



**Institute<sub>of</sub>  
Data**

---

2022



# Software Engineering

## Module 4

---

## More on Front-End Development

---



# Introduction

As you start your journey as a web developer, you will be tempted to improvise and hope for the best. While this is a common practice for beginners, soon you will discover that planning is the most important thing you can do in this industry.

Whether you are working on your side hustle or a client project, you should always have a good plan. In web development, the plan start with the **Design**.



# What is Design?

Design is many things. Following the dictionary **“a plan or drawing produced to show the look and function or workings of a building, garment, or other object before it is made.”**

Then what do we mean by design?

In the IT industry we have many types of designs, here we will focus only on a few:

- Application flow design
- Sketches
- LoFi (low fidelity) designs
- HiFi (High Fidelity) designs
- Paper prototype Design

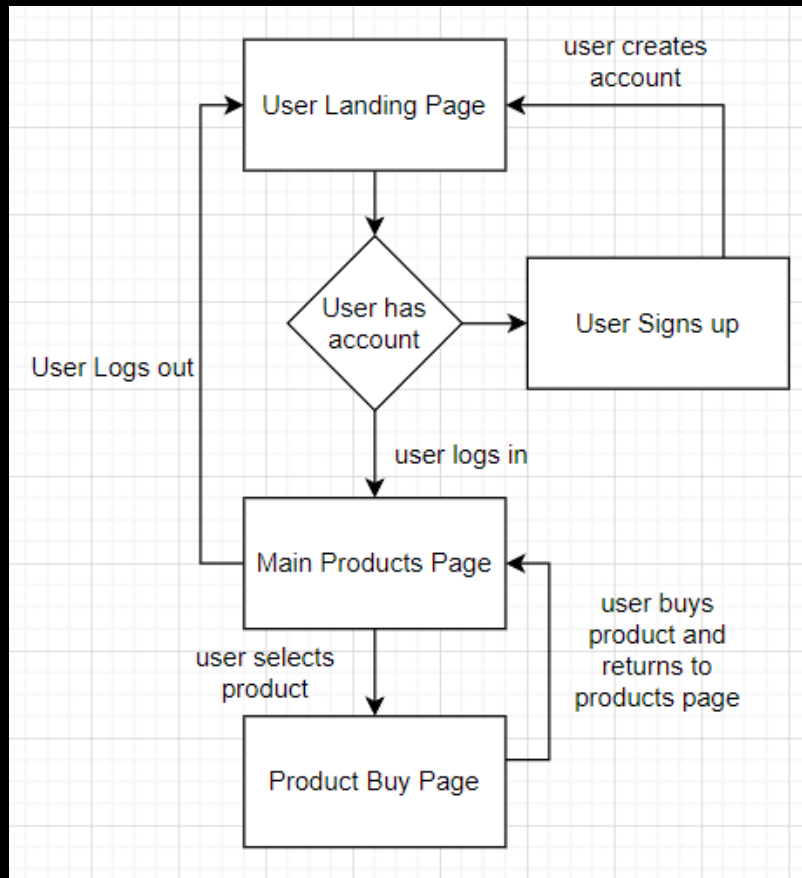


# Application Design

The application flow design is the most important part of your application. Here you can find out if your application will do what is meant to do. Do you know how many developers **forget to create a sign up page?**

Having the correct structure you will be able to also understand how much work there will be.

It is very hard not to put your hands on the actual code, but we are still very far away.

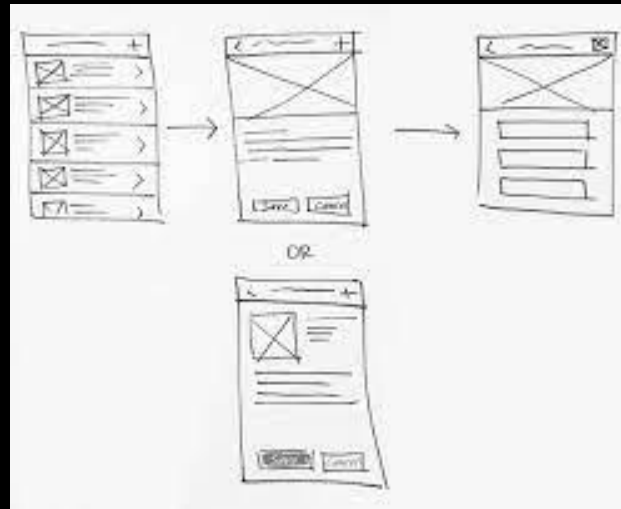




# Sketches

Pen and paper, or pencil. Or an iPad, whatever you like, but start drawing it out. The design for our previous application should not take you more than 5 minutes to sketch.

