SCTR's

PUNE INSTITUTE OF COMPUTER TECHNOLOGY

THIRD YEAR
INFORMATION TECHNOLOGY
(2019 COURSE)

LABORATORY MANUAL
FOR

LABORATORY PRACTICE- II
(Web Development Application)

SEMESTER - VI

[Subject code: 314458]

[Prepared By]

Mrs. Rachana A. Karnavat
Mr. Sandip R. Warhade
Mrs. Radhika S. Malpani

DEPARTMENT OF INFORMATION TECHNOLOGY

INSTITUTE VISION AND MISSION

VISION

Pune Institute of Computer Technology aspires to be the leader inhigher technical education and research of international repute.

MISSION

To be leading and most sought-after Institute of education and research in emerging engineering and technology disciplines that attracts, retains and sustains gifted individuals of significant potential.

DEPARTMENT VISION AND MISSION

VISION

The department endeavors to be recognized globally as a center of academic excellence & research in Information Technology.

MISSION

To inculcate research culture among students by imparting information technology related fundamental knowledge, recent technological trends and ethics to get recognized as globally acceptable and socially responsible professionals.

Savitribai Phule Pune University, Pune Third Year Information Technology (2019 Course) 314458: Laboratory Practice-II (Web Application Development)		
Teaching Scheme:	**Credit Scheme:**	**Examination Scheme:**
Practical (PR) : 4 hrs/week	**02 Credit**	**PR : 25 Marks** **TW : 50 Marks**

HOME

Prerequisites: Programming languages C++, Java

Course Objectives:

1. To understand basic concepts of web programming and scripting languages.
2. To learn Version Control Environment.
3. To learn front end technologies and back end technologies.
4. To understand mobile web development.
5. To comprehend web application deployment.

Course Outcomes:

On completion of the course, students will be able to–

CO1: Develop Static and Dynamic responsive website using technologies HTML, CSS, Bootstrapand AJAX.
CO2: Create Version Control Environment.
CO3: Develop an application using front end and backend technologies.
CO4: Develop mobile website using JQuery Mobile.
CO5: Deploy web application on cloud using AWS.

Guidelines for Instructor's Manual

Lab Assignments: Following is a list of suggested laboratory assignments for reference. Laboratory Instructors may design a suitable set of assignments for their respective courses at their level. Beyond curriculum assignments, the mini-project is also included as a part of laboratory work. The Inclusion of few optional assignments that are intricate and/or beyond the scope of curriculum will surely be the value addition for the students and it will satisfy the intellectuals within the group of the learners and will add to the perspective of the learners. For each laboratory assignment, it is essential for students to draw/write/generate flowchart, algorithm, test cases, mathematical model, Test data set and comparative/complexity analysis (as applicable).

Guidelines for Student's Lab Journal

Program codes with sample output of all performed assignments are to be submitted as softcopy. Use of DVD or similar media containing students programs maintained by Laboratory In-charge is highly encouraged. For reference one or two journals may be maintained with program prints in the Laboratory. As a conscious effort and little contribution towards Green IT and environment awareness, attaching printed papers as part of write-ups and program listing to journals may be avoided. Submission of journal/ term work in the form of softcopy is desirable and appreciated.

Guidelines for Lab /TW Assessment
Term work is continuous assessment that evaluates a student's progress throughout the semester. Term work assessment criteria specify the standards that must be met and the evidence that will be gathered to demonstrate the achievement of course outcomes. Categorical assessment criteria for the term work should establish unambiguous standards of achievement for each course outcome. They should describe what the learner is expected to perform in the laboratories or on the fields to show that the course outcomes have been achieved. It is recommended to conduct an internal monthly practical examination as part of continuous assessment.
Guidelines for Laboratory Conduction
Following is a list of suggested laboratory assignments for reference. Laboratory Instructors may design a suitable set of assignments for respective courses at their level. Beyond curriculum assignments and mini-project may be included as a part of laboratory work. The instructor may set multiple sets of assignments and distribute among batches of students. It is appreciated if the assignments are based on real world problems/applications. The Inclusion of few optional assignments that are intricate and/or beyond the scope of curriculum will surely be the value addition for the students and it will satisfy the intellectuals within the group of the learners and will add to the perspective of the learners. For each laboratory assignment, it is essential for students to draw/write/generate flowchart, algorithm, test cases, mathematical model, Test data set and comparative/complexity analysis (as applicable). Batch size for practical and tutorials may be as per guidelines of authority.
Guidelines for Practical Examination
Students' work will be evaluated typically based on the criteria like attentiveness, proficiency in execution of the task, regularity, punctuality, use of referencing, accuracy of language, use of supporting evidence in drawing conclusions, quality of critical thinking and similar performance measuring criteria.
List of Laboratory Assignments
Group A-(WAD)
Assignment 1 **a.** Create a responsive web page which shows the ecommerce/college/exam admin dashboard with sidebar and statistics in cards using HTML, CSS and Bootstrap. **b.** Write a JavaScript Program to get the user registration data and push to array/local storage with AJAX POST method and data list in new page.

Assignment 2

a. Create version control account on GitHub and using Git commands to create repository and push your code to GitHub.
b. Create Docker Container Environment (NVIDEIA Docker or any other).
c. Create an Angular application which will do following actions: Register User, Login User, Show User Data on Profile Component

Assignment 3

a. Create a Node.JS Application which serves a static website.
b. Create four API using Node.JS, ExpressJS and MongoDB for CURD Operations on assignment 2.C.

Assignment 4

a. Create a simple Mobile Website using jQuery Mobile.
b. Deploy/Host Your web application on AWS VPC or AWS Elastic Beanstalk. Mini Project

Develop a web application using full stack development technologies in any of the following domains:

1. Social Media
2. ecommerce
3. Restaurant
4. Medical
5. Finance
6. Education
7. Any other

Reference Books:

1. Kogent Learning Solutions Inc, Web Technologies: HTML, JAVASCRIPT, PHP, JAVA, JSP, XML and AJAX, Blackbook, Dreamtech Press, Second Edition, ISBN: 9788177228496.
2. Raymond Camden, Andy Matthews, jQuery Mobile Web Development Essentials, Packt Publishing, Second Edition, 9781782167891.
3. Steven M. Schafer, "HTML, XHTML and CSS", Wiley India Edition, Fourth Edition, 978- 81-265-1635-3
4. Dr.HirenJoshi, Web Technology and Application Development, DreamTech, First,ISBN:978-93-5004-088-1
5. Steven M. Schafer, "HTML, XHTML and CSS", Wiley India Edition, Fourth Edition, 978- 81-265-1635-3
6. Ivan Bayross,"Web Enabled Commercial Application Development Using HTML, JavaScript, DHTML and PHP,BPB Publications,4th Edition,ISBN:978-8183330084.
7. Brain Fling, Mobile Design and Development, O'REILLY, First Edition, ISBN: 13:978-81- 8404-817-
8. Adam Bretz & Colin J Ihrig, Full Stack Javascript Development with MEAN, SPD, First Edition, ISBN:978-0992461256.

- Books / E- Learning References

1. https://www.meanacademy.in/web-technologies
2. https://www.meanacademy.in/angular
3. https://www.meanacademy.in/mongodb
4. https://www.meanacademy.in/nodejs
5. https://www.meanacademy.in/aws

Assignment 1a

Title: Web Page Design

Problem Statement: Create a responsive web page which shows the ecommerce/ college/ exam admin dashboard with sidebar and statistics in cards using HTML, CSS and Bootstrap.

Objective: Apply HTML, CSS and Bootstrap classes and demonstrate a web page that is responsive.

S/W Packages and H/W apparatus used:

Linux OS: Ubuntu/Windows, VSCode editor.

PC with the configuration as Pentium IV 1.7 GHz. 128M.B RAM, 40 G.B HDD, 15''Color Monitor, Keyboard, Mouse.

References:

1. Steven M. Schafer, "HTML, XHTML and CSS", Wiley India Edition, Fourth Edition,978- 81-265-1635-3

Theory:

1. HTML

- HTML stands for Hypertext Markup Language. HTML is rendered on a Web Browser. HTML has a set of elements that describes the structure of a Web page.

a. HTML Elements

- The HTML **element** comprises of a start tag, the content and the end tag:
- <tagname>Content</tagname>
- The main HTML element is <html> … </html>. All the HTML will be enclosed within it.

b. HTML Boilerplate

```
<!DOCTYPE html>
```

```
<html lang="en">
  <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>HTML 5 Boilerplate</title>
      <link rel="stylesheet" href="style.css">
  </head>
  <body>
      Other HTML elements are per choice of User
      <script src="index.js"></script>
  </body>
</html>
```

- HTML Boilerplate is the HTML Standard structure using which the user can create Web Pages.
- <head> and <body> are the two main tags. <head> is where the meta information, title of Web Page and CSS files can be embedded. <body> holds the user's code and any JavaScript files can also come below.

c. Block-Level vs. Inline Elements

- **The Block-level elements** occupy the entire width of the Web Page (height is automatic as per the content within element, if not explicitly mentioned).
- Common Block-Level Elements are , <section>, <form>, <main>, <table>, <h1> to <h6>, <nav>, <header>, <p>, <hr>, <ol>, <ul>, <li> etc.
- **The inline elements** take up the width on the browser as much its own width only.
- Inline Elements can sit next each other in a single row.
- Inline Elements can also be nested within the Block-level elements
- Common Inline Elements are <a>, <b>, <i>, <img>, , <input>, <label>, <button> etc.

d. HTML Attributes

- An attribute is a key-value pair written within an element.
- An attribute is used to add more specific details to an element.
- For e.g. You may use a "style" attribute in element to change the color of text or add a background …. and much more
- **Case Study: The <a> tag**
  - The <a> or anchor tag is used to add a hyperlink.
  - In this slide we shall see the different attributes associated with <a> tag.

1. **href** – used to mention the link
   i. e.g. CLICK ME
2. **target** – indicates where to open the linked document.
   i. e.g. CLICK ME

   The web page is opened in same tab

   ii. e.g. CLICK ME

   The web page is opened in a new window or tab

3. **style** – used to add styles
   1. e.g. CLICK ME

2. Cascading Style Sheets (CSS)

- CSS, also known as Cascading Style Sheets is used to style or achieve the desired look and feel of the web page.
- CSS has a wide range of properties that can facilitate the user from changing the color of text to applying animations.

a. CSS Syntax

selector{ property:value; property:value; }

Selector could be element selector or id selector or class selector or universal selector etc..

A property may have various values.

e.g. if property is “font-style”

It can have 2 values: normal, italic

b. Types of CSS

- **Inline CSS** – It is written within the HTML element itself using the style attribute.

  e.g. I am a div element

- **Internal CSS** – It is written within the HTML document using <style> tag.
- **External CSS** – It is written in a separate CSS file. The CSS can then be linked to the HTML file. CSS file is saved with .css extension

c. CSS Selectors

- CSS Selectors are the various ways for writing a CSS.
- The common selectors are:

1. Element Selector

The element selector uses the name of HTML tag.

e.g.

HTML

```
<div>I am div 1</div>
<div>I am div 2</div>
```

CSS

```
div
{
width: 100px;
border: 1px solid grey;
}
```

2. ID Selector

The id selector uses the unique id of HTML tag to apply CSS.

e.g.

HTML

```
<div id="div1">I am div 1</div>
```

CSS

```
#div1
{
width: 100px;
border: 1px dashed orange;
}
```

3. Class Selector

The class selector uses the class of HTML tag to apply CSS.

e.g. **HTML**

```
<div class="mydiv">I am div 1</div>
```

CSS

```
.mydiv{
width: 50%;
border: 1px dashed orange;}
```

4. Universal Selector

The universal selector applies CSS to each HTML element of the Web Page. Common styling like font-family, page width can be specified using universal selector

e.g. **CSS**

```
*{
font-family: sans-serif;
font-weight: 700;
text-align: justify;
}
```

5. Grouping Selector

In grouping selector all the elements which require same style can be grouped together and a common set CSS rules can be applied to them.

e.g. **HTML**

```
<div>I am a div</div>
<p>I am paragraph 1</p>
<p id="p1">I am paragraph 2</p>
```

CSS

```
div, #p1
{
text-align: center;
color: red;
}
```

6. Combination Selector

When there is a relationship between HTML elements, the combination selector can be used to apply rules.

There are four different combination selectors in CSS:

- descendant selector (space)
- child selector (>)
- adjacent sibling selector (+)
- general sibling selector (~)
-

7. Pseudo-class Selector

A pseudo-class is used to define a specific state of an HTML element.

:link, :hover

```
<p><a href="#" target="_blank">Click ME</a></p>
e.g.
a:link {
  color: red;
```

```
}
a:hover {
  color: hotpink;
}
```

:checked

```
<form action="#">
<input type="radio" value="male" name="gender"> Male<br>
<input type="radio" value="female" name="gender"> Female<br>
</form>
input:checked {
  outline: 2px solid deeppink;
}
```

3. Bootstrap

- Bootstrap is a light-weight library of CSS.
- Bootstrap 5 is common used version as of now.
- It has a wide range of class that works perfectly well on all browsers.
- These classes have in-built CSS associated with them.
- e.g. class="card" automatically applies the following CSS:

```
position: relative;
display: flex;
flex-direction: column;
min-width: 0;
```

a. Bootstrap Grid System

- Bootstrap Grid allows 12 columns in a row on the web page.
- The Bootstrap 5 grid system has six classes:
    - .col- (extra small devices - screen width less than 576px)
    - .col-sm- (small devices - screen width equal to or greater than 576px)

- .col-md- (medium devices - screen width equal to or greater than 768px)
- .col-lg- (large devices - screen width equal to or greater than 992px)
- .col-xl- (xlarge devices - screen width equal to or greater than 1200px)
- .col-xxl- (xxlarge devices - screen width equal to or greater than 1400px)

-

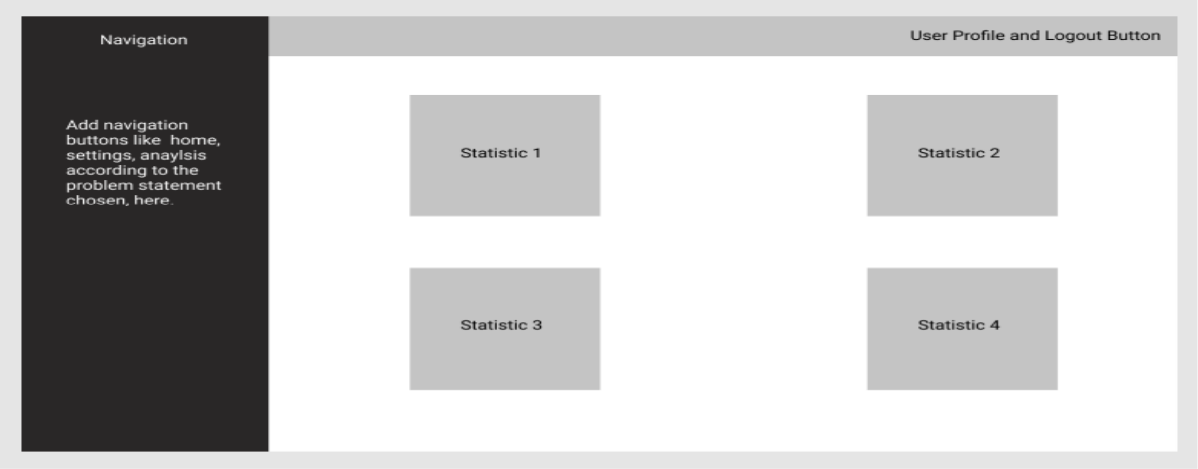
- The first * may hold one of the values: xs, sm, md, lg, xl, xxl
- The second * may hold a number (the summation of all numbers in a row must be less than or equal to 12)

b. How to add Bootstrap CDN links to your code

```
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/css/bootstrap.min.css" rel="stylesheet">
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/js/bootstrap.bundle.min.js"></script>
 </head>
```

Source: getbootstrap.com

Implementation: Sample Dashboard



Sample HTML tags to be used

1. p
2. body
3. All table tags
4. All heading tags
5. a
6. div
7. title
8. head
9. li
10. ol
11. ul
12. html
13. br
14. hr
15. img
16. link
17. header etc…

Sample Bootstrap classes to be used

1. container
2. container-fluid
3. card
4. card-body
5. card-title
6. card-text
7. btn
8. btn-primary (with variations) etc…

Conclusion: Thus, we have applied HTML, CSS and Bootstrap classes and demonstrated a web page that is responsive.

Assignment 1b

Title: JavaScript AJAX Implementation

Problem Statement: Write a JavaScript Program to get the user registration data and push to array/local storage with AJAX POST method and data list in new page.

Objective: To understand the working of JS, AJAX, XmlHttpRequest by accepting the User registration data.

S/W Packages and H/W apparatus used:

Linux OS: Ubuntu/Windows, VSCode editor.

