# Learning Objectives

- **Add the Bootstrap framework to a Django project**
- **Insert the following components into a Django project:**
  - **Alerts**
  - **Buttons**
  - **Dropdowns**
  - **Modal**
  - **NavBar**
  - **Pagination**
- **Add a container to the `<body>` element**
- **Change the default width of the container**
- **Create columns within the container**
- **Manually set the width of a column**
- **Set the width of a column relative to a breakpoint**

# Clone Blango Repo

## Clone Blango Repo

Before we continue, you need to clone the `blango` repo so you have all of your code. You will need the SSH information for your repo.

### In the Terminal

- Clone the repo. Your command should look something like this:

```
git clone git@github.com:<your_github_username>/blango.git
```

- You should see a `blango` directory appear in the file tree.

You are now ready for the next assignment.

# Introduction

## HTML Frameworks Intro

You have definitely used a framework before: Django! A framework is a project that contains common functionality that can be used by many different people for their own projects. You just need to "fill in the blanks". Without Django, you'd have to write a lot of things from scratch. All the way from the HTTP server interface, to the database connection, URL routing and template engine.

HTML frameworks work in a similar manner. Without using one, you'd have to write a lot of custom CSS and Javascript, before you could even start on the parts of the site that are unique to your project.

HTML frameworks might also be referred to as *CSS Frameworks*, as most of their functionality is achieved with custom CSS, so you might see these terms used interchangeably. Note though that *Javascript Frameworks*, like Vue.js, React or Angular are completely different and serve a different purpose: to provide interactive or dynamic web applications with Javascript code.

By using an HTML framework, we can get the following things:

- A *reset* look and feel, so that elements, font sizes and spacing look the same across different browsers.
- Components like navigation bars, panels, cards and modals.
- Automatic Javascript hooks for simple interactive elements
- Utilities for element spacing, text size, colors, and more.
- A grid system for layout.

Usually components and styles are built by adding classes to normal HTML elements – generally HTML frameworks don't provide new types of tags.

Using frameworks can have some downsides though:

- Including a framework can bring in large CSS files which would slow down your page load. If you only wanted a couple of features from a framework this can seem inefficient.
- Websites built with a particular framework can end up looking similar. If you're building a portfolio site to showcase your web design skills, this might not be what you want.
  – On the flipside though, the familiarity can make your UI easier for users to understand.
- If you want to customize the look of some of the elements, it can be a lot of work to write the correct CSS.

There are many HTML frameworks available, to fit many different needs. But once you learn one, the concepts are quite similar between them. Here's a super short look at three of them:

### Bootstrap

Bootstrap is "the most popular HTML, CSS and JS library in the world.". It includes many different components and Javascript functionality for interactivity.

### Foundation

Foundation is the "most advanced responsive front-end framework in the world". While not as popular as Bootstrap some people prefer it for its semantic markup and mobile-first approach.

### Bulma

Bulma is newer than Bootstrap and Foundation, and is more lightweight. It's not as full-featured and doesn't include its own Javascript helpers, so you'll need to write your own for interaction. Many people prefer it for this reason.

### Tailwind CSS

Tailwind CSS is growing in popularity and loved for its utility approach to classnames and its small size. Like Bulma, you'll need to provide your own Javascript.

In this and the subsequent courses, we'll use Bootstrap, because of its popularity and the number of components it provides. Plus, it's used with the Django Rest Framework UI which will look at in Course 2.

# Bootstrap Setup

## Bootstrap Setup

How do we "install" Bootstrap into our project? It's simple, the only requirement is to link in the Bootstrap CSS, by adding a `<link>` element to the `<head>` of our HTML. Like this:

```
<link
        href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap
        .min.css" rel="stylesheet" integrity="sha384-
        EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
        crossorigin="anonymous">
```

And the vast majority of the time you'll want to include the Bootstrap Javascript files, by adding a `<script>` element to the end of your `<body>` in the HTML, like this:

```
<script
        src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle
        js" integrity="sha384-
        MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
        crossorigin="anonymous"></script>
```

Note that the version (*5.0.2* in our case) and integrity hashes (the values that start with *sha-384*) will change with new releases of Bootstrap. You find the latest versions to use on the Bootstrap introduction page.

Bootstrap also provide a starter template which can be very handy to copy and paste to get a new site up and running. In fact, we'll do that now to get our first view in Blango. You should be familiar with the process of setting up templates and views in Django, so we'll just briefly go over the steps.

1. Create a `templates` directory inside the root of the project. We'll use this to store global templates. Do this by right-clicking on `blango` and selecting `New Folder...`.
2. Inside the `templates` project directory, create a new file called `base.html`. Right-click on `templates` and select `New File...`. In here we'll basically copy and paste the starter Bootstrap template, but make a couple of edits:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
        scale=1">
    <link
        href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap
        .min.css" rel="stylesheet" integrity="sha384-
        EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
        crossorigin="anonymous">


    <title>Hello, world!</title>
  </head>
  <body>
    <h1>Hello, world!</h1>
    {% block content %}


    {% endblock %}
    <script
        src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle
        js" integrity="sha384-
        MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
        crossorigin="anonymous"></script>