You need to start thinking professionally, **any time you spend on your application must create value**, if the value you create in 5 minutes is the same as 2 hours, then you should only spend 5 minutes. What is value? What helps you move forward, pretty pictures don't do that.





# Lo-Fi Designs

Lo-Fi is the key to acceptance, and also shows that you are professionally trained.

Lo-Fi should be as Lo as possible.

Hints - don't use colours, in fact, they should be ugly, as plain and ugly as possible, because you are showcasing functionalities.

When people see your lo-fi, you don't need to explain it is lo-fi. The moment you try to make lo-fi look attractive, you will move the attention from functionality to look and feel. **You don't want that.**

These are lo-fi designs created through Balsamiq. There are many applications you can use, we will explore FIGMA, which is the standard in the industry, but overall , you should find something you like.

Also remember, you are a **web developer**, not a designer, you need to know the process and the tools, but ultimately, it is not your job to make things look pretty.





# What are frameworks in Javascript?

Frameworks can be defined as “scaffolding” for larger applications. Javascript is a “badly” designed language, remember, it was developed in two weeks. In order to overcome this, a number of encapsulating frameworks have been created to “abstract” many modern functionalities and make it much more robust and simpler to develop. For example, there is not a simple way to “update” the UI dynamically, frameworks like react solve this problem. We normally refer as Vanilla, an application which uses no framework.

Today the most popular are React, Angular and Vue, but many others exist, with pros and cons. React is a UI library, Angular is a fully-fledged front-end framework, while Vue.js is a progressive framework. In this program we will focus on React.

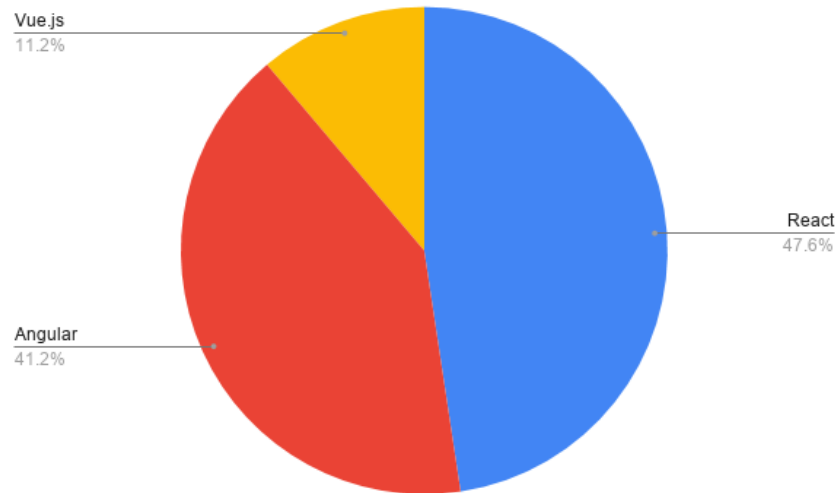
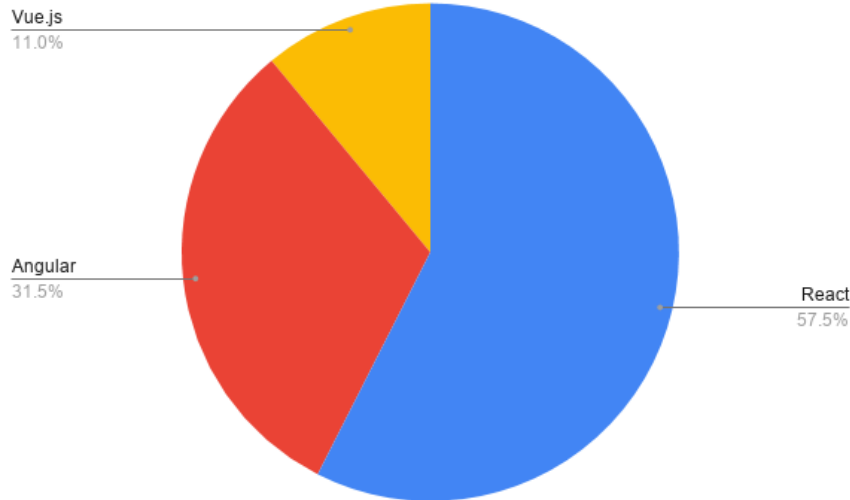




# The Job Market

When entering the job market, it is important to look at the trends, but remember, this may change very quickly. What is trendy today may not exist tomorrow.

As a person in IT, take these numbers with a pinch of salt, also, it is very normal for a good software engineer to be requested to jump from one framework to another, or even change programming language altogether.





# React

React is the least complex of the three frameworks. That is because you only need to import the library, then you can start writing your React application with a few lines of code.