PC with the configuration as Pentium IV 1.7 GHz. 128M.B RAM, 40 G.B HDD, 15''Color Monitor, Keyboard, Mouse.

References:

1. Ivan Bayross,"Web Enabled Commercial Application Development Using HTML, JavaScript, DHTML and PHP, BPB Publications,4th Edition,ISBN:978-8183330084.

Theory:

1. **HTML Form**

An HTML form is collection of

- Textbox (to fill info like Name, Roll No etc)
- Textarea (to fill info like Address or lengthy text)
- Select (to select value/s from a dropdown list)
- Radio Button (to make one selection like Yes/No, Male/Female)
- Check Box (to select one or more options from limited choices)
- Submit Button (to submit the filled form)

..and many more elements all contributing to the collection of info from user.

The **<form> tag** is the starting point for any HTML form

The attributes used within <form> are:

1. action: <form action="welcome.html">

After the form is submitted, filename/URL mentioned in "action" shall hold the output/processing.

2. method: <form action="welcome.html" method= "get/post">

The data could be sent as URL variables (using method="get") or as HTTP post transaction (using method="post").

2. JavaScript

JavaScript is a front-end scripting language.

JavaScript can be added to the HTML file in two ways:

- Internal JavaScript
- External JavaScript

Internal JavaScript: JS code is directly added to the HTML file using the <script> & </script> tags. The <script> tag can either be placed inside the <head> or the <body> tag according to the programming requirement.

External JavaScript: A separate file with a **.js** extension is prepared, where the only the JavaScript code is written. This JS file can be embedded in HTML file by using <script src="file_name.js"> tag, and this <script> may reside inside the <head> or <body> tag of the HTML file.

JavaScript Used to:

It is primarily used to develop websites and web-based applications. JavaScript is a front-end scripting language.

- **Create Interactive/Dynamic Websites:** JavaScript is used to make web pages dynamic and interactive. Using JavaScript the DOM can be altered on the fly based on the inputs or activities of the user.
- **Building Applications:** JavaScript can be employed to make web and mobile applications. To build web and mobile apps, the most popular JavaScript frameworks like – ReactJS, React Native, Node.js are used.
- **Web Servers:** A robust server applications using JavaScript can be created. To be precise the JavaScript frameworks like Node.js and Express.js are used to build these servers.

3. XmlHttpRequest

XMLHTTPRequest object is an API used to fetch data from the server. XMLHTTPRequest is used in Ajax programming. It can retrieve any format of data such as JSON, XML, text, etc. It requests for data in the background without the page refresh and loads response on the page. An object of XMLHTTPRequest is used for asynchronous communication between client and server.

Properties of XMLHttpRequest Object:

- **onload:** When the request is served and the response is ready, it defines a function to be called.

- **onreadystatechange**: A function is called whenever the readyState property changes.

- **readyState**: It holds the current status of the XMLHttpRequest. There are five states of a request:
    - **readyState= 0**: It represents the Request not initialized.
    - **readyState= 1**: Establishment of server connection.
    - **readyState= 2**: Request has been received
    - **readyState= 3**: During the time of processing the request
    - **readyState= 4**: Response is ready after finishing the request
- **responseText**: It returns the response data received by the request in the form of a string.
- **responseXML**: It returns the response data received by the request in XML format.
- **status**: It returns the status number of the request. (i.e. 200 and 404 for OK and NOT FOUND respectively).
- **statusText**: It returns the status text in form of a string. (i.e. OK and NOT FOUND for 200 and 404 respectively).

4. Ajax

AJAX is abbreviation for Asynchronous JavaScript and XML. AJAX also hits the request to the server and loads the data on the HTML without the page refresh.

Examples of applications using AJAX: Gmail, Google Maps, Youtube, and Facebook tabs.

With the jQuery AJAX methods, one can request text, HTML, XML, or JSON from a remote server using HTTP Get and/or HTTP Post

The parameters specify one or more name/value pairs for the AJAX request.

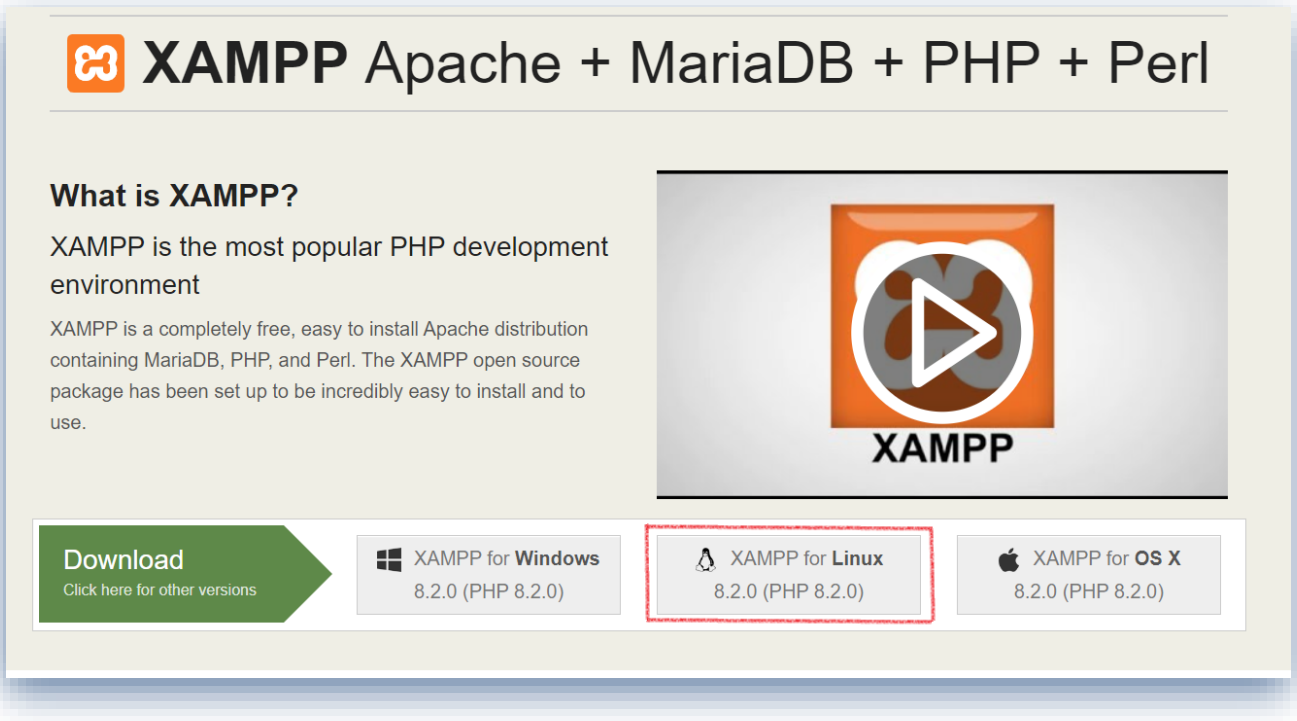
Possible names/values in the table below:

Name	**Value/Description**
async	A Boolean value indicating whether the request should be handled asynchronous or not. Default is true
contentType	The content type used when sending data to the server. Default is: "application/x-www-form-urlencoded"
data	Specifies data to be sent to the server
dataType	The data type expected of the server response.
error(*xhr*,*status*,*error*)	A function to run if the request fails.
success(*result*,*status*,*xhr*)	A function to be run when the request succeeds
type	Specifies the type of request. (GET or POST)
url	Specifies the URL to send the request to. Default is the current page

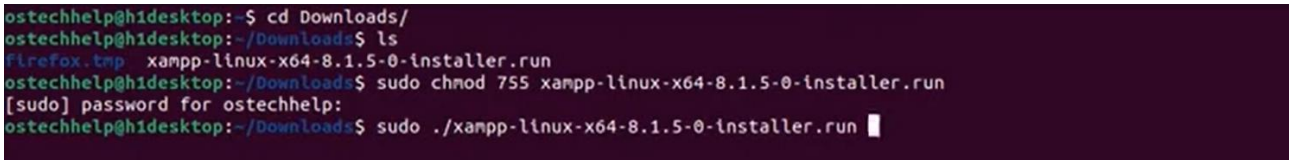
Steps to implement the program:

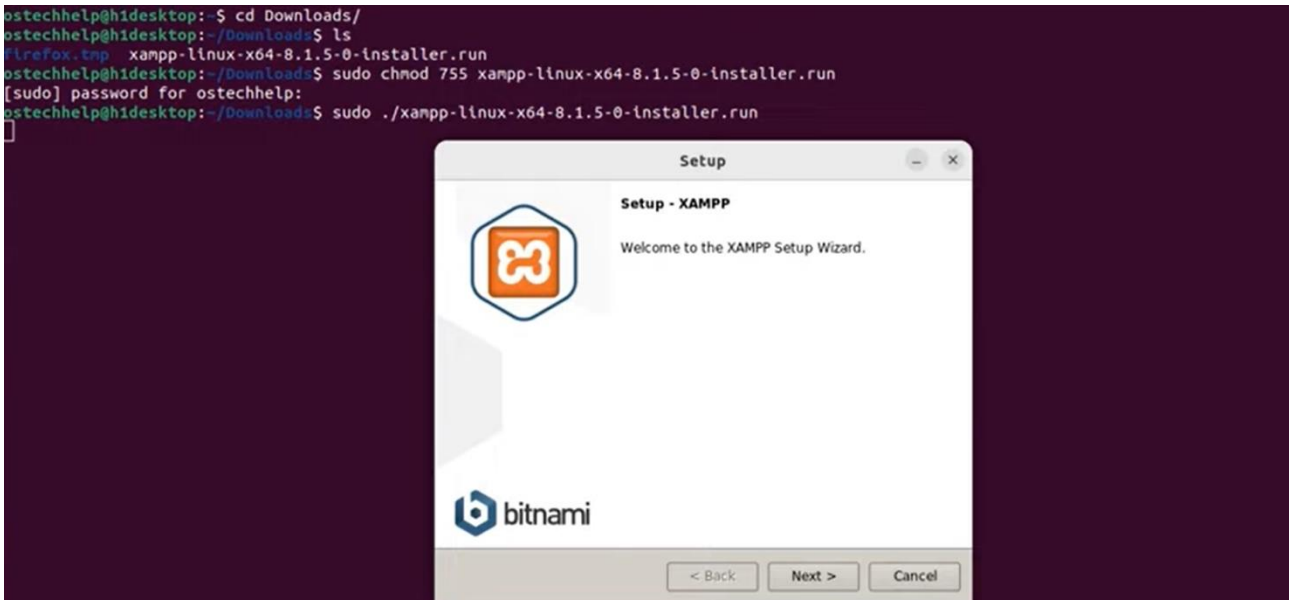
1. Install a web-server, here steps for installing XAMPP server on Ubuntu are mentioned

   Visit https://www.apachefriends.org/ and click on Ubuntu setup as highlighted below

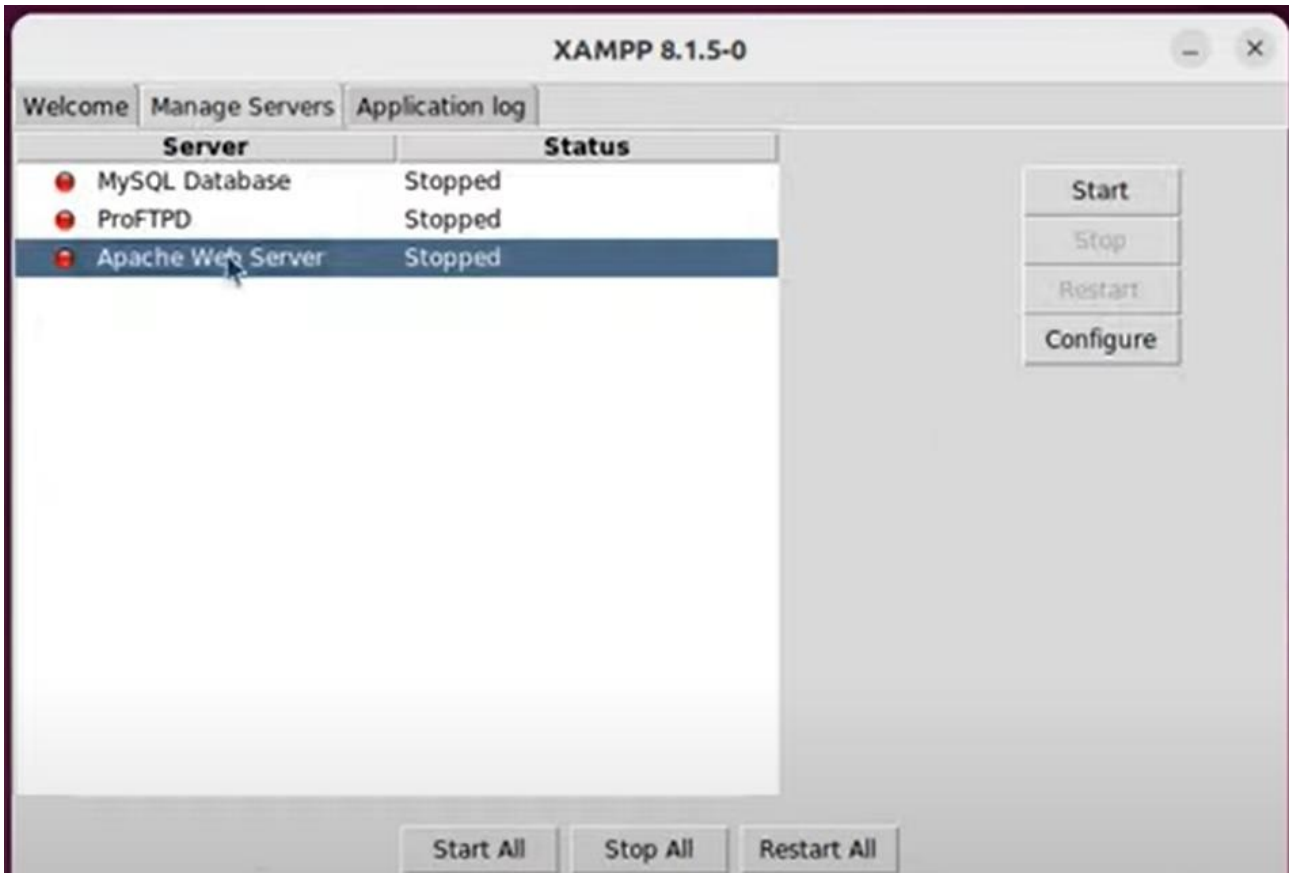


2. Assuming that the setup is in Downloads, follow the steps ahead





Go on clicking next. After the setup is installed a following window will appear, click “Manage Servers” -> click on Apache Web Server -> click on Start



Once above setting is done, again go to terminal and use following commands

Here “mysite” is name of the folder and it will be launched in VSCode.

