

SORORITY RECRUITER

**Annie Steenson & Zach
Newton**

**Application to manage sorority recruitment
logistics**

TABLE OF CONTENTS

EMPHASIS 1

Background..... 1

Complex Issues 1

NEW AND COMPLEX WORK..... 3

Dart & Angular 2 3

Responsive User Interface 5

Complex Code Descriptions..... 7

Responsive Forms.....12

BLOOM'S TAXONOMY 14

Analyzing14

Evaluating.....14

Creating15

OVERVIEW

BACKGROUND

Typically, every sorority participates in formal recruitment every year. Some sororities have an extensive matching process to ensure potential new members are talking to the sorority's active members in which they have the most in common with.

Our application alleviates the overhead of matching and automates it into a user friendly web application.

COMPLEX ISSUES

The majority of the complexity of our app comes from our use of Dart with Angular 2, both of which we have never used before.

Dart is Google's class-based, object-oriented, optionally-typed application programming language. We converted our Angular logic into vanilla JavaScript using Dart.

We were introduced this new type of web technology called transformers. Transformers prepare and "transform" package assets before building the application for deployment. In order, to use the

dart transformer, we had to introduce another piece called a package manager. Dart uses Pub as a package manager. New to this technology, we had to figure out how to install dart using Pub. Afterwards, we used Pub to pull packages/dependencies down such as the Dart to JavaScript transformer.

Angular 2 is a JavaScript framework that implements a Model View Controller design pattern. Angular 2 is different because it implements two-way data binding between View and Model layers, which enables applications to be dynamic.

NEW AND COMPLEX WORK

DART & ANGULAR 2

Both Dart and Angular2 also ended up proving to be a complete pain to run, debug, edit, and overall develop with in combination with each other. Specifically, we were unable to successfully add Angular2+Dart routing, so everything is on the same page. It also felt that most, if not all, of what we accomplished with Dart and Angular2 would have been much more easily accomplished by simply using AngularJS.

The majority of the complexity of our app comes from our use of Dart with Angular2, both of which we have never used before. Both Dart and Angular2 also ended up proving to be a complete pain to run, debug, edit, and overall develop with in combination with each other. Specifically, we were unable to successfully add Angular2+Dart routing, so everything is on the same page. It also felt that most, if not all, of what we accomplished with Dart and Angular2 would have been much more easily accomplished by simply using AngularJS.

With that being said, we did our best to develop a somewhat functional app, but most importantly, we kept the spirit of Angular in mind and developed our app to be extremely modular through the use of Templates, Components (similar to a Directive), Objects, and Services.

Through the use of modular code, we were able to separate each part into a specific role that only accomplishes one thing, and each of these modules corresponded to a usable HTML component so that our main page template looked like this:

```
<app-login-form></app-login-form>
<hr>
<app-active-form></app-active-form>
<hr>
<app-pnm-form></app-pnm-form>
```

and our overall app template looked like this

```
<div class="">
  <app-navbar></app-navbar>

  <app-drawer></app-drawer>

  <main>
    <router-outlet></router-outlet>
  </main>
  <app-footer></app-footer>
</div>
```

Most importantly in the subject of our codes modularity was the use of Services for storing anything that could possibly pertain to more than one directive. For example, we used a Menu Service to hold the links that could go both in the Navbar and in the Drawer. Another example is our use of an Auth Service to check the authentication of a user. This would be important if routing worked to make sure that the session was stored between page visits, but also ensure that if a page

is visited by a user who hadn't logged in, that they would not be allowed.

We also split out content pieces into individual modular components. A key example of this is the form components for Active and Potential New Members so that we could keep each of their submit functions separate for cleaner code and greater flexibility.

One of the surprises we came across using Dart (which is based on JavaScript) was lack of a similar object structure to JavaScript and the need to use classes as objects instead. While at first this felt cumbersome, the ability to separate objects into specific structures that behave more like Java classes became a handy tool for maintaining object consistency throughout the app. This object structure was fantastic for developing without worrying about accidentally adding an attribute that shouldn't exist.

