ICP5

Keenan Flynn – kpfxn8@umsystem.edu – https://github.com/kfly2fly/Web-Mobile-Spring-2022
Jasmine Naraine- jnytc@umstyem.edu -https://github.com/JNaraine/Web

This ICP focuses on the Angular framework. We used this framework to build a simple To Do list and countdown timer.

To Do List

A user can define a list of items they wish to accomplish. They can add, remove, edit, and check off these items as completed. This was accomplished by creating 2 components: the standard app component and an Item component. The app component focuses on the general style of the page and user interface (such as the Add item feature). The item component focuses on each individual to do items and how to dynamically display those items.

app.component: To add an item, the addItem() method is called in app.component.ts. This method appends an input description and the boolean done=false to the allItems[] list. This list keeps track of the item info. The number of items is displayed below the 'Add' button and dynamically changes based on the number of items and uses a ngIf directive. Finally a ngFor directive loops through allItems[] and calls the Item component to display each individual item.

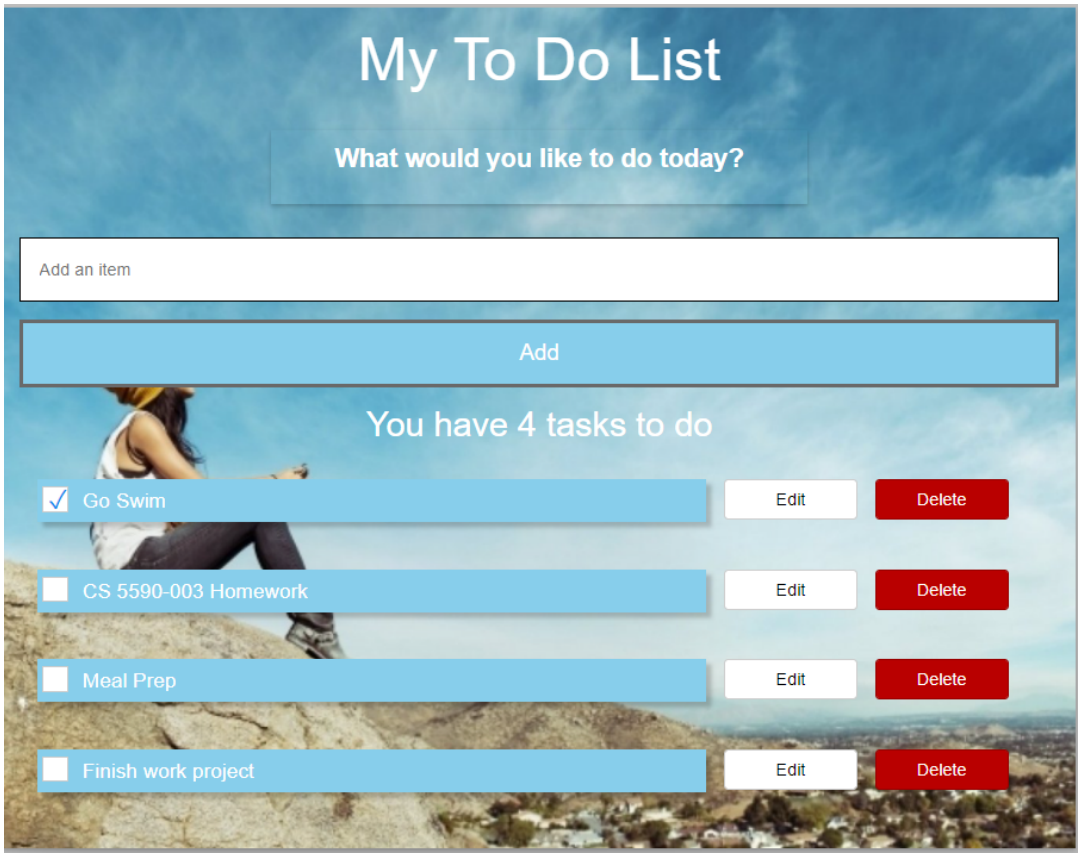
item.component: A checkbox and text box is displayed with the items description. The checkbox displays if the item is 'done'. The description can be edited with an Edit button which inverts a boolean variable named 'editable'. When editable is true, 2 new buttons appear: Cancel and Save. Cancel turns editable to false and returns to default view. Save calls the saveItem function which updates the items status. If the Delete button is pressed, an event emitter object is created which calls remove() in app.component and deletes the item.