Suppose if you create “index.html” file in “mysite” folder, the on browser use the URL as:

localhost/mysite/index.html

Sample Code:

1. **Using XmlHttpRequest**

index.html

```
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.3/jquery.min.js"></script>
    <script src="https://code.jquery.com/jquery-3.6.3.js" integrity="sha256-
nQLuAZGRRcILA+6dMBOvcRh5Pe310sBpanc6+QBmyVM="
crossorigin="anonymous"></script>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
rel="stylesheet">
    <title>Assignment 1b Code</title>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/js/bootstrap.bundle.min.js"></script>
    <style>
      form, h1{
        width:30%;
        margin: 5% auto;
      }
      input{
        margin: 1%;
      }
    </style>
  </head>
  <body>
    <h1>USER REGISTRATION</h1>
```

```
        <form action="" method="POST">
         <label>First Name</label>
            <input type="text" name="fullName" id="fullName" placeholder="Name Surname
Lastname" pattern="[A-Za-z ]*" class="form-control" required>
         <label>Email</label>
            <input type="email" name="userEmail" id="userEmail" class="form-control" required>
         <label>User Name</label>
            <input type="text" name="userName" id="userName" class="form-control" required>
         <label>Password</label>
            <input type="password" name="userPassword" id="userPassword" class="form-control"
required>
            <input type="button" onclick="submitForm()" name="submit" value="SUBMIT
DETAILS" id="submit" class="btn btn-primary w-100">
      </form>
      <!-- <div id="outputs"></div> -->
      <script>
         function submitForm(){
            var name = $('input[name=fullName]').val();
            var email = $('input[name=userEmail]').val();
            var username = $('input[name=userName]').val();
            var password = $('input[name=userPassword]').val();

            var data = 'name=' + name + '&email=' + email + '&username=' + username;

            xhr = new XMLHttpRequest();
            xhr.open('POST','output.php',true);
            xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
```

```
            xhr.onload = function(){
              //document.getElementById('outputs').innerHTML = xhr.responseText;
              var new_window = window.open(null, '','_blank');
              new_window.document.write(xhr.responseText);
            }
               xhr.send(data);          }
      </script>
   </body>
</html>
```

Output.php

```
<h1>With XmlHttpRequest</h1>
<?php
        $name = $_POST['name'];
        $email = $_POST['email'];
        $username = $_POST['username'];
        echo "<div> Hi,".$name."</div>";
        echo "<div>Your email is: ".$email."</div>";
        echo "<div>Your username is: ".$username."</div>";
?>
```

---

2. Using $.ajax() **index.html**

```
<html>
   <head>
      <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.3/jquery.min.js"></script>
```

```
        <script src="https://code.jquery.com/jquery-3.6.3.js" integrity="sha256-
    nQLuAZGRRcILA+6dMBOvcRh5Pe310sBpanc6+QBmyVM="
    crossorigin="anonymous"></script>
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
    rel="stylesheet">
        <title>Assignment 1b Code</title>
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
    alpha1/dist/js/bootstrap.bundle.min.js"></script>
        <style>
            form, h1{
                width:30%;
                margin: 5% auto;
            }
            input{
                margin: 1%;          }
        </style>


    </head>
    <body>
        <h1>USER REGISTRATION</h1>
          <form action="" method="POST">
            <label>First Name</label>
                <input type="text" name="fullName" id="fullName" placeholder="Name Surname
    Lastname" pattern="[A-Za-z ]*" class="form-control" required>
            <label>Email</label>
                <input type="email" name="userEmail" id="userEmail" class="form-control" required>
            <label>User Name</label>
```

```
            <input type="text" name="userName" id="userName" class="form-control" required>
        <label>Password</label>
            <input type="password" name="userPassword" id="userPassword" class="form-control"
required>
            <input type="button" onclick="submitForm()" name="submit" value="SUBMIT
DETAILS" id="submit" class="btn btn-primary w-100">



</form>
    <!-- <div id="outputs"></div> -->
    <script>
        function submitForm(){
            var name = $('input[name=fullName]').val();
            var email = $('input[name=userEmail]').val();
            var username = $('input[name=userName]').val();
            var password = $('input[name=userPassword]').val();

            var formData = 'name=' + name + '&email=' + email + '&username=' + username;




  $.ajax({
                type: "POST",
                url: "output.php",
                data: formData,
                }).done(function (response) {
```

```
                //$("#outputs").html(response);
                var new_window = window.open(null, '','_blank');
                new_window.document.write(response);
            });
        }
    </script>
</body>
</html>
```

Output.php

```
<h1>With Ajax</h1>
<?php
    $name = $_POST['name'];
    $email = $_POST['email'];
    $username = $_POST['username'];
    echo "<div> Hi,".$name."</div>";
    echo "<div>Your email is: ".$email."</div>";
    echo "<div>Your username is: ".$username."</div>";
?>
```

Conclusion: Thus, we have implemented a JavaScript Program to get the user registration data and using AJAX POST method we have listed data on new page.

Assignment 2 a

Title: Create version control account on GitHub, Docker Container Environment and Angular application

Problem Statement:

a. Create version control account on GitHub and using Git commands to create repository and push your code to GitHub.

b. Create Docker Container Environment (NVIDEIA Docker or any other).

c. Create an Angular application which will do following actions: Register User, Login User, Show User Data on Profile Component

Objective: Create repositories on GitHub, Docker container environment and Angular application.

S/W Packages and H/W apparatus used:

Linux OS: Ubuntu/Windows, VSCode editor.

PC with the configuration as Pentium IV 1.7 GHz. 128M.B RAM, 40 G.B HDD, 15''Color Monitor, Keyboard, Mouse.

References:

1. Steven M. Schafer, "HTML, XHTML and CSS", Wiley India Edition, Fourth Edition,978- 81-265-1635-3

Theory:

a) Create version control account on GitHub and using Git commands to create repository and push your code to GitHub

Git is a version control system. It helps you keep track of code changes and it is used to collaborate on code.

Git commands: git –version

git version 2.30.2.windows.1

Git and GitHub are different things.

Git is a popular version control system. It was created by Linus Torvalds in 2005, and has been maintained by Junio Hamano since then.

It is used for:

- Tracking code changes
- Tracking who made changes
- Coding collaboration

What does Git do?

- Manage projects with **Repositories**
- **Clone** a project to work on a local copy
- Control and track changes with **Staging** and **Committing**
- **Branch** and **Merge** to allow for work on different parts and versions of a project
- **Pull** the latest version of the project to a local copy
- **Push** local updates to the main project

Working with Git

- Initialize Git on a folder, making it a **Repository**
- Git now creates a hidden folder to keep track of changes in that folder
- When a file is changed, added or deleted, it is considered **modified**
- You select the modified files you want to **Stage**
- The **Staged** files are **Committed**, which prompts Git to store a **permanent** snapshot of the files
- Git allows you to see the full history of every commit.
- You can revert back to any previous commit.
- Git does not store a separate copy of every file in every commit, but keeps track of changes made in each commit!

What is GitHub?

- Git is not the same as GitHub.
- GitHub makes tools that use Git.
- GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.

Steps to create version control account on GitHub and using Git commands to create repository and push your code to GitHub.

- Sign up for a GitHub account
- Install Git
- Create a new repository

- Initialize a Git repository
- Add files to the repository
- Commit changes
- Link your local repository to your GitHub repository
- Push changes to GitHub
- Using these steps, you can easily set up version control for your projects, keep track of changes to your code, and collaborate with others.

Steps to Push and PULL version control repository to GitHub

Step No	Command	Description
1	Git Installation	Download Git from the website: https://www.git-scm.com/
2	Command line >git –version	If Git is installed, it should show something like git version X.Y
3	git config --global user.name "w3schools-test" git config --global user.email "test@w3schools.com"	Configure Git Change the user name and e-mail address to your own
4	mkdir myproject cd myproject	*Creating Git Folder*
5	git init	*Initialize Git* *Initialized empty Git repository in /Users/user/myproject/.git/*
6	git status	*To check the status*
7	git add index.html	*Add file to staging environment*
8	git add --all	*add all files in the current directory to the Staging Environment:*
9	git commit -m "First release of Hello World!"	*The committ command performs a commit, and the -m "message" adds a message.*
10	git commit -a -m "Updated index.html with a new line"	*Skips staging environment*

11	git log	*To view the history of commits for a repository, you can use the log command*
12	git *command* -help	*See all the available options for the specific command*
13	git help --all	*See all possible commands*
14	git commit -help	*See help for specific command*
15	git branch hello-world-images	*a branch is a new/separate version of the main repository. This command creates a new branch hello-world-images*
16	git checkout hello-world-images	*checkout is the command used to check out/ move to a branch*
17	git checkout master	*Used to switch between branches*
18	https://github.com/	*Create a new account on github*
19		*Create a Repository on GitHub*
20	git remote add origin https://github.com/w3schools-test/hello-world.git	*Push Local Repository to GitHub*
21	git push --set-upstream origin master	*push master branch to the origin url,*
22		*go back into GitHub and see that the repository has been updated:*
23	git fetch origin	*fetch gets all the change history of a tracked branch/repo*
24	git merge origin/master	*merge combines the current branch, with a specified branch.*
25	git pull origin	*pull is a combination of fetch and merge* *It is used to pull all changes from a remote repository into the branch you are working on.*

Detailed Steps:

- Sign up for a GitHub account: Go to the GitHub website and sign up for an account by providing your email, username, and password. You can use either the free version or the paid version of GitHub, depending on your needs.

- Install Git: Download and install Git on your local machine if you haven't already. Git is a free and open-source version control system that allows you to manage your code locally and push it to GitHub.

- Create a new repository: Log in to your GitHub account and click on the "New" button to create a new repository. Give it a name and description, and choose whether it will be public or private.

- Initialize a Git repository: In your local project directory, run the command "git init" to initialize a Git repository. This will create a hidden .git directory in your project directory, which will be used to track changes to your code.

- Add files to the repository: Use the command "git add ." to add all the files in your local project directory to the Git repository. Alternatively, you can add individual files by using the command "git add filename".

- Commit changes: Use the command "git commit -m 'commit message'" to commit your changes to the repository. The commit message should be a brief description of the changes you made.

- Link your local repository to your GitHub repository: Run the command "git remote add origin https://github.com/your-username/repository-name.git" to link your local Git repository to the GitHub repository you created in step 3.

- Push changes to GitHub: Use the command "git push -u origin master" to push the changes in your local repository to the GitHub repository. This will upload your code to GitHub and make it available for others to see.

Once you've completed these steps, you can continue to use Git and GitHub to manage your code changes and collaborate with others.

Conclusion:

Git and GitHub provide a powerful and flexible platform for version control and collaborative software development. By learning these basic steps, you can start to use Git and GitHub for your own projects and join a global community of developers who are using these tools to build innovative software solutions.

Assignment 2 b

Create Docker Container Environment (NVIDEIA Docker or any other).

DOCKER:

Docker is a popular containerization platform that enables developers to package their applications and their dependencies into a container. These containers can then be deployed on any system with Docker installed, making it a highly portable and flexible solution for deploying applications.

NVIDIA Docker is an extension of the Docker platform that allows developers to run GPU-accelerated applications inside Docker containers. This is particularly useful for machine learning and data science applications, as these often require access to specialized hardware like GPUs.

FEATURES OF DOCKER

- Docker has the ability to reduce the size of development by providing a smaller footprint of the operating system via containers.
- With containers, it becomes easier for teams across different units, such as development, QA and Operations to work seamlessly across applications.
- You can deploy Docker containers anywhere, on any physical and virtual machines and even on the cloud.
- Since Docker containers are pretty lightweight, they are very easily scalable.

COMPONENTS:

Docker has the following components

- Docker for Mac – It allows one to run Docker containers on the Mac OS.
- Docker for Linux - It allows one to run Docker containers on the Linux OS.
- Docker for Windows - It allows one to run Docker containers on the Windows OS.
- Docker Engine – It is used for building Docker images and creating Docker containers.
- Docker Hub – This is the registry which is used to host various Docker images.
- Docker Compose – This is used to define applications using multiple Docker containers.

DOCKER – INSTALLING DOCKER ON LINUX

To install Docker, we need to follow the steps given below.

Step 1:

Before installing Docker, you first have to ensure that you have the right Linux kernel version running. Docker is only designed to run on Linux kernel version 3.8 and higher.

We can do this by running the following command:

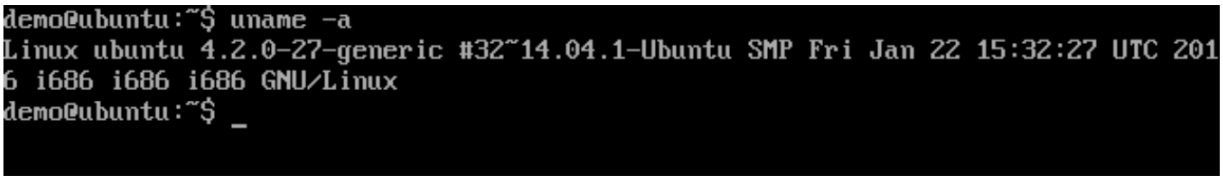
uname This method returns the system information about the Linux system

Example

```
uname -a
```

Output

When we run above command, we will get the following result:



Step 2: You need to update the OS with the latest packages, which can be done via the following command:

```
apt-get
```

This method installs packages from the Internet on to the Linux system.

Syntax

```
sudo apt-get update
```

Output

When we run the above command, we will get the following result:

```
Hit http://us.archive.ubuntu.com trusty-backports/universe Sources
Hit http://us.archive.ubuntu.com trusty-backports/multiverse Sources
Hit http://us.archive.ubuntu.com trusty-backports/main i386 Packages
Hit http://us.archive.ubuntu.com trusty-backports/restricted i386 Packages
Hit http://us.archive.ubuntu.com trusty-backports/universe i386 Packages
Hit http://us.archive.ubuntu.com trusty-backports/multiverse i386 Packages
Hit http://us.archive.ubuntu.com trusty-backports/main Translation-en
Hit http://us.archive.ubuntu.com trusty-backports/multiverse Translation-en
Hit http://us.archive.ubuntu.com trusty-backports/restricted Translation-en
Hit http://us.archive.ubuntu.com trusty-backports/universe Translation-en
Hit http://us.archive.ubuntu.com trusty Release
Hit http://us.archive.ubuntu.com trusty/main Sources
Hit http://us.archive.ubuntu.com trusty/restricted Sources
Hit http://us.archive.ubuntu.com trusty/universe Sources
Hit http://us.archive.ubuntu.com trusty/multiverse Sources
Hit http://us.archive.ubuntu.com trusty/main i386 Packages
Hit http://us.archive.ubuntu.com trusty/restricted i386 Packages
Hit http://us.archive.ubuntu.com trusty/universe i386 Packages
Hit http://us.archive.ubuntu.com trusty/multiverse i386 Packages
Hit http://us.archive.ubuntu.com trusty/main Translation-en
Hit http://us.archive.ubuntu.com trusty/multiverse Translation-en
Hit http://us.archive.ubuntu.com trusty/restricted Translation-en
Hit http://us.archive.ubuntu.com trusty/universe Translation-en
Ign http://us.archive.ubuntu.com trusty/main Translation-en_US
Ign http://us.archive.ubuntu.com trusty/multiverse Translation-en_US
Ign http://us.archive.ubuntu.com trusty/restricted Translation-en_US
Ign http://us.archive.ubuntu.com trusty/universe Translation-en_US
Fetched 3,906 kB in 21s (184 kB/s)
Reading package lists... Done
demo@ubuntu:~$
```

This command will connect to the internet and download the latest system packages for Ubuntu.

Step 3:

The next step is to install the necessary certificates that will be required to work with the Docker site later on to download the necessary Docker packages. It can be done with the following command:

```
sudo apt-get install apt-transport-https ca-certificates
```

```
demo@ubuntudemo:~$ sudo apt-get install apt-transport-https ca-certificates
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be upgraded:
  apt-transport-https ca-certificates
2 upgraded, 0 newly installed, 0 to remove and 105 not upgraded.
Need to get 215 kB of archives.
After this operation, 8,192 B disk space will be freed.
Get:1 http://us.archive.ubuntu.com/ubuntu/ trusty-updates/main apt-transport-
ps amd64 1.0.1ubuntu2.15 [25.0 kB]
Get:2 http://us.archive.ubuntu.com/ubuntu/ trusty-updates/main ca-certificate
ll 20160104ubuntu0.14.04.1 [190 kB]
Fetched 215 kB in 1s (152 kB/s)
Preconfiguring packages ...
(Reading database ... 57694 files and directories currently installed.)
Preparing to unpack .../apt-transport-https_1.0.1ubuntu2.15_amd64.deb ...
Unpacking apt-transport-https (1.0.1ubuntu2.15) over (1.0.1ubuntu2.11) ...
Preparing to unpack .../ca-certificates_20160104ubuntu0.14.04.1_all.deb ...
Unpacking ca-certificates (20160104ubuntu0.14.04.1) over (20141019ubuntu0.14.
1) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Setting up apt-transport-https (1.0.1ubuntu2.15) ...
Setting up ca-certificates (20160104ubuntu0.14.04.1) ...
Processing triggers for ca-certificates (20160104ubuntu0.14.04.1) ...
Updating certificates in /etc/ssl/certs... 19 added, 19 removed; done.
Running hooks in /etc/ca-certificates/update.d....done.
demo@ubuntudemo:~$
```

Step 4: The next step is to add the new GPG key. This key is required to ensure that al
is encrypted when downloading the necessary packages for Docker.

The following command will download the key with the
58118E89F3A912897C070ADBF76221572C52609D from the **keyserver** hkp://ha.poo
keyservers.net:80 and adds it to the **adv** keychain. Please note that this particular
required to download the necessary Docker packages.

```
sudo apt-key adv \
                --keyserver hkp://ha.pool.sks-keyservers.net:80 \
                --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
```

```
demo@ubuntudemo:~$ sudo apt-key adv \ --keyserver hkp://ha.pool.sks-keyservers.
et:80 \ --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
Executing: gpg --ignore-time-conflict --no-options --no-default-keyring --homed
r /tmp/tmp.KcaZ3WlmGt --no-auto-check-trustdb --trust-model always --keyring /e
c/apt/trusted.gpg --primary-keyring /etc/apt/trusted.gpg  --keyserver hkp://ha.
ool.sks-keyservers.net:80  --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
gpg: requesting key 2C52609D from hkp server ha.pool.sks-keyservers.net
gpg: key 2C52609D: public key "Docker Release Tool (releasedocker) <docker@dock
r.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
demo@ubuntudemo:~$
```

Step 5: Next, depending on the version of Ubuntu you have, you will need to add the relevant site to the **docker.list** for the **apt package manager**, so that it will be able to detect the Docker packages from the Docker site and download them accordingly.

- Precise 12.04 (LTS) — deb https://apt.dockerproject.org/repo ubuntu-precise main
- Trusty 14.04 (LTS) — deb https://apt.dockerproject.org/repo ubuntu-trusty main
- Wily 15.10 — deb https://apt.dockerproject.org/repo ubuntu-wily main
- Xenial 16.04 (LTS) — deb https://apt.dockerproject.org/repo ubuntu-xenial main

Since our OS is Ubuntu 14.04, we will use the Repository name as "deb https://apt.dockerproject.org/repo ubuntu-trusty main"

And then, we will need to add this repository to the **docker.list** as mentioned above.

```
echo "deb https://apt.dockerproject.org/repo ubuntu-trusty main" | sudo tee
/etc/apt/sources.list.d/docker.list
```

```
demo@ubuntudemo:~$ echo "deb https://apt.dockerproject.org/repo ubuntu-trusty
in" | sudo tee /etc/apt/sources.list.d/docker.list
deb https://apt.dockerproject.org/repo ubuntu-trusty main
demo@ubuntudemo:~$ _
```

Step 6: Next, we issue the apt-get update command to update the packages on the Ubuntu system.

```
Hit http://us.archive.ubuntu.com trusty-backports/main Translation-en
Hit http://us.archive.ubuntu.com trusty-backports/multiverse Translation-en
Hit http://us.archive.ubuntu.com trusty-backports/restricted Translation-en
Hit http://us.archive.ubuntu.com trusty-backports/universe Translation-en
Hit http://us.archive.ubuntu.com trusty Release
Hit http://us.archive.ubuntu.com trusty/main Sources
Hit http://us.archive.ubuntu.com trusty/restricted Sources
Hit http://us.archive.ubuntu.com trusty/universe Sources
Hit http://us.archive.ubuntu.com trusty/multiverse Sources
Hit http://us.archive.ubuntu.com trusty/main amd64 Packages
Hit http://us.archive.ubuntu.com trusty/restricted amd64 Packages
Hit http://us.archive.ubuntu.com trusty/universe amd64 Packages
Hit http://us.archive.ubuntu.com trusty/multiverse amd64 Packages
Hit http://us.archive.ubuntu.com trusty/main i386 Packages
Hit http://us.archive.ubuntu.com trusty/restricted i386 Packages
Hit http://us.archive.ubuntu.com trusty/universe i386 Packages
Hit http://us.archive.ubuntu.com trusty/multiverse i386 Packages
Hit http://us.archive.ubuntu.com trusty/main Translation-en
Hit http://us.archive.ubuntu.com trusty/multiverse Translation-en
Hit http://us.archive.ubuntu.com trusty/restricted Translation-en
Hit http://us.archive.ubuntu.com trusty/universe Translation-en
Ign http://us.archive.ubuntu.com trusty/main Translation-en_US
Ign http://us.archive.ubuntu.com trusty/multiverse Translation-en_US
Ign http://us.archive.ubuntu.com trusty/restricted Translation-en_US
Ign http://us.archive.ubuntu.com trusty/universe Translation-en_US
Fetched 3,333 kB in 36s (90.8 kB/s)
Reading package lists... Done
demo@ubuntudemo:~$
```

Step 7: If you want to verify that the package manager is pointing to the right repository, you can do it by issuing the **apt-cache command**.

```
apt-cache policy docker-engine
```

In the output, you will get the link to https://apt.dockerproject.org/repo/

```
     1.8.2-0~trusty 0
        500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packa
s
     1.8.1-0~trusty 0
        500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packa
s
     1.8.0-0~trusty 0
        500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packa
s
     1.7.1-0~trusty 0
        500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packa
s
     1.7.0-0~trusty 0
        500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packa
s
     1.6.2-0~trusty 0
        500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packa
s
     1.6.1-0~trusty 0
        500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packa
s
     1.6.0-0~trusty 0
        500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packa
s
     1.5.0-0~trusty 0
        500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packa
s
demo@ubuntudemo:~$
```

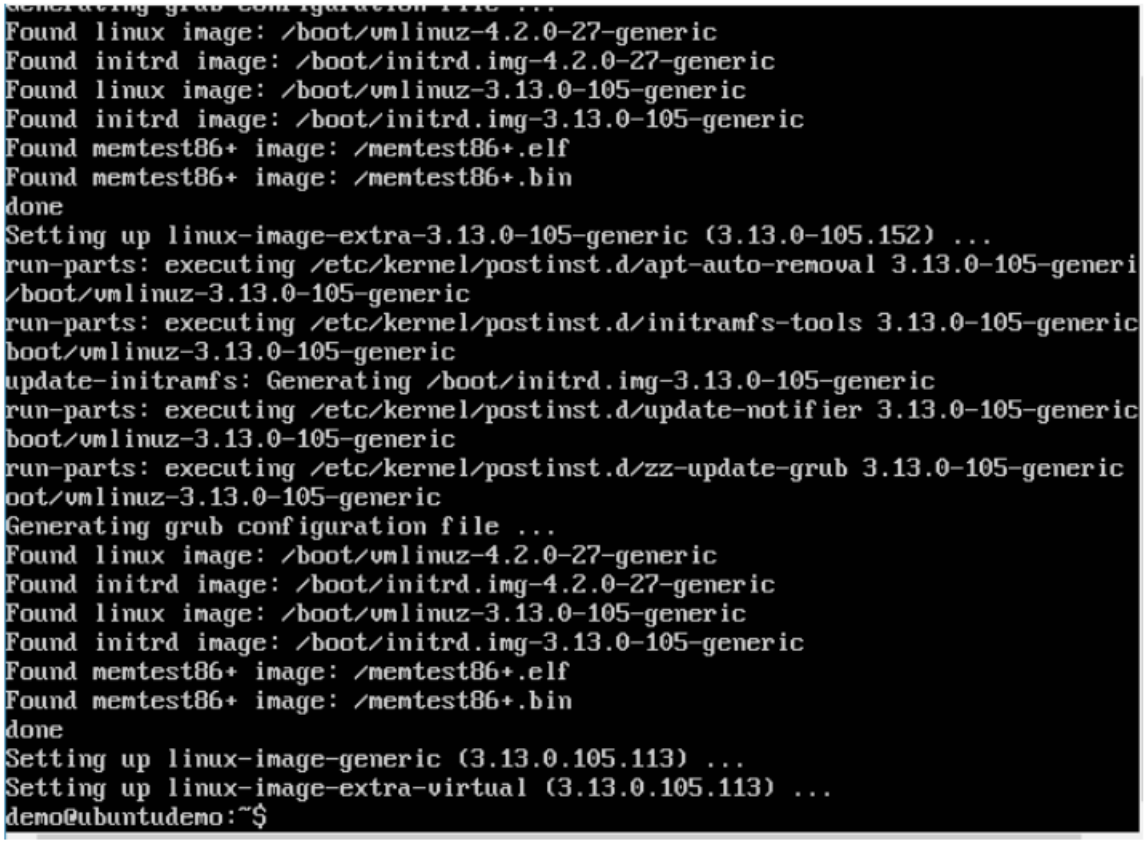
Step 8: Issue the **apt-get update command** to ensure all the packages on the local system are up to date.

```
Hit http://us.archive.ubuntu.com trusty-backports/main Translation-en
Hit http://us.archive.ubuntu.com trusty-backports/multiverse Translation-en
Hit http://us.archive.ubuntu.com trusty-backports/restricted Translation-en
Hit http://us.archive.ubuntu.com trusty-backports/universe Translation-en
Hit http://us.archive.ubuntu.com trusty Release
Hit http://us.archive.ubuntu.com trusty/main Sources
Hit http://us.archive.ubuntu.com trusty/restricted Sources
Hit http://us.archive.ubuntu.com trusty/universe Sources
Hit http://us.archive.ubuntu.com trusty/multiverse Sources
Hit http://us.archive.ubuntu.com trusty/main amd64 Packages
Hit http://us.archive.ubuntu.com trusty/restricted amd64 Packages
Hit http://us.archive.ubuntu.com trusty/universe amd64 Packages
Hit http://us.archive.ubuntu.com trusty/multiverse amd64 Packages
Hit http://us.archive.ubuntu.com trusty/main i386 Packages
Hit http://us.archive.ubuntu.com trusty/restricted i386 Packages
Hit http://us.archive.ubuntu.com trusty/universe i386 Packages
Hit http://us.archive.ubuntu.com trusty/multiverse i386 Packages
Hit http://us.archive.ubuntu.com trusty/main Translation-en
Hit http://us.archive.ubuntu.com trusty/multiverse Translation-en
Hit http://us.archive.ubuntu.com trusty/restricted Translation-en
Hit http://us.archive.ubuntu.com trusty/universe Translation-en
Ign http://us.archive.ubuntu.com trusty/main Translation-en_US
Ign http://us.archive.ubuntu.com trusty/multiverse Translation-en_US
Ign http://us.archive.ubuntu.com trusty/restricted Translation-en_US
Ign http://us.archive.ubuntu.com trusty/universe Translation-en_US
Fetched 30.2 kB in 15s (1,980 B/s)
Reading package lists... Done
demo@ubuntudemo:~$
```

Step 9: For Ubuntu Trusty, Wily, and Xenial, we have to install the linux-image-extra-* kernel packages, which allows one to use the **aufs storage driver**. This driver is used by the newer versions of Docker.

It can be done by using the following command:

```
sudo apt-get install linux-image-extra-$(uname -r) linux-image-extra-virtual
```



Step 10: The final step is to install Docker and we can do this with the following command:

```
sudo apt-get install -y docker-engine
```

Here, **apt-get** uses the install option to download the Docker-engine image from the Docker website and get Docker installed.

The Docker-engine is the official package from the Docker Corporation for Ubuntu-based systems.

```
Selecting previously unselected package liberror-perl.
Preparing to unpack .../liberror-perl_0.17-1.1_all.deb ...
Unpacking liberror-perl (0.17-1.1) ...
Selecting previously unselected package git-man.
Preparing to unpack .../git-man_1%3a1.9.1-1ubuntu0.3_all.deb ...
Unpacking git-man (1:1.9.1-1ubuntu0.3) ...
Selecting previously unselected package git.
Preparing to unpack .../git_1%3a1.9.1-1ubuntu0.3_amd64.deb ...
Unpacking git (1:1.9.1-1ubuntu0.3) ...
Selecting previously unselected package cgroup-lite.
Preparing to unpack .../cgroup-lite_1.9_all.deb ...
Unpacking cgroup-lite (1.9) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Processing triggers for ureadahead (0.100.0-16) ...
ureadahead will be reprofiled on next reboot
Setting up libltdl7:amd64 (2.4.2-1.7ubuntu1) ...
Setting up libsystemd-journal0:amd64 (204-5ubuntu20.20) ...
Setting up aufs-tools (1:3.2+20130722-1.1) ...
Setting up docker-engine (1.12.3-0~trusty) ...
docker start/running, process 22612
Setting up liberror-perl (0.17-1.1) ...
Setting up git-man (1:1.9.1-1ubuntu0.3) ...
Setting up git (1:1.9.1-1ubuntu0.3) ...
Setting up cgroup-lite (1.9) ...
cgroup-lite start/running
Processing triggers for libc-bin (2.19-0ubuntu6.7) ...
Processing triggers for ureadahead (0.100.0-16) ...
demo@ubuntudemo:~$
```

Creating a Docker container environment typically involves several steps :

Installing Docker: To create a Docker container, you must first install Docker on your system. You can follow the installation instructions for your operating system on the Docker website.

Creating a Dockerfile: A Dockerfile is a text file that contains instructions for building a Docker image. This includes installing dependencies, setting environment variables, and copying files into the container. You can create a Dockerfile in any text editor, such as Notepad or Visual Studio Code.

Building the Docker image: Once you have created a Dockerfile, you can use the "docker build" command to build a Docker image from it. This process involves downloading any necessary dependencies and configuring the container.

Running the Docker container: After the Docker image is built, you can use the "docker run" command to start a new container from the image. This will launch your application in a containerized environment.

If you are using NVIDIA Docker, you will need to follow additional steps to configure the container for GPU support. This typically involves installing the NVIDIA drivers and the NVIDIA Docker runtime on your system.

Here is an example Dockerfile for a Python machine learning application using TensorFlow and NVIDIA Docker:

```
FROM nvidia/cuda:11.0-base

RUN apt-get update && apt-get install -y \
    python3 \
    python3-pip \
    python3-dev \
    build-essential \
    libglib2.0-0 \
    libsm6 \
    libxext6 \
    libxrender-dev

RUN pip3 install tensorflow-gpu==2.5.0

WORKDIR /app

COPY . /app

CMD ["python3", "myapp.py"]
```

This Dockerfile installs the necessary dependencies for running a Python machine learning application with TensorFlow and NVIDIA GPU support. Once you have created the Dockerfile, you can build and run the Docker container using the following command

```
docker build -t myapp .
docker run --gpus all -it myapp
```

This will build a Docker image named "myapp" and run a new container from it, with GPU support enabled.

Conclusion:

Using Docker containers, including NVIDIA Docker, for machine learning and data science applications can improve portability, reproducibility, scalability, and security.

It can also simplify the deployment process and make it easier to share applications with others.

c. Create an Angular application which will do following actions: Register User, Login User, Show User Data on Profile Component

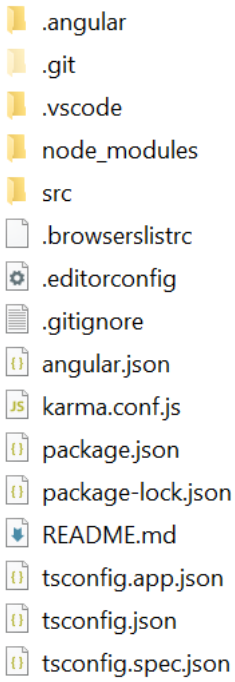
1. Angular requires a current, active LTS(long term support) or maintenance LTS version of Node.js and NPM.

   Install node.js https://nodejs.org/
   It will automatically install NPM - node package manager
2. Install Angular CLI

   npm install -g @angular/cli@latest
   To Create Angular 2 Application Angular CLI is required
3. To create new project

   through CLI go to folder of the new project
   Give command as -ng new project-name

The project will be created as directory structure below –



Open folder src/app

Modify app.module.ts for form application

```
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'
@NgModule({
 declarations: [
   AppComponent
 ],
 imports: [
```

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

Open app.component.html
Write html code for form (representative code is mentioned here, modify for multiple inputs)

```
<h1>Simple Form</h1>
<form #simpleForm = "ngForm" (ngSubmit) = "getValues(simpleForm.value)">
  <input type ="text" ngModel name = "user" placeholder = "Enter Name">
  <br> <br>
  <input type ="text" ngModel name = "age" placeholder = "Enter age">
  <br> <br>
  <input type ="text" ngModel name = "city" placeholder = "Enter city">
  <br> <br>
  <button>Get user value</button>
</form>
```

Make changes in app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'AngProj1';
  getValues(val:any)
  {
    console.log(val);
  }
}
```

Here getValue() function which is called in form file is defined.
You can check inputted values through form in console.
Step 1: Create an Angular application using the Angular CLI

ng new my-app

Step 2: Create a user service to handle user-related functionality. This service will be responsible for making API calls to your server to register, login, and retrieve user data. You can generate a service using the Angular CLI:

ng generate service user

Step 3: Define a user model to represent the user data. The user model will typically contain properties like username, email, password, etc.

Step 4: Create a registration component to allow users to register for an account. This component will have a form that accepts user input and submits it to the user service for processing. The registration component will also handle any error messages returned by the user service.

Step 5: Create a login component to allow users to log in to their account. This component will have a form that accepts user input and submits it to the user service for processing. The login component will also handle any error messages returned by the user service.

Step 6: Create a profile component to display the user data. This component will use the user service to retrieve the user data and display it on the screen. The profile component should only be accessible to users who are logged in.

Step 7: Create a routing module to define the routes for your application. The routing module should include routes for the registration, login, and profile components.

Step 8: Create a navigation bar to allow users to navigate between the different components. The navigation bar should be included in a shared component that can be used across the application.

Step 9: Update the app module to include the routing module and the user service.

Step10:Update the app component to include the navigation bar and the router outlet.

Once you have completed these steps, you should have an Angular application that allows users to register, log in, and view their user data on a profile component.

General outline:

The Angular documentation: The official Angular documentation is a great resource for learning Angular. It includes tutorials and guides for getting started with Angular, as well as documentation on specific features and functionality.

Angular CLI documentation: The Angular CLI is a powerful tool for generating new components, services, and other files in your Angular

application. The CLI documentation provides a detailed reference for all the available commands and options.

Angular Material: Angular Material is a library of pre-built UI components that can be used in your Angular application. It includes components for forms, buttons, navigation, and much more. You can find documentation and examples on the Angular Material website.

Stack Overflow: Stack Overflow is a popular Q&A site for developers. It can be a great resource for finding answers to specific questions or problems you encounter while developing your Angular application.

Web link/resources:

- https://angular.io/docs.
- https://angular-university.io/.
- https://www.ng-book.com/.
- https://blog.angular.io/.
- https://cli.angular.io/.

These are just a few examples of the many web resources available for Angular development. By leveraging these resources, you can improve your skills and build better Angular applications.

Conclusion:

Assignment 3a

Title: Static Web Server

Problem Statement: Create a Node JS application which serves a static web site.

Objective: Use necessory modules lije http, url, fs, path etc for creation of server and file operations on server side.

S/W Packages and H/W apparatus used:

OS/Application/code editor: Ubuntu/Windows, node js, VSCode editor.

PC with the configuration as Pentium IV 1.7 GHz. 128M.B RAM, 40 G.B HDD, 15''Color Monitor, Keyboard, Mouse.

References:

1. Adam Bretz & Colin J Ihrig, Full Stack Javascript Development with MEAN, SPD, First Edition, ISBN:978-0992461256.

Theory:

Here we will build a static file web server which will list out all the files in the directory and on clicking the file name it displays the file content. Steps for creating a static file server is as follows:

Step 1: Importing necessary modules, and defining **Multipurpose Internet Mail Extensions** which helps browser to understand the type of file that is being sent.

```
// Importing necessary modules
const http = require('http');
const url = require('url');
const fs = require('fs');
const path = require('path');

// Port on which the server will create
const PORT = 1800;

// Maps file extension to MIME types which
// helps the browser to understand what to
// do with the file
const mimeType = {
        '.ico': 'image/x-icon',
        '.html': 'text/html',
        '.js': 'text/javascript',
        '.json': 'application/json',
        '.css': 'text/css',
        '.png': 'image/png',
        '.jpg': 'image/jpeg',
        '.wav': 'audio/wav',
        '.mp3': 'audio/mpeg',
        '.svg': 'image/svg+xml',
        '.pdf': 'application/pdf',
        '.doc': 'application/msword',
        '.eot': 'application/vnd.ms-fontobject',
        '.ttf': 'application/font-sfnt'
};
```

Step 2: **Creating a server at the port specified (say 5000).**

```
// Creating a server and listening the port 5000
http.createServer( (req, res) => {
}).listen(PORT);
```

Step 3: **We will respond the** URL "/" **to list all the files in the directory. We will limit this article to the current working directory only. Add the below code to the server's function call.**

```
// Parsing the requested URL
const parsedUrl = url.parse(req.url);

// If requested url is "/" like "http://localhost:5000/"
if(parsedUrl.pathname==="/") {
        var filesLink="<ul>";
        res.setHeader('Content-type', 'text/html');
        var filesList=fs.readdirSync("./");

        filesList.forEach(element => {
                if(fs.statSync("./"+element).isFile()) {
                        filesLink +=`<br/><li><a href='./${element}'>
                                ${element}
                        </a></li>` ;
                }
        });
        filesLink+="</ul>";

        res.end("<h1>List of files:</h1> " + filesLink);
}
```

Step 4: **Preprocessing the requested file pathname to avoid directory traversal (like http://localhost:5000/../fileOutofContext.txt) by replacing '../' with ' '.**

```
/* processing the requested file pathname to
avoid directory traversal like,
http://localhost:1800/../fileOutofContext.txt
by limiting to the current directory only */
const sanitizePath =
path.normalize(parsedUrl.pathname).replace(/^(\.\.[\/\\])+/, '');
```

let pathname = path.join(__dirname, sanitizePath);

Step 5:**Finally, check whether the file exists. If exists then send the file with the proper header** 'Content-type' **having value as per the file extension mapped with the MIME type above. Else if not exist then send** File not found! **With** 404 **status code.**

```
if(!fs.existsSync(pathname)) {

        // If the file is not found, return 404
        res.statusCode = 404;
        res.end(`File ${pathname} not found!`);
}
else {

        // Read file from file system limit to the
        // current directory only.
        fs.readFile(pathname, function(err, data) {
                if(err) {
                        res.statusCode = 500;
                        res.end(`Error in getting the file.`);
                }
                else {

                        // Based on the URL path, extract the file
                        // extension. Ex .js, .doc, ...
                        const ext = path.parse(pathname).ext;

                        // If the file is found, set Content-type
                        // and send data
                        res.setHeader('Content-type',
                                        mimeType[ext] || 'text/plain' );
                        res.end(data);
                }
        });
}
```

Conclusion: Thus, we have used http, fs, url, path node js modules to create static file web server.

Assignment 3b

Title: API for CRUD operations

Problem Statement: Create four API using Node.JS, ExpressJS and MongoDB for CURD Operations for Register User, Login User, Show User Data.

Objective: Use node JS, express, mongo DB and mongoose libraries to create CRUD operations for user adminastration.

S/W Packages and H/W apparatus used:

OS: Ubuntu/Windows, node JS, Express, Mongo DB, Postman, VSCode editor.

PC with the configuration as Pentium IV 1.7 GHz. 128M.B RAM, 40 G.B HDD, 15''Color Monitor, Keyboard, Mouse.

References:

1. Adam Bretz & Colin J Ihrig, Full Stack Javascript Development with MEAN, SPD, First Edition, ISBN:978-0992461256.

Theory:

CRUD is an acronym for Create, Read, Update and Delete**. It is a set of operations we get servers to execute (POST, GET, PUT and DELETE requests respectively). This is what each operation does:**

- Create (POST)- **Make something**
- Read (GET)- **Get something**

- Update (PUT)- **Change something**
- Delete (DELETE)- **Remove something**

Step 1: Creating the Application

- Create a new folder for the application.

mkdir crud-node-express

- Initialize the application with a package.json file

Go to the root folder of your application and type npm init to initialize your app with a package.json file.

```
cd crud-node-express
```

```
npm init
```

Step 2: Install dependencies

Install express, mongoose, and body-parser modules.

```
npm install express body-parser mongoose --save
```

Setting up the Web Server

Create a new file named server.js in the root folder of the application with the following contents:

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express(); app.use(bodyParser.urlencoded({ extended: true }))
app.use(bodyParser.json())
app.get('/', (req, res) => { res.json({"message": "Hello Crud Node Express"}); });
app.listen(3000, () => { console.log("Server is listening on port 3000"); });
```

Step 3: Configuring and Connecting to the database

create a new folder config in the root folder of our application for keeping all the configurations

```
mkdir config
```
cd ***config***

Create a new file database.config.js inside config folder with the following contents:

```
module.exports = {
 url: 'mongodb://localhost:27017/crud-node-express' }
```

import the above database configuration in server.js and connect to the database using mongoose. Add the following code to the server.js file

after app.use(bodyParser.json()) line

```
const dbConfig = require('./config/database.config.js');
const mongoose = require('mongoose');
 mongoose.Promise = global.Promise;
mongoose.connect(dbConfig.url, {
 useNewUrlParser: true }).then(() => {
console.log("Databse Connected Successfully!!"); }).catch(err => { console.log('Could
    not connect to the database', err); process.exit(); });
```

run the server and make sure that you're able to connect to the database.

node server.js

Step 4: Create Mongoose Model

Create a folder called model inside the app folder. Create a user.js file and paste the below code.

```
var mongoose = require('mongoose');
var schema = new mongoose.Schema({ email: { type: String, required: true, unique:
    true },
firstName: { type: String, default: '' },
lastName: { type: String, default: '' }, phone: String, });
var user = new mongoose.model('User', schema);
module.exports = user;
```

Step 5: Create the Controller

Inside app/controllers **folder, let's create** User.js **with these CRUD functions:**

- **create**
- **findAll**
- **findOne**
- **update**
- **destroy**

```
const UserModel = require('../model/user') // Create and Save a new user

exports.create = async (req, res) => {
if (!req.body.email && !req.body.firstName && !req.body.lastName && !req.body.phone) {
res.status(400).send({ message: "Content can not be empty!" }); }

const user = new UserModel({ email: req.body.email, firstName: req.body.firstName,
lastName: req.body.lastName, phone: req.body.phone });

await user.save().then(data => { res.send({ message:"User created successfully!!",
user:data }); }).catch(err => { res.status(500).send({ message: err.message || "Some error
occurred while creating user" }); }); };
// Retrieve all users from the database.

exports.findAll = async (req, res) => { try { const user = await UserModel.find();
res.status(200).json(user); } catch(error) { res.status(404).json({message: error.message});
} };
// Find a single User with an id

exports.findOne = async (req, res) => { try { const user = await
UserModel.findById(req.params.id); res.status(200).json(user); } catch(error) {
res.status(404).json({ message: error.message}); } };
// Update a user by the id in the request

exports.update = async (req, res) => { if(!req.body) { res.status(400).send({ message:
"Data to update can not be empty!" }); } const id = req.params.id; await
UserModel.findByIdAndUpdate(id, req.body, { useFindAndModify: false }).then(data => {
if (!data) { res.status(404).send({ message: `User not found.` }); }else{ res.send({ message:
"User updated successfully." }) } }).catch(err => { res.status(500).send({ message:
err.message }); }); };
 // Delete a user with the specified id in the request

exports.destroy = async (req, res) => { await
UserModel.findByIdAndRemove(req.params.id).then(data => { if (!data) {
res.status(404).send({ message: `User not found.` }); } else { res.send({ message: "User
deleted successfully!" }); } }).catch(err => { res.status(500).send({ message: err.message });
}); };
```

Creating a new User

```
// Create and Save a new user
exports.create = async (req, res) => { if (!req.body.email && !req.body.firstName &&
    !req.body.lastName && !req.body.phone) { res.status(400).send({ message: "Content
    can not be empty!" }); }
 const user = new UserModel({ email: req.body.email, firstName: req.body.firstName,
lastName: req.body.lastName,
phone: req.body.phone });
await user.save().then(data => {
res.send({ message:"User created successfully!!", user:data }); }).catch(err => {
    res.status(500).send({ message: err.message || "Some error occurred while creating
    user"
});
});
};
```

Retrieving all Users

```
// Retrieve all users from the database.
exports.findAll = async (req, res) => { try { const user = await UserModel.find();
    res.status(200).json(user); }
catch(error) { res.status(404).json({message: error.message}); }
};
```

Retrieving a single User

```
// Retrieve all users from the database.
 exports.findAll = async (req, res) => {
try { const user = await UserModel.find(); res.status(200).json(user); } catch(error) {
    res.status(404).json({message: error.message}); }
};
```

Updating a User

```
// Update a user by the id in the request exports.update = async (req, res) => {
    if(!req.body) { res.status(400).send({ message: "Data to update can not be
```

empty!" }); } const id = req.params.id; await UserModel.findByIdAndUpdate(id, req.body, { useFindAndModify: false }).then(data => { if (!data) { res.status(404).send({ message: `User not found.` }); }else{ res.send({ message: "User updated successfully." }) } }).catch(err => { res.status(500).send({ message: err.message }); }); };

Deleting a User

// Delete a user with the specified id in the request exports.destroy = async (req, res) => { await UserModel.findByIdAndRemove(req.params.id).then(data => { if (!data) { res.status(404).send({ message: `User not found.` }); } else { res.send({ message: "User deleted successfully!" }); } }).catch(err => { res.status(500).send({ message: err.message }); }); };

Step 6: Define Routes

When a client sends a request for an endpoint using an HTTP request (GET, POST, PUT, DELETE), we need to determine how the server will respond by setting up the routes.

Create a User.js **inside** app/routes **folder with content like this:**

const express = ***require***(***'express'***) ***const*** UserController = ***require***(***'../controllers/User'***) ***const*** router = express.Router(); router.get(***'/'***, UserController.findAll); router.get(***'/:id'***, UserController.findOne); router.post(***'/'***, UserController.create); router.patch(***'/:id'***, UserController.update); router.delete(***'/:id'***, UserController.destroy); ***module***.exports = router

The last step before trying out our routes is to add the route class to the **server.js**

const UserRoute = require('./app/routes/User') app.use('/user',UserRoute)

restart your node.js server and now we have our API ready.

Conclusion: Thus, we have used node JS, express, mongo DB and mongoose libraries to create CRUD operations

Assignment 4 a

Title: Implementation of jQuery Mobile

Problem Statement: Create a simple Mobile Website using jQuery Mobile.

Objective: Apply jQuery Mobile library conventions and demonstrate a website.

S/W Packages and H/W apparatus used:

Software/Languange/Libraries used: HTML, jQuery Mobile

Linux OS: Ubuntu/Windows, VSCode editor.

PC with the configuration as Pentium IV 1.7 GHz. 128M.B RAM, 40 G.B HDD, 15''Color Monitor, Keyboard, Mouse.

References:

1. https://jquerymobile.com/
2. https://demos.jquerymobile.com/1.4.5/

Theory:

What is Mobile jQuery?

jQuery Mobile is the easiest way to build sites and apps that are accessible on all popular smartphone, tablet and desktop devices.

jQuery Mobile is a touch-optimized HTML5 UI framework designed to make responsive web sites and apps that are accessible on all smartphone, tablet and desktop devices.

jQuery Mobile has multiple Pages and Navigation, Widgets, Form Widgets etc.

1. Pages

The page is the primary unit of interaction in jQuery Mobile and is used to group content into logical views that can be animated in and out of view with page transitions. A HTML document may start with a single "page" and the Ajax navigation system will load additional pages on demand into the DOM as users navigate around. Alternatively, a HTML document can be built with multiple "pages"

inside it and the framework will transition between these local views with no need to request content from the server.

Mobile page structure

A jQuery Mobile site must start with an HTML5 doctype to take full advantage of all of the framework's features. (Older devices with browsers that don't understand HTML5 will safely ignore the 'doctype' and various custom attributes.)

In the head, references to jQuery, jQuery Mobile and the mobile theme CSS are all required to start things off. The easiest way to get started is to link to files hosted on the jQuery CDN or for best performance, build a custom bundle.

```
<!DOCTYPE html>
<html>
<head>
    <title>Page Title</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://code.jquery.com/mobile/[version]/jquery.mobile-[version].min.css" />
    <script src="https://code.jquery.com/jquery-[version].min.js"></script>
    <script src="https://code.jquery.com/mobile/[version]/jquery.mobile-[version].min.js"></script>
</head>

<body>
    ...content goes here...
</body>
</html>
```

Viewport meta tag

Note above that there is a meta viewport tag in the head to specify how the browser should display the page zoom level and dimensions. If this isn't set, many mobile browsers will use a "virtual" page width around 900 pixels to make it work well with existing desktop sites but the screens may look zoomed out and too wide. By setting the viewport attributes to content="width=device-width, initial-scale=1", the width will be set to the pixel width of the device screen.

<meta name="viewport" content="width=device-width, initial-scale=1">

These settings do not disable the user's ability to zoom the pages, which is nice from an accessibility perspective. There is a minor issue in iOS that doesn't properly set the width when changing orientations with these viewport settings, but this will hopefully be fixed in a future release. You can set other viewport values to disable zooming if required since this is part of your page content, not the library.

Inside the body: Pages

Inside the <body> tag, each view or "page" on the mobile device is identified with an element (usually a div) with the data-role="page" attribute.

```
<div data-role="page">
    ...
</div>
```

Within the "page" container, any valid HTML markup can be used, but for typical pages in jQuery Mobile, the immediate children of a "page" are divs with data-role="header", class="ui-content", and data-role="footer".

```
<div data-role="page">
    <div data-role="header">...</div>
    <div role="main" class="ui-content">...</div>
    <div data-role="footer">...</div>
</div>
```

Putting it together: Basic single page template

Putting it all together, this is the standard boilerplate page template you should start with on a project:

```
<!DOCTYPE html>
<html>
<head>
    <title>Page Title</title>

    <meta name="viewport" content="width=device-width, initial-scale=1">
```

```
    <link rel="stylesheet" href="https://code.jquery.com/mobile/[version]/jquery.mobile-[version].min.css" />
    <script src="https://code.jquery.com/jquery-[version].min.js"></script>
    <script src="https://code.jquery.com/mobile/[version]/jquery.mobile-[version].min.js"></script>
</head>
<body>

<div data-role="page">

    <div data-role="header">
        <h1>Page Title</h1>
    </div><!-- /header -->

    <div role="main" class="ui-content">
        <p>Page content goes here.</p>
    </div><!-- /content -->

    <div data-role="footer">
        <h4>Page Footer</h4>
    </div><!-- /footer -->
</div><!-- /page -->

</body>
</html>
```

Multi-page template structure

A single HTML document can contain multiple "pages" that are loaded together by stacking multiple divs with a data-role of "page". Each "page" block needs a unique id (id="foo") that will be used to link internally between "pages" (href="#foo"). When a link is clicked, the framework will look for an internal "page" with the id and transition it into view.

Here is an example of a two "page" site built with two jQuery Mobile divs navigated by linking to an id placed on each page wrapper. Note that the ids on the page wrappers are only needed to support the internal page linking, and are optional if each

page is a separate HTML document. Here is what two pages look like inside the body element.

```
<body>

<!-- Start of first page -->
<div data-role="page" id="foo">

    <div data-role="header">
        <h1>Foo</h1>
    </div><!-- /header -->

    <div role="main" class="ui-content">
        <p>I'm first in the source order so I'm shown as the page.</p>
        <p>View internal page called <a href="#bar">bar</a></p>
    </div><!-- /content -->

    <div data-role="footer">
        <h4>Page Footer</h4>
    </div><!-- /footer -->
</div><!-- /page -->

<!-- Start of second page -->
<div data-role="page" id="bar">

    <div data-role="header">
        <h1>Bar</h1>
    </div><!-- /header -->

    <div role="main" class="ui-content">
        <p>I'm the second in the source order so I'm hidden when the page loads.
I'm just shown if a link that references my id is beeing clicked.</p>
        <p><a href="#foo">Back to foo</a></p>
    </div><!-- /content -->

    <div data-role="footer">
```

```
            <h4>Page Footer</h4>
    </div><!-- /footer -->
</div><!-- /page -->
</body>
```

Ajax Navigation

The $.mobile.navigate method and the navigate event form the foundation of jQuery Mobile's navigation infrastructure. As such, they can function outside the confines of jQuery Mobile as a clean and intuitive navigation/history API.

Introduction

jQuery Mobile includes a navigation system to load pages into the DOM via Ajax, enhance the new content, then display pages with a rich set of animated transitions. The navigation system uses progressive enhancement to automatically 'hijack' standard links and form submissions and route them as an Ajax request.

One of jQuery Mobile's core features is the ability to load and view content from disparate pages into the initial document with support for standard navigation methods like anchors and the back button. To accomplish this the library has progressive support for hashchange and popstate coupled with internal history tracking which can be used à la carte.

An example use case would be something like Twitter's web client. The first step is to hijack link clicks on the page and use the URL that represents that UI state to track history with $.mobile.navigate. It's at this point that any additional information about the UI necessary for operation on return using the back button would be stored (see, foo property of the object argument to the navigate method).

```
// Define a click binding for all anchors in the page
$( "a" ).on( "click", function( event ) {

    // Prevent the usual navigation behavior
    event.preventDefault();

    // Alter the url according to the anchor's href attribute, and
    // store the data-foo attribute information with the url
    $.mobile.navigate( $(this).attr( "href" ), {
            foo: $(this).attr("data-foo")
    });
```

```
    // Hypothetical content alteration based on the url. E.g, make
    // an Ajax request for JSON data and render a template into the page.
    alterContent( $(this).attr("href") );
});
```

Next, a navigate event binding helps in responding to backward and forward navigation via the browsers history API. Here the alterContent function can address the direction in which the browser is navigating as well as any additional information stored on the data object when $.mobile.navigate was invoked to store the corresponding history entry.

```
// Respond to back/forward navigation
$( window ).on( "navigate", function( event, data ){
    if ( data.state.foo ) {
            // Make use of the arbitrary data stored
    }

    if ( data.state.direction == "back" ) {
            // Make use of the directional information
    }

    // reset the content based on the url
    alterContent( data.state.url );
});
```

2. Widgets

Navbar

jQuery Mobile has a very basic navbar widget that is useful for providing up to 5 buttons with optional icons in a bar.

Navbar basics

A navbar is coded as an unordered list of links wrapped in a container element that has the data-role="navbar" attribute. When a link in the navbar is clicked it gets the active (selected) state. The ui-btn-active class is first removed from all anchors in the

navbar before it is added to the activated link. If this is a link to another page, the class will be removed again after the transition has completed.

To set an item to the active state, add class="ui-btn-active" to an anchor in the markup. Additionally add a class of ui-state-persist to make the framework restore the active state each time the page is shown while it exists in the DOM.

Navbars with 1 item will render as 100% wide.

```
<div data-role=”navbar”>
    <ul>
        <li><a href=”#” class=”ui-btn-active”>One</a><li>
    </ul>
</div>
```

The navbar items are set to divide the space evenly so in this case, each button is 1/2 the width of the browser window:

```
<div data-role=”navbar”>
    <ul>
        <li><a href=”#” class=”ui-btn-active”>One</a><li>
<li><a href=”#”>Two</a></li>
    </ul>
</div>
```

And there are also ways to add third and fourth items.

3. Form Widgets

Checkbox

Checkbox inputs are used to provide a list of options where more than one can be selected. Checkbox buttons are enhanced by the checkboxradio widget.

Basic markup

To create a single checkbox, add an input with a type="checkbox" attribute and a corresponding label. If the input isn't wrapped in its corresponding label, be sure to set the for attribute of the label to match the id of the input so they are semantically associated.

```
<form>
    <label>
```

<input type=”checkbox” name=”checkbox-0”>Check Me

</label>

</form>

Radio buttons

Radio inputs are used to provide a list of options where only a single option can be selected. Radio buttons are enhanced by the checkboxradio widget.

Basic markup

To create a set of radio buttons, add an input with a type="radio" attribute and a corresponding label. Set the for attribute of the label to match the id of the input so they are semantically associated.

<form>

<label>

<input type=”radio” name=”radio-choice-0”>One

</label>

<label for=”radio-choice-0b”>Two</label>

<input type=”radio” name=”radio-choice-0” id=” radio-choice-0b” class=”custom”>

</form>

Sample Code:

Index.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=1.0">

   <link rel="stylesheet" href="https://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css" />
   <script src="https://code.jquery.com/jquery-1.11.1.min.js"></script>
```

```
    <script src="https://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js"></script>
    <title>Demo of jQuery Mobile</title>
</head>
<body>
  <div data-role="page" style="background-color:aliceblue;">
      <div data-role="header" data-theme="b">
         <h1>My Website</h1>
      </div>
      <div data-role="navbar">
         <ul>
            <li><a href="index.html">Home</a></li>
            <li><a href="about.html">About</a></li>
            <li><a href="login_register.html">Register for Newsletter</a></li>
         </ul>
      </div>
      <center><h3>Gallery</h3></center>
      <div class="content">
         <center>
            <div class="rows">
               <img src="img1.jpg" alt="Img1" width="240" height="200">
               <img src="img2.jpg" alt="Img2" width="240" height="200">
               <img src="img3.jpg" alt="Img3" width="240" height="200">
               <img src="img4.jpg" alt="Img4" width="240" height="200">
               <img src="img5.jpg" alt="Img4" width="240" height="200">
               <img src="img6.jpg" alt="Img4" width="240" height="200">
            </div>
            <br>

            <div class="rows">
               <img src="img1.jpg" alt="Img1" width="240" height="200">
               <img src="img2.jpg" alt="Img2" width="240" height="200">
```

```
    <img src="img3.jpg" alt="Img3" width="240" height="200">
    <img src="img4.jpg" alt="Img4" width="240" height="200">
    <img src="img5.jpg" alt="Img4" width="240" height="200">
    <img src="img6.jpg" alt="Img4" width="240" height="200">
</div>
<br>

<div class="rows">
    <img src="img1.jpg" alt="Img1" width="240" height="200">
    <img src="img2.jpg" alt="Img2" width="240" height="200">
    <img src="img3.jpg" alt="Img3" width="240" height="200">
    <img src="img4.jpg" alt="Img4" width="240" height="200">
    <img src="img5.jpg" alt="Img4" width="240" height="200">
    <img src="img6.jpg" alt="Img4" width="240" height="200">
</div>
<br>

<div class="rows">
    <img src="img1.jpg" alt="Img1" width="240" height="200">
    <img src="img2.jpg" alt="Img2" width="240" height="200">
    <img src="img3.jpg" alt="Img3" width="240" height="200">
    <img src="img4.jpg" alt="Img4" width="240" height="200">
    <img src="img5.jpg" alt="Img4" width="240" height="200">
    <img src="img6.jpg" alt="Img4" width="240" height="200">
</div>
<br>

<div class="rows">
    <img src="img1.jpg" alt="Img1" width="240" height="200">
    <img src="img2.jpg" alt="Img2" width="240" height="200">
    <img src="img3.jpg" alt="Img3" width="240" height="200">
```

```
    <img src="img4.jpg" alt="Img4" width="240" height="200">
    <img src="img5.jpg" alt="Img4" width="240" height="200">
    <img src="img6.jpg" alt="Img4" width="240" height="200">
</div>
<br>

<div class="rows">
    <img src="img1.jpg" alt="Img1" width="240" height="200">
    <img src="img2.jpg" alt="Img2" width="240" height="200">
    <img src="img3.jpg" alt="Img3" width="240" height="200">
    <img src="img4.jpg" alt="Img4" width="240" height="200">
    <img src="img5.jpg" alt="Img4" width="240" height="200">
    <img src="img6.jpg" alt="Img4" width="240" height="200">
</div>
<br>

<div class="rows">
    <img src="img1.jpg" alt="Img1" width="240" height="200">
    <img src="img2.jpg" alt="Img2" width="240" height="200">
    <img src="img3.jpg" alt="Img3" width="240" height="200">
    <img src="img4.jpg" alt="Img4" width="240" height="200">
    <img src="img5.jpg" alt="Img4" width="240" height="200">
    <img src="img6.jpg" alt="Img4" width="240" height="200">
</div>
<br>

<div class="rows">
    <img src="img1.jpg" alt="Img1" width="240" height="200">
    <img src="img2.jpg" alt="Img2" width="240" height="200">
    <img src="img3.jpg" alt="Img3" width="240" height="200">
    <img src="img4.jpg" alt="Img4" width="240" height="200">
```

```
    <img src="img5.jpg" alt="Img4" width="240" height="200">
    <img src="img6.jpg" alt="Img4" width="240" height="200">
</div>
<br>

<div class="rows">
    <img src="img1.jpg" alt="Img1" width="240" height="200">
    <img src="img2.jpg" alt="Img2" width="240" height="200">
    <img src="img3.jpg" alt="Img3" width="240" height="200">
    <img src="img4.jpg" alt="Img4" width="240" height="200">
    <img src="img5.jpg" alt="Img4" width="240" height="200">
    <img src="img6.jpg" alt="Img4" width="240" height="200">
</div>
<br>

<div class="rows">
    <img src="img1.jpg" alt="Img1" width="240" height="200">
    <img src="img2.jpg" alt="Img2" width="240" height="200">
    <img src="img3.jpg" alt="Img3" width="240" height="200">
    <img src="img4.jpg" alt="Img4" width="240" height="200">
    <img src="img5.jpg" alt="Img4" width="240" height="200">
    <img src="img6.jpg" alt="Img4" width="240" height="200">
</div>
<br>

<div class="rows">
    <img src="img1.jpg" alt="Img1" width="240" height="200">
    <img src="img2.jpg" alt="Img2" width="240" height="200">
    <img src="img3.jpg" alt="Img3" width="240" height="200">
    <img src="img4.jpg" alt="Img4" width="240" height="200">
    <img src="img5.jpg" alt="Img4" width="240" height="200">
```

```
            <img src="img6.jpg" alt="Img4" width="240" height="200">
        </div>
        <br>

        <div class="rows">
            <img src="img1.jpg" alt="Img1" width="240" height="200">
            <img src="img2.jpg" alt="Img2" width="240" height="200">
            <img src="img3.jpg" alt="Img3" width="240" height="200">
            <img src="img4.jpg" alt="Img4" width="240" height="200">
            <img src="img5.jpg" alt="Img4" width="240" height="200">
            <img src="img6.jpg" alt="Img4" width="240" height="200">
        </div>
        <br>

        <div class="rows">
            <img src="img1.jpg" alt="Img1" width="240" height="200">
            <img src="img2.jpg" alt="Img2" width="240" height="200">
            <img src="img3.jpg" alt="Img3" width="240" height="200">
            <img src="img4.jpg" alt="Img4" width="240" height="200">
            <img src="img5.jpg" alt="Img4" width="240" height="200">
            <img src="img6.jpg" alt="Img4" width="240" height="200">
        </div>
        <br>
    </center>
</div>
<div data-role="footer">
    <center>
        <a href="index.html" class="ui-btn ui-btn-inline">Home</a>
        <a href="about_us.html" class="ui-btn ui-btn-inline">About</a>
        <a href="login_register.html" class="ui-btn ui-btn-inline">Register</a>
    </center>
```

```
        </div>
    </div>
</body>
</html>
```

Register.html

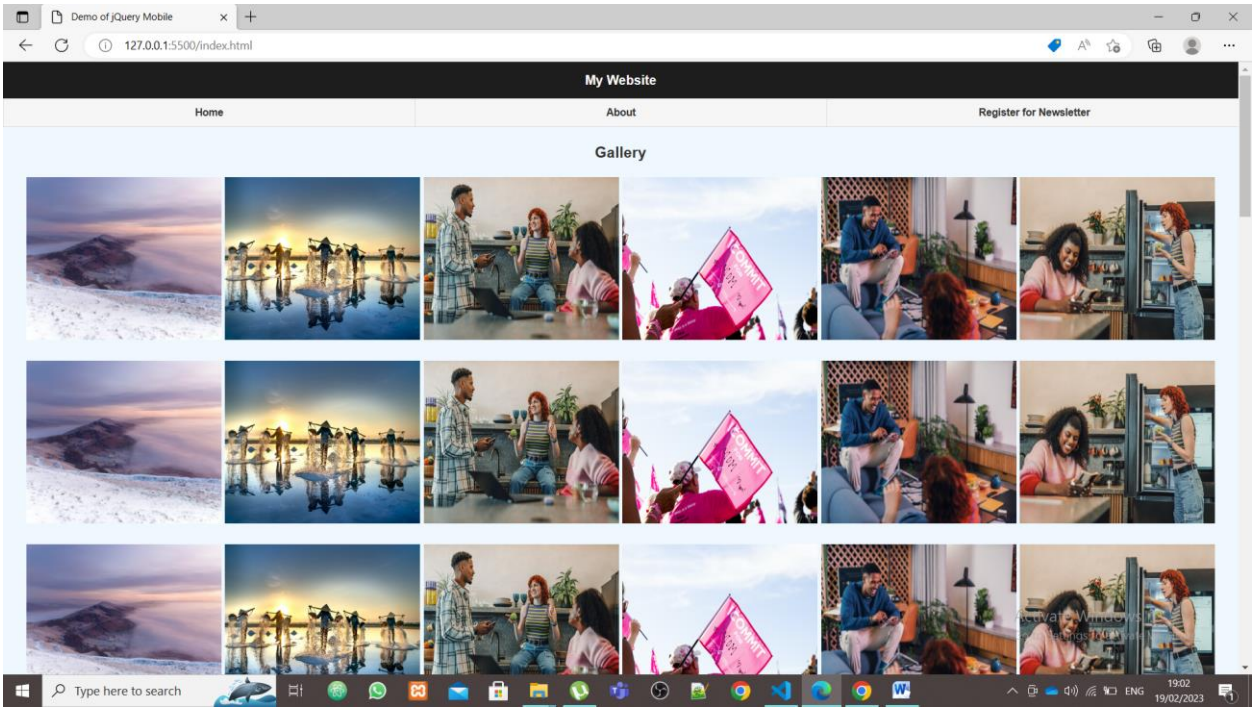
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="https://code.jquery.com/mobile/1.4.5/jquery.mobile-
1.4.5.min.css" />
    <script src="https://code.jquery.com/jquery-1.11.1.min.js"></script>
    <script src="https://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js"></script>
    <title>Demo of jQuery Mobile</title>
</head>
<body>
    <div data-role="page" style="background-color:aliceblue;">
        <div data-role="header" data-theme="b">
            <h1>My Website</h1>
        </div>

            <div data-role="navbar">
                <ul>
                    <li><a href="index.html">Home</a></li>
                    <li><a href="about.html">About</a></li>
                    <li><a href="login_register.html">Register for Newsletter</a></li>
                </ul>
```