  </body>
</html>
```

Here, we have removed the comments and commented out code, and added a template `block` called `content` that can be overridden by child templates.

`base.html` can now be saved.

3. Now we want Django to be able to load templates from this directory, so we need to add it to the `TEMPLATES` settings in our `settings.py`. Open this file, and then find the `TEMPLATES` setting. You should see it has a key of `DIRS`, with an empty list:

Open settings.py

```
"DIRS": [],
```

Add the path to the `templates` directory into the list, like this:

```
"DIRS": [BASE_DIR / "templates"],
```

▼ **Path Objects and Strings**
Since `BASE_DIR` is a Path object and not a string, we can join paths by using the `/` operator like this.

While we're here, notice the setting `"APP_DIRS": True`. This means that Django will automatically try to find templates in app directories. We'll see what this means in the next step.

`settings.py` can now be saved and closed.

4. Now, we'll add a templates directory for the `blog` app, to hold blog specific templates. Go to the `templates` directory in the `blog` app directory. Inside this, create another directory called `blog` (so you should have the path `blango/blog/templates/blog`). The inner `blog` directory is for template namespacing. It means templates in here won't conflict with the names of other templates in our project.

Since we had the setting `APP_DIRS": True`, Django is automatically going to look in the `blog/templates` directory for templates, so we don't need to make any more settings changes.

5. Create a new files in the `blog/templates/blog` directory called `index.html`. It should inherit from the global `base.html` and override the `content` block. The contents aren't too important, so you can use something like:

Open index.html

```
{% extends "base.html" %}
{% block content %}
<h2>Index Template</h2>
{% endblock %}
```

Save and close `index.html`.

6. We need a view to render this template. Create one called `index` in the `blog/views.py`:

Open views.py

```
def index(request):
    return render(request, "blog/index.html")
```

7. Finally, let's route the root path to this view. Add a route for `""` (empty string) to this view in `urls.py`. Don't forget to also import the views file. Here's a truncated view of `urls.py`

Open urls.py

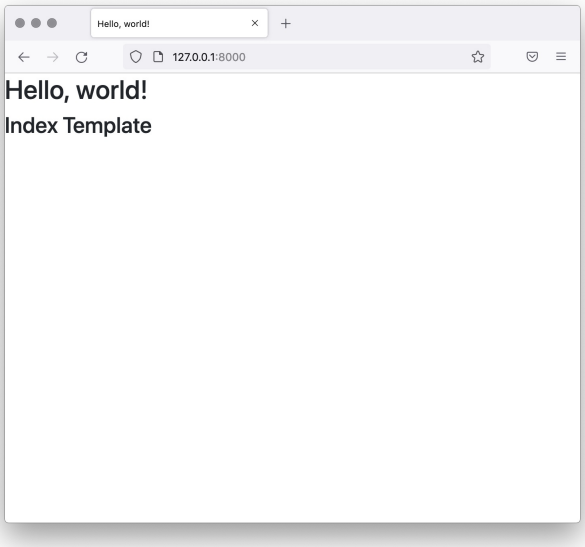
```
# other imports
import blog.views

urlpatterns = [
    # other patterns
    path("", blog.views.index)
]
```

Save `urls.py`.

8. Now, you can start the Django dev server and check out the template. You should see something like this.

```
python3 manage.py runserver 0.0.0.0:8000
```

View index.html




Bootstrap Hello World Template

Now we have Bootstrap included and ready to go in our template, and we can start using Bootstrap classes and components. Next we'll take a quick look at some of the Bootstrap components.

# Bootstrap Components 1

## Bootstrap Components 1

Bootstrap has a number of included components to add useful features to your website. There are quite a few, but here are some that are probably most useful (your mileage may vary based on the type of project you're building!)

In this section we will show the HTML that is added to our template to demonstrate the component, and a screenshot of what it looks like. You can try it out on your own Blango project if you want. In the bottom panel, select the tab with the terminal and start the Django dev server.

```
python3 manage.py runserver 0.0.0.0:8000
```

### Alerts

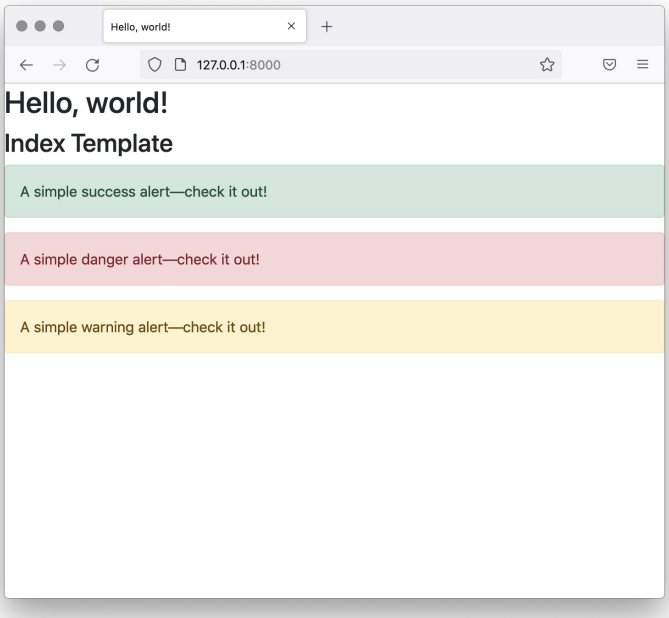
Alerts are used to add an alert or message to a page, and can change color to indicate success, failure or warning. They consist of a `<div>` with an `alert` class and then a modifier class, like `alert-success`, `alert-warning`, etc. Add the following code to `{% block content %}`