Most React applications are component-based and don't just render a few elements on the page.

You can start using React with just a few lines of code.

```
ReactDOM.render(  
  <h1>Hello, world!</h1>,  
  document.getElementById('root')  
)
```



# React Continued

React Elements are the smallest building blocks of React apps. They are more powerful than DOM elements because the React DOM makes sure to update them efficiently whenever something changes.

Components are larger building blocks that define independent and reusable pieces to be used throughout the application. They accept inputs called props and produce elements that are then displayed to the user.

React is based on JavaScript, but it's mostly combined with JSX (JavaScript XML), a syntax extension that allows you to create elements that contain HTML and JavaScript at the same time.

Anything you create with JSX could also be created with the React JavaScript API, but most developers prefer JSX because it's more intuitive.



# React Problem

React is a client-side rendering framework, which means all data are handled and rendered on the client's browser. This cause problems when:

- the amount of data becomes too large (e.g. 10000 items to be displayed as a list).
- the application/website needs to be found by search engines (because JS pages have low indexing and ranking).
- there are a too many components in the app, then re-rendering management becomes extremely difficult to make only needed components are re-rendered.



# Angular

Angular has the most complex project structure out of the three, and since it's a fully-blown front-end framework, it relies on more concepts.

Since Angular works best with TypeScript, it's important that you know TypeScript when building an Angular project.

Similar to React Angular is also component-based.



# Angular Continued

Projects in Angular are structured into Modules, Components, and Services. Each Angular application has at least one root component and one root module.

Each component in Angular contains a Template, a Class that defines the application logic, and MetaData (Decorators). The metadata for a component tells Angular where to find the building blocks that it needs to create and present its view.



## Other Frameworks - Meteor / Next

Meteor.js : Used develop web and mobile apps, other JS frameworks (like React, Angular) can be embedded as well. Mobile Applications are made possible using Apache Cordova.

NextJS : Based on the React framework. Next.js is a server-rendered React framework that requires little or no configuration.



Institute of  
Data

# Style Guides





# Style Guides

Style Guides are distinct from design pattern libraries, which are a long-standing tool used by UX practitioners to define broad design ideas, rather than specific implementation details.

You may wonder “why so many applications all look the same?” That is because we make use of style guides. Almost every major company has a well defined and DOCUMENTED style guide.

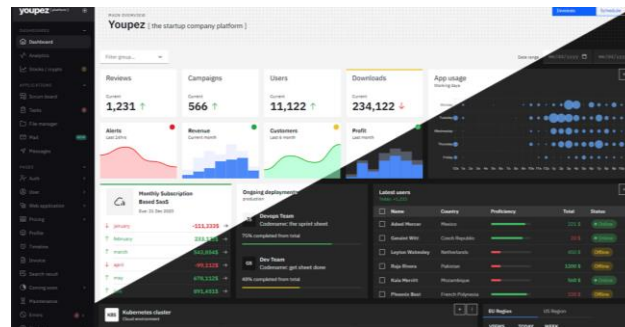
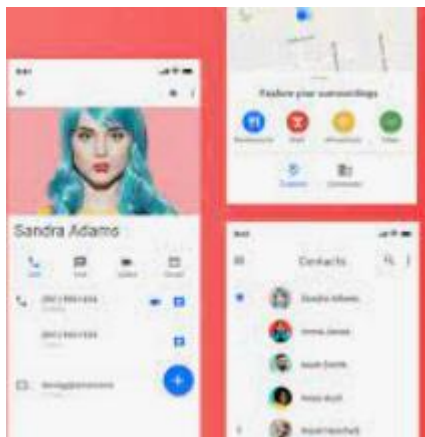
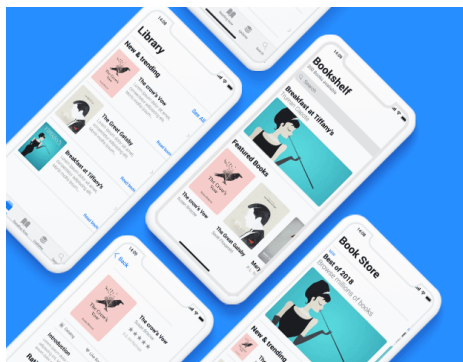
Google has **Material**, Microsoft has **Fluent**, IBM has **Carbon**, Apple has IOS style guides (because Apple is Apple). In this course we will use “Bootstrap”, because it is more generic and it will help you to easily transition everywhere.



# Exercise 1: Some examples

Can you tell which company they belong to?

Do some research and discuss with the class the current trends and a design that you really like.





