px-comp


Hello! Answer the prompts to scaffold a Px component.

The generated component itself is not fancy (it makes a circle on the scree
n that increments a counter when clicked),
but contains the Bower config, Gruntfile, tests, etc. common to all Px comp
onents...


? What is the component's name, must have a "-", e.g. 'px-thing'? my-table
? Optional: Local paths to mix-ins the component uses, comma-separated (e.g
. '../px-my-mixin,../px-my-other-mixin')
? Which of these common PXd Sass modules does your component need? (You can
 add more later in bower.json)
 ◯ Buttons
 ◯ Lists
 ◯ Forms
 ◯ Headings
❯◉ Tables
```

- To download and install the Bower dependencies for the component, in the Terminal, run the following commands:
  - **bower install**

  bower link

Lab 3: Connecting to Microservices

Learning Objectives

By the end of the lab, students will be able to:

- Describe how to fetch data from a Polymer web component

Lab Exercises

- *Connecting a Microservice*, page 29

Directions

Complete the exercises that follow.

Exercise 1: Connecting a Microservice

Overview

Before Polymer components, most API calls were made from Angular controllers or services and this is still a supported pattern in Predix. However, in this exercise, you will fetch data from a Polymer iron-ajax element.

You use the following syntax to bring data into a Polymer web component. You use the URL of a microservice to do this, but in our lab we'll bring data in from a json file as the URL.

```
<iron-ajax auto url="{{microservice_Url}}"
last-response="{{data}}"></iron-ajax>
```

Connecting from a Polymer Web Component

1. Install a Polymer element.

- In a Terminal window, change to the `predix-seed-1.1.3` directory and run the following command:

  `bower install polymerelements/iron-ajax --save`

- Enter **1** when prompted

  This command installs the polymer iron-ajax element. It also provides a reference to the polymer iron-ajax elements in the bower.json file.

2. Use the iron-ajax component to fetch data and pass the info to the hospital-info component.

- Navigate to the `index.html` file in the `predix-seed-1.1.3/public/views/hospital` folder
- Replace all of the code in the file with the code provided and save the file.

  Note that the hospital-details jason file (object) is in place of a URL that would normally provide an endpoint into a microservice.

  The options used below the `<iron-ajax` tag are auto, url, handle-as and last-response. The auto option tells the system to make the rest call when the iron-ajax element loads or when the URL or params options changes. The URL tells the program to go to that location to get data. Handle-as tells the program if the data returning will be XML, json, a blob, a document, or other data type. Last-response refers to the most recent response from the ajax request.

3. Create a mock-data file (replaces data from a URL).

- In the `predix-seed-1.1.3/public` directory, create a new file called `hospital-details.json`
- Copy and paste the code provided into the new file and save the file.
- In the Terminal, from the `predix-seed-1.1.3` directory, run the **`grunt serve`** command

  The details in the hospital-details file display in the Polymer element in the web browser.