```
<div class="alert alert-success" role="alert">
  A simple success alert—check it out!
</div>
<div class="alert alert-danger" role="alert">
  A simple danger alert—check it out!
</div>
<div class="alert alert-warning" role="alert">
  A simple warning alert—check it out!
</div>
```

▼ **Refresh the Website**

Click on the blue, circular arrows to refresh the website.

blue, circular arrows




Bootstrap Alerts

## Buttons

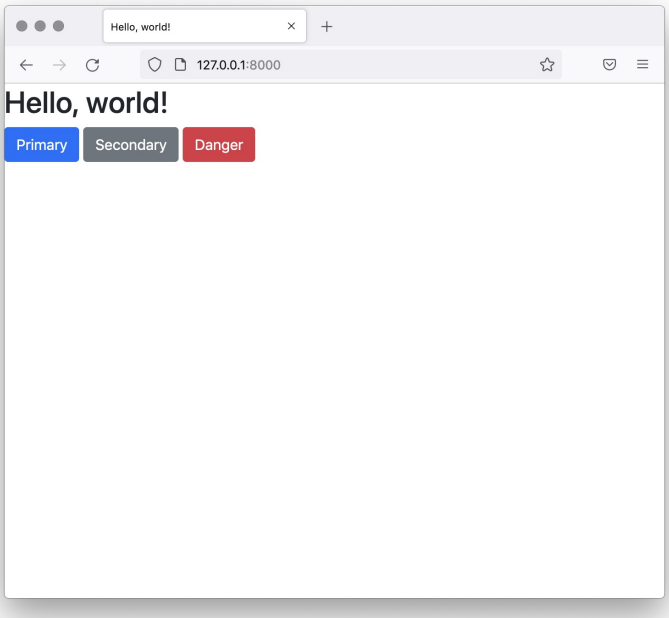
The Button components allows for consistent looking buttons across browsers, with different coloring for primary, secondary, warning and error buttons. Buttons can created from `<button>` or `<a>` elements and are done so with the `btn` class and then a modifer class, like `btn-primary`, `btn-secondary`, `btn-danger`, etc.

```
<button type="button" class="btn btn-primary">Primary</button>
<a href="#" class="btn btn-secondary">Secondary</a>
<button type="button" class="btn btn-danger">Danger</button>
```

### ▼ Refresh the Website

Click on the blue, circular arrows to refresh the website.

blue, circular arrows




Bootstrap Buttons

Note that Bootstraps buttons look the same regardless of the element used to create them (`<a>` or `<button>`).

## Dropdowns

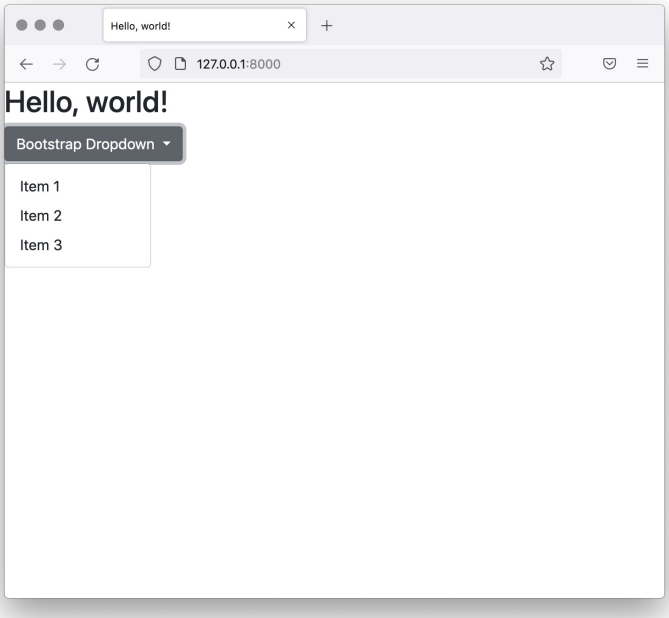
Dropdowns are used when you want a dropdown menu from which you can select an option. It consists of `<div>` with the class `dropdown`. This `<div>` then contains: a Bootstrap button with class `dropdown-toggle` and special attribute `data-bs-toggle="dropdown"`; and a `<ul>` with class `dropdown-menu`. Bootstrap adds all the Javascript to make it interactive automatically. For accessibility, ARIA attributes are also used.

```
<div class="dropdown">
  <button class="btn btn-secondary dropdown-toggle"
        id="dropdownMenuButton1" type="button" data-bs-
        toggle="dropdown" aria-expanded="false">
    BootStrap Dropdown
  </button>
  <ul class="dropdown-menu" aria-
        labelledby="dropdownMenuButton1">
    <li><a class="dropdown-item" href="#">Item 1</a></li>
    <li><a class="dropdown-item" href="#">Item 2</a></li>
    <li><a class="dropdown-item" href="#">Item 3</a></li>
  </ul>
</div>
```

▼ **Refresh the Website**

Click on the blue, circular arrows to refresh the website.

blue, circular arrows




Bootstrap Dropdown

# Bootstrap Components 2

## Bootstrap Components 2

### Modals

In the bottom panel, select the tab with the terminal and start the Django dev server.