Institute of  
Data

# Intro to Paper Prototyping



# Paper prototyping

Paper prototyping is a process where design teams create paper representations of digital products to help them realise concepts and test designs. They draw sketches or adapt printed materials and use these low-fidelity screenshot samples to cheaply guide their designs and study users' reactions from early in projects.





Institute of  
Data

FIGMA



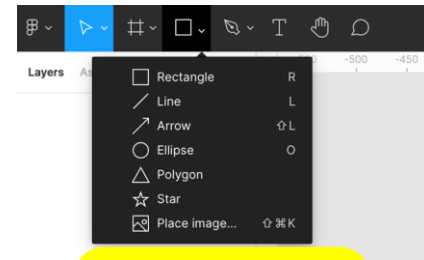
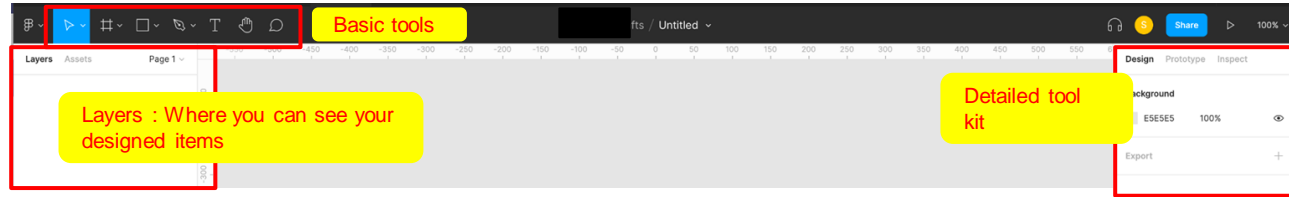
# What is Figma / Why it has become very popular

- Figma is a design / prototyping tool that can be easily shared with multiple team members and collaboratively work together.
- Note/Feedbacks from collaborator helps the designer /design teams to get feedbacks easily.
- One of the strong points of Figma is that Figma is providing Developer's code for UI
- Figma is also cloud-based design/prototyping tool which allows the designers to work on their files, both desktop app & web app. It also has an autosave function.
- Figma can be used for free for an individual.
- Figma provides free membership for students or educators.

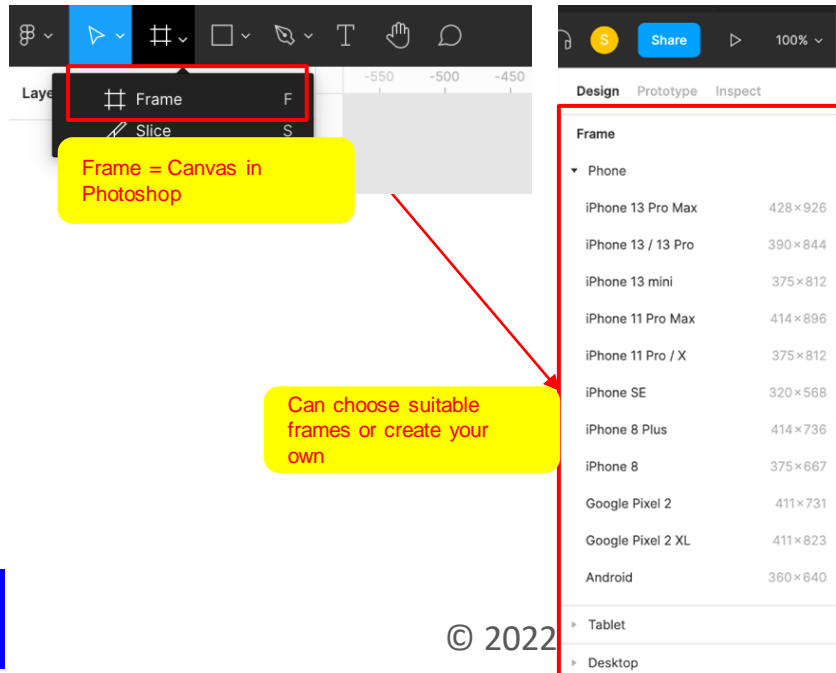
**Most enterprises make use of Figma, it is industry Standard and highly recognised in the designers community as well.**

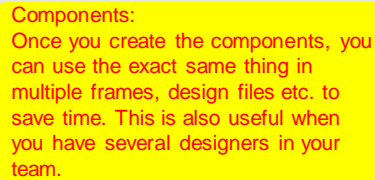
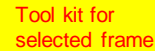


# Figma editing tools



Low/Mid fi prototype is basically created with Rectangles/Circles just to show the concept & ideas