RESPONSIVE USER INTERFACE

While not as new as using dart, our use of JS to manipulate DOM elements to make the user experience of our application feel smooth was fairly complex and advanced. Zach's mobile-first, vanilla JS based style library that was created during this project however, does rely heavily on CSS for animation purposes, which is complex in its own right, just not in the eyes of some.

Due to the short window of time and mobile first mindset that was exercised in Zach's library, there was not enough time to format breakpoints properly for desktop, so it is recommended that viewing our app be done on a screen width that is less than 600px.

A large part of the complexity of this Responsive User Interface is the drag out side menu. With mobile being in mind through the entire process, when the site is viewed on a mobile device (accessible on Chrome through the use of the hotkeys Ctrl+Shift+I > Ctrl+Shift+M), one can drag from right to left on the screen to first see an indication bubble, then the full menu. The menu is of course accessible also from

the hamburger button at the top of the screen, but that is often a long stretch for a thumb, especially on a tablet.

A smaller part of the complexity, but a key component in the user experience, is the use of JavaScript to perform front end form validation and provide immediate response to the user, which prevents the need to submit before seeing if there are errors. This behavior can be seen if one types an invalid email address on a member creation form.

Create Active Member

First Name

Last Name

Email

Phone Number

Year

0

Hometown City

Hometown State

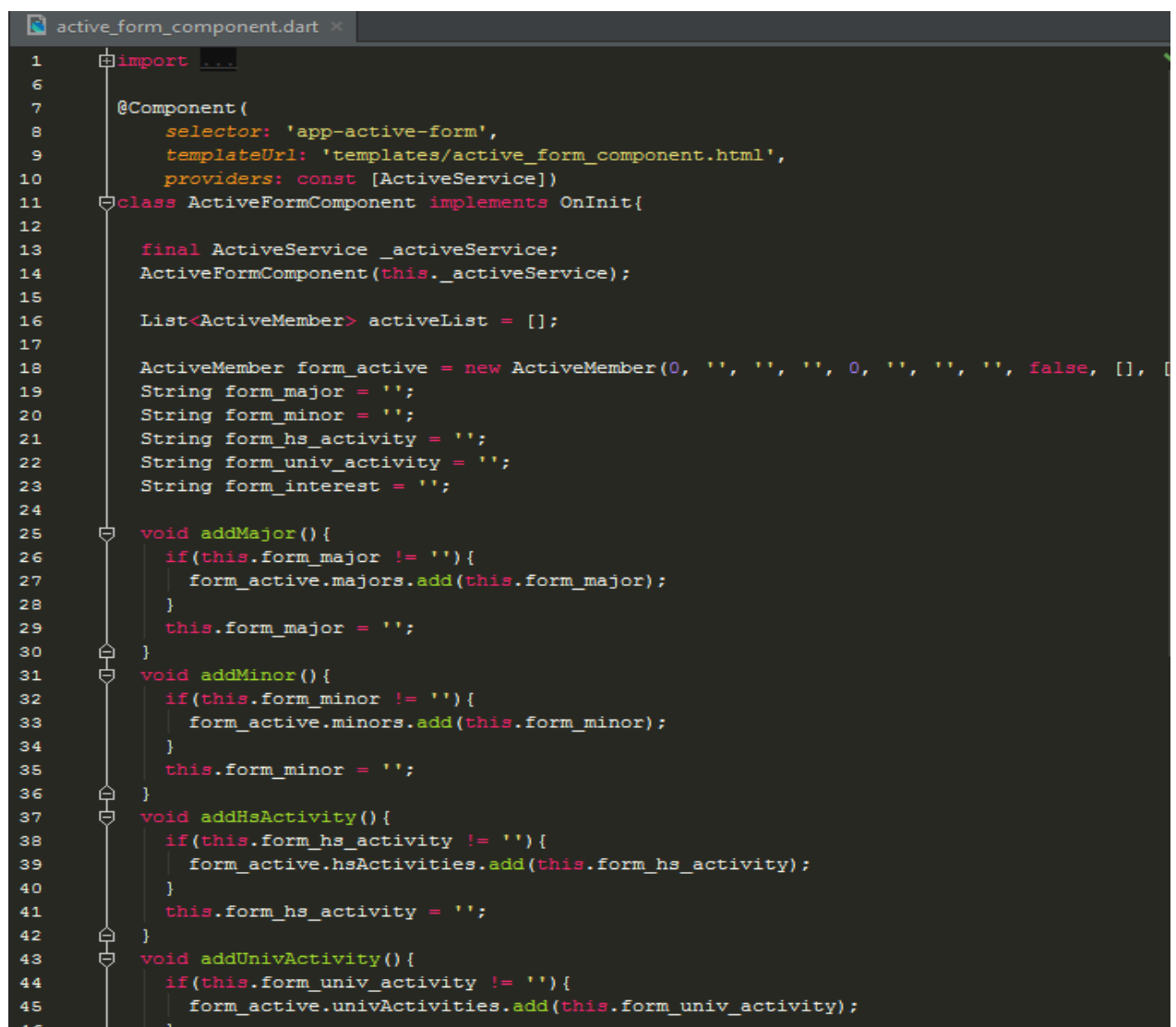
Hometown High School

☐ Admin?

Lastly, along the lines of form experience, vanilla JS is again used to allow for the user to toggle the password field into a text field in order to view what is already displayed. This is common on many sites and apps online, but does not come built in, which is why Zach built it into the library.

COMPLEX CODE DESCRIPTIONS

Active Member Form Component with shown methods to dynamically add to Member's activity and major lists.



```
1 import ...
6
7 @Component(
8   selector: 'app-active-form',
9   templateUrl: 'templates/active_form_component.html',
10  providers: const [ActiveService])
11 class ActiveFormComponent implements OnInit{
12
13   final ActiveService _activeService;
14   ActiveFormComponent(this._activeService);
15
16   List<ActiveMember> activeList = [];
17
18   ActiveMember form_active = new ActiveMember(0, '', '', '', 0, '', '', '', false, [], [
