

Completion: Required

Submission: Required

Last Name: _____

First Name: _____

This *tutorial* is to help you learn REACT, AngularJS, AJAX and ChartJS for creating better interactive & visual websites.

1) How and what to submit?

*Submit the following (upload in Blackboard to the available container) in **"one"** PDF document (not in docx or any other format):*

- i) The certification page (see next page) should be the first page, followed by*
- ii) your solution to the problems given on this assignment.*

One way is to copy the certification page and your solution to the problems into a Word document and then save the Word document as PDF, and upload the PDF version (not docx version). Only the PDF version will be graded.

2) Only ONE upload attempt is allowed: *Before submitting a document through Blackboard, you should review the document being uploaded to make sure that you are uploading the correct document (e.g. do not upload the assignment belonging to another course). To help you prevent uploading wrong documents, notes (titled **"HelpOnSubmissionThroughBlackboard"** on how to save & review drafts before final submission have been uploaded under Reference Material folder.*

Certification Page

This page must be the first page of your uploaded document.

Your assignment will not be graded without this page (completed with your full name in the area provided) as the first page of your uploaded document.

I, _____, certify that the work I am uploading represents my own efforts, and is not copied from anyone else or any other resource (such as Internet). *Furthermore, I certify that I have not let anyone copy from my work.*

Tutorial Portion

Learning Objectives!

React, AngularJS, AJAX, and ChartJS are all meant to create better visual and interactive websites easily. Below is a summary of each one:

-React (*reacts* to state changes) is a JS library for building user interfaces:

Video: <https://www.youtube.com/watch?v=N3AkSS5hXMA>

-AngularJS is a framework to develop a Single-Page Application (SPA) letting one render dynamic views in web applications:

Video: <https://www.youtube.com/watch?v=WAZTZUgeLhQ>

-AJAX (Asynchronous JAVa and XML) helps autorefresh data on websites:

Video: <https://www.youtube.com/watch?v=3l13qGLTgNw>

-ChartJS lets one easily make and include graphs on websites:

Video: <https://www.youtube.com/watch?v=W6ai7wu5Vlk>

In this tutorial we will be learning how the above may be used to make websites better, interactive, and visual.

Below is a summary of the sections in this tutorial:

Section 1: Introduction to React.

Section 2: Introduction to AngularJS.

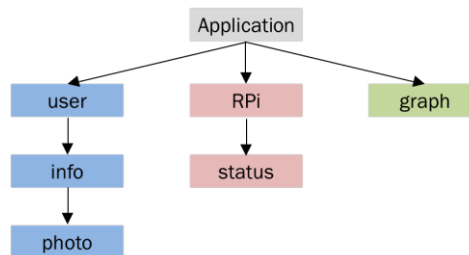
Section 3: Introduction to AJAX.

Section 4: Introduction to ChartJS.

1) React (Reference: <https://www.youtube.com/watch?v=Ke90Tje7VS0>)

- React is *component* based where a component is *independent & reusable* entity (fragment of code).
- React is preferred for UI due to its *fast rendering times*.
- One can use *HTML within JavaScript*.
- A React component's *container is an HTML DIV*.

An application is made up of independent components that can be in different parts of a website or in different websites. Each component may have child components. A tree may be representative of components with application at its root. Consider three components for an application: *user* (with logged-in user's info & photo), *RPi* (with status of sensors & actuators), a *graph* (of data received from a certain critical sensor). This may be represented by the following tree:



Let's make a simple website with a button which when clicked responds by showing the text "Lighting up LED1."

a) Access 000webhost.com and create index.php with the code given below where we are **i)** creating an **Led1On** button, **ii)** loading **React** followed by **iii)** loading the **required component code**:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>React</title>
  </head>
  <body>
    <h2>LED ON Button</h2>
    <div id="LED1"></div>

    <!-- Load React. -->
    <script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin></script>
    <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>

    <!-- Load React component. -->
    <script src="ledon.js"></script>

  </body>
</html>
```

b) In the above code, the React component was identified by "*ledon.js*". Here we need to describe the component's functionality. The following code needs to be stored under _____ in *ledon.js* file:

```
'use strict';
const e = React.createElement;
class ledOnButton extends React.Component {
  constructor(props) {
    super(props);
    /* initial state of ledOnButton */
    this.state = { on: false };
  }

  render() {
    if (this.state.on) {
      /* result when button is clicked */
      return 'LED is ON';
    }

    return e(
      'button',
      { onClick: () => this.setState({ on: true }) },
      'On'
    );
  }
}

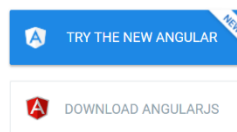
/* define button container */
const domContainer = document.querySelector('#LED1');
/* render the ledOnButton in the defined Data Object Model container */
ReactDOM.render(e(ledOnButton), domContainer);
```

2) AngularJS (Reference: <https://www.youtube.com/watch?v=zKkUN-mJtPQ&list=PL6n9fhu94yhWKHkL7RJmmXyxkuFB3KSI>)

- A *Model* consists of data, variables, and methods.
- A *View* is what is displayed to the user.
- Two-way data binding synchronizes *model* with the respective *view*.
- *Ng-app* or another *AngularJS directive* is required in HTML tags to use AngularJS.
- AngularJS is embedded in HTML, as has been the case while using traditional JS scripts. React, on the other hand, is embedded HTML within JS.

Due to AngularJS' two-way *data binding* synchronization capability between *model* and *view*, when one gets updated the other gets updated as well. *PayPal* and *GitHub Community Forum* are examples where AngularJS was used to develop the websites.

a) Using a **Windows** machine, access <https://www.angularjs.org> and click on **DOWNLOAD ANGULARJS** to download AngularJS.



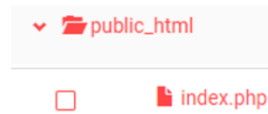
In the dialog box that pops up, accept the default settings. Alternatively, one can use CDN (Content Delivery Network) as the AngularJS script source rather than downloading the AngularJS script.



Right click anywhere on the popped-up window with *AngularJS script file*, and save it on your Windows machine.

```
/*
AngularJS v1.7.9
(c) 2010-2018 Google, Inc. http://angularjs.org
License: MIT
*/
(function(C){'use strict';function re(a){if(D(a))=(a.objectMaxDepth)&&(b.objectMaxDepth=Xb(a.objectMaxDepth)?a.objectMaxDepth:NaN),v
Xb(a){return W(a)&&0(a)}function F(a,b){b||Error;return function(){var d=arguments[0],c;c="[+({a+*':\""}+d+" http://errors.angular
f.js?="+arguments[d];f="function"==typeof f?f.toString():replace(/\\[\\s\\$]/,"");"undefined"==typeof f?"undefined":"string"==typeof f?
b="length" in Object(a)&&a.length;return W(b)&&(0<b&&b-1 in a)||"function"==typeof a.item)}function r(a,b,d){var c,e;if(a){if(a){for
ya(a){var f="object"!=typeof a;c=0;for(e=a.length;c++;)(f||c in a)&&b.call(d,a[c],c,a)}else if(a.forEach&&a.forEach!=""a.forEac
a).hasOwnProperty(c)&&b.call(d,a[c],c,a);else for(c in a)ta.call(a,c)&&b.call(d,a[c],c,a);return a}function Oc(a,b,d){for(var c=Obje
{return+pb}
function Zb(a,b,d){for(var c=a.$$hashKey,e=0,f=b.length;c++;e){var ge=b[e];if(D(ge)||B(ge))for(var k=Object.keys(g),h=0,l=k.length;h<
a[m]=p.clone();"__proto__"!=m&&(D(a[m])||((a[m]=H(p)?[{}]:{}),Zb(a[m],[p],l0)):a[m]=p);c?a.$$hashKey=c:delete a.$$hashKey;return a}func
parseInt(a,
10)}function ac(a,b){return S(Object.create(a),b)}function E(){function Ta(a){return a}function Ia(a){return function(){return a}}fu
{return"undefined"!=typeof a}function D(a){return null!=a&&"object"==typeof a}function Hc(a){return null!=a&&"object"==typeof a&
Date]"==Ia.call(a)}
function H(a){return Array.isArray(a)||a instanceof Array}function cc(a){switch(Ia.call(a)){case "[object Error]":return!0;case "[ob
{return"function"==typeof a}function ab(a){return"[object RegExp]"==Ia.call(a)}function $a(a){return a&&a.window==a}function bb(a)
```

b) Access your *000webhost.com* account and create an *index.php* under *public_html* folder to start developing a website based on HTML, CSS and AngularJS.



Drag and drop the *AngularJS* script file that was just downloaded to the *public_html* folder. For security reasons, *AngularJS* script file should be located under **scripts** folder and its path provided in HTML. Once you develop the intended website, you can move *AngularJS* script file to the **scripts** folder.



c) Code *index.php* with the following:

```
<!doctype html>
<html>
  <head>
    <script src="angular.min.js"></script>
  </head>
</html>
```

Later, when *AngularJS* script file is moved under the scripts folder, make sure to provide the correct path in the above code.

d) Modify *index.php* according to the code given below to add our local AngularJS script. Note that AngularJS script will only be recognized and executed within its specified "*ng-app*" tag:

```
<!doctype html>
<html>
  <head>
    <script src="angular.min.js"></script>
  </head>
  <body ng-app>
    <div>
      {{ ['John', 'Jane', 'Ben'][2] }} </br>
      50+25 = {{ 50 + 25 }}
    </div>
  </body>
</html>
```

Note that the double-curly braces hold expressions. The first set of double-curly braces defines an array and selects element with index "2", and the second set computes the addition of 50 and 25. The displayed result should be as follows:

A screenshot of a web browser window. The browser has a title bar that says 'Ben'. The main content area of the browser displays the text '50+25 = 75'.

e) Module and Controller in AngularJS: A **module** in AngularJS is a container for an area of an application. It has two arguments, the first being the name of your module (e.g. **myApp**) and the second being dependencies ("[]" for no dependencies):

```
var myApp = angular.module("myApp",[]);
```

A **controller** in AngularJS is a JS function meant to build the required *model* for the intended *view*. Recall that a *model* consists of data, variables, and methods, whereas a *view* is what the user sees. Therefore, a *controller* in AngularJS is supposed provide data, variables, and methods and build the required *model*. In the code below, *\$scope* is an AngularJS *object* passed to the controller by AngularJS *framework*. The code is setting *\$scope*'s *message* property to "AngularJS" text:

```
var myApp = angular.module("myApp",[]);  
myApp.controller("myController", function($scope){  
    $scope.message = "AngularJS";  
});
```

Save the above code under *script1.js*. Wherever you save this file, make sure to provide the correct path in the *modified index.php* code below:

```
<!doctype html>  
<html ng-app = "myApp">  
<head>  
    <script src="angular.min.js"></script>  
    <script src="script1.js"></script>  
</head>  
<body>  
    <div ng-controller="myController">  
        {{ message }}  
    </div>  
</body>  
</html>
```

Refresh the page to check the output which should be the result of processing the expression within the double-curly braces:

AngularJS

f) Having the basic knowledge of how AngularJS works, let's build a **to-do-list** web application where a user can add items to the list and delete items from the list (of course, dynamically). Let's name our **module** and **controller** as **todoModule** and **todoController**, respectively, in the modified *index.php* code below:

```
<!doctype HTML>
<html>
  <style>
    body {text-align: center;}
    #App {height: 500px; width: 500px; background-color: powderblue;display: inline-block;}
    ul {text-align: justify;}
  </style>
  <script src="angular.min.js"></script>
  <link rel="stylesheet" href="style.css">
    <script>
      angular.module('todoApp', [])
        .controller('todoController', function($scope) {
          $scope.activity = [];
          $scope.add = function() {
            $scope.activity.push($scope.title);
          }
          $scope.delete = function() {
            $scope.activity.splice(this.$index, 1);
          }
        })
    </script>
</html>
```

Don't refresh the webpage yet since we still need to add **structure** (form, buttons, etc.) to the webpage. Let's modify the above to the code shown below:

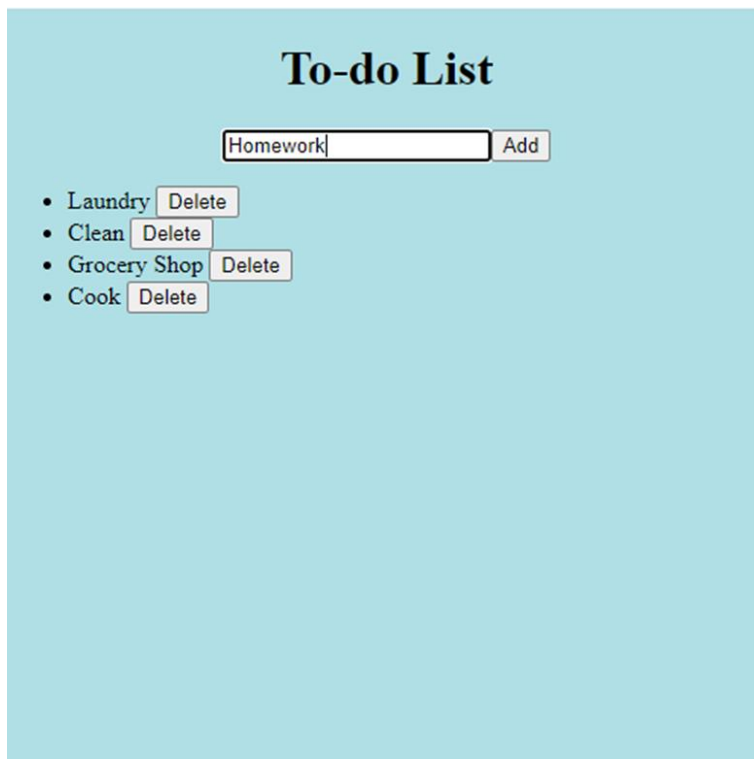
```
<!doctype HTML>
<html>
  <style>
    body {text-align: center;}
    #App {height: 500px; width: 500px; background-color: powderblue;display: inline-block;}
    ul {text-align: justify;}
  </style>
  <script src="angular.min.js"></script>
  <link rel="stylesheet" href="style.css">
    <script>
      angular.module('todoApp', [])
        .controller('todoController', function($scope) {
          $scope.activity = [];
          $scope.add = function() {
            $scope.activity.push($scope.title);
          }
          $scope.delete = function() {
```

```

        $scope.activity.splice(this.$index, 1);
    }
})
</script>
<body>
  <div id= "App">
    <h1>To-do List</h1>
    <div ng-app="todoApp" ng-controller="todoController">
      <form ng-submit="add()">
        <input ng-model="title"><button>Add</button>
      </form>
      <ul>
        <li ng-repeat="x in activity track by $index">{{ x }} <button ng-click="delete()">Delete</button>
      </li>
    </ul>
  </div>
</div>
</body>
</html>

```

Refresh the webpage. An interface which accepts items to be added and deleted should appear. Below is an example snapshot:



To-do List

Homework

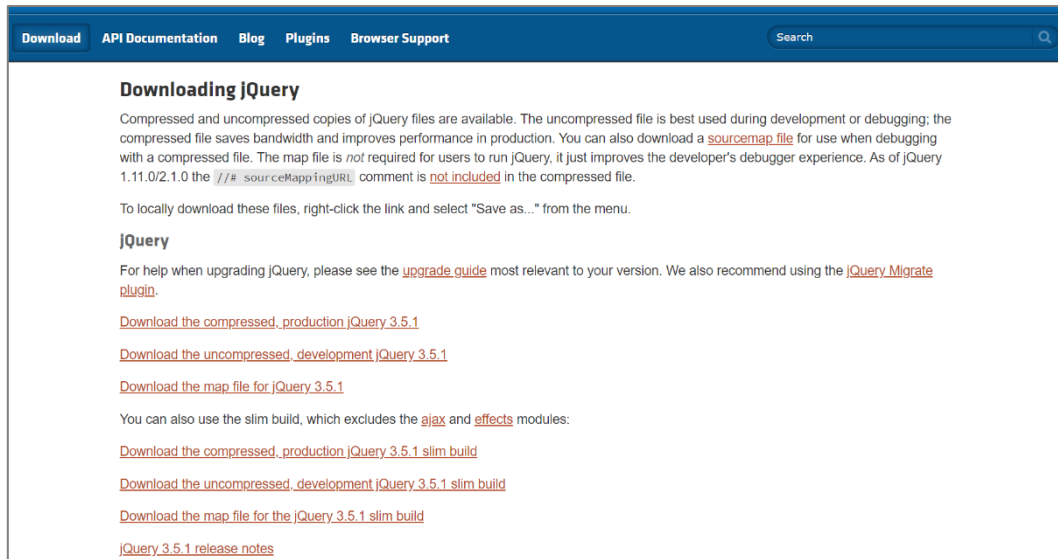
- Laundry
- Clean
- Grocery Shop
- Cook

3) AJAX (Reference: <https://www.youtube.com/watch?v=XhMGV8PzyOg>)

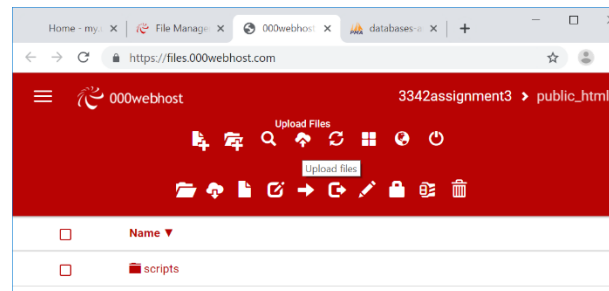
Here we will be using AJAX to autorefresh our webpage as soon as the relevant content gets updated in a DBOR table.

a) We need to download jQuery, a library of functions and methods, which includes AJAX methods.

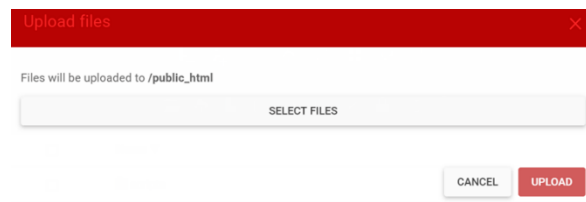
Access <https://jquery.com/download/> and right-click on **Download the compressed, production jQuery 3.5.1** and save the file as **jquery.js** on your Windows machine.



On your **000webhost.com** account, access **file manager** and click on **Upload files** icon:



Click on **SELECT FILES**, and on your Windows machine select/open **jquery.js** file downloaded earlier to upload to 000webhost.com:



b) Let's build a website to display records from **person** table of **insurance** DBOR. However, we want the website to *autorefresh* every 3 seconds to catch & display any changes occurring in **person** table. Under your *000webhost.com* account, enter the following code for *index.php*, and don't refresh the webpage as yet because we still need to make *DBpeople.php* file:

```
<?php
require_once __DIR__ . '/../required/db_connect.php';
?>
<html>
<head>
<title>Welcome</title>
</head>
<body>
Welcome to my Insurance DB </br>
===== </br>
<div id="people"></div>
    <script type="text/javascript" src="jquery.js"></script>
    <script type="text/javascript">
        $(document).ready(function() {           //once the page is ready, runs the code
            setInterval(function() {             //calls the function to be executed
                $('#people').load('DBpeople.php') //loads info from DBpeople.php into this DIV
            }, 3000);                             //repeat every 3000 milliseconds
        });
    </script>
===== </br>
</body>
</html>
```

Create and add *DBpeople.php* given below to your *000webhost.com* account. This file loads records from **person** table of **insurance** DBOR:

```
<?php
require_once __DIR__ . '/../required/db_connect.php';
?>
<?php
    if ($stmt=$mysqli->prepare("SELECT * FROM person LIMIT 100")) {
        $stmt->execute();
        $stmt->bind_result($pname,$street,$city);
        printf("Name Street City<br>");
        while ($stmt->fetch()) {
            echo $pname . " " . $street . " " . $city . "<br>";
        }
        $stmt->close();
    }
    else {
        echo "error";
        $mysqli->close();
    }
?>
```

c) Let's test our webpage using the following steps:

i) Refresh the webpage which should show the existing records of **person** table:

```
=====
Name Street City
ben zed edinburg
dolly pstreet Brownsville
gloria pstreet Brownsville
marisol zenith harlingen
puente winder edinburg
sunny zenith harlingen
zapata media edinburg
=====
```

ii) Add a record in your **person** table of **insurance** DBOR:



iii) Observe the webpage without refreshing it. In 3 seconds, the records should include the newly added record:

```
=====
Name Street City
ben zed edinburg
dolly pstreet Brownsville
gloria pstreet Brownsville
john zenith edinburg
marisol zenith harlingen
puente winder edinburg
sunny zenith harlingen
zapata media edinburg
=====
```

4) ChartJS (Reference: <https://www.youtube.com/watch?v=sE08f4iuOhA>)

As the name implies, it is a JS library used for visualizing data. It allows different types of charts/graphs including line, pie, bar and scatter. Instead of downloading the library as we have been doing previously for AngularJS and AJAX, let's learn how we can use CDN (Content Delivery Network) to accomplish making charts on our webpage.

a) Add the following *index.php* to your *000webhost.com* account (note the link to CDN... recall that we used ArcGIS service in a similar way):

```
<!doctype HTML>
<html>
    <canvas id="myChart"></canvas>
    <script src="https://cdn.jsdelivr.net/npm/chart.js@2.8.0"></script>
</html>
```

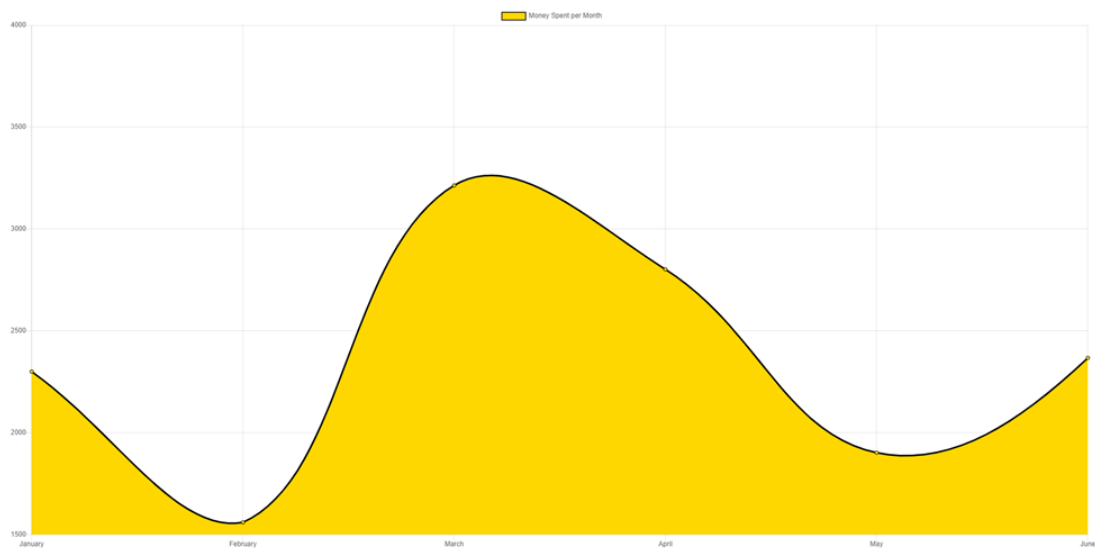
Note that a chart needs a **canvas** which the above code has. Let's define the chart's context to be 2-dimensional, and create the chart **template** for a **line type** chart with **data** (consisting of **labels** & **datasets**) and **options** (such as title and legend). This is accomplished by the JS script shown in the modified *index.php* file below:

```
<!doctype HTML>
<html>
    <canvas id="myChart"></canvas>
    <script src="https://cdn.jsdelivr.net/npm/chart.js@2.8.0"></script>
    <script>
        var ctx = document.getElementById('myChart').getContext('2d');
        var chart = new Chart(ctx, {
            type: 'line',
            data: {
                labels: [],
                datasets: [{ }]
            },
            options: {}
        });
    </script>
</html>
```

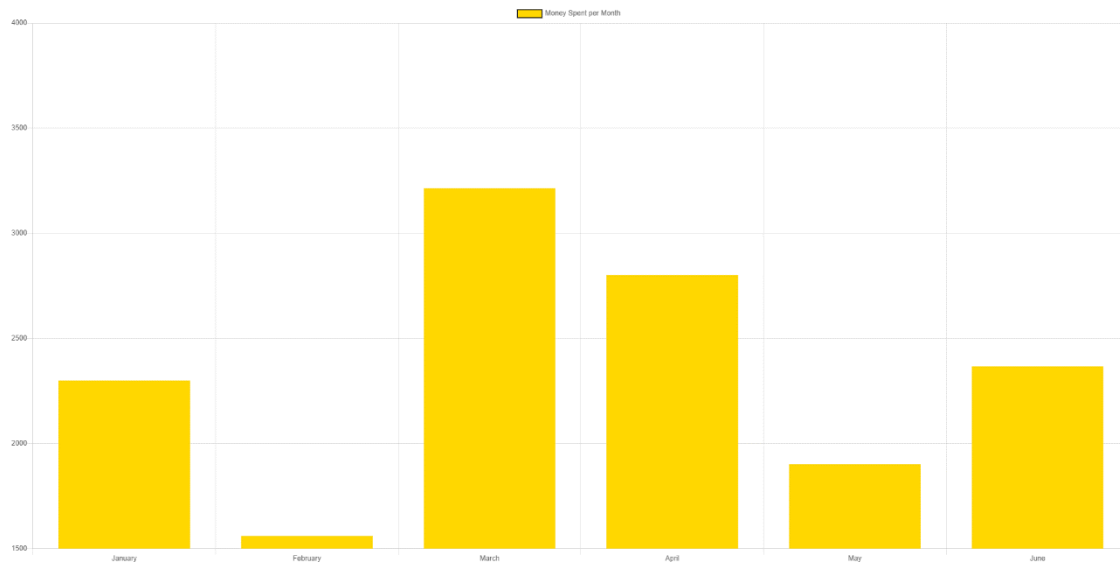
Let's populate the chart template with some real data, for example, the modified *index.php* given below graphs *money* spent over *6 months* (background color is set as gold and border color as black):

```
<!doctype HTML>
<html>
    <canvas id="myChart"></canvas>
    <script src="https://cdn.jsdelivr.net/npm/chart.js@2.8.0"></script>
    <script>
        var ctx = document.getElementById('myChart').getContext('2d');
        var chart = new Chart(ctx, {
            type: 'line',
            data: {
                labels: ['January', 'February', 'March', 'April', 'May', 'June'],
                datasets: [{label: 'Money Spent per Month',
                    backgroundColor: 'rgb(255, 215, 0)',
                    borderColor: 'rgb(0, 0, 0)',
                    data: [2300, 1560, 3214, 2802, 1902, 2367]
                }]
            },
            options: {}
        });
    </script>
</html>
```

Refresh the webpage for the above *index.php*. The result should be the chart shown below:



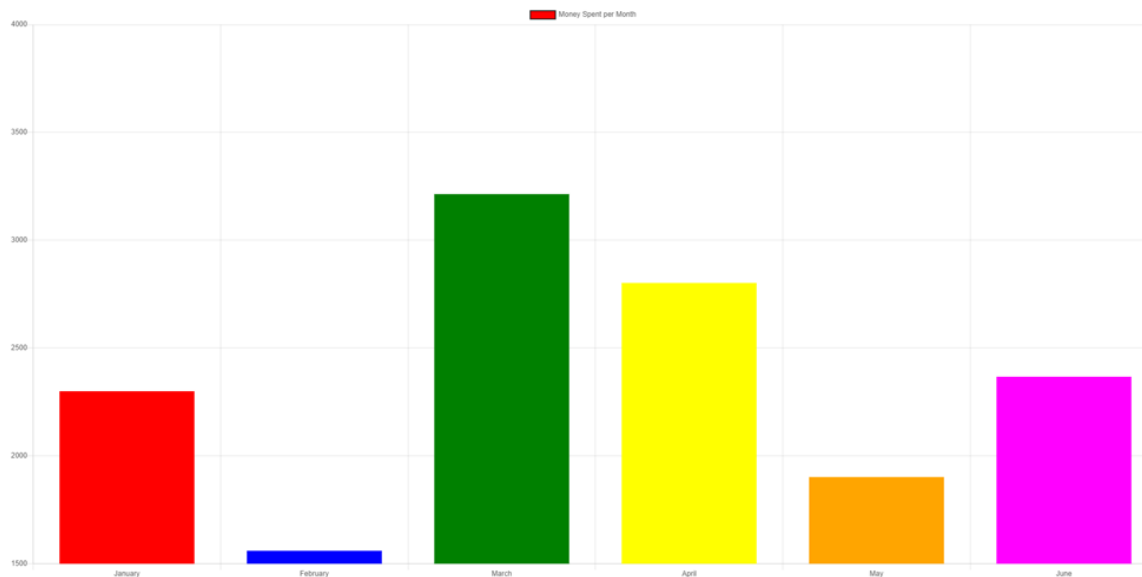
b) Change the chart type from *line* to *bar* in the above *index.php*, and refresh the webpage. The result should be the following bar chart:



One can attribute different colors to each *bar* by replacing the *backgroundColor* line in *index.php* with the following line:

```
backgroundColor: ['red', 'blue', 'green', 'yellow', 'orange', 'magenta'],
```

Refreshing the webpage with the modified *index.php* should produce the following:



c) Let's add *dynamic characteristics* to the chart, so that when the mouse is hovered over a **bar**, the border width of that **bar** is adjusted:

```
<!doctype HTML>
<html>
  <canvas id="myChart"></canvas>
  <script src="https://cdn.jsdelivr.net/npm/chart.js@2.8.0"></script>
  <script>
    var ctx = document.getElementById('myChart').getContext('2d');
    var chart = new Chart(ctx, {
      type: 'bar',
      data: {
        labels: ['January', 'February', 'March', 'April', 'May', 'June'],
        datasets: [{label: 'Money Spent per Month',
          backgroundColor: ['red', 'blue', 'green', 'yellow', 'orange', 'magenta'],
          borderColor: 'rgb(0, 0, 0)',
          data: [2300, 1560, 3214, 2802, 1902, 2367],
          borderWidth: 1,
          borderColor: 'black',
          hoverBorderWidth: 3,
          hoverBorderColor: 'black'
        }]
      },
      options: {}
    });
  </script>
</html>
```

Refresh the webpage and see if hovering of the mouse over different bars results in what is expected.

d) One can add *global* chart options. Below is an example of **globally** using *Lato* font family with font size 18 and font color as *grey*:

```
<!doctype HTML>
<html>
  <canvas id="myChart"></canvas>
  <script src="https://cdn.jsdelivr.net/npm/chart.js@2.8.0"></script>
  <script>
    var ctx = document.getElementById('myChart').getContext('2d');
    Chart.defaults.global.defaultFontFamily = 'Lato';
    Chart.defaults.global.defaultFontSize = 18;
    Chart.defaults.global.defaultFontColor = '#777';
    var chart = new Chart(ctx, {
      type: 'bar',
      data: {
        labels: ['January', 'February', 'March', 'April', 'May', 'June'],
        datasets: [{label: 'Money Spent per Month',
          backgroundColor: ['red', 'blue', 'green', 'yellow', 'orange', 'magenta'],
          borderColor: 'rgb(0, 0, 0)',
          data: [2300, 1560, 3214, 2802, 1902, 2367],
          borderWidth: 1,
          borderColor: 'black',
          hoverBorderWidth: 3,
          hoverBorderColor: 'black'
        }]
      },
      options: {}
    });
  </script>
</html>
```

Refresh the webpage and see the changes in the chart.

e) Let's add some options, such as **title** and **legend** (let's position legend to the right). Also, let's display a **pie** chart instead:

```
<!doctype HTML>
<html>
  <canvas id="myChart"></canvas>
  <script src="https://cdn.jsdelivr.net/npm/chart.js@2.8.0"></script>
  <script>
    var ctx = document.getElementById('myChart').getContext('2d');
    Chart.defaults.global.defaultFontFamily = 'Lato';
    Chart.defaults.global.defaultFontSize = 18;
    Chart.defaults.global.defaultFontColor = '#777';
    var chart = new Chart(ctx, {
      type: 'pie',
      data: {
        labels: ['January', 'February', 'March', 'April', 'May', 'June'],
        datasets: [{label: 'Money Spent per Month',
          backgroundColor: ['red', 'blue', 'green', 'yellow', 'orange', 'magenta'],
          borderColor: 'rgb(0, 0, 0)',
          data: [2300, 1560, 3214, 2802, 1902, 2367],
          borderWidth: 1,
          borderColor: 'black',
          hoverBorderWidth: 3,
          hoverBorderColor: 'black'
        }]
      },
      options: {
        title: {
          display: true,
          text: 'Money Spent per Month'
        },
        legend: {
          display: true,
          position: 'right'
        }
      }
    });
  </script>
</html>
```

Refresh the webpage and see the changes. The output should look like the following:

