

**Guru Nanak Dev  
Engineering  
College, Ludhiana**



---

**Department of Information Technology**

**ADVANCED WEB TECHNOLOGIES  
LABORATORY LPEIT-102 PRACTICAL FILE  
(5<sup>th</sup> Sem 2025 batch)**

**Student Details**

**Arshpreet Singh**

**2104478 (URN)**

**2121021(CRN)**

**I.T A1**

**Submitted to:**

**Dr. Akshay Girdhar**

# INDEX

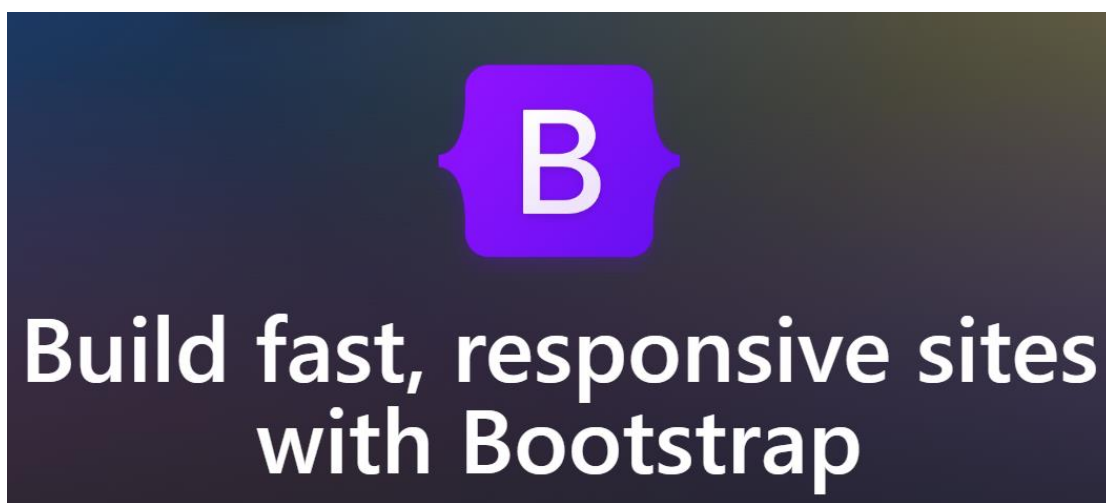
S.NO	PRACTICAL	DATE	PAGE	SIGN
1	To install and setup the HTML5 based Bootstrap framework and to deploy basic HTML elements using Bootstrap CSS.			
2	To understand and deploy the multicolumn grid layout of Bootstrap.			
3	To deploy different types of buttons, progress bars, modals and navigation bars using Bootstrap.			
4	To create and setup the Git repository on Bitbucket or Github using SSH.			
5	To perform push, c lone and patch operation to Git repository			
6	To install and setup the CodeIgniter Framework and to understand its MVC architecture.			
7	To construct a simple login page web application to authenticate users using CodeIgniter Framework and also perform CRUD operations.			
8	To install and setup, configure the Laravel Framework.			
9	To construct the any simple web application using Laravel Framework			

# Experiment: 1

To install and setup the HTML5 based Bootstrap framework and to deploy basic HTML elements using Bootstrap CSS.

## Introduction:

Bootstrap is a popular open-source front-end framework for building responsive and visually appealing websites and web applications. It was originally created by Twitter and is now maintained by the Bootstrap community. Bootstrap simplifies web development by providing a collection of pre-designed and pre-built HTML, CSS, and JavaScript components and utilities that you can easily incorporate into your projects.



## Method: 1

### Including via CDN

Get started by including Bootstrap's production-ready CSS and JavaScript via CDN without the need for any build steps.

Step 1) **Create a new index.html file in your project root.** Include the <meta name="viewport"> tag as well for proper responsive behaviour in mobile devices.

## Get started with Bootstrap

[View on GitHub](#)

Bootstrap is a powerful, feature-packed frontend toolkit. Build anything—from prototype to production—in minutes.

### Quick start

Get started by including Bootstrap's production-ready CSS and JavaScript via CDN without the need for any build steps. See it in practice with this [Bootstrap CodePen demo](#).

Step 2) **Include Bootstrap's CSS and JS.** Place the <link> tag in the <head> for our CSS, and the <script> tag for our JavaScript bundle (including Popper for positioning dropdowns, poppers, and tooltips) before the closing </body>.

Step 3) **Hello, world!** Open the page in your browser of choice to see your Bootstrapped page. Now you can start building with Bootstrap by creating your own layout.

## CDN links

Description	URL
CSS	<a href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/CSS/bootstrap.min.CSS">https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/CSS/bootstrap.min.CSS</a>
JS	<a href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/JS/bootstrap.bundle.min.JS">https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/JS/bootstrap.bundle.min.JS</a>

## Method: 2

### Downloading Compiled CSS and JS

Download ready-to-use compiled code for **Bootstrap v5.3.1** to easily drop into your project.

# Download

[View on GitHub](#)

Download Bootstrap to get the compiled CSS and JavaScript, source code, or include it with your favorite package managers like npm, RubyGems, and more.

## Compiled CSS and JS

Download ready-to-use compiled code for **Bootstrap v5.3.1** to easily drop into your project, which includes:

- Compiled and minified CSS bundles (see [CSS files comparison](#))
- Compiled and minified JavaScript plugins (see [JS files comparison](#))

This doesn't include documentation, source files, or any optional JavaScript dependencies like Popper.

[Download](#)

Below is the file structure of minified file system for CSS and JavaScript. Bootstrap includes a handful of options for including some or all of our compiled CSS and JS.

Compiled and minified CSS bundles	Compiled and minified JS plugins
bootstrap.CSS bootstrap.min.CSS bootstrap.rtl.CSS bootstrap.rtl.min.CSS bootstrap-grid.CSS  bootstrap-grid.rtl.CSS bootstrap-grid.min.CSS bootstrap-grid.rtl.min.CSS bootstrap-utilities.CSS bootstrap-utilities.rtl.CSS bootstrap-utilities.min.CSS bootstrap-utilities.rtl.min.CSS	bootstrap.bundle.JS bootstrap.bundle.min.JS bootstrap.JS bootstrap.min.JS

### Method: 3

Downloading Source files for CSS and JS

Compile Bootstrap with your own asset pipeline by downloading our source Sass, JavaScript, and documentation files. This option requires some additional tooling:

- Sass compiler for compiling Sass source files into CSS files
- Autoprefixer for CSS vendor prefixing

Should you require our full set of build tools, they are included for developing Bootstrap and its docs, but they're likely unsuitable for your own purposes.

## Source files

Compile Bootstrap with your own asset pipeline by downloading our source Sass, JavaScript, and documentation files. This option requires some additional tooling:

- [Sass compiler](#) for compiling Sass source files into CSS files
- [Autoprefixer](#) for CSS vendor prefixing

Should you require our full set of [build tools](#), they are included for developing Bootstrap and its docs, but they're likely unsuitable for your own purposes.

[Download source](#)

## Containers

Containers are a fundamental building block of Bootstrap that contain, pad, and align your content within a given device or viewport.

## How they work

Containers are the most basic layout element in Bootstrap and are required when using our default grid system. Containers are used to contain, pad, and (sometimes) center the content within them. While containers can be nested, most layouts do not require a nested container.

Bootstrap comes with three different containers:

`.container`, which sets a max-width at each responsive breakpoint

`.container-{breakpoint}`, which is width: 100% until the specified breakpoint

`.container-fluid`, which is width: 100% at all breakpoints

## Default container

Our default `.container` class is a responsive, fixed-width container, meaning its max-width changes at each breakpoint.

```
<div class="container">  
  <!-- Content here -->  
</div>
```

## Responsive containers

Responsive containers allow you to specify a class that is 100% wide until the specified breakpoint is reached, after which we apply max-widths for each of the higher breakpoints. For example, `.container-sm` is 100% wide to start until the sm breakpoint is reached, where it will scale up with md, lg, xl, and xxl.

```
<div class="container-sm">100% wide until small breakpoint</div>  
<div class="container-md">100% wide until medium breakpoint</div>
```

## Fluid containers

Use `.container-fluid` for a full width container, spanning the entire width of the viewport.

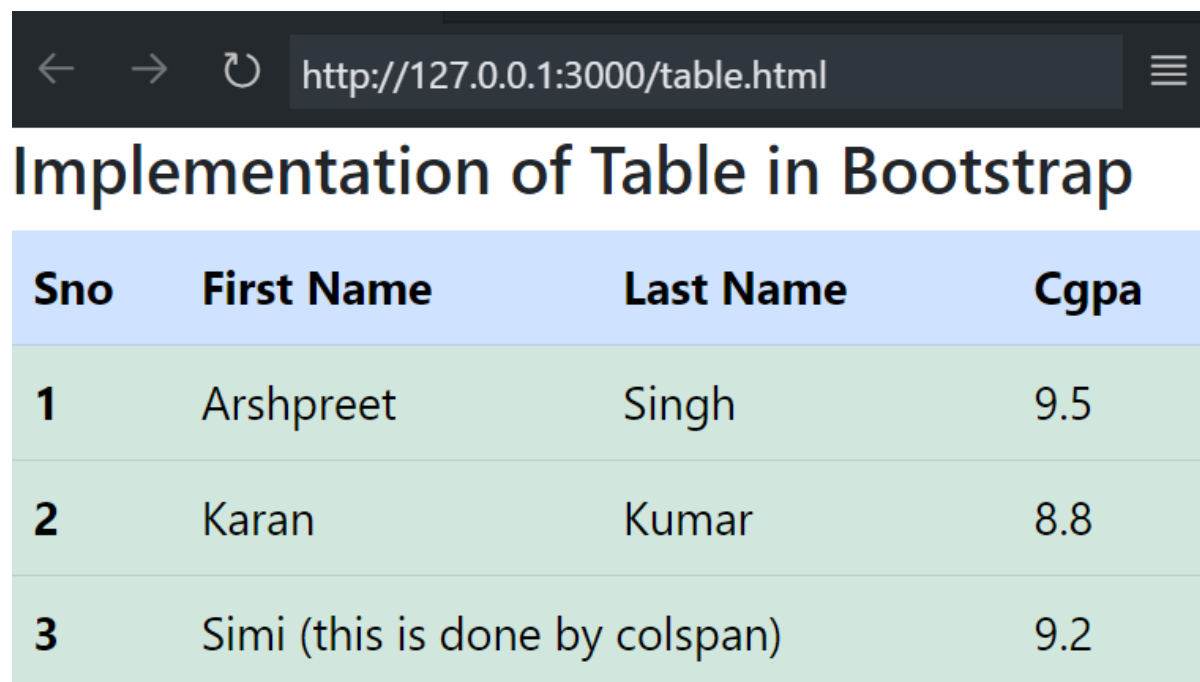
```
<div class="container-fluid">  
  ...  
</div>
```

## Tables

Documentation and examples for opt-in styling of tables (given their prevalent use in JavaScript plugins) with Bootstrap.

Due to the widespread use of `<table>` elements across third-party widgets like calendars and date pickers, Bootstrap's tables are **opt-in**. Add the base class `.table` to any `<table>`, then extend with our optional modifier classes or custom styles. All table styles are not inherited in Bootstrap, meaning any nested tables can be styled independent from the parent.

Implementation Sample:



The screenshot shows a web browser window with the address bar displaying `http://127.0.0.1:3000/table.html`. The page content features a table with the title "Implementation of Table in Bootstrap". The table has four columns: "Sno", "First Name", "Last Name", and "Cgpa". It contains three data rows. The first row shows "1", "Arshpreet", "Singh", and "9.5". The second row shows "2", "Karan", "Kumar", and "8.8". The third row shows "3", "Simi (this is done by colspan)", and "9.2".