19   String form_major = '';
20   String form_minor = '';
21   String form_hs_activity = '';
22   String form_univ_activity = '';
23   String form_interest = '';
24
25   void addMajor(){
26     if(this.form_major != ''){
27       form_active.majors.add(this.form_major);
28     }
29     this.form_major = '';
30   }
31   void addMinor(){
32     if(this.form_minor != ''){
33       form_active.minors.add(this.form_minor);
34     }
35     this.form_minor = '';
36   }
37   void addHsActivity(){
38     if(this.form_hs_activity != ''){
39       form_active.hsActivities.add(this.form_hs_activity);
40     }
41     this.form_hs_activity = '';
42   }
43   void addUnivActivity(){
44     if(this.form_univ_activity != ''){
45       form_active.univActivities.add(this.form_univ_activity);
46     }
47   }
48 }
```

Majors

Major Name

Add Major

Minors

Minor Name

Add Minor

High School Activities

HS Activity Name

Add HS Activity

University Activities

University Activity Name

Add University Activity

Interests

Active Member Object

```
ActiveMember.dart x
1 class ActiveMember {
2     int id = 0;
3     String firstName = '';
4     String lastName = '';
5     String email = '';
6     String phoneNumber = '';
7     int year = 0;
8     String hometownCity = '';
9     String hometownState = '';
10    String hometownHs = '';
11    bool admin = false;
12    List<String> majors = [];
13    List<String> minors = [];
14    List<String> hsActivities = [];
15    List<String> univActivities = [];
16    List<String> interests = [];
17
18    //new ActiveMember(0, '', '', '', 0, '', '', '', false, [], [], [], [], []);
19
20    ActiveMember(int id,
21        String firstName,
22        String lastName,
23        String phoneNumber,
24        int year,
25        String hometownCity,
26        String hometownState,
27        String hometownHs,
28        bool admin,
29        List<String> majors,
30        List<String> minors,
31        List<String> hsActivities,
32        List<String> univActivities,
33        List<String> interests) {
34        this.id = id;
35        this.firstName = firstName;
36        this.lastName = lastName;
37        this.phoneNumber = phoneNumber;
38        this.year = year;
39        this.hometownCity = hometownCity;
40        this.hometownState = hometownState;
41        this.hometownHs = hometownHs;
42        this.admin = admin;
43        this.majors = majors;
44        this.minors = minors;
45        this.hsActivities = hsActivities;
46        this.univActivities = univActivities;
47        this.interests = interests;
48    }
49 }
```