There were several challenges during this ICP. One such challenge was getting the dependencies and config files to work together. We had add this line into the tsconfig.json file '"strictPropertyInitialization": false' because we did not use constructors in the To Do list. It was also challenging getting the Items formatted correctly, especially the checkbox and text.

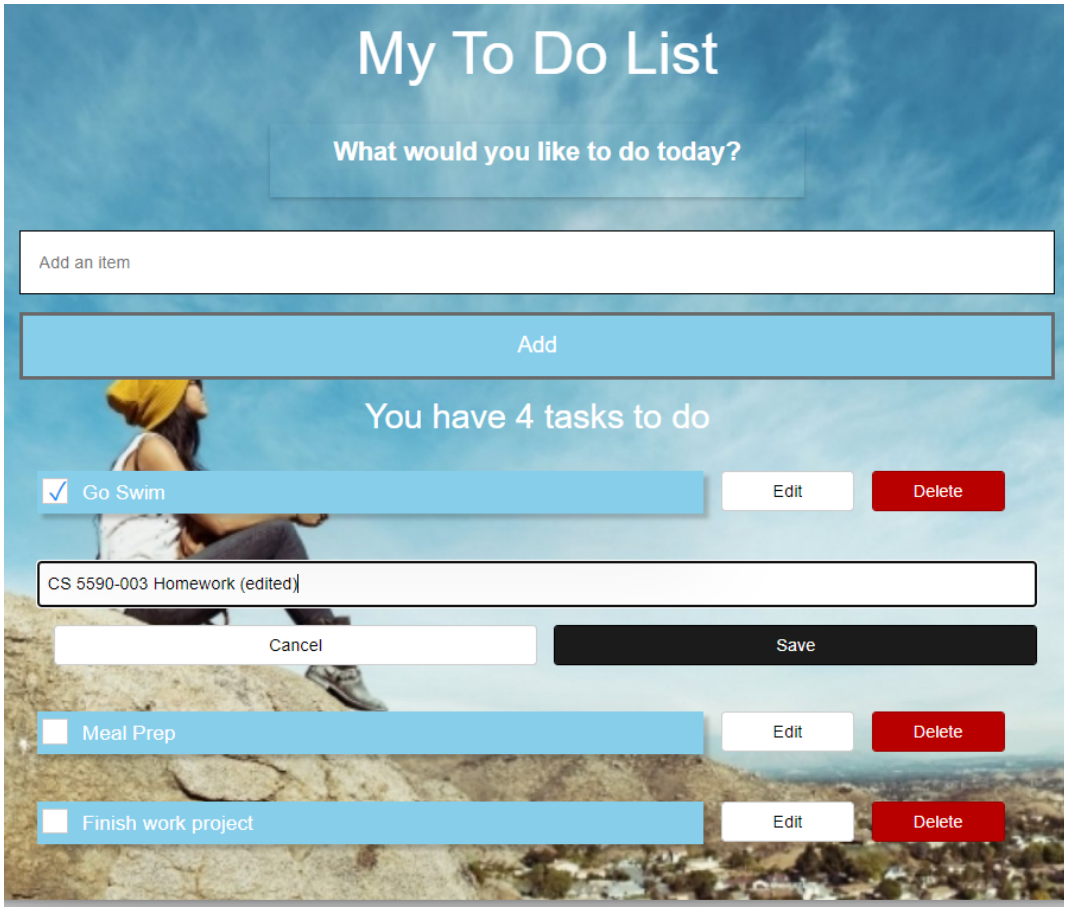
*The countdown timer is explained below the screenshots for the To Do list

TO DO List

Main Page



Main page with item edit



Index.html

```html
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>MyTodoList</title>
  <base href="/">
  <!-- Make webpage responsive -->
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body class="bg">
  <!-- Call app.component.html -->
  <app-root></app-root>
</body>
</html>
```