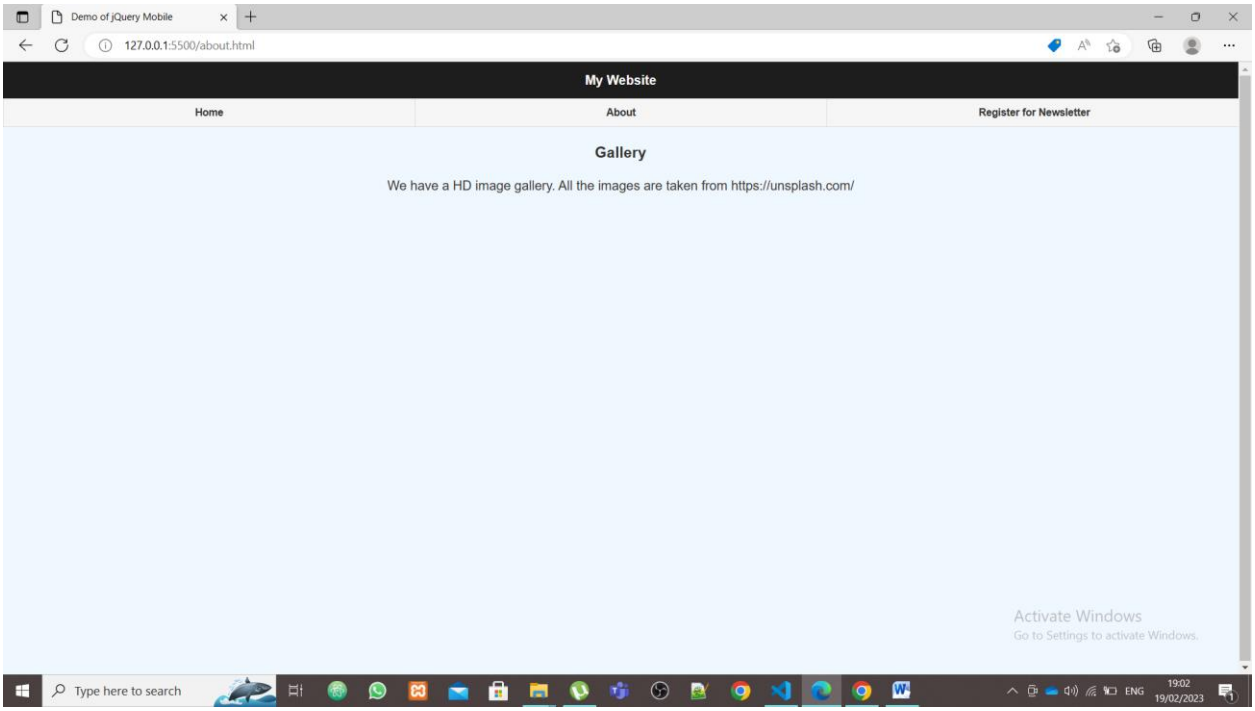
```
        </div>

      <div class="content" style="width:50%; margin: 5% auto;">
        <center><h3>Register</h3></center>
          <div data-role="content">
            <label for="userName">Username</label>
            <input type="text" name="userName">
            <label for="email">Email</label>
            <input type="text" name="email">
            <label for="password">Password</label>
            <input type="text" name="password">
            <center><a href="index.html" class="ui-btn ui-btn-inline">Register
Me</a></center>
          </div>

      </div>
  </div>
</body>
</html>
```

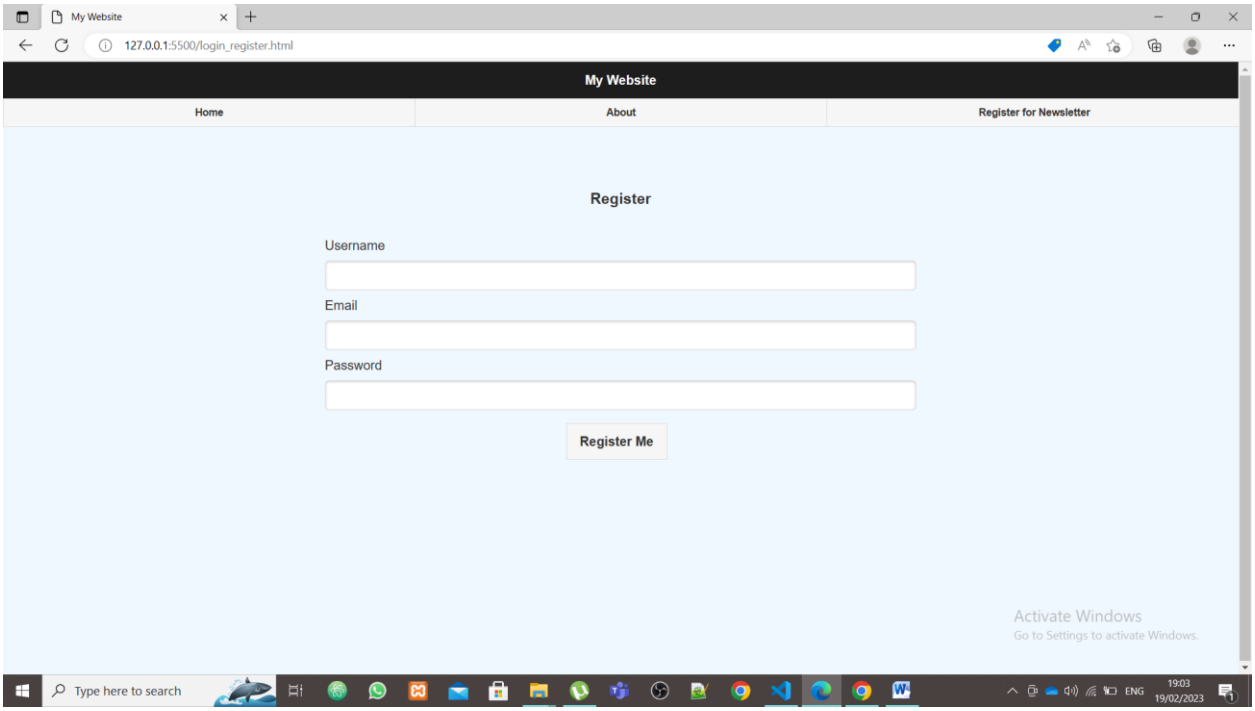
About.html

```
<!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=1.0">

    <link rel="stylesheet" href="https://code.jquery.com/mobile/1.4.5/jquery.mobile-
1.4.5.min.css" />
    <script src="https://code.jquery.com/jquery-1.11.1.min.js"></script>
```

```
    <script src="https://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js"></script>
    <title>Demo of jQuery Mobile</title>
</head>
<body>
  <div data-role="page" style="background-color:aliceblue;">
      <div data-role="header" data-theme="b">
         <h1>My Website</h1>
      </div>
      <div data-role="navbar">
         <ul>
            <li><a href="index.html">Home</a></li>
            <li><a href="about.html">About</a></li>
            <li><a href="login_register.html">Register for Newsletter</a></li>
         </ul>
      </div>
      <center><h3>Gallery</h3></center>
      <div class="content">
         <center>
            We have a HD image gallery. All the images are taken from https://unsplash.com/
         </center>
      </div>
  </div>
</body>
</html>
```


Demo of jQuery Mobile
127.0.0.1:5500/index.html
My Website
Home
About
Register for Newsletter
Gallery
Type here to search
ENG
19:02
19/02/2023


Demo of jQuery Mobile
127.0.0.1:5500/about.html
My Website
Home
About
Register for Newsletter
Gallery
We have a HD image gallery. All the images are taken from https://unsplash.com/
Activate Windows
Go to Settings to activate Windows.
Type here to search
ENG
19:02
19/02/2023



Conclusion: Thus, we are able to create a simple Mobile Website using jQuery Mobile.

Assignment 4b

Title: Work on AWS VPC or AWS Elastic Beanstalk for deployment

Problem Statement: Deploy/Host your web application on AWS VPC or AWS Elastic Beanstalk.

Objective: Understand the working of AWS VPC or AWS Elastic Beanstalk.

S/W Packages and H/W apparatus used:

Software/Languange/Libraries used: AWS VPC or AWS Elastic Beanstalk

Linux OS: Ubuntu/Windows, VSCode editor.

PC with the configuration as Pentium IV 1.7 GHz. 128M.B RAM, 40 G.B HDD, 15''Color Monitor, Keyboard, Mouse.

References:

1. https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html
2. https://docs.aws.amazon.com/vpc/latest/userguide/working-with-vpcs.html
3. https://docs.aws.amazon.com/elastic-beanstalk/index.html

Theory:

What is Amazon VPC?

> *Amazon Virtual Private Cloud (Amazon VPC) enables you to launch AWS resources into a virtual network that you've defined. This virtual network closely resembles a traditional network that you'd operate in your own data center, with the benefits of using the scalable infrastructure of AWS.*

Features

The following features help you configure a VPC to provide the connectivity that your applications need:

Virtual private clouds (VPC)

A VPC is a virtual network that closely resembles a traditional network that you'd operate in your own data center. After you create a VPC, you can add subnets.

Subnets

A subnet is a range of IP addresses in your VPC. A subnet must reside in a single Availability Zone. After you add subnets, you can deploy AWS resources in your VPC.

IP addressing

You can assign IPv4 addresses and IPv6 addresses to your VPCs and subnets. You can also bring your public IPv4 and IPv6 GUA addresses to AWS and allocate them to resources in your VPC, such as EC2 instances, NAT gateways, and Network Load Balancers.

Routing

Use route tables to determine where network traffic from your subnet or gateway is directed.

Gateways and endpoints

A gateway connects your VPC to another network. For example, use an internet gateway to connect your VPC to the internet. Use a VPC endpoint to connect to AWS services privately, without the use of an internet gateway or NAT device.

Peering connections

Use a VPC peering connection to route traffic between the resources in two VPCs.

Traffic Mirroring

Copy network traffic from network interfaces and send it to security and monitoring appliances for deep packet inspection.

Transit gateways

Use a transit gateway, which acts as a central hub, to route traffic between your VPCs, VPN connections, and AWS Direct Connect connections.

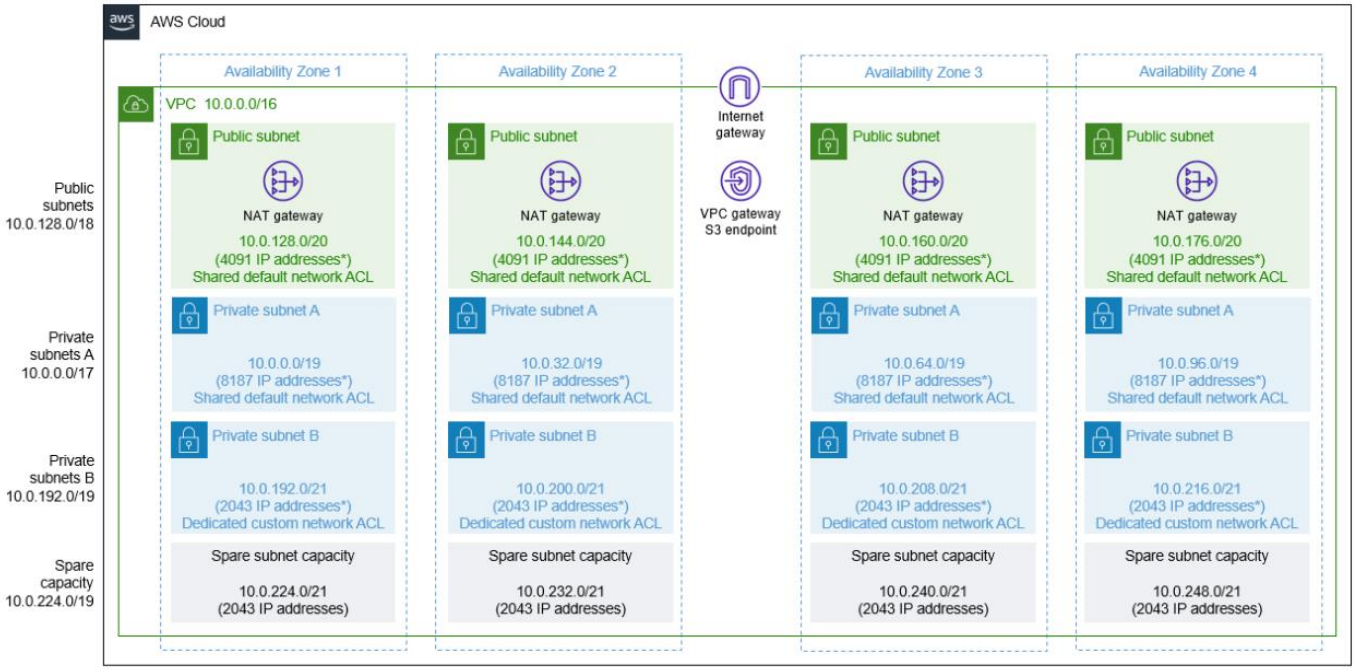
VPC Flow Logs

A flow log captures information about the IP traffic going to and from network interfaces in your VPC.

VPN connections

Connect your VPCs to your on-premises networks using AWS Virtual Private Network (AWS VPN)

Architecture for Amazon VPC on AWS



Getting started with Amazon VPC

Your AWS account includes a default VPC in each AWS Region. Your default VPCs are configured such that you can immediately start launching and connecting to EC2 instances.

Working with Amazon VPC

You can create and manage your VPCs using any of the following interfaces:

- AWS Management Console — Provides a web interface that you can use to access your VPCs.
- AWS Command Line Interface (AWS CLI) — Provides commands for a broad set of AWS services, including Amazon VPC, and is supported on Windows, Mac, and Linux.
- AWS SDKs — Provides language-specific APIs and takes care of many of the connection details, such as calculating signatures, handling request retries, and error handling.
- Query API — Provides low-level API actions that you call using HTTPS requests. Using the Query API is the most direct way to access Amazon VPC, but it requires that your application handle low-level details such as generating the hash to sign the request, and error handling.

How Amazon VPC works

Amazon Virtual Private Cloud (Amazon VPC) enables you to launch AWS resources into a virtual network that you've defined. This virtual network closely resembles a traditional network that you'd operate in your own data center, with the benefits of using the scalable infrastructure of AWS.

VPCs and subnets

A *virtual private cloud* (VPC) is a virtual network dedicated to your AWS account. It is logically isolated from other virtual networks in the AWS Cloud. You can specify an IP address range for the VPC, add subnets, add gateways, and associate security groups.

A *subnet* is a range of IP addresses in your VPC. You launch AWS resources, such as Amazon EC2 instances, into your subnets. You can connect a subnet to the internet, other VPCs, and your own data centers, and route traffic to and from your subnets using route tables.

Create a VPC

Follow the steps in this section to create a VPC. When you create a VPC, you have two options:

- VPC, subnets, and other VPC resources: Creates a VPC, subnets, NAT gateways, and VPC endpoints.
- VPC only: Creates only a VPC without any additional resources like subnets or NAT gateways within the VPC.

Create a VPC, subnets, and other VPC resources

In this step, you create a VPC, subnets, Availability Zones, NAT gateways, and VPC endpoints.

To create a VPC, subnets, and other VPC resources

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.

2. In the navigation pane, choose Your VPCs, Create VPC.
3. Under Resources to create, choose VPC and more.
4. Modify the options as needed:

- Name tag auto-generation: Choose the Name tag that will be applied to the resources you create. The tag can either be automatically generated for you or you can define the value. The defined value will be used to generate the Name tag in all resources as "name-resource". For example if you enter "Preproduction", each subnet will be tagged with a "Preproduction-" Name tag.
- IPv4 CIDR block: Choose an IPv4 CIDR for the VPC. This option is required.
- IPv6 CIDR block: Choose an IPv6 CIDR for the VPC.
- Tenancy: Choose the tenancy option for this VPC.
  - Select Default to ensure that EC2 instances launched in this VPC use the EC2 instance tenancy attribute specified when the EC2 instance is launched.
  - Select Dedicated to ensure that EC2 instances launched in this VPC are run on dedicated tenancy instances regardless of the tenancy attribute specified at launch.

- Number of Availability Zones (AZs): Choose the number of AZs in which you want to create subnets. An AZ is one or more discrete data centers with redundant power, networking, and connectivity in an AWS Region. AZs give you the ability to operate production applications and databases that are more highly available, fault tolerant, and scalable than would be possible from a single data center. If you partition your applications running in subnets across AZs, you are better isolated and protected from issues such as power outages, lightning strikes, tornadoes, earthquakes, and more.
- Customize AZs: Choose which AZs your subnets will be created in.
- Number of public subnets: Choose the number of subnets you would like to be considered "public" subnets. A "public" subnet is a subnet that has a route table entry that points to an internet gateway. This enables EC2 instances running in the subnet to be publicly accessible over the internet.
- Number of private subnets: Choose the number of subnets you would like to be considered "private" subnets. A "private" subnet is a subnet that does not have a route table entry that points to an internet gateway. Use private subnets to secure backend resources that do not need to be publicly accessible over the internet.
- Customize subnets CIDR blocks: Choose the CIDR blocks for the public and or private subnets.

- NAT gateways: Choose the number of AZs in which to create Network Address Translation (NAT) gateways. A NAT gateway is an AWS-managed service that enables EC2 instances in private subnets to send outbound traffic to the internet. Resources on the internet, however, cannot establish a connection with the instances. Note that there is cost associated with NAT gateways.
- VPC endpoints: Choose whether to create a VPC endpoint for Amazon S3. A VPC endpoint enables you to privately connect your VPC to supported AWS services and VPC endpoint services powered by PrivateLink without requiring an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC do not require public IP addresses to communicate with resources in the service.
- DNS options: Choose the domain name resolution options for the EC2 instances launched into this VPC.
  - Enable DNS hostnames: Enables hostnames to be provisioned for EC2 instance public IPv4 addresses.
  - Enable DNS resolution: Enables hostnames to be provisioned for EC2 instance public IPv4 addresses and enables domain name resolution of the hostnames.

5. In the Preview pane, you can visualize the relationships between resources inside a VPC. Solid lines represent relationships between resources. Dotted lines represent network traffic to NAT gateways, internet gateway and gateway endpoints. Once you create the VPC, you can visualize the resources in your VPC in this format at any time using the Resource map tab.
6. Choose Create VPC.