Sno	First Name	Last Name	Cgpa
1	Arshpreet	Singh	9.5
2	Karan	Kumar	8.8
3	Simi (this is done by colspan)		9.2

Code:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.
css" rel="stylesheet" integrity="sha384-
4bw+aeP/YC94hEpVNVgiZdgIC5+VKNBQNGChEKQN+PtmoHDEXuppvnDJzQIu9"
crossorigin="anonymous">
  </head>
  <body>
```

```

<h3>Implementation of Table in Bootstrap</h3>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle
.min.js" integrity="sha384-
HwwvtgBNo3bZJJLYd8oVXjrBZt8cqVSpeBNS5n7C8IVInixGAoxmnlMuBnhbgrkm"
crossorigin="anonymous"></script>
<table class="table">
  <thead class="table-primary">
    <tr>
      <th scope="col">Sno</th>
      <th scope="col">First Name</th>
      <th scope="col">Last Name</th>
      <th scope="col">Cgpa</th>
    </tr>
  </thead>
  <tbody class="table-success">
    <tr>
      <th scope="row">1</th>
      <td>Arshpreet</td>
      <td>Singh</td>
      <td>9.5</td>
    </tr>
    <tr>
      <th scope="row">2</th>
      <td>Karan</td>
      <td>Kumar</td>
      <td>8.8</td>
    </tr>
    <tr>
      <th scope="row">3</th>
      <td colspan="2">Simi (this is done by colspan)</td>
      <td>9.2</td>
    </tr>
  </tbody>
</table>
</body>
</html>

```

## Forms

Examples and usage guidelines for form control styles, layout options, and custom components for creating a wide variety of forms.

Bootstrap's form controls expand on form styles with classes. Use these classes to opt into their customized displays for a more consistent rendering across browsers and devices.



Be sure to use an appropriate type attribute on all inputs (e.g., email for email address or number for numerical information) to take advantage of newer input controls like email verification, number selection, and more.

Implementation Sample:

## Implementation of Form

Student Email address

arshpreet@gndec.ac.in

Password

Branch

Select ▾

☐ Check box

Submit

Code:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1">
    <title>Bootstrap demo</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstra
p.min.css" rel="stylesheet" integrity="sha384-
4bw+/aepP/YC94hEpVNVgiZdgIC5+VKNBQNGCHeKRQN+PtmoHDEXuppvnDJzQIu9"
crossorigin="anonymous">
```

```

</head>
<body>
  <h3>Implementation of Form</h3>
  <br>
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.
bundle.min.js" integrity="sha384-
HwwvtgBNo3bZJJLYd8oVXjrBZt8cqVSpeBNS5n7C8IVInixGAoxmnlMuBnhbgrkm"
crossorigin="anonymous"></script>
  <form>
    <div class="mb-3">
      <label for="exampleInputEmail1" class="form-label">Student
Email address</label>
      <input type="email" class="form-control"
id="exampleInputEmail1" aria-describedby="emailHelp"
placeholder="arshpreet@gndec.ac.in">
    </div>
    <div class="mb-3">
      <label for="exampleInputPassword1" class="form-
label">Password</label>
      <input type="password" class="form-control"
id="exampleInputPassword1">
    </div>
    <div class="mb-3">
      <label for="exampleInputEmail1" class="form-
label">Branch</label>
      <div class="dropdown">
        <button class="btn btn-secondary dropdown-toggle"
type="button" data-bs-toggle="dropdown" aria-expanded="false">
          Select
        </button>
        <ul class="dropdown-menu">
          <li><a class="dropdown-item"
href="#">I.T.</a></li>
          <li><a class="dropdown-item"
href="#">C.S.E</a></li>
          <li><a class="dropdown-item"
href="#">others</a></li>
        </ul>
      </div>
    </div>

    <div class="mb-3 form-check">
      <input type="checkbox" class="form-check-input"
id="exampleCheck1">
      <label class="form-check-label" for="exampleCheck1">Check
box</label>
    </div>
  </form>

```

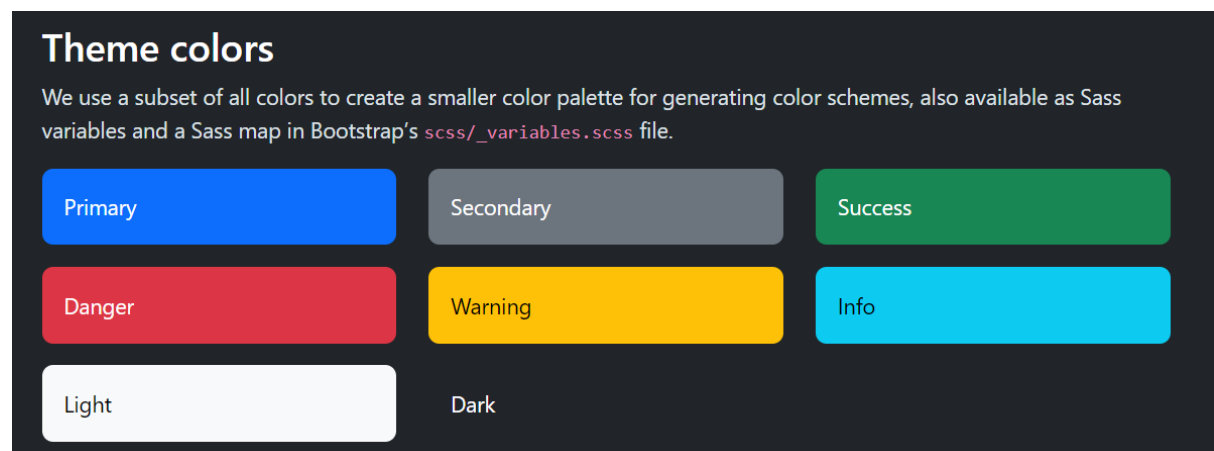
```

        <button type="submit" class="btn btn-
primary">Submit</button>
    </form>
</body>
</html>

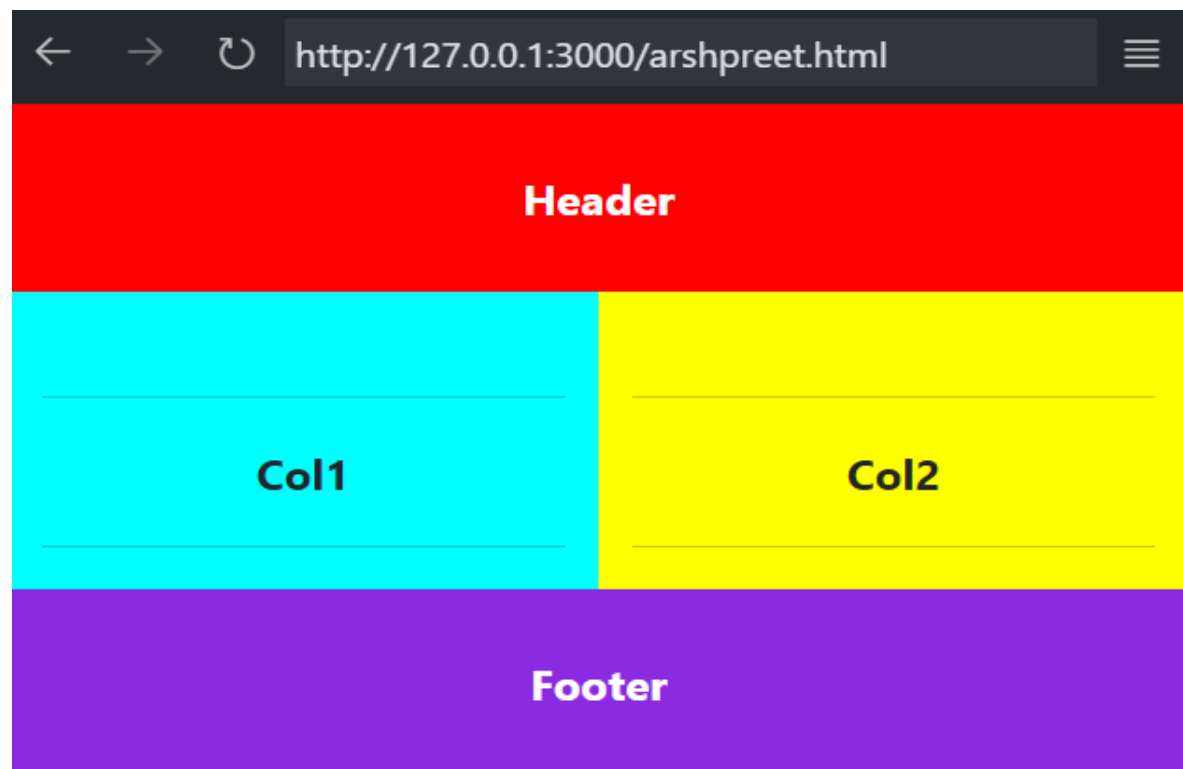
```

## Color System:

Bootstrap is supported by an extensive color system that themes our styles and components. This enables more comprehensive customization and extension for any project.



Below are the basic HTML elements that is implemented using Bootstrap Framework.



Code:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1">
    <title>Bootstrap demo</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstra
p.min.css" rel="stylesheet" integrity="sha384-
4bw+/aepP/YC94hEpVNVgiZdgIC5+VKNBQNGCHeKRQN+PtmoHDEXuppvnDJzQIu9"
crossorigin="anonymous">
  </head>
  <body>
    <div class="container" style="text-align: center;background-
color: red;color: white;">
      <br><b>Header</b><br><br>
    </div>
    <div class = "container">
      <div class = "row">
        <div class = "col-6" align="center" style="background-
color: aqua;"><br><hr><b>Col1</b><br><hr></div>
        <div class = "col-6" align="center" style="background-
color: yellow;"><br><hr><b>Col2</b><br><hr></div>
      </div>
    </div>
    <div class="container" style="text-align: center;background-
color:blueviolet;color: white;">
      <br><b>Footer</b><br><br>
    </div>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.
bundle.min.js" integrity="sha384-
HwwvtgBNo3bZJJLYd8oVXjrBZt8cqVSpeBNS5n7C8IVInixGAoxmnlMuBnhbgrkm"
crossorigin="anonymous"></script>
  </body>
</html>
```

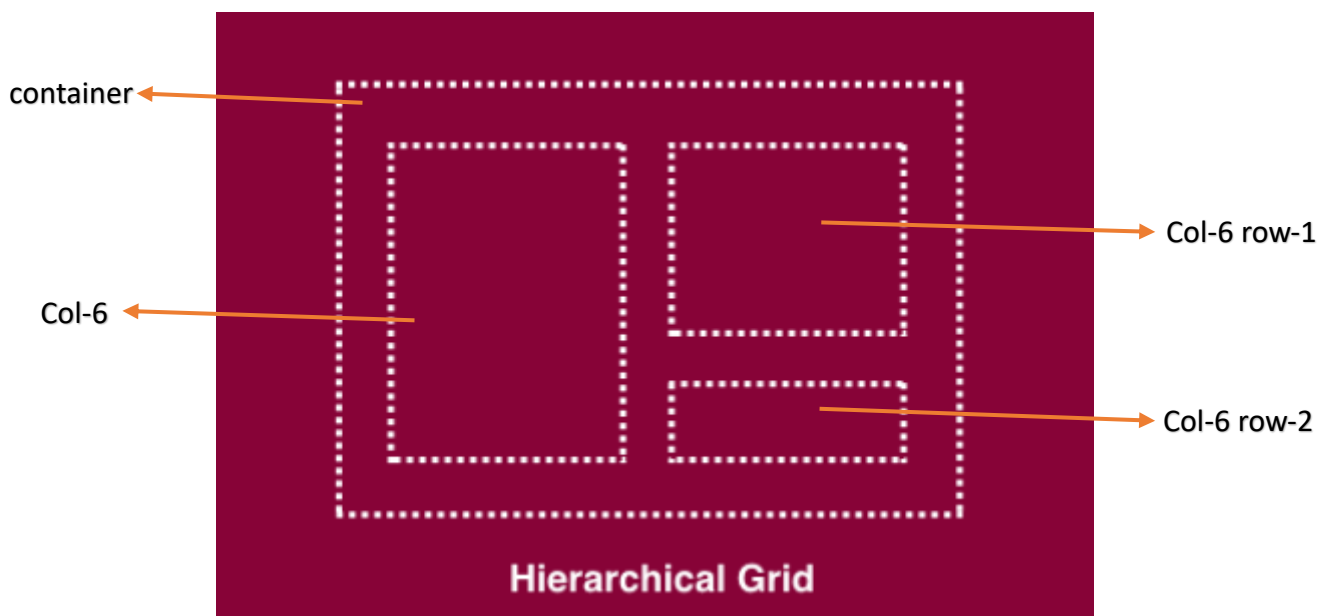
# Experiment: 2

To understand and deploy the multicolumn grid layout of Bootstrap.

## Grid system

Use our powerful mobile-first flexbox grid to build layouts of all shapes and sizes thanks to a twelve-column system, six default responsive tiers, Sass variables and mixins, and dozens of predefined classes. Bootstrap's grid system uses a series of containers, rows, and columns to layout and align content. It's built with flexbox and is fully responsive. Below is an example and an in-depth explanation for how the grid system comes together.

```
<div class="container text-center">  
  <div class="row">  
    <div class="col">  
      Column  
    </div>  
    <div class="col">  
      Column  
    </div>  
    <div class="col">  
      Column</div></div></div>
```



## How Grid System works:

- Containers provide a means to center and horizontally pad your site's contents. Use `.container` for a responsive pixel width or `.container-fluid` for width: 100% across all viewport and device sizes.
- Rows are wrappers for columns. Each column has horizontal padding (called a gutter) for controlling the space between them. This padding is then counteracted on the rows with negative margins. This way, all the content in your columns is visually aligned down the left side.
- In a grid layout, content must be placed within columns and only columns may be immediate children of rows.
- Thanks to flexbox, grid columns without a specified width will automatically layout as equal width columns. For example, four instances of `.col-sm` will each automatically be 25% wide from the small breakpoint and up. See the `auto-layout columns` section for more examples.
- Column classes indicate the number of columns you'd like to use out of the possible 12 per row. So, if you want three equal-width columns across, you can use `.col-4`.
- Column widths are set in percentages, so they're always fluid and sized relative to their parent element.
- Columns have horizontal padding to create the gutters between individual columns, however, you can remove the margin from rows and padding from columns with `.no-gutters` on the `.row`.
- To make the grid responsive, there are five grid breakpoints, one for each responsive breakpoint: all breakpoints (extra small), small, medium, large, and extra large.
- Grid breakpoints are based on minimum width media queries, meaning **they apply to that one breakpoint and all those above it** (e.g., `.col-sm-4` applies to small, medium, large, and extra large devices, but not the first xs breakpoint).
- You can use predefined grid classes (like `.col-4`) or Sass mixins for more semantic markup.
- Be aware of the limitations and bugs around flexbox, like the inability to use some HTML elements as flex containers.

## Grid options

While Bootstrap uses ems or rems for defining most sizes, pxs are used for grid breakpoints and container widths. This is because the viewport width is in pixels and does not change with the font size. See how aspects of the Bootstrap grid system work across multiple devices with a handy table.

	<b>Extra small</b> <576px	<b>Small</b> ≥576px	<b>Medium</b> ≥768px	<b>Large</b> ≥992px	<b>Extra large</b> ≥1200px
<b>Max container width</b>	None (auto)	540px	720px	960px	1140px
<b>Class prefix</b>	.col-	.col-sm-	.col-md-	.col-lg-	.col-xl-
<b># of columns</b>	12				
<b>Gutter width</b>	30px (15px on each side of a column)				
<b>Nestable</b>	Yes				
<b>Column ordering</b>	Yes				

### Auto-layout columns

Utilize breakpoint-specific column classes for easy column sizing without an explicit numbered class like .col-sm-6.

### Equal-width

For example, here are two grid layouts that apply to every device and viewport, from xs to xl. Add any number of unit-less classes for each breakpoint you need and every column will be

```

<div class="container">
  <div class="row">
    <div class="col">
      </div>
    <div class="col">
      </div>
    </div>
  <div class="row">
    <div class="col">
      </div>
    <div class="col">
      </div>
    <div class="col">
      </div>
    </div>
  </div> </div>

```

Equal-width columns can be broken into multiple lines, but there was a Safari flexbox bug that prevented this from working without an explicit flex-basis or border. There are workarounds for older browser versions, but they shouldn't be necessary if you're up-to-date.

```
<div class="container">
  <div class="row">
    <div class="col">Column</div>
    <div class="col">Column</div>
    <div class="w-100"></div>
    <div class="col">Column</div>
    <div class="col">Column</div>
  </div>
</div>
```

### Setting one column width

Auto-layout for flexbox grid columns also means you can set the width of one column and have the sibling columns automatically resize around it. You may use predefined grid classes (as shown below), grid mixins, or inline widths. Note that the other columns will resize no matter the width of the center column.

```
<div class="container">
  <div class="row">
    <div class="col"> 1 of 3 </div>
    <div class="col-6"> 2 of 3 (wider) </div>
    <div class="col"> </div> </div>
  <div class="row">
    <div class="col"> 1 of 3 </div>
    <div class="col-5"> 2 of 3 (wider) </div>
    <div class="col"> 3 of 3 </div>
  </div> </div>
```

### Variable width content

Use col-{breakpoint}-auto classes to size columns based on the natural width of their content.

```
<div class="container">
```



```

<div class="row justify-content-md-center">
  <div class="col col-lg-2">
    1 of 3 </div>
  <div class="col-md-auto">
    Variable width content </div>
<div class="col col-lg-2"> 3 of 3
  </div> </div>

<div class="row">
  <div class="col"> 1 of 3 </div>
  <div class="col-md-auto">
    Variable width content </div>
<div class="col col-lg-2"> 3 of 3 </div>
  </div> </div>

```

### Equal-width multi-row

Create equal-width columns that span multiple rows by inserting a `.w-100` where you want the columns to break to a new line. Make the breaks responsive by mixing the `.w-100` with some responsive display utilities.

```

<div class="row">
  <div class="col">col</div>
  <div class="col">col</div>
  <div class="w-100"></div>
  <div class="col">col</div>
  <div class="col">col</div>
  </div>

```

### Responsive classes

Bootstrap's grid includes five tiers of predefined classes for building complex responsive layouts. Customize the size of your columns on extra small, small, medium, large, or extra-large devices however you see fit.

### All breakpoints

For grids that are the same from the smallest of devices to the largest, use the .col and .col-\* classes. Specify a numbered class when you need a particularly sized column; otherwise, feel free to stick to .col.

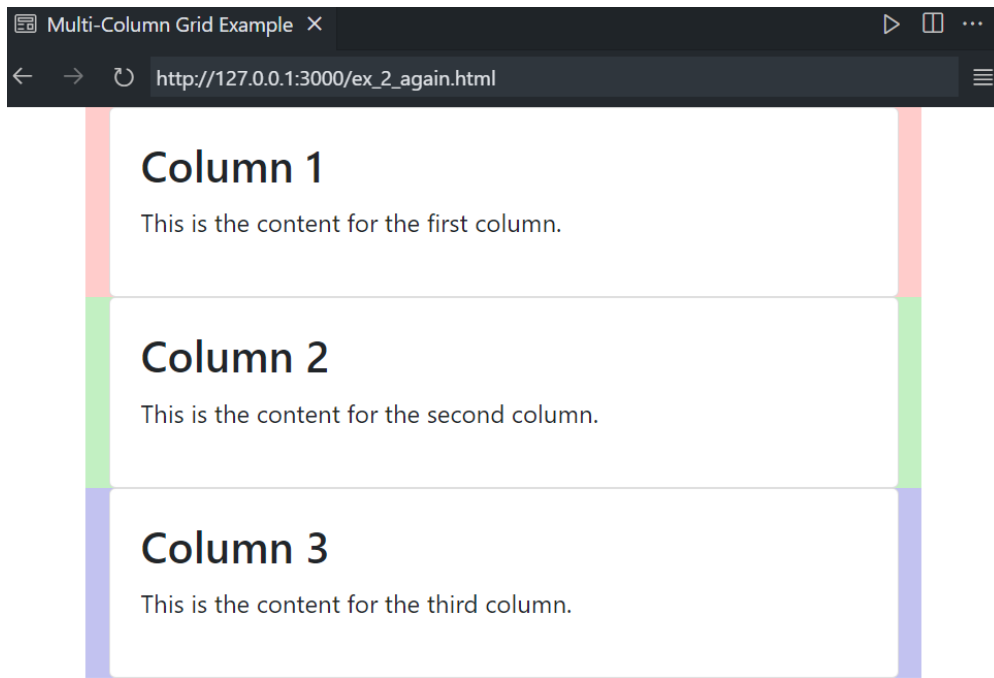
```
<div class="row">
  <div class="col">col</div>
  <div class="col">col</div>
  <div class="col">col</div>
  <div class="col">col</div>
  <div class="col">col</div>
</div>

