Documentation – feedback-app

Architecture

- There is a single web component defined, called <feedback-app>. The index.html page served by the server by default has this web component declared as a tag, once in its body section. So, this component encapsulates everything in the page.

- Each web component needs to be defined in its own HTML file with file name matching the name of the web component. This component <feedback-app> is defined in src/feedback-app/feedback-app.html.

- The definition consists of two parts – defining the structure and defining some programmability, within <dom-module>. The structure is defined in <template> tag. Script code for interactions is placed in the <script> tag.

- The most important idea in Polymer is that within <script>, we define an ES6 class to associate with the web component. In this case, the class FeedbackApp is defined, and registered to be associated with the string "feedback-app". So, whenever the component <feedback-app> is imported and used, an instance of FeedbackApp class is created in the browser.

- Styling, which is part of the structure, is included in the <template> tag. Custom styles can also be applied using <custom-style> tag.

```
<dom-module id="my-app">

    <template>
        <custom-style>
            <style is="custom-style">
                 /* CSS style rules go here */
             </style>
        </custom-style>

        <!-- The structure of the document is created by putting in tags -->
        <web-component-one on-click="eventHandler">
             "Content"
        </web-component-one>
        <web-component-two attribute="value"></web-component-two>
      </template>
```

```
  <script>
    /**
     * @customElement
     * @polymer
     */
    class MyApp extends Polymer.Element {
      static get is() { return 'my-app'; }
      static get properties(){
            /*
             Return properties as an object
            */
      }
      /*
        Event handlers go here
      */
      eventHandler(e){
        /*
        Code to handle events, especially accessing shadow DOM
        */
      }
    }
    /*
      Most crucially, registration
    */
    window.customElements.define(MyApp.is, MyApp);
  </script>
</dom-module>
```

Code Snippet 1 – The template for a Polymer component

- The above mentioned layout implicitly enforces Model View Controller (MVC) architecture. The structure forms the view and the scripts constitute the controlling logic. The data model is made up of the attributes of web components/tags in the structure as well as the properties of the associated class. The model and view are agnostic of each other and the controller. The controller handles changes in the other components by responding to events triggered in the view or responding to notifications from the model.

- Since each web-component is a class, it can have properties – both data and methods. The data properties are useful to bind to attributes of web-components in the structure or to place data in the structure. The method properties are usually event handlers and any utility methods. Properties are defined in the

static getter method properties() of FeedbackApp. There are two types of binding. One way binding, where data flows only from the host (say, the FeedbackApp class) to the target (say, an attribute of a component defined in <template>). Two way binding is where data flows in both directions.

- Another important concept in Polymer is that of shadow DOM. This means that for each component its child elements form a DOM tree rooted at the shadow root. This tree is visible and the child elements can be manipulated from within the web-component, but is not visible as part of the browser's DOM tree. Therefore, the FeedbackApp instance can access its shadow DOM tree using many methods in Polymer's DOM API.