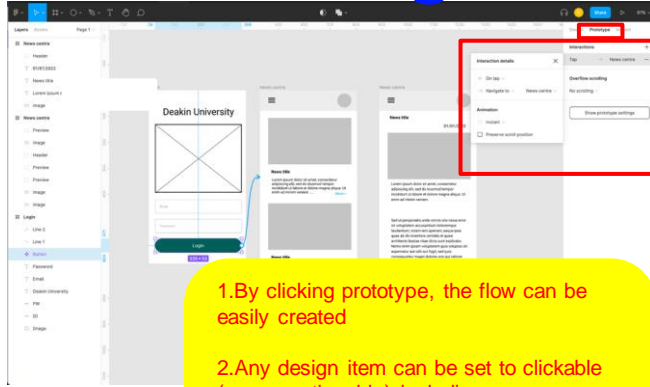








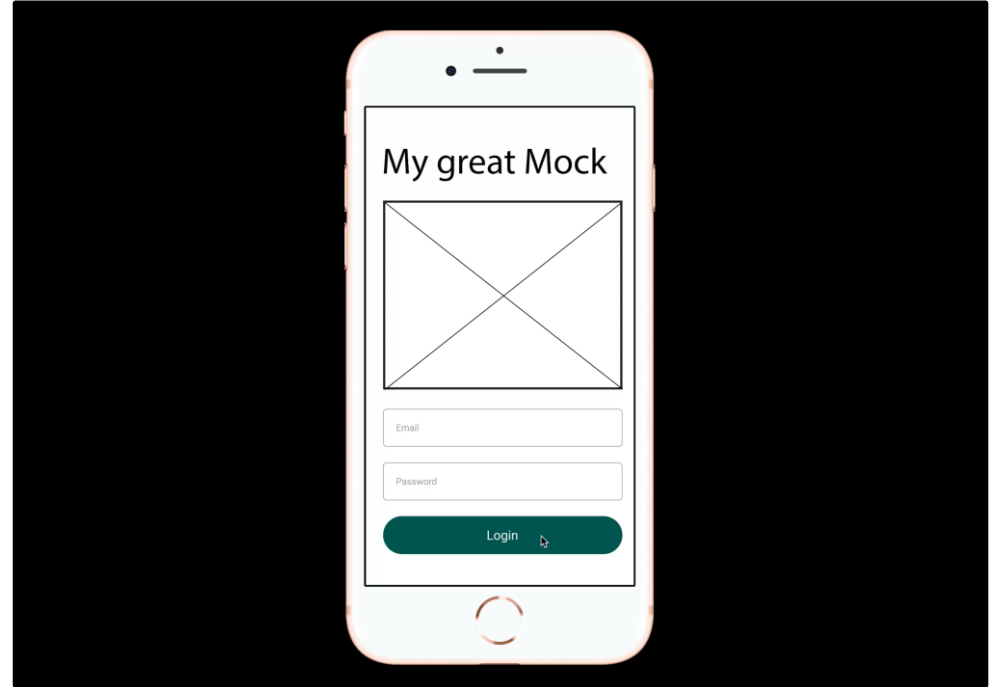
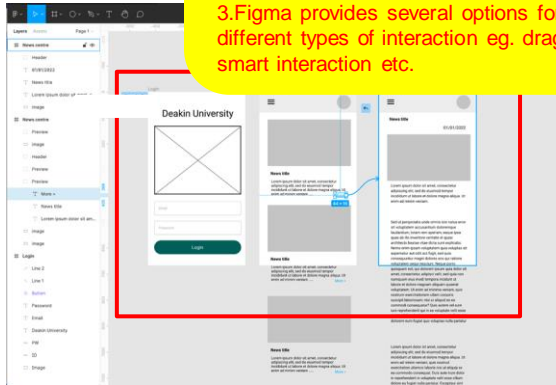
# Creating interactive prototypes