Style.css (there are more classes that are not included in this picture)

```css
/* You can add global styles to this file, and also import other style files */

/*Import bootstrap*/
@import 'bootstrap/dist/css/bootstrap.css';

/*Global Classes*/
body {
  font-family: Helvetica, Arial, sans-serif;
}

.bg{
  background-color: #C0C0C0;
}

.btn-wrapper {
  /* flexbox */
  display: flex;
  flex-wrap: nowrap;
  justify-content: space-between;
}

.btn {
  color: #000;
  background-color: #fff;
  border: 2px solid #cececе;
  padding: .35rem 1rem .25rem 1rem;
  font-size: 1rem;
}

.btn:hover {
  background-color: #ecf2fd;
```

Item.ts (not part of a component)

```
// Create the definition for a to-do item so our application can use this type
//This item has a description and can be 'done'
export interface Item {
  description: string;
  done: boolean;
}

```

app.component.ts

```
import { Component } from '@angular/core';

// Specify selector, templateUrl, and stylesheet
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

// Define and export class
export class AppComponent {

// Hard code some items
  allItems = [
    { description: 'Go Swim', done: true },
    { description: 'CS 5590-003 Homework', done: false },
    { description: 'Meal Prep', done: false },
    { description: 'Finish work project', done: false },
  ];

// Fucntion that returns all items
  get items() {
      return this.allItems;
}

// Add an item to allItems list
  addItem(description: string) {
  this.allItems.unshift({
    description,
    done: false
  });
}

//remove an item from the list
remove(item: any) {
  this.allItems.splice(this.allItems.indexOf(item), 1);
}

}
```

app.component.html

```html
<!-- Create the main div with a background picture -->
<div class="main bg-cover text-white" style="background: url(https://www.incimages.com/uploaded_files/image/1920x1080/getty_468868827_970566970450047_60099.jpg)">
    <!-- Header -->
    <h1 class="display-4 font-weight-bold">My To Do List</h1>
    <!-- Sub header -->
    <div class="main2">
        <label for="addItemInput">What would you like to do today?</label>
    </div>
    <!-- Add a input bar where the user can define a new to-do item description -->
    <input
      #newItem
      placeholder="Add an item"
      (keyup.enter)="addItem(newItem.value); newItem.value = ''"
      class="lg-text-input"
      id="addItemInput"
    />
    <!-- Add a button that appends the input description to the item list with done=false -->
    <button class="btn-primary" (click)="addItem(newItem.value)">Add</button>

   <!--  Dynamically display # of items left -->
    <h2>You have {{items.length}} <span *ngIf="items.length === 1; else elseBlock">task</span>
    <ng-template #elseBlock>tasks</ng-template> to do</h2>

    <!-- ngFor directive loops through list of items and calls the Item component to display each item -->
    <ul>
      <li *ngFor="let item of items">
        <app-item (remove)="remove(item)" [item]="item"></app-item>
      </li>
    </ul>
</div>
```

app.component.css (there are more classes that are not included in this picture)

```css
/*Dynamically change screen width for responsive design*/
@media screen and (min-width: 600px) {
  .main {
    width: 70%;
  }
}

/*Define classes for the App componenet*/
body {
  color: #4d4d4d;
  background-color: #f5f5f5;
}

.main {
  max-width: 1000px;
  width: 85%;
  margin: 2rem auto;
  padding: 1rem;
  text-align: center;
  box-shadow: 0 2px 4px 0 rgba(0,0,0,.2), 0 2.5rem 5rem 0 rgba(0,0,0,.1);
}

.bg-cover {
    background-size: cover !important;
}

.main2 {
  max-width: 500px;
  width:  60%;
  margin: 2rem auto;
  padding: .5rem .2rem;
  text-align: center;
  box-shadow: 0 2px 4px 0 rgba(0,0,0,.2), 0 2.5rem 5rem 0 rgba(0,0,0,.1);
}
```