Create a VPC only

Follow the steps in this section to create only a VPC and no additional resources.

To create a VPC only

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose Your VPCs, Create VPC.
3. Under Resources to create, choose VPC only.
4. Specify the following VPC details as needed.

- Name tag: Optionally provide a name for your VPC. Doing so creates a tag with a key of Name and the value that you specify.

- IPv4 CIDR block: Specify an IPv4 CIDR block (or IP address range) for your VPC. Choose one of the following options:
  - IPv4 CIDR manual input: Manually input an IPv4 CIDR. The CIDR block size must have a size between /16 and /28. We recommend that you specify a CIDR block from the private (non-publicly routable) IP address ranges as specified in RFC 1918; for example, 10.0.0.0/16, or 192.168.0.0/16 .

    You can specify a range of publicly routable IPv4 addresses. However, we currently do not support direct access to the internet from publicly routable CIDR blocks in a VPC. Windows instances cannot boot correctly if launched into a VPC with ranges from 224.0.0.0 to 255.255.255.255 (Class D and Class E IP address ranges).
  - IPAM-allocated IPv4 CIDR block: If there is an Amazon VPC IP Address Manager (IPAM) IPv4 address pool available in this Region, you can get a CIDR from an IPAM pool. If you select an IPAM pool, the size of the CIDR is limited by the allocation rules on the IPAM pool (allowed minimum, allowed maximum, and default).
- IPv6 CIDR block: Optionally associate an IPv6 CIDR block with your VPC. Choose one of the following options, and then choose Select CIDR:
  - No IPv6 CIDR block: No IPv6 CIDR will be provisioned for this VPC.
  - IPAM-allocated IPv6 CIDR block: If there is an Amazon VPC IP Address Manager (IPAM) IPv6 address pool available in this Region, you can get a CIDR from an IPAM pool. If you select an IPAM pool, the size of the CIDR is limited by the allocation rules on the IPAM pool (allowed minimum, allowed maximum, and default).
  - Amazon-provided IPv6 CIDR block: Requests an IPv6 CIDR block from an Amazon pool of IPv6 addresses. For Network Border Group, select the group from which AWS advertises IP addresses. Amazon provides a fixed IPv6 CIDR block size of /56. You cannot configure the size of the IPv6 CIDR that Amazon provides.
  - IPv6 CIDR owned by me: (BYOIP) Allocates an IPv6 CIDR block from your IPv6 address pool. For Pool, choose the IPv6 address pool from which to allocate the IPv6 CIDR block.
- Tenancy: Choose the tenancy option for this VPC.
  - Select Default to ensure that EC2 instances launched in this VPC use the EC2 instance tenancy attribute specified when the EC2 instance is launched.
  - Select Dedicated to ensure that EC2 instances launched in this VPC are run on dedicated tenancy instances regardless of the tenancy attribute specified at launch.

    Note

    If your AWS Outposts require private connectivity, you must select Default.

Tags: Add optional tags on the VPC. A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

5. Choose Create VPC.

Alternatively, you can use a command line tool.

To create a VPC using a command line tool

- create-vpc (AWS CLI)
- New-EC2Vpc (AWS Tools for Windows PowerShell)

To describe a VPC using a command line tool

- describe-vpcs (AWS CLI)
- Get-EC2Vpc (AWS Tools for Windows PowerShell)

After you have created a VPC, you can create subnets.

View your VPC details

Use the following steps to view the details about your VPCs.

To view VPC details using the console

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose VPCs.
3. Select the VPC, and then choose View Details to see the configuration details of your VPC.

To describe a VPC using a command line tool

- describe-vpcs (AWS CLI)
- Get-EC2Vpc (AWS Tools for Windows PowerShell)

To view all of your VPCs across Regions

Open the Amazon EC2 Global View console at https://console.aws.amazon.com/ec2globalview/home.

Visualize the resources in your VPC

Use the following steps to see a visual representation of the resources in your VPC using the Resource map tab. The resource map shows relationships between resources inside a VPC and how traffic flows from subnets to NAT gateways, internet gateway and gateway endpoints.

You can use the resource map to understand the architecture of a VPC, see how many subnets it has in it, which subnets are associated with which route tables, and which route tables have routes to NAT gateways, internet gateways, and gateway endpoints.

You can also use the resource map to spot undesirable or incorrect configurations, such as private subnets disconnected from NAT gateways or private subnets with a route directly to the internet gateway. You can choose resources within the resource map, such as route tables, and edit the configurations for those resources.

To visualize the resources in your VPC

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose VPCs.
3. Select the VPC, and then choose Resource map to view a visualization of the resources in your VPC.
4. Choose Show details to view details for each resource.

- VPC: The VPC ID and IP address CIDRs assigned to the VPC.
- Subnets: The subnets in your VPC, the subnet IDs, and the IP address CIDRs assigned to the subnets.
- Route tables: The route tables in your VPC, the route table IDs, the subnet associations, and the number of entries in the route tables.
- Network functions: The internet gateways, NAT gateways, and gateway endpoints in your VPC. You can view details related to the each connection, like if an internet gateway enables your public subnets to connect to the internet or a VPC endpoint enables instances in your private subnet to connect privately to Amazon S3.

5. Hover over a resource to see the relationship between the resources. Solid lines represent relationships between resources. Dotted lines represent network traffic to network connections.

To describe a VPC using a command line tool

- describe-vpcs (AWS CLI)
- Get-EC2Vpc (AWS Tools for Windows PowerShell)

To view all of your VPCs across Regions

Open the Amazon EC2 Global View console at https://console.aws.amazon.com/ec2globalview/home.

What is AWS Elastic Beanstalk?

Amazon Web Services (AWS) comprises over one hundred services, each of which exposes an area of functionality. While the variety of services offers flexibility for how you want to manage your AWS infrastructure, it can be challenging to figure out which services to use and how to provision them.

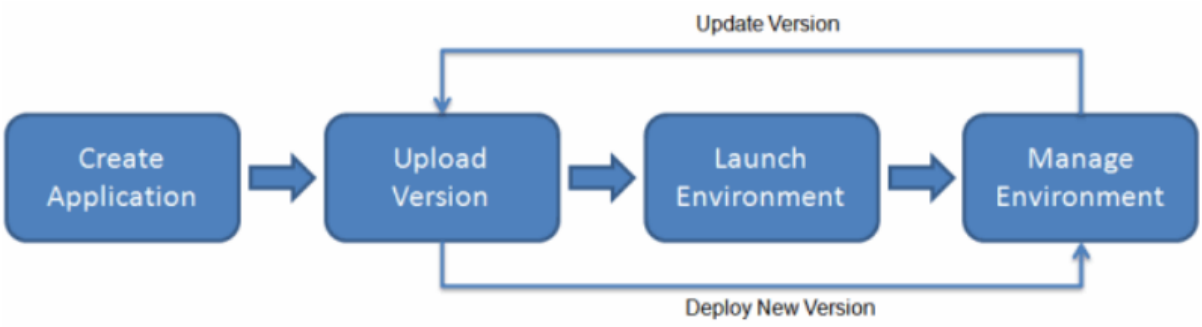
With Elastic Beanstalk, you can quickly deploy and manage applications in the AWS Cloud without having to learn about the infrastructure that runs those applications. Elastic Beanstalk reduces management complexity without restricting choice or control. You simply upload your application, and Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring.

Elastic Beanstalk supports applications developed in Go, Java, .NET, Node.js, PHP, Python, and Ruby. When you deploy your application, Elastic Beanstalk builds the selected supported platform version and provisions one or more AWS resources, such as Amazon EC2 instances, to run your application.

You can interact with Elastic Beanstalk by using the Elastic Beanstalk console, the AWS Command Line Interface (AWS CLI), or **eb**, a high-level CLI designed specifically for Elastic Beanstalk.

You can also perform most deployment tasks, such as changing the size of your fleet of Amazon EC2 instances or monitoring your application, directly from the Elastic Beanstalk web interface (console).

To use Elastic Beanstalk, you create an application, upload an application version in the form of an application source bundle (for example, a Java .war file) to Elastic Beanstalk, and then provide some information about the application. Elastic Beanstalk automatically launches an environment and creates and configures the AWS resources needed to run your code. After your environment is launched, you can then manage your environment and deploy new application versions. The following diagram illustrates the workflow of Elastic Beanstalk.



After you create and deploy your application, information about the application—including metrics, events, and environment status—is available through the Elastic Beanstalk console, APIs, or Command Line Interfaces, including the unified AWS CLI.

Setting up: Create an AWS account

Step 1: Create an example application

Create an application and an environment

To create your example application, you'll use the **Create application** console wizard. It creates an Elastic Beanstalk application and launches an environment within it. An environment is the collection of AWS resources required to run your application code.

To create an example application (in the old console)

1. Open the Elastic Beanstalk console using this link: https://console.aws.amazon.com/elasticbeanstalk/home#/gettingStarted?applicationName=getting-started-app
2. Optionally add application tags.
3. For **Platform**, choose a platform, and then choose **Create application**.

To deploy and run the example application on AWS resources, Elastic Beanstalk takes the following actions. They take about five minutes to complete.

1. Creates an Elastic Beanstalk application named **getting-started-app**.
2. Launches an environment named **GettingStartedApp-env** with these AWS resources:
    - An Amazon Elastic Compute Cloud (Amazon EC2) instance (virtual machine)
    - An Amazon EC2 security group
    - An Amazon Simple Storage Service (Amazon S3) bucket
    - Amazon CloudWatch alarms
    - An AWS CloudFormation stack

- A domain name

3. Creates a new application version named **Sample Application**. This is the default Elastic Beanstalk example application file.
4. Deploys the code for the example application to the **GettingStartedApp-env** environment.

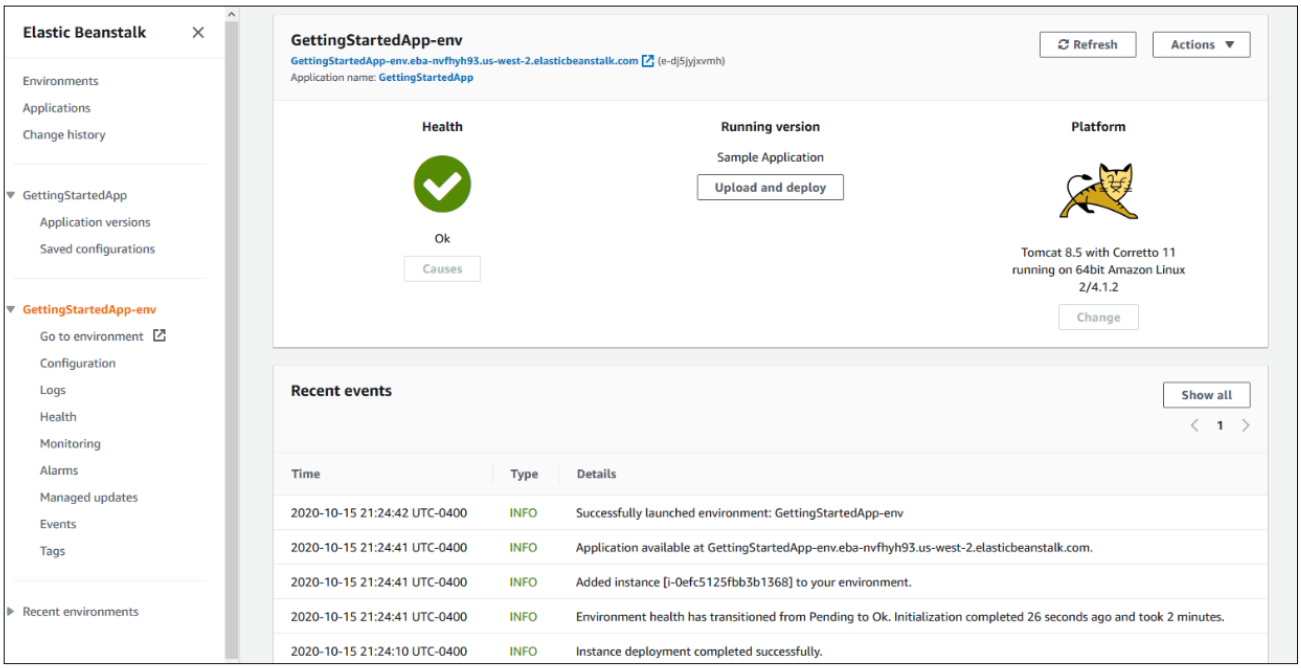
During the environment creation process, the console tracks progress and displays events.

When all of the resources are launched and the EC2 instances running the application pass health checks, the environment's health changes to Ok. You can now use your web application's website.

Step 2: Explore your environment

The environment overview pane shows top level information about your environment. This includes its name, its URL, its current health status, the name of the currently deployed application version, and the platform version that the application is running on. Below the overview pane you can see the five most recent environment events.

88



While Elastic Beanstalk creates your AWS resources and launches your application, the environment is in a Pending state. Status messages about launch events are continuously added to the overview.

The environment's **URL** is located at the top of the overview, below the environment name. This is the URL of the web application that the environment is running. Choose this URL to get to the example application's *Congratulations* page. The navigation pane on the left lists a **Go to environment** link that launches the same application page.

The other links on the left navigation pane contain more detailed information about your environment and provide access to additional features:

- **Configuration** – Shows the Configuration overview page. This page displays a summary of environment configuration option values, grouped by category.
- **Logs** – Retrieve and download logs from the Amazon EC2 in your environment. You can retrieve full logs or recent activity. The retrieved logs are available for 15 minutes.
- **Health** – Shows the status of and detailed health information about the Amazon EC2 instances running your application.
- **Monitoring** – Shows statistics for the environment, such as average latency and CPU utilization. You can use this page to create alarms for the metrics that you are monitoring.
- **Alarms** – Shows the alarms that you configured for environment metrics. You can add, modify or delete alarms on this page.
- **Managed updates** – Shows information about upcoming and completed managed platform updates and instance replacement.
- **Events** – Shows information or error messages from the Elastic Beanstalk service and from other services whose resources this environment uses.
- **Tags** – Shows environment tags and allows you to manage them. Tags are key-value pairs that are applied to your environment.

Step 3: Deploy a new version of your application

Periodically, you might need to deploy a new version of your application. You can deploy a new version at any time, as long as no other update operations are in progress on your environment.

The application version that you started this tutorial with is called **Sample Application**.

To update your application version

1. Download the sample application that matches your environment's platform. Use one of the following applications.
    - **Docker** – docker.zip
    - **Multicontainer Docker** – docker-multicontainer-v2.zip
    - **Preconfigured Docker (Glassfish)** – docker-glassfish-v1.zip
    - **Go** – go.zip
    - **Corretto** – corretto.zip
    - **Tomcat** – tomcat.zip
    - **.NET Core on Linux** – dotnet-core-linux.zip

- **.NET** – dotnet-asp-v1.zip
- **Node.js** – nodejs.zip
- **PHP** – php.zip
- **Python** – python.zip
- **Ruby** – ruby.zip

2. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
3. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
4. On the environment overview page, choose **Upload and deploy**.
5. Choose **Choose file**, and then upload the sample application source bundle that you downloaded. The console automatically fills in the **Version label** with a new unique label. If you type in your own version label, ensure that it's unique.
6. Choose **Deploy**.

Step 4: Configure your environment

You can configure your environment to better suit your application. For example, if you have a compute-intensive application, you can change the type of Amazon Elastic Compute Cloud (Amazon EC2) instance that is running your application. To apply configuration changes, Elastic Beanstalk performs an environment update.

Some configuration changes are simple and happen quickly. Some changes require deleting and recreating AWS resources, which can take several minutes. When you change configuration settings, Elastic Beanstalk warns you about potential application downtime.

Step 5: Clean up

To delete the application and all associated resources

1. Delete all application versions.

a. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
b. In the navigation pane, choose **Applications**, and then choose **getting-started-app**.
c. In the navigation pane, find your application's name and choose **Application versions**.
d. On the **Application versions** page, select all application versions that you want to delete.
e. Choose **Actions**, and then choose **Delete**.
f. Turn on **Delete versions from Amazon S3**.
g. Choose **Delete**, and then choose **Done**.

2. Terminate the environment.

a. In the navigation pane, choose **getting-started-app**, and then choose **GettingStartedApp-env** in the environment list.
b. Choose **Actions**, and then choose **Terminate Environment**.
c. Confirm that you want to terminate **GettingStartedApp-env** by typing the environment name, and then choose **Terminate**.

3. Delete the getting-started-app application.

a. In the navigation pane, choose the **getting-started-app**.

b. Choose **Actions**, and then choose **Delete application**.

c. Confirm that you want to delete **getting-started-app** by typing the application name, and then choose **Delete**.

Conclusion:

Thus, we studied how to deploy/host web application on AWS VPC or AWS Elastic Beanstalk.