1. By clicking prototype, the flow can be easily created

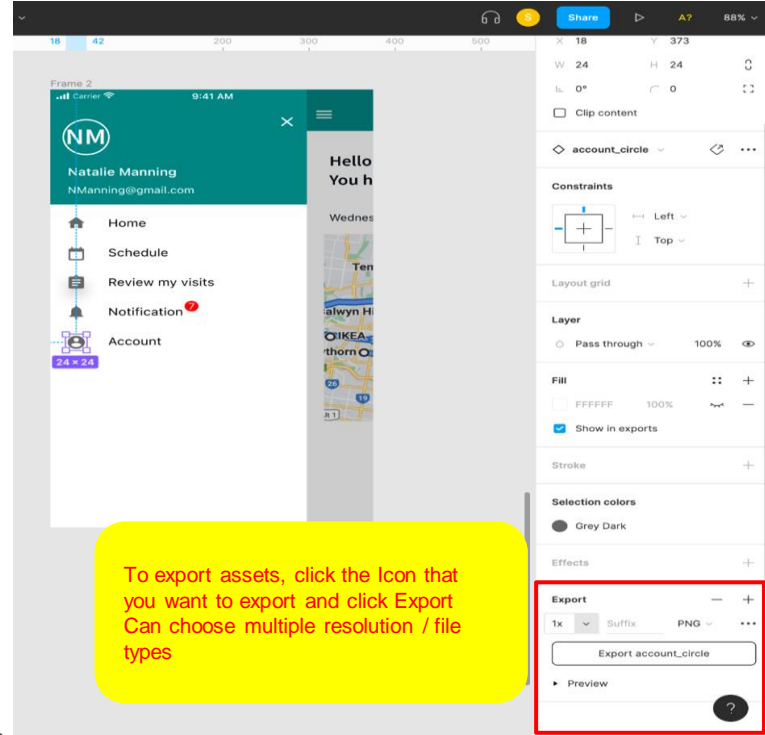
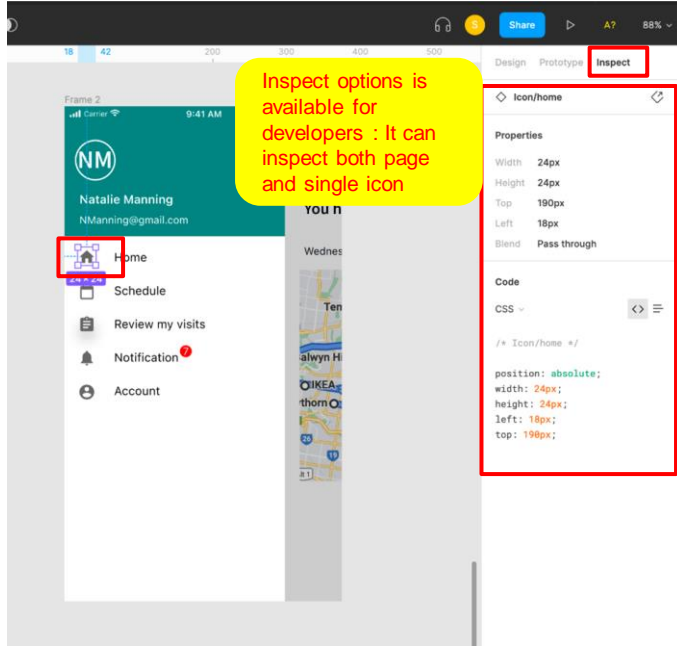
2. Any design item can be set to clickable (or any actionable) including page

3. Figma provides several options for different types of interaction eg. drag, tap, smart interaction etc.



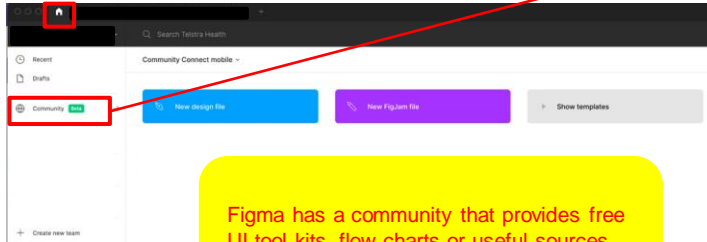


# Figma editing tools

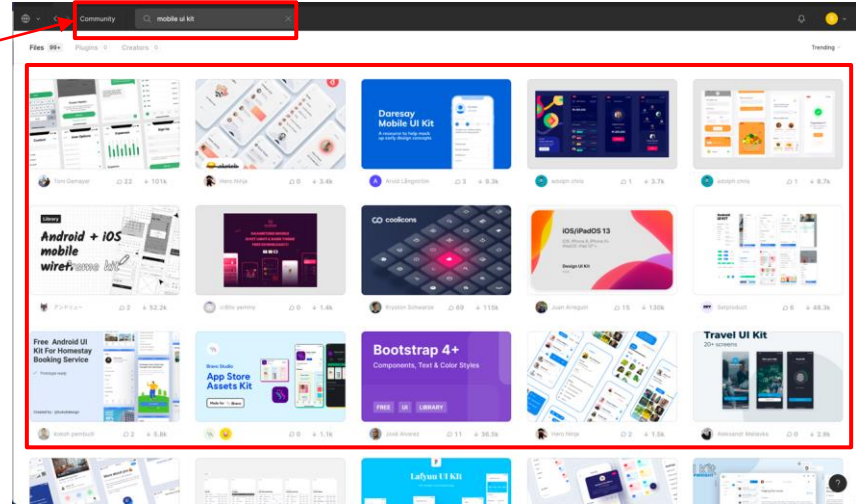




# Free UI tool kit or useful source from community



Figma has a community that provides free UI tool kits, flow charts or useful sources that you can download.