<div class="row">
  <div class="col-8">col-8</div>
  <div class="col-4">col-4</div>
</div>
```

### Output Images:

The images presented below showcase the output generated by a multicolumn grid system. Each visual representation illustrates the effectiveness and structure of the grid system in organizing content across multiple columns. These images provide a clear glimpse into the functionality and layout of the multicolumn grid, highlighting its visual appeal and practical application in design or data presentation.





## References

1. Getbootstrap – display: <https://getbootstrap.com/docs/4.5/utilities/display/>
2. Grid system from Javatpoint: <https://www.javatpoint.com/bootstrap-grid>
3. Grid layout from geetbootstrap: <https://getbootstrap.com/docs/4.0/layout/grid/>

# Experiment: 3

To deploy different types of buttons, progress bars, modals and navigation bars using Bootstrap.

## Deployment of Bootstrap Components - Introduction

Bootstrap is a popular front-end framework that simplifies the process of creating responsive and visually appealing web applications. This practical exercise aims to demonstrate how to deploy various Bootstrap components, including buttons, progress bars, modals, and navigation bars, in a web project. These components play a crucial role in enhancing user experience and interactivity on web applications.

## Materials Required

- A computer with a text editor and web browser
- Internet connection (for linking Bootstrap via CDN)

## Procedure

### Step 1: Setting up Bootstrap

1. Begin by creating an HTML document (e.g., `index.html`) using your preferred text editor.
2. Link the Bootstrap CSS and JavaScript files by adding the following code within the

`<head>` section of your HTML document:`

`<!-- Link to Bootstrap CSS -->`

`<link rel="stylesheet"  
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">`

`<!-- Link to Bootstrap JS and Popper.js for some Bootstrap features -->`

`<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>`

`<script  
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js">  
</script>`

## Step 2: Deploying Buttons

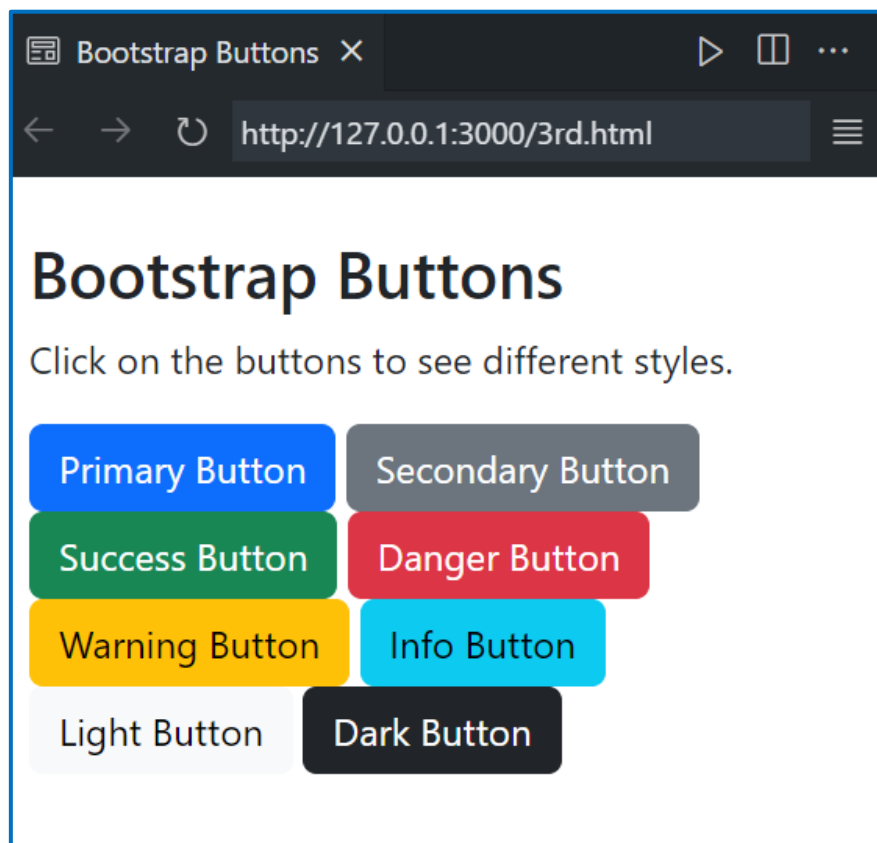
1. Create buttons using Bootstrap classes. For example:

```
<button class="btn btn-primary">Primary Button</button>
```

```
<button class="btn btn-secondary">Secondary Button</button>
```

```
<button class="btn btn-success">Success Button</button>
```

2. Customize the buttons by experimenting with different classes and styles provided by Bootstrap.



## Step 3: Creating Progress Bars

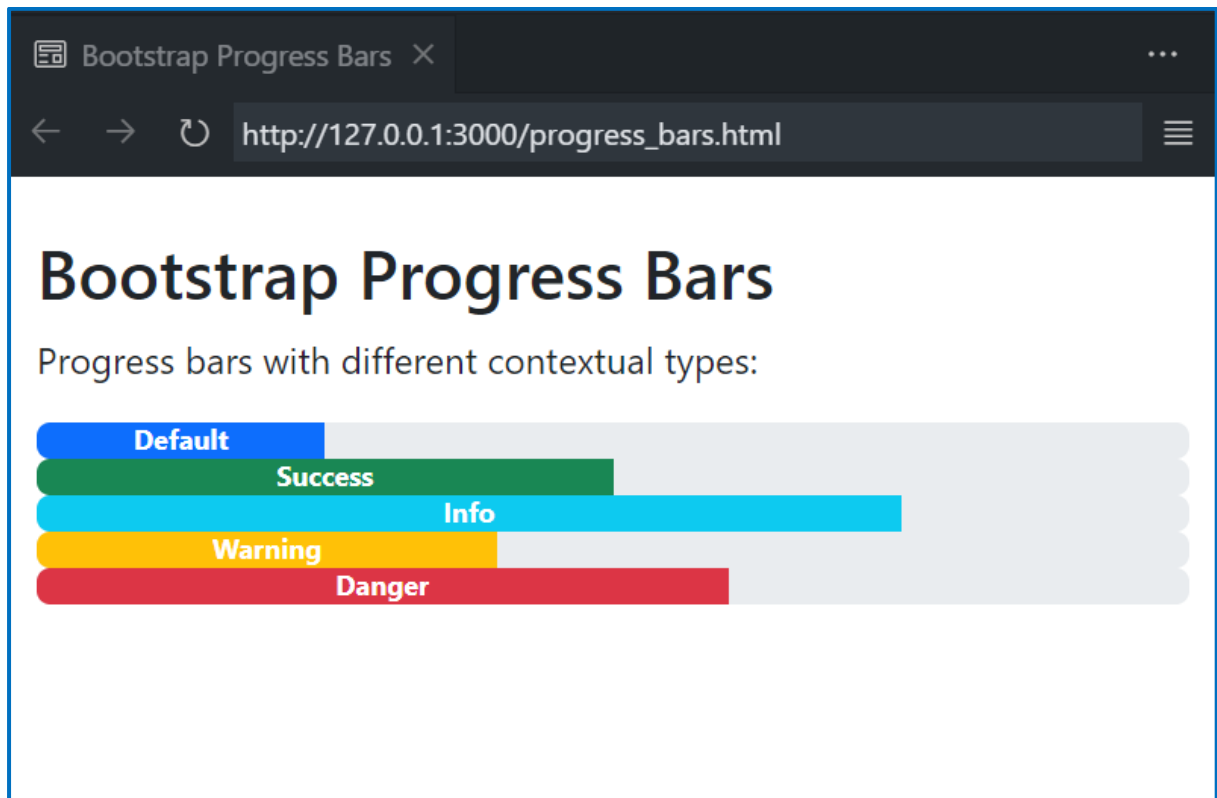
1. Develop a progress bar using Bootstrap's classes. For instance:

```
html
```

```
<div class="progress">
```

```
<div class="progress-bar" style="width: 70%;" aria-valuenow="70" aria-valuemin="0" aria-  
valuemax="100"></div>  
</div>
```

2. Adjust the progress bar's width and appearance to suit your application's needs.



#### Step 4: Designing Modals

1. Create a modal dialog using Bootstrap. This modal can display additional information or perform specific actions when triggered. Example modal code:

```
<!-- Button to trigger modal -->  
  
<button type="button" class="btn btn-primary" data-bs-toggle="modal" data-bs-  
target="#myModal">  
    Open Modal  
</button>  
  
<!-- Modal -->
```

```

<div class="modal fade" id="myModal" tabindex="-1" role="dialog" aria-
labelledby="exampleModalLabel" aria-hidden="true">

  <div class="modal-dialog" role="document">

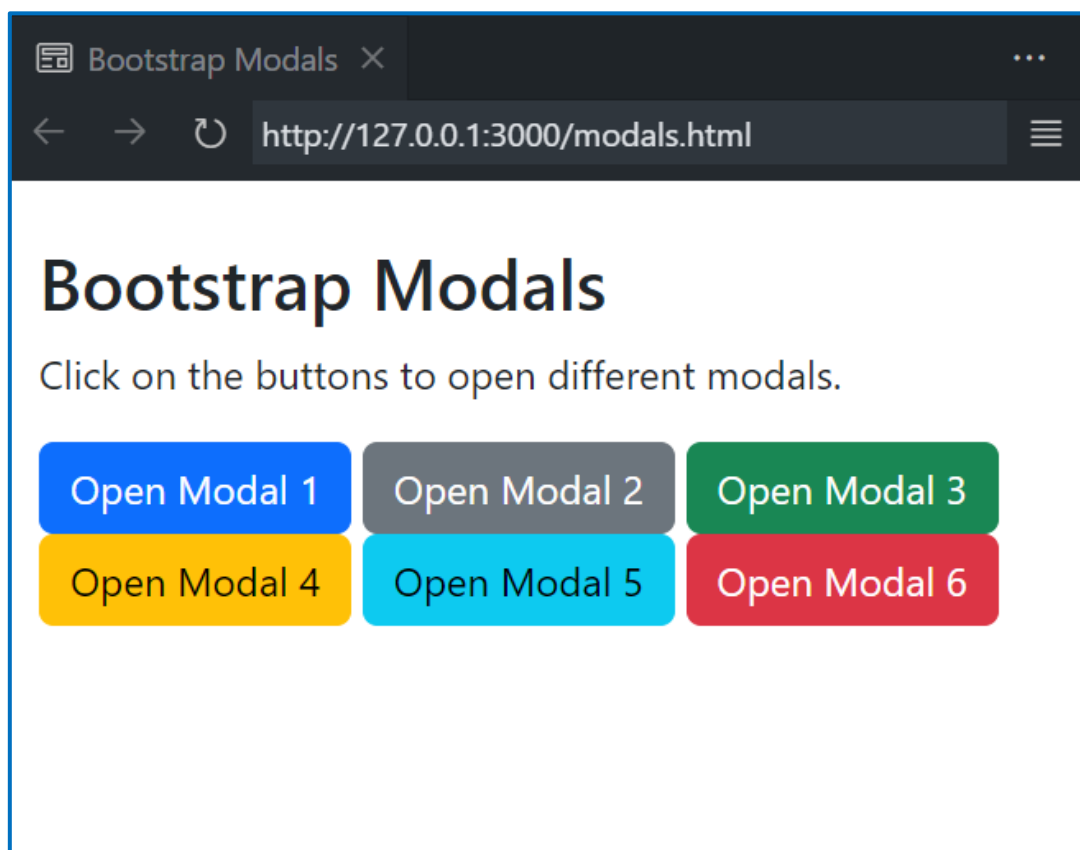
    <!-- Modal content goes here -->

  </div>

</div>

```

2. Customize the modal's content and appearance based on your project's requirements.



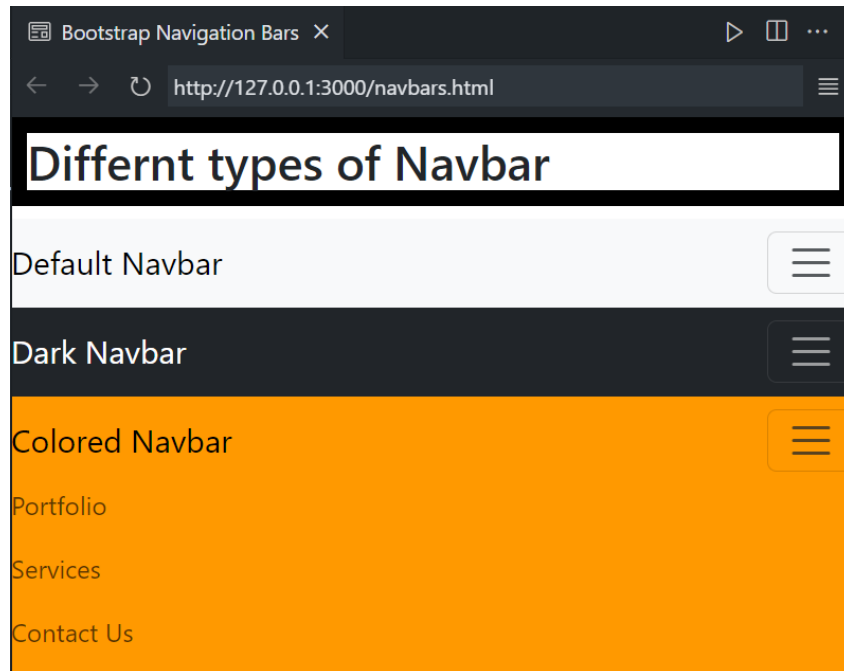
## Step 5: Implementing Navigation Bars

1. Create a navigation bar using Bootstrap. A navigation bar is essential for providing easy access to various sections of your website.  
Example navigation bar code:

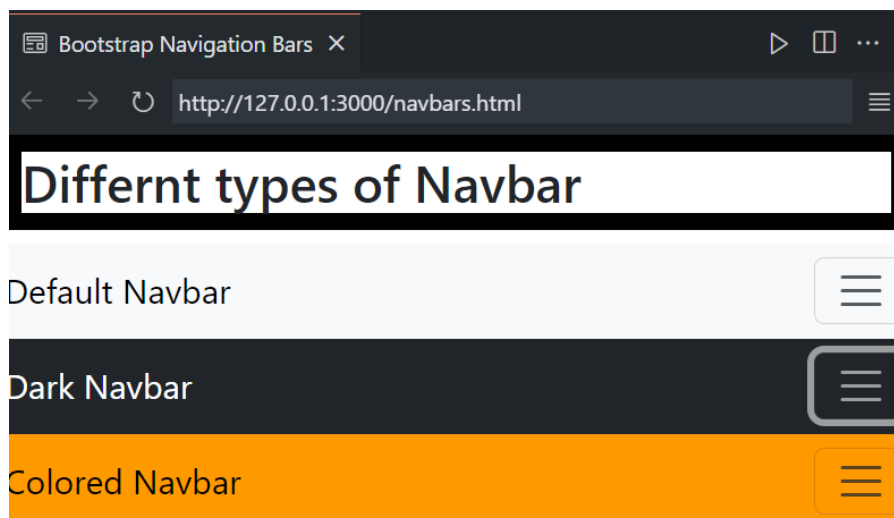
```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
```