```
python3 manage.py runserver 0.0.0.0:8000
```

If you want to display a pop up screen on the page, use a Modal. You can set a title, body and footer, plus lots of options for how it behaves.

To set it up, you'll need something to trigger the modal to open, like a `<button>`. Set its `data-bs-toggle` attribute to "modal" and `data-bs-target` to a CSS selector for the modal to open (for example `#example-modal` will open the modal with ID `example-modal`).

See the code below for an example of the button and the modal container and content – it must be contained in a `<div>` with class `modal`.

```
<button type="button" class="btn btn-primary" data-bs-
        toggle="modal" data-bs-target="#example-modal">
  Open Modal
</button>

<div class="modal" id="example-modal" tabindex="-1" aria-
        labelledby="example-modal-label" aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="example-modal-label">Modal
        Title</h5>
        <button type="button" class="btn-close" data-bs-
        dismiss="modal" aria-label="Close"></button>
      </div>
      <div class="modal-body">
        Modal Body
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-bs-
        dismiss="modal">Close</button>
      </div>
    </div>
  </div>
</div>
```

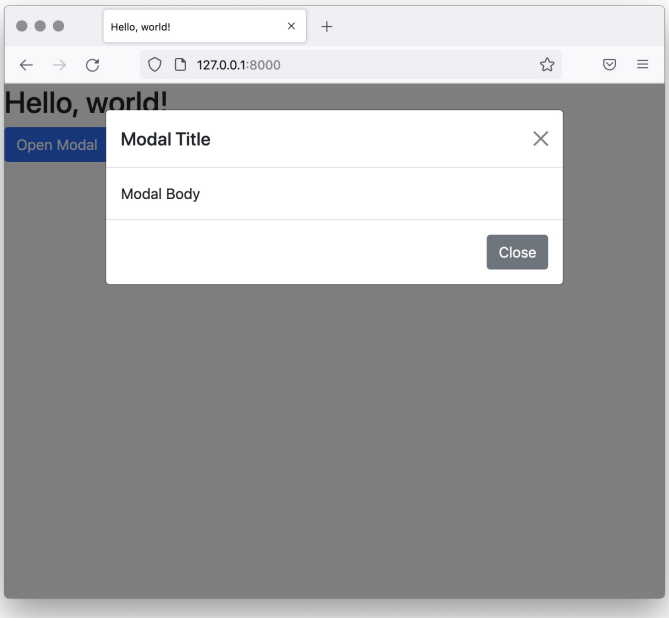
Here's how the modal looks open:

▼ **Refresh the Website**

Click on the blue, circular arrows to refresh the website.

blue, circular arrows

|||




Bootstrap Modal

You can see the *Open Modal* button in the background.

## Navbar

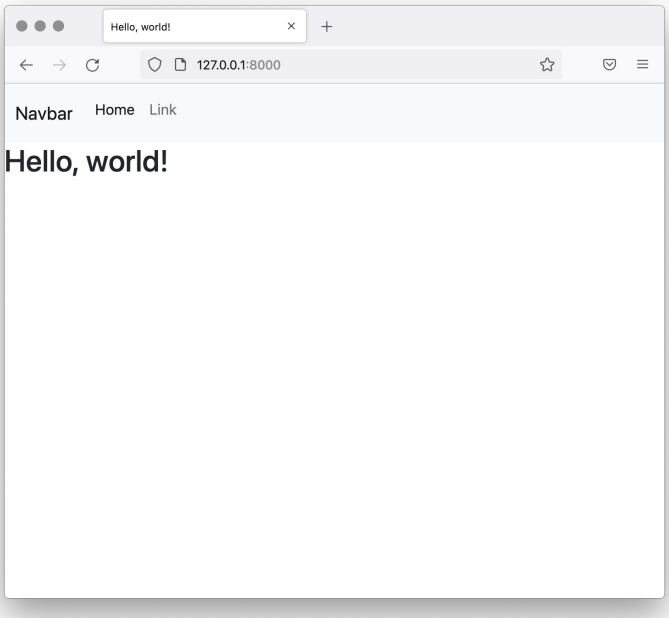
Navbars are essential on most pages. They are usually placed on the top of the page, and contain links to the main pages in your site. They have a lot of options for customization, which are better explained in the official docs, but here's a short example:

```
<nav class="navbar navbar-expand-sm navbar-light bg-light">
  <div class="container-fluid">
      <a class="navbar-brand" href="#">Navbar</a>
      <button class="navbar-toggler" type="button" data-bs-
      toggle="collapse"
              data-bs-target="#navbarSupportedContent" aria-
      controls="navbarSupportedContent"
              aria-expanded="false" aria-label="Toggle
      navigation">
          <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse"
      id="navbarSupportedContent">
          <ul class="navbar-nav me-auto mb-2 mb-lg-0">
              <li class="nav-item">
                  <a class="nav-link active" aria-
      current="page" href="#">Home</a>
              </li>
              <li class="nav-item">
                  <a class="nav-link" href="#">Link</a>
              </li>
          </ul>
      </div>
  </div>
</nav>
```

▼ **Refresh the Website**

Click on the blue, circular arrows to refresh the website.

blue, circular arrows




Bootstrap Navbar

Note that this has been placed into the `base.html` template directly inside the `<body>`, so that it appears above the *Hello, world!* text.

## Pagination

The last component we'll look at is used for pagination. It's used to display an unordered list (`<ul>`) as a horizontal set of buttons, and intended for showing the current page and switching pages.

The `<ul>` element should have the class `pagination`, and the `<li>` elements it contains will have the class `page-item`. You can also add the class `disabled` to disable an item and make it unclickable, or add the class `active` to highlight it to show the current page. You can see an example of both of those here:

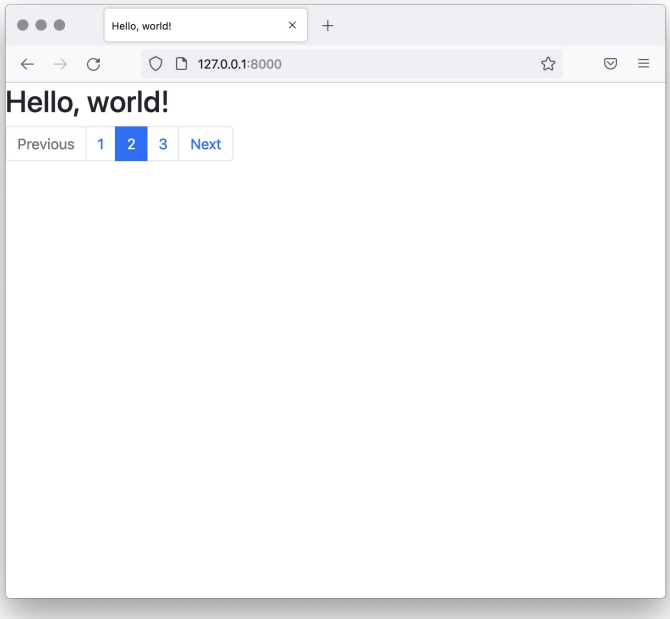
```
<ul class="pagination">
    <li class="page-item disabled">
        <a class="page-link" href="#" tabindex="-1" aria-
        disabled="true">Previous</a>
    </li>
    <li class="page-item"><a class="page-link" href="#">1</a>
        </li>
    <li class="page-item active" aria-current="page">
        <a class="page-link" href="#">2</a>
    </li>
    <li class="page-item"><a class="page-link" href="#">3</a>
        </li>
    <li class="page-item">
        <a class="page-link" href="#">Next</a>
    </li>
</ul>
```

Here's how it displays:

▼ **Refresh the Website**

Click on the blue, circular arrows to refresh the website.

blue, circular arrows




Bootstrap Pagination

There are quite a few more components available, and it is worth browsing the Bootstrap documentation to see which components will work best for your particular needs.

Now we'll have a look at how to lay out the page with the Bootstrap grid system.

# Grid and Container System 1

## Grid and Container System 1

In the bottom panel, select the tab with the terminal and start the Django dev server.

```
python3 manage.py runserver 0.0.0.0:8000
```

In using Bootstrap so far, we've put the components directly inside the `<body>`. This means we end up with them directly against the side of the browser window. This can make our layout feel cramped. Bootstrap solves this with a number of container classes that can be applied to keep your content a consistent size regardless of the browser window size.

A containing element (like a `<div>`) can have the `container` class added. It will jump between a number of set sizes at certain breakpoints as the window resizes. For example, it's the full width of the window at up to 576px wide, then jumps to 540px wide up to a 768px wide browser, at which point it resizes to 720px wide. This is easier to understand visually. So let's set up one you can experiment with.

Open your `base.html` file (click the tab in the bottom panel). We'll add a `<div>` around the content `block`. It should have the class `container bg-dark text-light`. The two latter classes are utility classes that Bootstrap provides, that set the container to have dark background, and light text, respectively, just so we can see what's going on.

In addition, remove "Hello World" and nav bar from the file. After these modifications, this part of your `base.html` should look like this:

```
<div class="container bg-dark text-light">
    {% block content %}

    {% endblock %}
</div>
```

Now update your `blog` `index.html`'s (click the tab in the bottom panel) content `block` to have a simple message in it, something like this is fine:

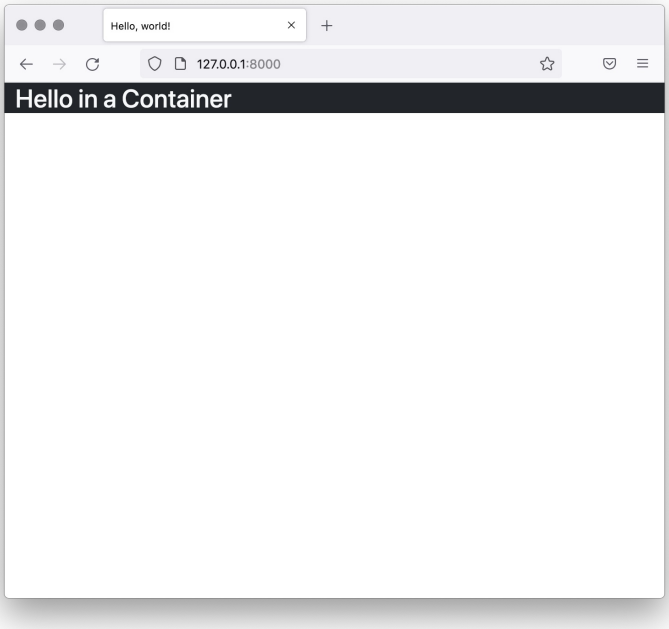
```
{% block content %}
    <h2>Hello in a Container</h2>
{% endblock %}
```

Load the main page of your project in a browser and you should see something like this, at small sizes:

▼ **Refresh the Website**

Click on the blue, circular arrows to refresh the website.

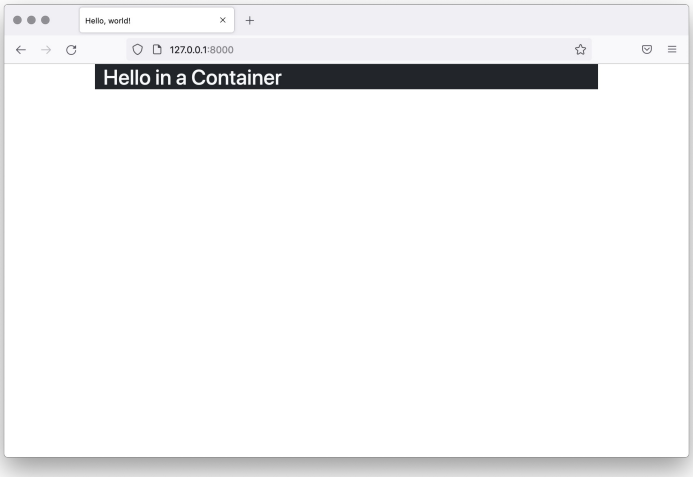
blue, circular arrows




Bootstrap Container Small

Notice the container spans the full width, but also has a small margin to add some space around its content.

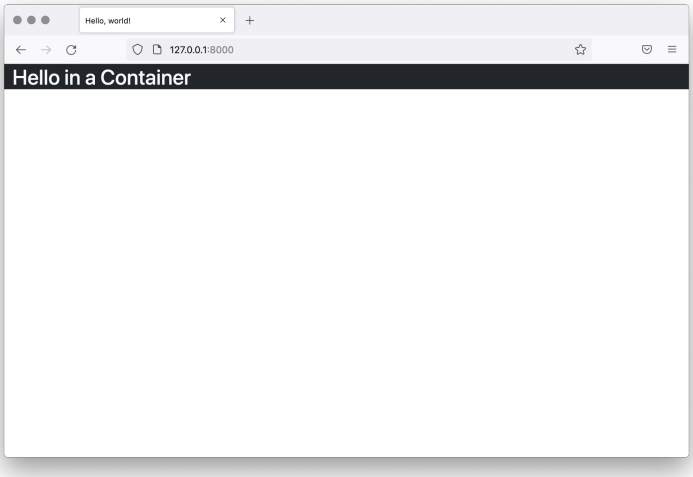
As you resize the window, the container should jump between different widths as you cross the breakpoints.




Bootstrap Container Medium

Another option is to use the `container-fluid` class. This will make a container that's full width at all sizes.

In `base.html`, try changing the class from `container` to `container-fluid` on your containing `<div>`, then refresh the page. You should see it stay full width at all sizes, but still with a margin around the content to keep it looking better.




Bootstrap Container Fluid

There are many options for containers, and you can switch to one that stays full width to larger sizes, then starts jumping to set sizes at breakpoints. There are too many options to replicate here, and the containers documentation lists all of them and the sizes at which they switch breakpoints.

In Blango, we're going to stick with a `container-fluid` class, so you can leave that there, but we're finished with the `bg-dark` and `text-light` utilities, so those classes can be removed.

Bootstrap (and many other HTML frameworks) use a grid system to lay out columns. Usually, this consist of *rows* that can contain up to 12 *columns* each. Rows are just `<div>`s with a `row` class; likewise columns are `<divs>` with a `col` class – although the width can be set manually, more on that in a minute.

In their simplest form, each column inside a row is of equal width. Let's try this out byt first building a row with three equal columns. We'll also use some color utility classes to see the columns. Open the `blog` `index.html`. Remove everything in the `content` `block` and then build a row with three columns, by putting in this HTML:

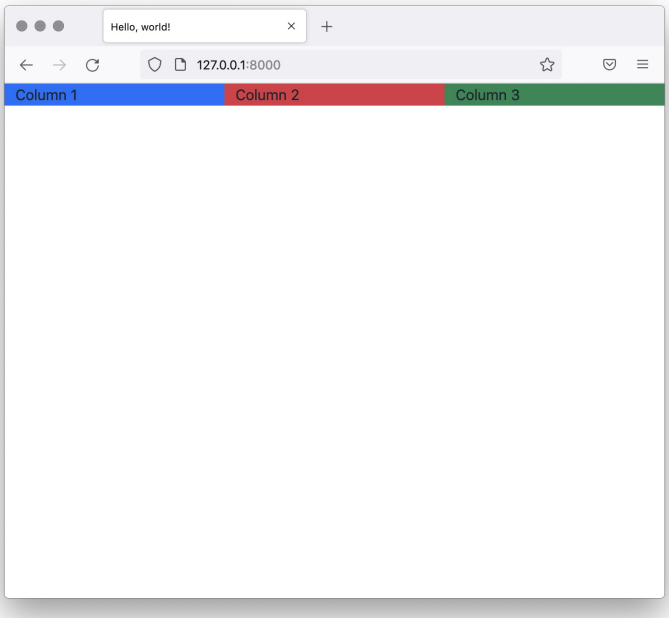
```
<div class="row">
    <div class="col bg-primary">Column 1</div>
    <div class="col bg-danger">Column 2</div>
    <div class="col bg-success">Column 3</div>
</div>
```

You should see three equal size columns, like this:

▼ **Refresh the Website**

Click on the blue, circular arrows to refresh the website.

blue, circular arrows




Bootstrap Columns Three Equal Sizes

# Grid and Container System 2

## Grid and Container System 2

In the bottom panel, select the tab with the terminal and start the Django dev server.

```
python3 manage.py runserver 0.0.0.0:8000
```

Columns can also have their width set manually with width modifier column classes. Use classes from `col-1` to `col-12` to set a width of between 1 and 12 columns. You can set a manual width on any number of columns in a row, and any without a width will adjust themselves to be equally sized in the remaining space. If columns start taking up a width more than 12, they start stacking vertically.

For example, we could set the left column to have a size of 6, meaning it would take up half of the 12 columns. The middle and right column would then take up a quarter of the width each. Let's try that out, change the first column to have the class `col-6`.

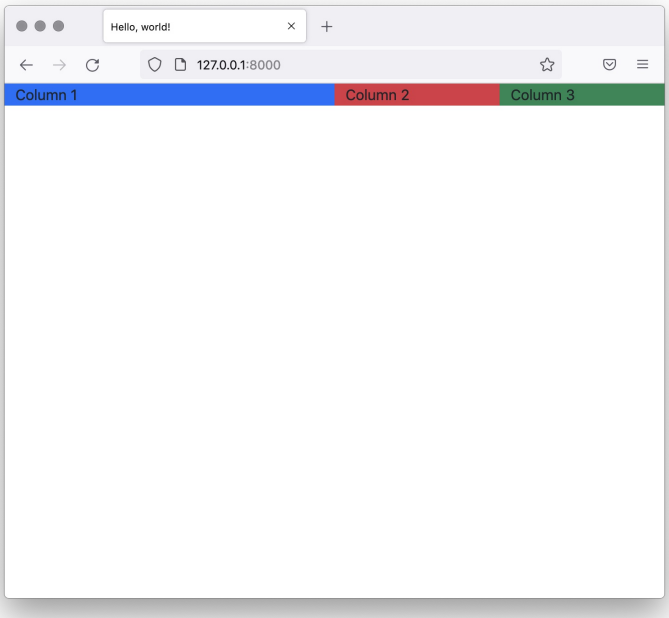
```
<div class="row">
    <div class="col-6 bg-primary">Column 1</div>
    <div class="col bg-danger">Column 2</div>
    <div class="col bg-success">Column 3</div>
</div>
```

Once you save and refresh the page, it should look like this:

▼ **Refresh the Website**

Click on the blue, circular arrows to refresh the website.

blue, circular arrows




Bootstrap Columns Wide Left Column

Each row's columns act independently, so you can have 3 columns in one row, then 8 in the next, then just 1 in the 3rd row, for example.

The last thing to see with columns is that we can have columns of different widths for different breakpoints (screen widths). This means we can have our content look good at a certain width, and just have it stack vertically instead of getting squished at small browser window sizes. For example, a 6-wide column takes up 50% of the grid width, or 600px on a 1200px wide browser window. On a 600px wide browser, we'd want it to be 12-wide, or take up 100% of the grid, for the same absolute column size. Note that these sizes were chosen arbitrarily as no Bootstrap breakpoint is exactly twice as wide as another, but hopefully it illustrates the concept.

We can set different column widths at different breakpoints with breakpoint specific classes. Bootstrap's breakpoint tiers and sizes are:

- Extra small (xs) <576px
- Small (sm) ≥576px
- Medium (md) ≥768px
- Large (lg) ≥992px
- Extra large (xl) ≥1200px
- Extra extra large (xxl) ≥1400px

The breakpoint identifier (the letters in the brackets) can be used in the column class to set its behaviour in that breakpoint. For example, these columns will behave as normal 1/3rd width columns above the medium breakpoint. Below 768px browser size, they become full-width and stack.

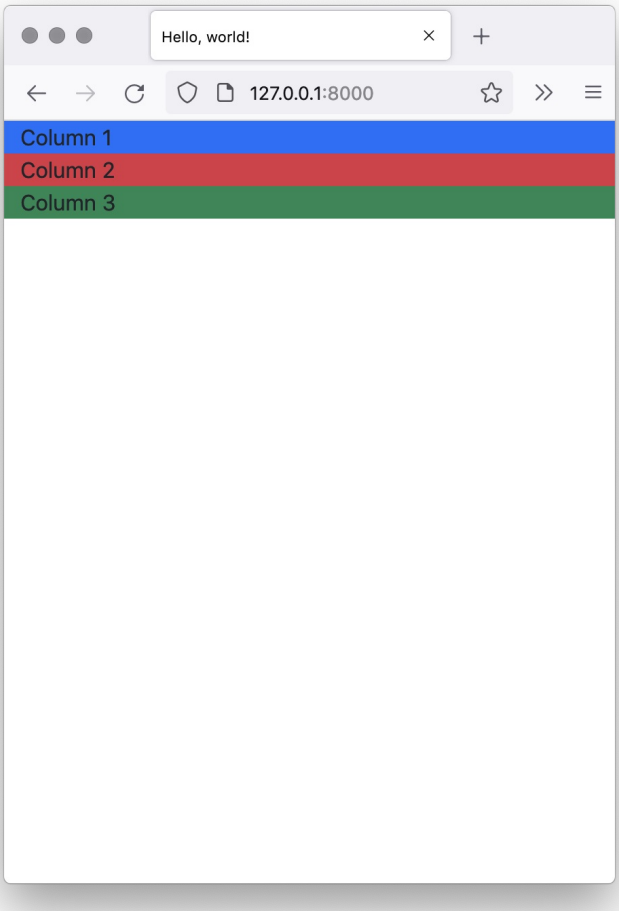
```