Item.component.ts

```
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';
// Import item.ts
import { Item } from "../item";

// Specify selector, templateUrl, and stylesheet
@Component({
  selector: 'app-item',
  templateUrl: './item.component.html',
  styleUrls: ['./item.component.css']
})

// Define and export class
export class ItemComponent {

  // Set editable to false
  editable = false;

  // Import specific objects from App component
  @Input() item: Item;
  @Input() newItem: string;
  //Export object that removes item
  @Output() remove = new EventEmitter<Item>();

  // Function that saves new description of item
  saveItem(description: any) {
    if (!description) return;
    this.editable = false;
    this.item.description = description;
  }
}
```

item.component.html

```
<div class="item">
<!-- Create a bootstrap grid for each item -->
  <div class="btn-wrapper flex-row" *ngIf="!editable">
    <!-- Create a div for each item description and checkbox -->
    <div class="col-8 desc">
      <!-- Add a checkbox and item description element for each item -->
      <input [id]="item.description" type="checkbox" (change)="item.done = !item.done" [checked]="item.done" />
      <label [for]="item.description" class="text-center">{{item.description}}</label>
    </div>
    <!-- Create column for edit and delete button -->
    <div class="col-4">
      <!-- Change editable to true -->
      <button class="btn" (click)="editable = !editable">Edit</button>
      <!-- Tell app component to remove an item -->
      <button class="btn btn-warn" (click)="remove.emit()">Delete</button>
    </div>
  </div>
  <!-- This section shows only if user clicks Edit button -->
  <div *ngIf="editable">
    <!-- Create input bar with previous description -->
    <input class="sm-text-input" placeholder="edit item" [value]="item.description" #editedItem (keyup.enter)="saveItem(editedItem.value)">z
    <div class="btn-wrapper">
      <!-- Cancel the edit operation -->
      <button class="btn" (click)="editable = !editable">Cancel</button>
      <!-- Export new item description if saved -->
      <button class="btn btn-save" (click)="saveItem(editedItem.value)">Save</button>
    </div>
  </div>
</div>
```

Item.component.css (there are more classes that are not included in this picture)

```css
.item {
  padding: .5rem 1rem .75rem 1rem;
  text-align: left;
  font-size: 1.2rem;
}

.desc {
  box-shadow: 5px 5px 4px 0 rgba(0,0,0,.2), 0 2.5rem 5rem 0 rgba(0,0,0,.1);
  background-image: none;
  background-color: skyblue;

}


.btn-wrapper {
  margin-top: 1rem;
  margin-bottom: .5rem;
}

.btn {
  /* menu buttons flexbox styles */
  flex-basis: 49%;
  margin-left: 1rem;
  width: 40%;
}

.btn-save {
  background-color: #000;
  color: #fff;
  border-color: #000;

}

.btn-save:hover {
  background-color: #444242;
}

.btn-save:focus {
  background-color: #fff;
  color: #000;
}
```

app.module.ts

```typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms';
import { ItemComponent } from './item/item.component'

@NgModule({
  declarations: [
    AppComponent,
    ItemComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Countdown

In the countdown we had to create a countdown spanning from days, hours, minutes, and seconds. When a user gives a specific date. For the countdown we had to use Angular to set up the countdown. The component.ts is the setup for the countdown. Math.floor returns the largest integer less than or equal to, like the days, hours, minutes and seconds. For the component.ts to work the component.html has to have the same function word as in the component.ts, as for this project the demo is the key that connects each component. For the countdown to be fully functional, there had to be imports in the module.ts. The program was functional with the imports of CountdownModule and a HttpModule.

Countdown

module.ts

app > ts app.module.ts > AppModule

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
//import { CountdownModule } from 'ngx-countdown';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import {HttpClientModule } from '@angular/common/http'

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
    //CountdownModule



  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Component.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale
  <title>Countdown</title>
</head>

<body>
  <h1> Camping</h1>
  <section class="register">
    <div class="container">

      <h2>{{demo }}</h2>
    </div>

  </section>
</body>
</html>
```

Component.ts

```ts
import { Component, OnInit } from '@angular/core';



@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements OnInit {
  title = 'ngx-countdown';


 constructor() {}

 ngOnInit(): void {

 }
  countDownDate = new Date("Mar 28, 2022 12:37:25").getTime();

  demo:any;
  x= setInterval(()=>{

    var now = new Date().getTime();
    var distance = this.countDownDate - now;
    var days = Math.floor(distance/(1000*60*60*24));
    var hours = Math.floor((distance %(1000*60*60*24)) / (1000*60*60));
    var minutes = Math.floor((distance % (1000*60*60)) / (1000*60));
    var seconds = Math.floor((distance % (1000*60)) / (1000));

    this.demo = days + " Days    " + hours + " Hours   " + minutes + " Minutes  " + seconds + " Seconds";
  })
```