## Lab: Figma

Create a Figma prototype for a social media application, where you can post your content and posts from other users are visible. Don't just jump into Figma, try to follow the design procedure.

Also, this is going to be part of your portfolio, try to do some research and come up with a personal design. For example, it could be a social media application dedicated to the Warhammer Community, or the Ferrari Lovers.



Institute of  
Data

# CUSTOM ELEMENTS

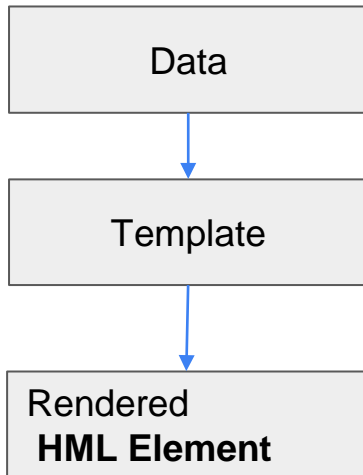


# Templating

In the last 20 years, we have become much more data-oriented. For this reason, we have moved towards the MVC model (Model View Controller).

To simplify this, the various frameworks like React and Angular have become data-centric. To satisfy this Datacentrism, we have moved towards the concept of **Templates**.

Templates are reusable pieces of code that are normally data-centric.

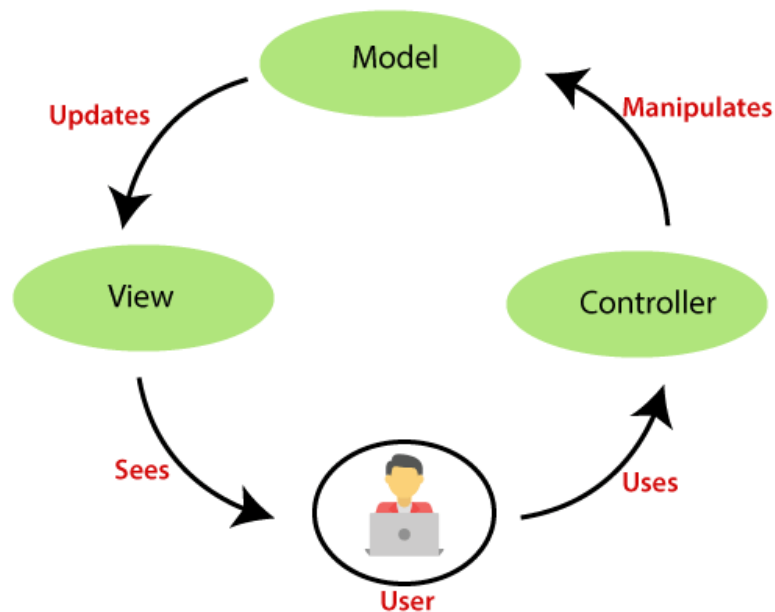




# Short intro to MVC

The MVC model is everywhere, try to get a high level understanding, it is not straight forward the first time you see it.

- Model: Our data is defined in this section. It's where we save our schemas and models, or the blueprint for our app's data.
- View: This includes templates as well as any other interaction the user has with the app. It is here that our Model's data is given to the user.
- Controller: This section is where the business logic is handled. This includes database reading and writing, as well as any other data alterations. This is the link between the Model and the View.





# HTML Template

The HTML `<template>` element represents a template in your markup. It contains "template contents"; essentially inert chunks of cloneable DOM. Templates act as pieces of scaffolding that you can use and reuse throughout the lifetime of an app.

To create a templated content, declare some markup and wrap it in the `<template>` element:

```
<template id="mytemplate">  
  <img src="" alt="great image">  
  <div class="comment"></div>  
</template>
```





# Custom templates

Start by creating a template for a card alongside the CSS required for the template:

You can do this using more than one template. You can use one single file.

## Creating a template :

Below is a template for a card with the CSS written alongside.

```
<html>
  <body>
    <div id="card-list"></div>
  </body>
</html>

<template id="card-template">
  <div class="card">
    <div class="card-body">
      <div class="card-title"></div>
      <div class="card-text"></div>
    </div>
  </div>
</template>

<style>
  body {
    font-family: Arial, Helvetica, sans-serif;
    background-color: #e3f2fd;
  }

  .card {
    padding: 10px;
  }

  .card-title {
    font-weight: 600;
    font-size: 3em;
    padding: 0 0 10px 0;
  }

  .card-description {
    font-weight: 400;
    font-size: 2em
  }
</style>
```



## Custom templates Cont.

In the same file, you can add your JavaScript.