<div class="row">
    <div class="col-md bg-primary">Column 1</div>
    <div class="col-md bg-danger">Column 2</div>
    <div class="col-md bg-success">Column 3</div>
</div>
```

Click and drag the right edge of the website to change its width. This is how they look below the medium breakpoint width:

▼ **Refresh the Website**

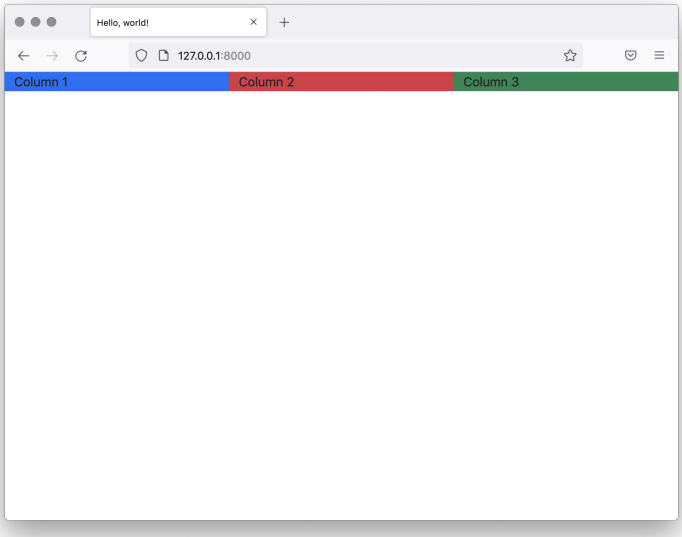
|||info
## Refresh the Website
Click on the blue, circular arrows to refresh the website.

blue, circular arrows




Bootstrap Vertical Stacked Columns

And above the medium breakpoint width:




Bootstrap Medium Columns in Screen Greater than 768 pixels

Manual width columns and breakpoints can also be combined, and even multiple classes at once to set different manual sizes at different breakpoints. This final example:

- at the extra small breakpoint, stacks the columns.
- between the small and medium breakpoints, sets the left column to 10, and middle column to 2. Since this adds to 12 the final column is stacked.
- at the medium breakpoint and above, sets the left column to 6, the middle column to 2, and the right column takes up the remaining space (4).

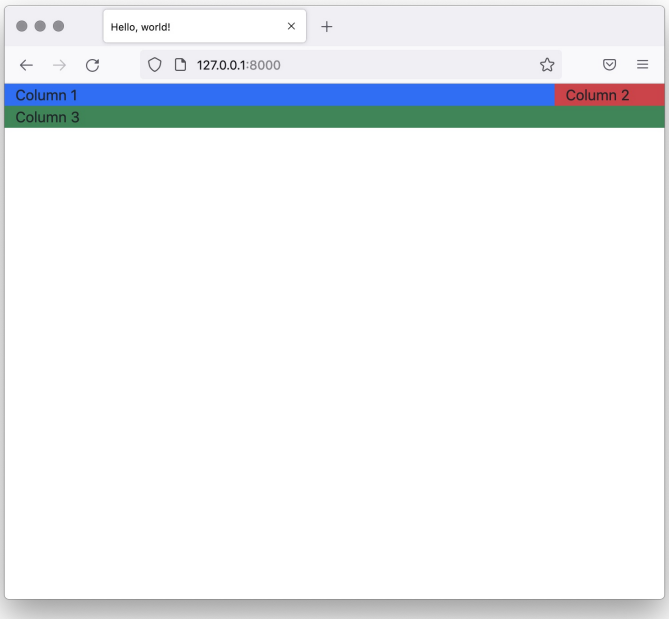
```
<div class="row">
    <div class="col-xs-12 col-sm-10 col-md-6 bg-primary">Column
        1</div>
    <div class="col-xs-12 col-sm-2 col-md-2 bg-danger">Column
        2</div>
    <div class="col bg-success">Column 3</div>
</div>
```

This is an example you should try for yourself and adjust the size up and down to see how it behaves. Here's a screenshot in the small breakpoint:

▼ **Refresh the Website**

Click on the blue, circular arrows to refresh the website.

blue, circular arrows




Bootstrap Different Breakpoints

As you can see, the grid system is very flexible and very powerful, once you get the hang of it. The grid documentation contains all the options and info about the breakpoints, and includes different ways to align and justify your content too. But these basics should cover most of your use cases.

Throughout the course we may introduce some other small utility classes the Bootstrap provides, but we will explain their purpose when we do. Luckily they're quite easy to pick up.
Now that we know how to layout our page with Bootstrap, in the next section we're going to look at Django's custom filters.

# Pushing to GitHub

## Pushing to GitHub

Before continuing, you must push your work to GitHub. In the terminal:

- Commit your changes:

```
git add .
git commit -m "Finish Introduction and Django Admin"
```

- Push to GitHub:

```
git push
```