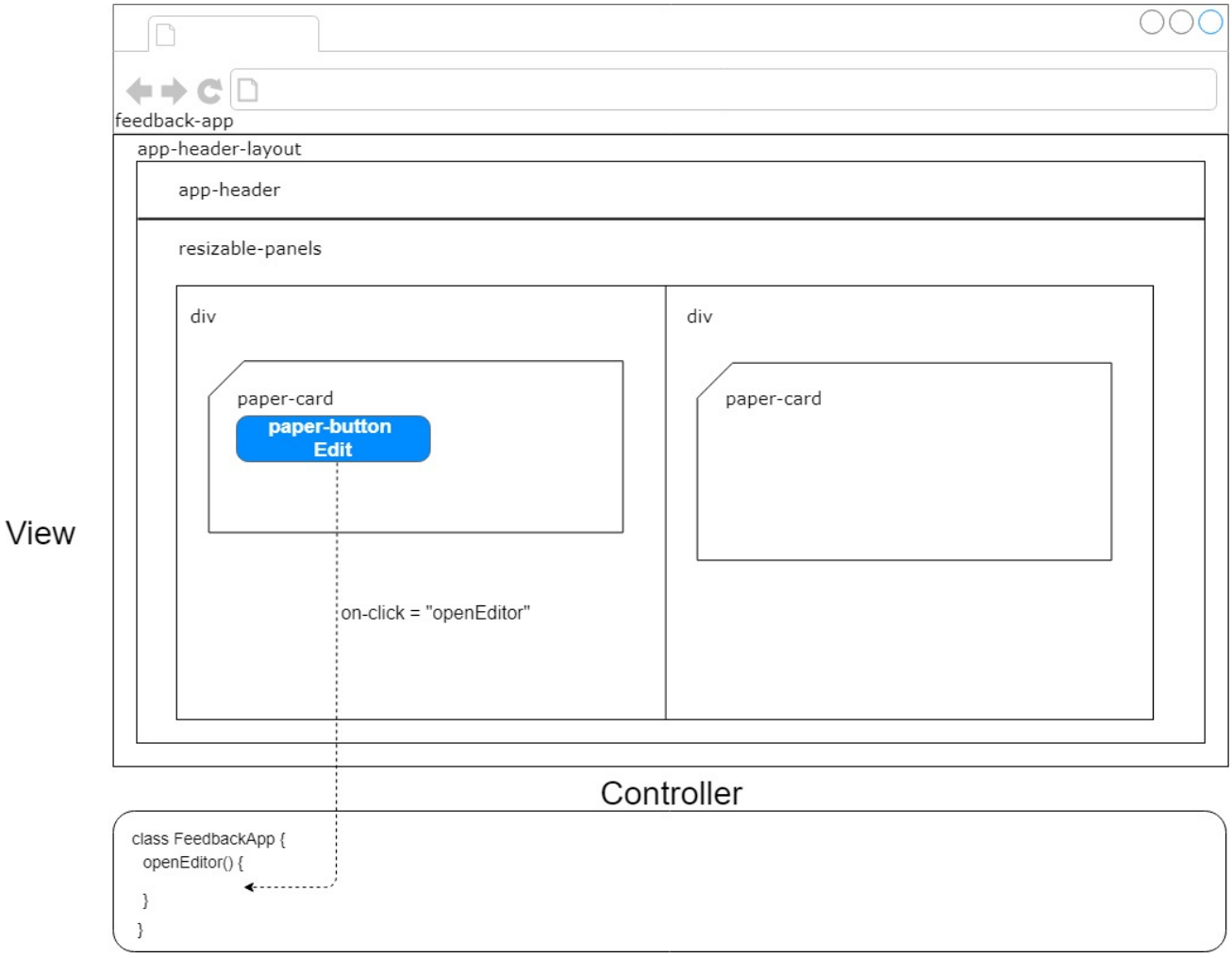

Fig. 1 – Architecture overview

- Coming to <feedback-app>, the layout of the page contains a header and two resizable panels side by side. For resizable panels, <resizable-panels> component

(from www.webcomponents.org) is used. The layout is made by wrapping the header content in <app-header> and wrapping the <app-header> and the <resizable-panels> components in <app-header-layout>.

- Within <resizable-panels>, the panels are actually containers, the left one is for holding the narrative views and the right one is for holding the narrative editor.

- Within both of the panels, a suitable web-component called <paper-card> is used to hold the narrative views and the editor. For the views, a <paper-card> contains a heading wrapped in <paper-toolbar>, tooltip display via <paper-tooltip>, followed by the actual body of the card. This body is encapsulated in a div, which is further wrapped in a Polymer behavior component called <iron-collapse>, which is to be able to expand/hide the narrative view. The buttons at the top are made with <paper-button> component.

- The <paper-card> that holds the narrative editor is similarly structured, except the collapsible behavior.

- To deliver some of the functionalities, the appropriate event handlers are registered for Polymer events on the target elements. All event handlers are properties of FeedbackApp class, defined in <script> tag.

Functionalities

The functionalities so far are -

1. Collapsible <paper-card> for holding the views - tap on blue toolbar to expand or collapse. On loading the page it is initially collapsed.
2. <paper-card> that holds narrative view has 'See More' and 'See Less' functionality.
3. Most importantly, clicking on 'Edit' button opens up the narrative editor with the content loaded. Clicking on 'Close' in the narrative editor will collapse the editor and clear it of the loaded content. The other buttons in the editor don't have any functionality associated yet.
4. Above all, the narrative data holds the data loaded from the JSON file data_narratives.json. The data that has been read using an AJAX call and is inserted in its appropriate place by manipulating the shadow DOM and setting innerHTML attributes.