```
<!-- Navbar content goes here -->
```

```
</nav>
```



2. Customize the navigation bar by adding menu items and adjusting the appearance to match your application's branding.





## Conclusion

In this practical exercise, we have successfully deployed different Bootstrap components, including buttons, progress bars, modals, and navigation bars. Bootstrap simplifies the process of creating visually appealing and responsive web applications, enhancing the overall user experience. These components are highly customizable, allowing developers to tailor them to specific project requirements. By mastering these Bootstrap elements, you can create interactive and engaging web applications.

## References

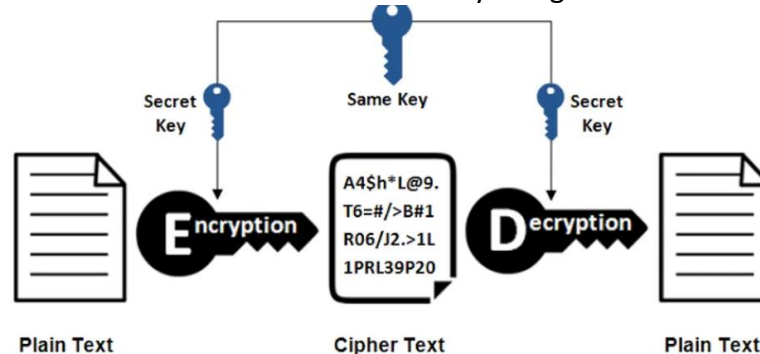
1. Bootstrap Documentation: <https://getbootstrap.com/docs/5.3/getting-started/introduction/>
2. W3schools navbars in Bootstrap: [https://w3schools.com/bootstrap/bootstrap\\_navbar.asp](https://w3schools.com/bootstrap/bootstrap_navbar.asp)

# Experiment: 4

To create and setup the Git repository on Bitbucket or GitHub using SSH.

## SSH (Secure Shell Protocol)

Using the SSH protocol, you can connect and authenticate to remote servers and services. With SSH keys, you can connect to GitHub without supplying your username and personal access token at each visit. You can also use an SSH key to sign commits.



To create and set up a Git repository on GitHub using SSH, you'll need to follow these steps:

1. **Generate an SSH Key:** If you haven't already, you'll need to generate an SSH key pair. This consists of a public key (which you'll upload to GitHub) and a private key (which you'll keep on your local machine). You can generate an SSH key using the following command in your terminal:

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

2. **Add your SSH Key to the SSH Agent:** You'll want to add your private key to the SSH agent to manage your keys. You can do this with the following command

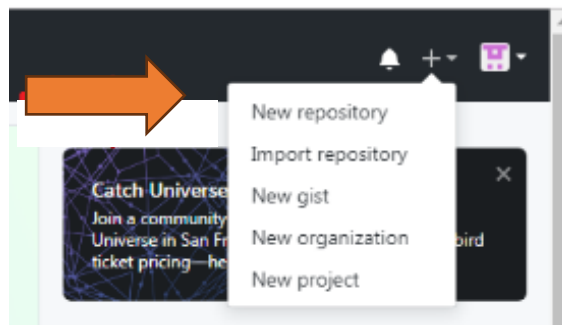
```
eval "$(ssh-agent -s)"  
ssh-add ~/.ssh/id_rsa
```

3. **Copy your Public Key:** Use the following command to copy your public key to your clipboard:

```
pbcopy < ~/.ssh/id_rsa.pub
```

4. **Add SSH Key to Your GitHub Account:** Log in to your GitHub account and go to "Settings" > "SSH and GPG keys." Click on the "New SSH key" button and paste your public key into the provided field. Give it a descriptive title to help you identify it.

5. **Create a New Repository:** On GitHub, click on the "+" icon in the upper right corner and select "New repository." Fill in the repository name, description, and other details as needed.



6. **Initialize Your Local Repository:** On your local machine, navigate to the directory where you want to create your Git repository. Run the following commands:

```
git init //initialise your repo
git add . //Stage Files
git commit -m "Initial commit" //Commit Files
```

7. **Set the Remote Origin:** On the GitHub repository page, you'll see a section called "Quick setup." Copy the URL provided for the repository, which should start with "git@github.com." In your local terminal, set the remote origin using the following command (replace the URL with your repository's URL):

```
git remote add origin git@github.com:yourusername/your-repo.git
```

8. **Push Your Code to GitHub:** Finally, push your code to the GitHub repository using the following command:

```
git push -u origin master
```

## Conclusion:

In the world of collaborative software development, setting up a Git repository on platforms like GitHub using the Secure Shell Protocol (SSH) is not only a best practice but also a key to efficient and secure version control. By following the steps outlined above, you can establish a strong foundation for your project, ensuring smooth

interactions between your local development environment and the remote repository. The use of SSH keys enhances security and eliminates the need for constantly providing usernames and tokens, streamlining your development process. With your repository now in place and securely connected, you are well-equipped to work on your projects with confidence.

**References:**

1. GitHub. "Generating a new SSH key and adding it to the SSH agent."  
<https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent>
2. GitHub. "Adding a new SSH key to your GitHub account."  
<https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account>
3. GitHub. "Adding an existing project to GitHub using the command line."  
<https://docs.github.com/en/get-started/importing-your-projects-to-github/adding-an-existing-project-to-github-using-the-command-line>

# Experiment: 5

To perform push, clone and patch operation to Git repository.

## PUSH

In Git, "push" is a command used to send your locally committed changes to a remote repository, typically hosted on a platform like GitHub, GitLab, Bitbucket, or a private Git server. When you push your changes, you're essentially updating the remote repository with the latest commits from your local repository. This is a fundamental part of collaborative development and version control in Git. Here's how the git push command works:

1. **Local Repository:** You have a Git repository on your local machine where you work on your code.
2. **Remote Repository:** This is a Git repository hosted on a server, which could be a platform like GitHub, GitLab, or a company's internal Git server. Remote repositories allow multiple developers to collaborate on a project.
3. **git commit:** Before you can push your changes, you need to commit them to your local repository using the git commit command. This creates a new commit with the changes you've made.
4. **git push:** Once you have local commits that you want to share with others or store in a remote repository, you use the git push command. The basic syntax for pushing is:

```
git push <remote> <branch>
```

- **<remote>** is the name of the remote repository (often named "origin" by default), which represents the URL of the remote server where your code is hosted.
- **<branch>** is the branch you want to push. It's often "master" for the main branch, but you can push any branch.

For example, if you want to push your local "master" branch to the "origin" remote repository:

```
git push origin master
```

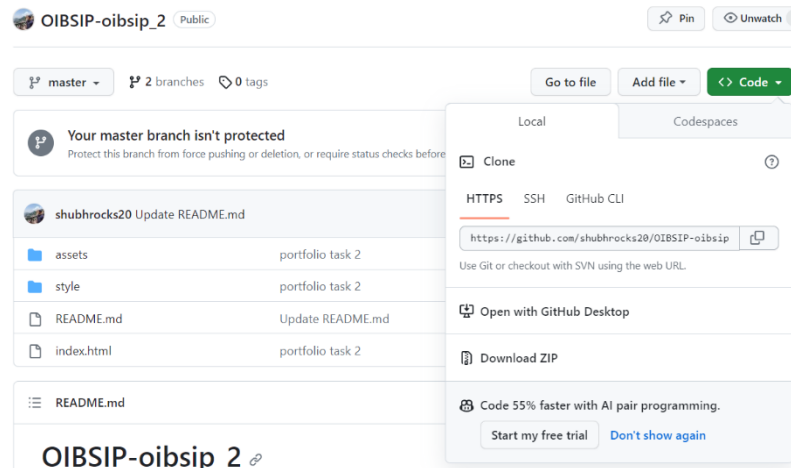
When you push, Git uploads your commits to the remote repository, making your changes available to others who can then pull those changes into their own local repositories. This process is a central part of collaboration and keeping all contributors in sync.

## Clone

In Git, "clone" is a command used to create a copy of a remote Git repository on your local machine. This command is commonly used when you want to start working on an existing project hosted on a remote Git server (such as GitHub, GitLab, Bitbucket, or a private Git server). Cloning a repository allows you to have a local copy of the codebase, complete with the commit history, branches, and all the version control features provided by Git. Here's how you use the git clone command:

```
git clone <repository URL> <optional directory>
```

- **<repository URL>** is the URL of the remote Git repository you want to clone. It can be an HTTPS or SSH URL, depending on your authentication and access settings.
- **<optional directory>** is the name of the local directory where you want to clone the repository. If you don't specify a directory name, Git will create a new directory with the same name as the remote repository.



## Patch

Git patch is a feature in git which enables you to create a patch file from a feature in one branch and apply it in another branch.

A patch file has all the differences between the two branches. Using the patch file, we can apply the changes in a different branch.

1. Creating a Patch:
  - a. Make changes to your code.
  - b. Stage the changes with git add.
  - c. Create a patch using git format-patch:

```
git format-patch -1 # -1 indicates the last commit
```

This will create a patch file in the current directory. If you want to create a patch for a specific commit, replace -1 with the commit hash.

2. Applying a Patch:
  - a. Ensure you have the patch file you want to apply.
  - b. Use the git apply command:

```
git apply patchfile.patch
```

## Conclusion:

In this experiment, we delved into essential Git operations - push, clone, and patch. Git is a powerful version control system that enables collaboration, code sharing, and code history tracking.

**Push** is a command used to synchronize your local repository with a remote repository. It allows you to share your commits with others and maintain a centralized codebase. This is a critical step for teamwork and keeping everyone's work in sync.

**Clone** is the process of creating a local copy of a remote Git repository. Cloning is indispensable when you want to start working on an existing project hosted on a remote Git server. It gives you access to the entire commit history and branches, empowering you to contribute to a project collaboratively.

**Patch** is a feature in Git that permits the creation of patch files from one branch and their application to another. These patch files contain the differences between two branches, making it easier to transfer changes between different branches.

Each of these operations is a fundamental part of Git's utility in version control and collaboration. Whether you're pushing your code to a remote repository, cloning a project to work on it locally, or using patch files to share specific changes, Git simplifies and streamlines these processes, making it an indispensable tool for developers.

## References:

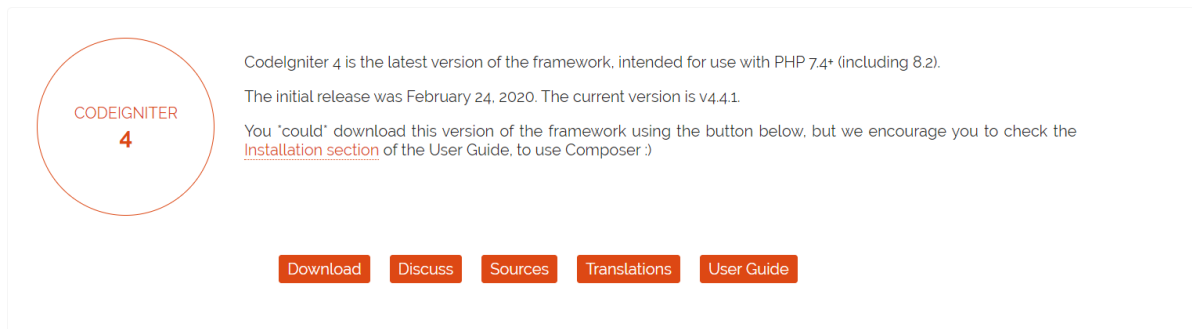
1. Git Documentation: <https://git-scm.com/docs>
2. GitHub Guides - Getting Started with Git:  
<https://guides.github.com/introduction/getting-your-project-on-github/>
3. Pro Git Book: <https://git-scm.com/book/en/v2>
4. Git Patch Documentation: <https://git-scm.com/docs/git-format-patch>

# Experiment: 6

To install and setup the CodeIgniter Framework and to understand its MVC architecture.

## Download CodeIgniter:

- Visit the [CodeIgniter website](#) and download the latest version.



## Extract Files:

- Once downloaded, extract the files to your desired directory.

	codeigniter4-framework-v4.4.1-0-g84461...	15-10-2023 09:01	File folder	
	ADVANCED WEB TECH LAB	15-10-2023 22:37	Microsoft Word D...	1,743 KB
	codeigniter4-framework-v4.4.1-0-g84461...	15-10-2023 09:00	WinRAR ZIP archive	1,111 KB

## Configure Base URL:

```
public string $baseUrl = 'http://localhost:8080/codeigniter/';
```

## Model:

### 1. Responsibility:

- The Model is responsible for managing the data, its structure, and the operations that can be performed on it. This includes retrieving data from a database, performing calculations, and applying business logic.

### 2. Characteristics:

- Data Representation:** It represents the data in the application. This could be data retrieved from a database, an API, or any other source.



- **Data Operations:** It handles operations like reading, writing, updating, and deleting data. It ensures data integrity and consistency.
- **Business Logic:** Contains the application's business logic, which is the set of rules that govern how data is processed and manipulated.
- **No Knowledge of UI:** The Model has no knowledge of how data is presented or displayed to the user.
- **Observable:** In some implementations, Models can notify interested parties (observers) when their state changes.

### **View:**

#### **1. Responsibility:**

- The View is responsible for presenting data to the user in a human-readable format. It displays the user interface, including buttons, forms, text, and graphics.

#### **2. Characteristics:**

- **User Interface:** It encompasses all the elements that the user interacts with, such as buttons, forms, text fields, and graphics.
- **Presentation Logic:** Contains code for rendering data in a specific way. This could involve formatting dates, currency, or other data for display.
- **No Business Logic:** The View should not contain any business logic. It's purely concerned with displaying data.
- **No Direct Interaction with Data:** It does not directly interact with the database or manipulate data. It receives data from the Controller.
- **Observable (Optional):** In some implementations, Views can be observers that are notified when the underlying data changes.

### **Controller:**

#### **1. Responsibility:**

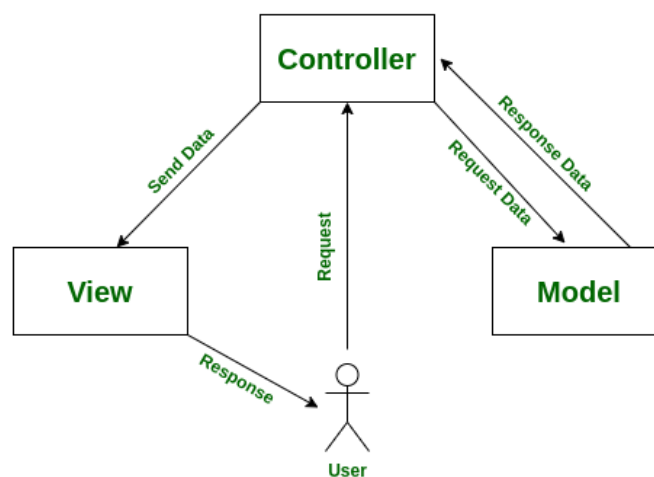
- The Controller acts as an intermediary between the Model and the View. It receives input from the user, processes that input (potentially interacting with the Model), and determines what the View should display.

#### **2. Characteristics:**

- **Input Handling:** Listens for user input, which could be a mouse click, keyboard event, or any form of interaction with the application.

- **Business Logic Orchestration:** Orchestrates the flow of data between the Model and the View, applying business logic when needed.
- **Determines the View:** Decides which View should be displayed based on the user's actions or the state of the application.
- **No UI Logic:** While it may decide which View to display, it doesn't contain UI code. It delegates that responsibility to the View.
- **Updates the Model:** In response to user actions, the Controller may instruct the Model to update its state.

#### Flow of Data:



#### 1. User Interaction:

- The user interacts with the View, such as clicking a button or filling out a form.

#### 2. Controller Receives Input:

- The Controller receives the input and decides how to handle it.

#### 3. Controller Updates Model (Optional):

- Depending on the action, the Controller may instruct the Model to update its data.

#### 4. Controller Determines View:

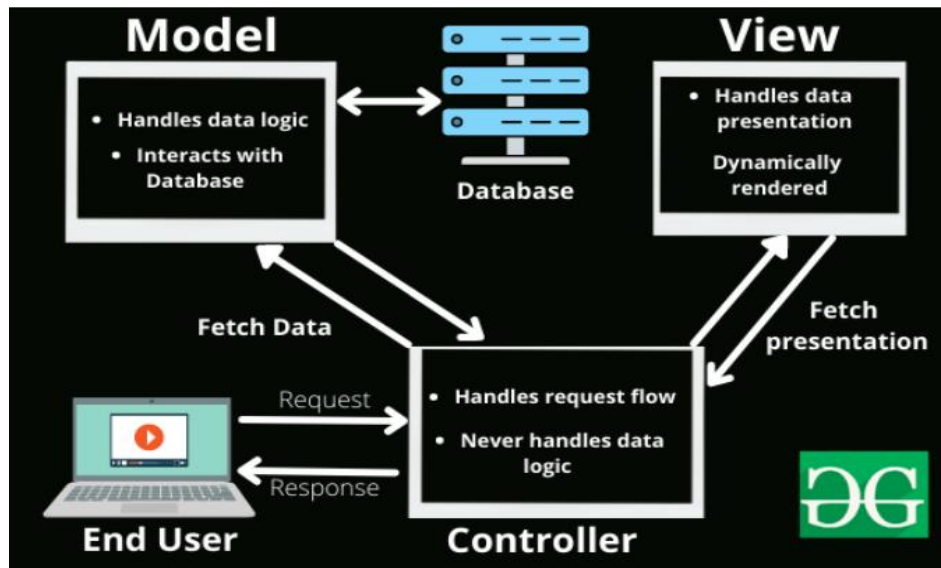
- The Controller decides which View to display based on the user's action and the state of the application.

#### 5. View Displays Data:

- The View receives data from the Controller and renders it for the user.

#### 6. User Sees Updated Interface:

- The user sees the updated interface in the View.



#### Benefits of MVC:

- **Separation of Concerns (SoC):** It separates the application logic into distinct parts, making it easier to manage and maintain.
- **Code Reusability:** Components can be reused across different parts of the application or in different applications.
- **Testability:** Each component (Model, View, Controller) can be tested independently, which makes it easier to identify and fix issues.
- **Scalability:** It allows for scaling different components independently. For example, if the application needs to handle more traffic, you can scale the Controller or the View layer independently of the Model.
- **Flexibility:** Changes in one component do not necessarily affect the others, providing flexibility in development and maintenance.
- **Collaboration:** Different teams or developers can work on different components simultaneously, as long as they adhere to the defined interfaces.

#### Conclusion:

In conclusion, the experiment focused on the installation and setup of the CodeIgniter Framework, emphasizing a comprehensive understanding of its Model-View-Controller

(MVC) architecture. The process involved downloading the framework from the CodeIgniter website, extracting files to a designated directory, and configuring the base URL. The subsequent exploration of the MVC architecture revealed distinct responsibilities and characteristics of each component.

The Model, responsible for managing data and related operations, ensures data integrity, consistency, and contains the application's business logic. The View, focused on presenting data to the user, encompasses the user interface and presentation logic, avoiding any business logic and direct interaction with the database. The Controller acts as an intermediary, handling user input, orchestrating the flow of data between the Model and View, determining the View to display, and updating the Model as necessary.

The flow of data through user interaction, Controller input handling, potential Model updates, and View data display was elucidated, illustrating the seamless communication between components. The benefits of adopting the MVC architecture were highlighted, including the separation of concerns, code reusability, testability, scalability, flexibility, and the facilitation of collaboration among different development teams or individuals. Overall, the experiment provided a practical exploration of CodeIgniter's MVC architecture, underscoring its advantages in terms of organization, maintainability, and collaborative development.

## References:

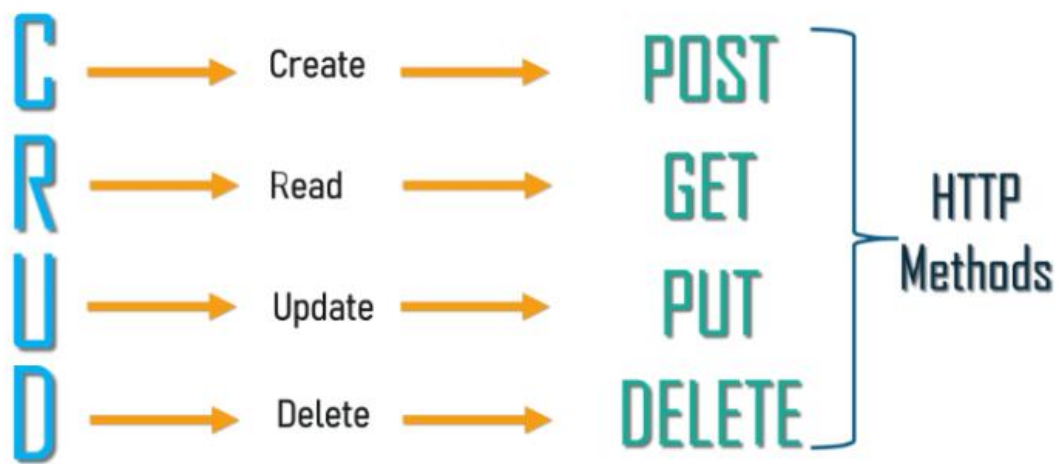
1. Geekfs for Geeks CodeIgniter: <https://www.geeksforgeeks.org/introduction-to-codeignitor-php/>
2. CodeIgniter's official website: <https://codeigniter.com/>
3. CodeIgniter MVC architecture: <https://codeigniter.com/userguide3/overview/mvc.html>

# Experiment: 7

To construct a simple login page web application to authenticate users using CodeIgniter Framework and also perform CRUD operations.

## Introduction:

**CRUD** is the acronym for **CREATE**, **READ**, **UPDATE** and **DELETE**. These terms describe the four essential operations for creating and managing persistent data elements, mainly in relational and NoSQL databases. CRUD is extensively used in database applications. This includes Relational Database Management Systems (RDBMS) like Oracle, MySQL, and PostgreSQL. It also includes NoSQL databases like MongoDB, Apache Cassandra, and AWS DynamoDB.



## CREATE

The Create operation adds a new record to a database. In RDBMS, a database table row is referred to as a record, while columns are called attributes or fields. The CREATE operation adds one or more new records with distinct field values in a table.

## READ

Read returns records (or documents or items) from a database table (or collection or bucket) based on some search criteria. The READ operation can return all records and some or all fields.

## UPDATE

Update is used to modify existing records in the database. For example, this can be the change of address in a customer database or price change in a product database. Similar to READ, UPDATES can be applied across all records or only a few, based on criteria.

## DELETE

Delete operations allow the user to remove records from the database. A hard delete removes the record altogether, while a soft delete flags the record but leaves it in place. For example, this is important in payroll where employment records need to be maintained even after an employee has left the company.

### Step-wise approach to create a CodeIgniter based Login form:

Here are the steps related specifically to creating the Model, View, and Controller for a simple login page web application with user authentication and CRUD operations using the CodeIgniter Framework:

#### **Step 1:** Install CodeIgniter

Download and extract CodeIgniter into your web server's root directory.

#### **Step 2:** Configure Database

Open ``application/config/database.php`` and configure the database settings.

#### **Step 3:** Create Database Tables

-Create a ``users`` table in your database with fields like ``id``, ``username``, ``password``, etc.

#### **Step 4:** Create Model for User

Create a new model file in ``application/models`` (e.g., ``User_model.php``).

Define functions for CRUD operations (create, read, update, delete) and user authentication.

#### **User\_Model.php**