Active Members

- 👤 Filippa Spiros
- 👤 Dora Michel
- 👤 Efimia Floros
- 👤 Natasa Xanthopoulos
- 👤 Elene Iordanou
- 👤 Pelagia Pachis
- 👤 Marina Giannopoulos
- 👤 Kyriake Kokinos
- 👤 Agathe Michelakos
- 👤 Efthymia Katsaros

Menu Service holding Link Objects that are used in an ngRepeat to populate menu links, similar to a configuration file.

```
menu_service.dart x
1
2  import ...
5
6  @Injectable()
7  class MenuService {
8      List<Link> navLinks = [
9          new Link('Login', '#', ''),
10         new Link('Actives', '#', ''),
11         new Link('Potentials', '#', ''),
12     ];
13
14     List<Link> menuItems = [
15         new Link('About', '', ''),
16     ];
17
18     List<Link> getNavLinks() => navLinks;
19     List<Link> getMenuItems() => menuItems;
20 }
21
```

JavaScript for dragging out the menu from the right.

Assigning window touch event listeners.

```
window.addEventListener('touchmove', function(e) {
    swipeMenu(e.touches[0].clientX);
}, true);
window.addEventListener('touchend', function(e) {
    document.getElementById('drawer-indicator').style.width = '0';
}, true);
```

Function for handling the swipe event

```
45     var oldMouseX = 0;
46     var startMouseX = 0;
47     var deltaMouseX = 0;
48     var swipeDist = window.outerWidth*4/7;
49     function swipeMenu(x) {
50         var ham = document.getElementById('ham-menu');
51         var drawer = document.getElementById('drawer');
52
53         if (oldMouseX != 0) {
54             deltaMouseX = x - oldMouseX;
55         }
56         oldMouseX = x;
57         startMouseX += deltaMouseX;
58
59         var swipeRatio = (Math.abs(startMouseX/swipeDist));
60         if (swipeRatio > 1) {
61             swipeRatio = 1;
62         }
63
64         if (deltaMouseX < 0 && startMouseX < -swipeDist) {
65             openMenu(drawer, ham);
66             startMouseX = 0;
67         } else if (startMouseX > swipeDist) {
68             closeMenu(drawer, ham);
69             startMouseX = 0;
70         } else if (drawer.className != 'open') {
71             document.getElementById('drawer-indicator').style.width = window.outerWidth/7 * swipeRatio + 'px';
72         }
73     }
74 }
75
76 function openMenu(drawer, ham) {
77     drawer.className = 'open';
78     ham.className = 'open';
79     navbar = document.getElementById('navbar');
80     navbar.classList.remove('scrolled');
81 }
82 function closeMenu(drawer, ham) {
83     drawer.className = 'close';
84     ham.className = '';
85     scrollNav();
86 }
87
```

RESPONSIVE FORMS

Assignment for form input checks. The functions are also run on each element on load in order to account for the existence of default values.

```
27     var formTextElements = document.getElementsByClassName('form-element-text');
28     for(var i = 0; i < formTextElements.length; i++){
29         inputTextContentCheck(formTextElements[i].getElementsByTagName('input')[0]);
30         inputTextValidityCheck(formTextElements[i].getElementsByTagName('input')[0]);
31         formTextElements[i].addEventListener('keyup', function(e) {
32             inputTextContentCheck(e.target);
33             inputTextValidityCheck(e.target);
34         });
35     }
```

Interests

Interest Name

Add Interest

+ Create Active Member

Interests

- Coding

Interest Name

Add Interest

+ Create Potential New Member

Input content checking functions to tell the input element to keep the label above the input

```
106 function inputTextContentCheck(elem){
107     if(elem.value != ''){
108         elem.parentElement.classList.add('has-text');
109     } else {
110         elem.parentElement.classList.remove('has-text');
111     }
112 }
```

Input validity check to use built in html checks such as email or number, or a custom pattern attribute for things like phones.

```

113 function inputTextValidityCheck(elem) {
114     var valid = elem.checkValidity();
115     if(valid && (elem.classList.contains('invalid') || elem !== document.activeElement)){
116         elem.parentElement.classList.remove('invalid');
117     } else if(!valid){
118         elem.parentElement.classList.add('invalid');
119     }
120 }

```

Assignment for password toggling click function

```

36     var passwordToggles = document.getElementsByClassName('password-toggle');
37     for (i = 0; i < passwordToggles.length; i++){
38         passwordToggles[i].addEventListener('click', function(e) {
39             passwordToggle(e.target);
40         })
41     }
42 };

```

Password field toggle function to show text or dots

```

121 function passwordToggle(elem) {
122     var passwordInput = elem.parentElement.getElementsByTagName('input')[0];
123     if(passwordInput.getAttribute('type')=='text'){
124         passwordInput.setAttribute('type', 'password');
125     } else {
126         passwordInput.setAttribute('type', 'text');
127     }
128 }

```

PORTFOLIO2

Login

Username

Password

Login

Authorized: false

BLOOM'S TAXONOMY

ANALYZING

We **experimented** with new technologies such as Angular JS 2, Dart JavaScript, and Pub Package Manager. We decided to use Angular JS 2 because the framework provided a clean implementation for two-way data binding. This can be **compared** to creating a dynamic web application using JQuery to manipulate DOM elements. In **contrast**, JQuery is not a complete framework unlike Angular JS. Angular JS comes with a model, view, and controller layer. JQuery would be considered more of a library for the view layer.

We also **tested** playing around with creating a mobile-first CSS library. Mobile-first means that styles are first applied to mobile device sizes and then you extend those styles to larger screens using media queries.

EVALUATING

We **commend** Dart for its more useful functionality such as object-oriented classes and developer friendly abstraction of JavaScript. There are many different benefits Dart brings to the table such as a better DOM library. For example:

```
Element newDiv = new DivElement()..style.color = "white";  
Document.body.children.add(newDiv);
```

The above code creates a new div element and sets the text color to white and adds it to the document body. The object oriented syntax is very intuitive to most programmers rather than vanilla JavaScript's version.

We **support** and **recommend** mobile-first CSS design because the CSS constraints and code for larger screens is usually more complicated. Mobile-first simplifies the code base and is better for long term production.

CREATING

We **designed** an application to be user friendly and help eliminate unnecessary calculations for Sorority Recruitment Officers. We **constructed** our own CSS library in order to accommodate to mobile-first and have a better tailored User Design Experience. **Written** with the Angular JS 2 framework, we **constructed** a modular code base and **formulated** Dart code and **assembled** JavaScript using the Dart transformer.

DICUSSION

Analyzing

Comparing & Contrast:

Angular proved to be useful for creating a dynamic web page. In our previous portfolio, we used JQuery to provide dynamic functionality. However, Angular simplifies a lot of overhead with managing DOM elements, unlike with JQuery.

Evaluating

Appraise & Argue:

In vanilla JavaScript, you have many different types of false values such as: null, undefined, "", 0, NaN, and false. This property is beneficial for certain applications that are more type dependent.

Dart uses one false value (falsify), which reduces the complexity of the code base.

Dart analyzed dart and stripped it down to two methods. Using the JQuery library as a dependency for a long term application could cause issues if JQuery changes its implementation.

Creating

Assemble:

We created a user-friendly application using Angular JS and Dart. This application manages Active Members of a sorority and Potential New Members.

It helps reduce overhead for Sorority Recruitment managers.