```
<?php namespace App\Models;
    use CodeIgniter\Model;
    class UserModel extends Model{
        protected $table = 'sample';
        protected $primaryKey = 'id';
        protected $allowedFields = ['name','email','password']; }?>
```

### Step 5: Create Controller for Authentication

- Create a new controller file in `application/controllers` (e.g., `Home.php`).
- Define functions for login, logout, and user registration.
- Load the user model in the constructor for database operations.

#### Home.php

```
<?php namespace App\Controllers;

use App\Models\UserModel;

class Home extends BaseController
{
    public function index(): string
    {
        // return view('welcome_message');
        return view('crud');
    }
}
```

#### C - Create

```
public function insert(){

    $data = [

        'name' => $this->request->getVar('name'),
        'email' => $this->request->getVar('email'),
        'password' => $this->request->getVar('password'), ];

    $model = new UserModel();

    $model->insert($data);

    echo "<h1>Data sent successfully...</h1>";
}
```

#### R - Read

```
public function show(){

    $model = new UserModel();

    $data['users'] = $model->findAll();

    return view('show',$data);
}
```

### U - Update

```
public function update(){  
    $data = [  
        'name' => $this->request->getVar('name'),  
        'email' => $this->request->getVar('email'),  
        'password' => $this->request->getVar('password'),];  
    $id = $this->request->getVar('id');  
    $model = new UserModel();  
    $model->update($id,$data);  
    return redirect()->to( base_url('Home/show') ); } }
```

### D - Delete

```
public function delete($id = null){  
    $model = new UserModel();  
    $data['user'] = $model->where('id', $id)->delete();  
    return redirect()->to( base_url('Home/show') );}  
  
    public function edit($id = null){  
        $model = new UserModel();  
        $data['user'] = $model->where('id',$id)->first();  
        return view('edit',$data);}
```

### Step 6: Create Views

- Create views in `application/views` for login, registration, user listing, user details, etc.
- Design forms using HTML and integrate with CodeIgniter's form helper.
- Utilize CodeIgniter's form helper for form creation, validation, and error handling.
- Integrate with the user controller to process registration requests and store new user data in the database.
- Create views in `application/views` for login, registration, user listing, user details, etc.



- Design forms using HTML and integrate with CodeIgniter's form helper.

### Crud.php (view)

```
<!DOCTYPE html> <html lang="en">
    <head> <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Registration form</title>
    </head> <body>
        <h1>Registration Form</h1>
        <form method="POST" action="<?php echo site_url('Home/insert'); ?>">
            <label><b>Name : </b></label>
            <input type="text" name="name"><br/><br/>
            <label><b>Email : </b></label>
            <input type="email" name="email"><br/><br/>
            <label><b>Password : </b></label>
            <input type="password" name="password"><br/><br/>
            <input type="submit" name="submit" value="Insert">
        </form> </body> </html>
```

### Step 7: Secure Routes

- Utilize CodeIgniter's route configuration to restrict access to certain pages to authenticated users.

### Routes.php

```
$routes->get('/', 'Home::index');
$routes->post('Home/insert', 'Home::insert');
$routes->get('Home/show', 'Home::show');
$routes->get('Home/delete/(:num)', 'Home::delete/$1');
$routes->get('Home/edit/(:num)', 'Home::edit/$1');
$routes->post('Home/update', 'Home::update');
```

### Step 11: Test Your Application

- Thoroughly test your login page, authentication, and CRUD operations.
- Verify that user data is stored and retrieved correctly from the database.

These steps provide a focused guide on creating the Model, View, and Controller for a simple login page web application with user authentication and CRUD operations using CodeIgniter. Adjustments may be needed based on specific project requirements and design considerations.

### Create Operation (User Registration)

Screenshot 1: Registration Form and Success



The screenshot displays a web application interface for user registration. It features a title 'Registration Form' in a large, bold, black serif font. Below the title are three input fields: 'Name' with the value 'Arshpreet Singh', 'Email' with the value 'arshpree8051@gmail.com', and 'Password' with a masked value '.....'. Each input field is a light blue rectangle with a thin border. Below the password field is a light gray rectangular button labeled 'Insert'. Below the form is a separate box containing the text 'Data sent successfully...' in a bold, black serif font.

**Registration Form**

**Name :**

**Email :**

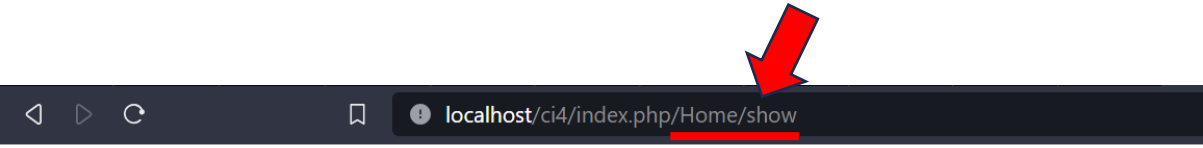
**Password :**

**Data sent successfully...**

This screenshot captures the user registration process. The registration form is displayed, allowing users to input their details. Upon successful registration, a confirmation message or success page is presented.

**Read Operation (User Listing and Details)**

Screenshot 2: User Listing and Details




User Id	Name	Email	Password	Delete	Edit
5	Arshpreet Singh	arshpree8051@gmail.com	21212121212121	<a href="#">Delete</a>	<a href="#">Edit</a>
6	Arshpreet Singh	arshpree8051@gmail.com	21212121212121	<a href="#">Delete</a>	<a href="#">Edit</a>
7	sample 2	sample2@gmail.com	sdad	<a href="#">Delete</a>	<a href="#">Edit</a>
8	sample3	sample3@gmail.com	dakfhwf	<a href="#">Delete</a>	<a href="#">Edit</a>
9	sample4	sample4@gmail.com	adlk;amd	<a href="#">Delete</a>	<a href="#">Edit</a>

This combined screenshot showcases the user listing page along with user details. Users can view a list of registered users, and by clicking on a user, they can access detailed information about that specific user.

**Update Operation (Edit User Details)**

Screenshot 3: Edit User Form and Success

User Id	Name	Email	Password	Delete	Edit
5	Arshpreet Singh	arshpree8051@gmail.com	21212121212121	<a href="#">Delete</a>	<a href="#">Edit</a>
6	Arshpreet Singh	arshpree8051@gmail.com	21212121212121	<a href="#">Delete</a>	<a href="#">Edit</a>
7	sample 2	sample2@gmail.com	sdad	<a href="#">Delete</a>	<a href="#">Edit</a>



## Update Form

**Name :**

**Email :**

**Password :**

This screenshot covers the user details editing process. Users with appropriate permissions can access the edit user form, modify details, and receive a success message upon updating.

### Delete Operation (Delete User)

Screenshot 4: Delete Confirmation and Success

#### Before:

User Id	Name	Email	Password	Delete	Edit
5	Arshpreet Singh	arshpree8051@gmail.com	21212121212121	<a href="#">Delete</a>	<a href="#">Edit</a>
9	sample4	sample4@gmail.com	adlk;amd	<a href="#">Delete</a>	<a href="#">Edit</a>



#### After:

User Id	Name	Email	Password	Delete	Edit
5	Arshpreet Singh	arshpree8051@gmail.com	21212121212121	<a href="#">Delete</a>	<a href="#">Edit</a>

In this screenshot, the deletion process is illustrated. A confirmation dialog appears before deleting a user. After confirmation, users are directed to a success page indicating the successful deletion.

### References:

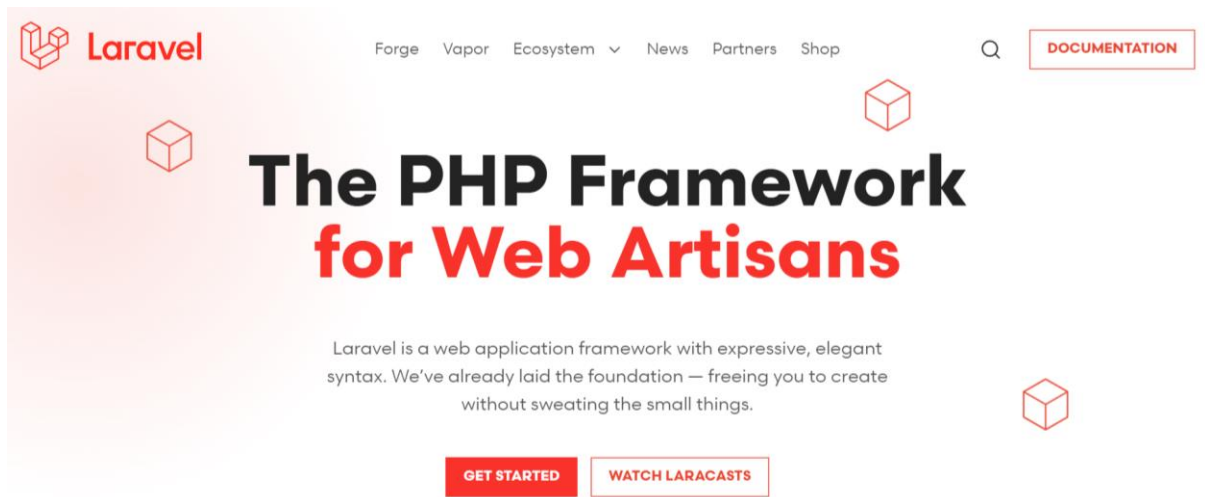
1. CrowdStrike: <https://www.crowdstrike.com/cybersecurity-101/observability/crud/>
2. Medium Article on CRUD: <https://medium.com/geekculture/crud-operations-explained-2a44096e9c88>

# Experiment: 8

To install and setup, configure the Laravel Framework.

## Introduction:

The Laravel framework has a few system requirements. All of these requirements are satisfied by the Laravel Homestead virtual machine, so it's highly recommended that you use Homestead as your local Laravel development environment.



However, if you are not using Homestead, you will need to make sure your server meets the following requirements:

- PHP  $\geq$  7.2.5
- BCMath PHP Extension
- Ctype PHP Extension
- Fileinfo PHP extension
- JSON PHP Extension
- Mbstring PHP Extension
- OpenSSL PHP Extension
- PDO PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension

## Installing Laravel

Laravel utilizes Composer to manage its dependencies. So, before using Laravel, make sure you have Composer installed on your machine.

## Via Laravel Installer

First, download the Laravel installer using Composer:

```
composer global require laravel/installer
```

Make sure to place Composer's system-wide vendor bin directory in your \$PATH so the laravel executable can be located by your system. This directory exists in different locations based on your operating system; however, some common locations include:

macOS: `$HOME/.composer/vendor/bin`

Windows: `%USERPROFILE%\AppData\Roaming\Composer\vendor\bin`

GNU / Linux Distributions: `$HOME/.config/composer/vendor/bin` or  
`$HOME/.composer/vendor/bin`

You could also find the composer's global installation path by running `composer global about` and looking up from the first line.

Once installed, the `laravel new` command will create a fresh Laravel installation in the directory you specify. For instance, `laravel new blog` will create a directory named `blog` containing a fresh Laravel installation with all of Laravel's dependencies already installed:

```
laravel new blog
```

## Via Composer Create-Project

Alternatively, you may also install Laravel by issuing the Composer `create-project` command in your terminal:

```
composer create-project --prefer-dist laravel/laravel:^7.0 blog
```

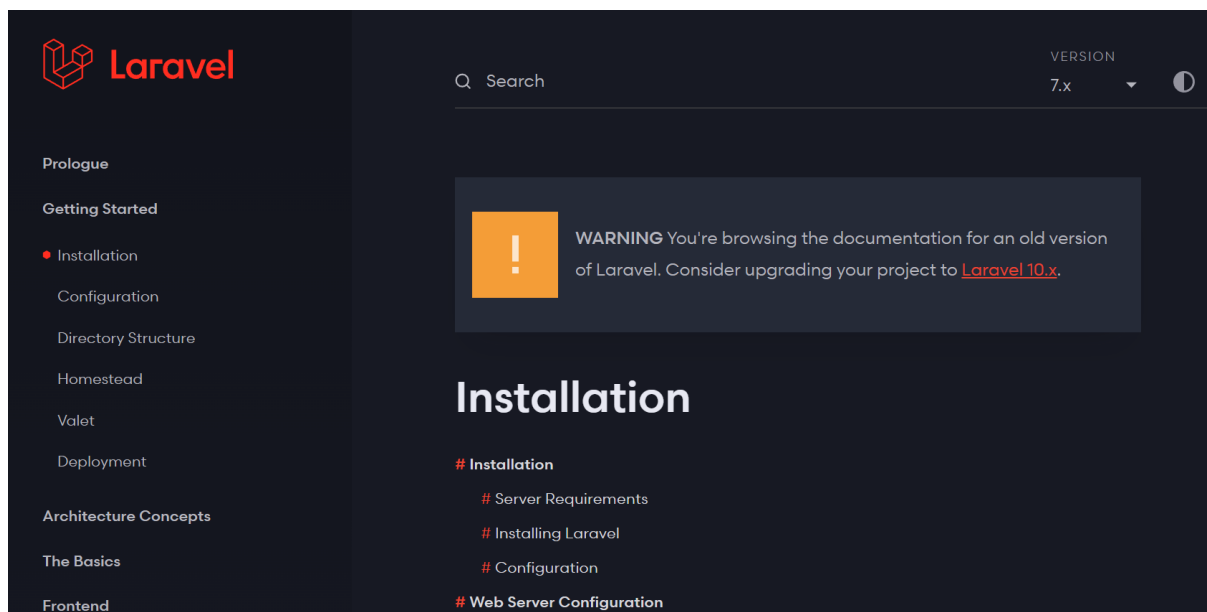
## Local Development Server

If you have PHP installed locally and you would like to use PHP's built-in development server to serve your application, you may use the `serve` Artisan command. This command will start a development server at `http://localhost:8000`:

```
php artisan serve
```

For the installation and setup of Laravel, it is highly recommended to visit the official website of Laravel at <https://laravel.com>. The Laravel documentation provides clear and comprehensive guidance on getting started with the framework. By following the instructions outlined on the official website, users can ensure they are using the latest stable version of Laravel and have access to up-to-date documentation and community support.

The installation process is streamlined through Composer, a widely used dependency manager for PHP, making it convenient for developers to set up their Laravel projects quickly and efficiently. Utilizing the official Laravel documentation ensures a reliable foundation for building robust web applications and allows developers to take full advantage of the features and best practices endorsed by the Laravel community.



## References:

1. Official documentation: <https://laravel.com/docs/10.x/installation>
2. Instalation guide: <https://kinsta.com/knowledgebase/install-laravel/>

# Experiment: 9

**To construct the any simple web application using Laravel Framework:**

## **Introduction:**

In this practical, the Laravel framework is used to build a basic to-do list application. Laravel is a powerful PHP framework that simplifies the process of developing robust web applications. The to-do list application will demonstrate fundamental concepts like routing, controllers, views, and database interactions.

Introduction:

The purpose of this practical file is to guide you through the process of creating a simple To-Do List application using the Laravel framework. This project will provide hands-on experience in building a basic web application, including the setup of models, views, controllers, and routes.

## **Stepwise Explanation to Make ToDo:**

### **Step 1: Install Laravel**

Begin by installing Laravel using Composer. Open a terminal and run the following command:

```
composer create-project --prefer-dist laravel/laravel todo-list
```

Navigate into the project directory:

```
cd todo-list
```

### **Step 2: Create a Task Model**

Generate a Task model using Artisan:

```
php artisan make:model Task -m
```

This command will create a migration file for the `tasks` table.



### Step 3: Define Task Migration

Open the migration file (`database/migrations/YYYY\_MM\_DD\_create\_tasks\_table.php`) and define the `tasks` table schema:

```
public function up(){
    Schema::create('tasks', function (Blueprint $table) {
        $table->id();
        $table->string('title');
        $table->text('description')->nullable();
        $table->boolean('completed')->default(false);
        $table->timestamps();});}
```

Run the migration to create the table:

```
php artisan migrate
```

### Step 4: Create Task Controller

Generate a controller for handling tasks:

```
php artisan make:controller TaskController
```

Edit the `TaskController` to include basic CRUD operations for tasks.

### Step 5: Set Up Routes

Define routes in `routes/web.php` to handle task-related actions:

```
use App\Http\Controllers\TaskController;

Route::get('/', [TaskController::class, 'index']);
Route::post('/tasks', [TaskController::class, 'store']);
Route::put('/tasks/{task}', [TaskController::class, 'update']);
Route::delete('/tasks/{task}', [TaskController::class, 'destroy']);
```

## Step 6: Create Views

Create Blade views for displaying tasks. You can use `resources/views/tasks.blade.php` as your main view.

### Detailed Code for Model, View, Controller, and Routes:

#### Model - Task.php (app\Models\Task.php):

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Task extends Model{

    use HasFactory;

    protected $fillable = ['title', 'description', 'completed'];}
```

#### Controller - TaskController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Task;

class TaskController extends Controller{

    public function index() { $tasks = Task::all();
        return view('tasks', compact('tasks')); }

    public function store(Request $request){

        Task::create($request->validate([

            'title' => 'required',

            'description' => 'nullable',]));

        return redirect('/');}
```

```

public function update(Request $request, Task $task)
{
    $task->update($request->validate([
        'title' => 'required',
        'description' => 'nullable',
        'completed' => 'boolean', ]));
    return redirect('/');
}

public function destroy(Task $task)
{
    $task->delete(); return redirect('/');
}

```

### Sample Screenshots:

Todo Create Todo

Todo Name  
Name here

Todo Description  
Description here

Update

Todo Create Todo

TODO DETAILS

Todo name here  
Todo description here.

Edit Delete

Todo Create Todo

Dummy todo here

Dummy todo here

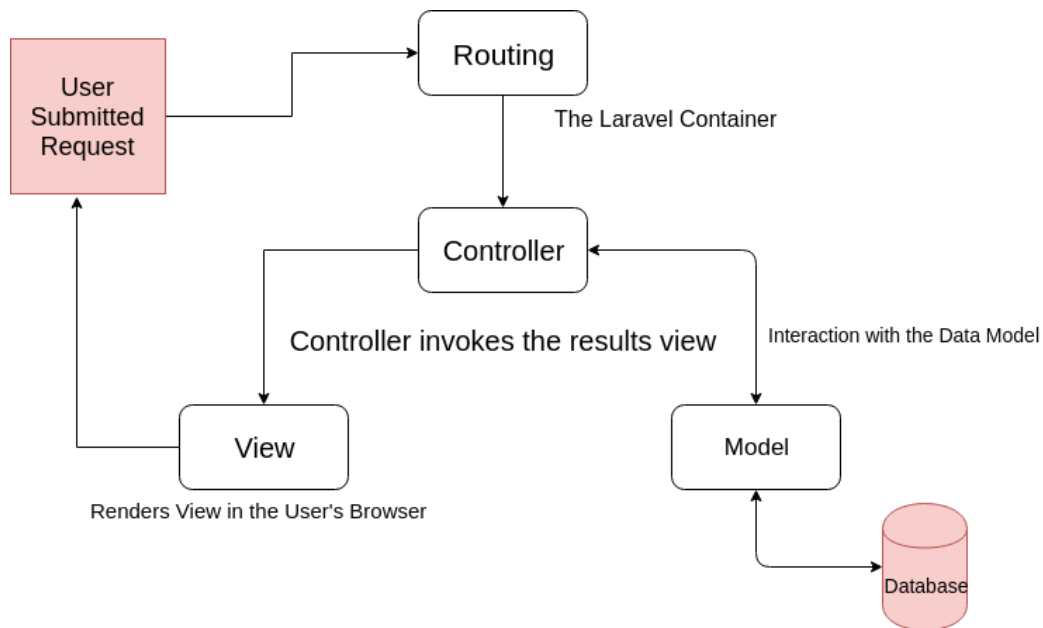
Dummy todo here

Dummy todo here

Dummy todo here

### MVC flow in Laravel based web app:

Model-View-Controller (MVC) is an architectural pattern used by Laravel that separates an application into three main logical components: the model, the view, and the controller.



### Conclusion:

In this practical, I have learned how to set up a basic Laravel project, create a model for tasks, implement controllers for CRUD operations, define routes, and create views. This To-Do List application serves as a foundation for more complex web applications and demonstrates the fundamental concepts of Laravel development. Consider expanding the functionality and styling to enhance your skills further.

### References:

1. Laravel official documentation: <https://laravel.com/docs/10.x/installation>
2. Installation guide: <https://kinsta.com/knowledgebase/install-laravel/>
3. Laravel Todo article on medium: <https://ibrajix.medium.com/understanding-laravel-build-a-simple-todo-app>
4. CRUD operations in Todo: <https://impulsivecode.com/laravel-crud-tutorial>