Then JavaScript can be used to add the cards with the same template.

### Example of using a template:

We can use the below script to add a card to your HTML page using template.

```
<script>

function addCard() {
  const template = document.getElementById("card-
template").content.cloneNode(true);
  template.querySelector('.card-title').innerText = 'My Card Title';
  template.querySelector('.card-text').innerText = 'lorem ipsum ble bla';
  document.querySelector('#card-list').appendChild(template);
}
if ('content' in document.createElement('template')) {
  addCard();
}
</script>
```



# Lab - Template

## Exercise 1 :

Modify the function so that you can pass the content for the card dynamically.

## Exercise 2 :

Modify the code so that your page is dynamically generated based on the code coming from an array. This way you will create different as many cards as you need.

## Exercise 3 - the artist's portfolio:

Generate multiple templates and populate the page dynamically. Create profile card and generate cards representing an artist's portfolio.

## Exercise 2

Use the following array

```
Const data=[{name:'bob', age:23},{name:'alice',age:39}]
```

For exercise 3 use the following

```
const artist={  
  name:"Van Gogh",  
  portfolio:[  
    {title:"portrait",  
url:"https://collectionapi.metmuseum.org/api/  
collection/v1/iiif/436532/1671316/main-  
image"},  
    {title:"sky",  
url:"https://mymodernmet.com/wp/wp-  
content/uploads/2020/11/White-house-night-  
van-gogh-worldwide-2.jpg"},  
  ]  
}
```



Institute of  
Data

# CONCEPT TO DESIGN



# Requirements Gathering

When building your application, try to find the best way for yourself to “make others” understand what it is you want to deliver. Not one single path is the best, but here is the author’s way of doing things.

Later on we will try to build our own twitter app!

- 1) What are the key requirements ? For example, an e-shop requires that you have stock, a buy button, or a cart management system. List them all and describe the main parts.
- 2) Sketch the application on paper, trying to understand how everything communicates, what causes something else to trigger. For example, buying a product needs to automatically update the stock list, or if you have a login, you should have signup.
- 3) Define the data, everything spins around data. Define your data models and find any edge cases.
- 4) Use any tool you prefer to create a prototype, and present it to someone who is not you to test, testing your own designs is a “terrible” practice.
- 5) Find ideas and define a color scheme
- 6) Create the HI-FI designs.



## Lab - Let's create a calculator

In this exercise you will need to create a calculator. The application is quite simple, but it requires that you understand the basic flow.

### Requirements

The application takes 2 numbers, and can do 4 operations (+, /, x, -). You need to press the equal button to get the result displayed, and reset to clear it.

- 1) List the requirements, in this case you have a total of 4 requirements
  - a) Get data
  - b) Choose an operator
  - c) Get the result
  - d) Reset the screen
- 2) Sketch the application, so that you are sure about the correct functioning.
- 3) You may use a flow diagram to help.
- 4) Use a tool of your choice, like Figma, to design the application.
- 5) Use the prototype ability and test it.

**Develop the application** . Start from GIT ,it is good practice to do things in a standard way.

- Create a repository
- Clone the repository locally
- Create a branch for each feature



Institute of  
Data

# BOOTSTRAP



# Enter Bootstrap

Have you ever wondered why so many websites all look the same?

Bootstrap is a CSS framework based on HTML, CSS and Javascript. It is used for developing responsive layout and mobile first web projects.

**Bootstrap includes:**

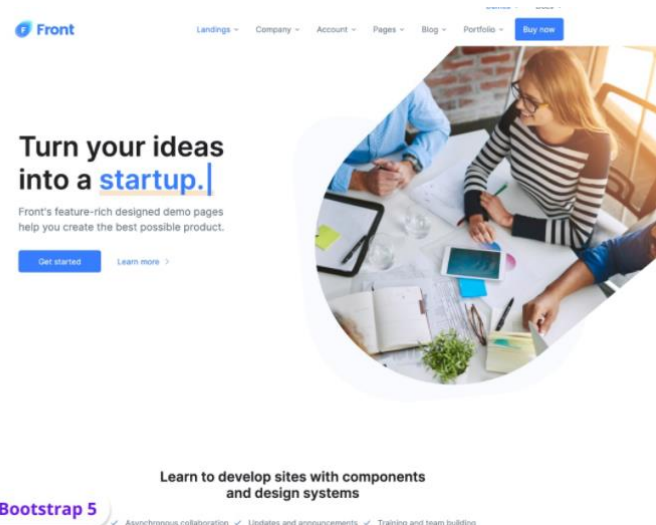
**Basic structure** with a grid system, link styles, and backgrounds etc.

**CSS:** Global CSS settings, the definition of basic HTML element styles, extensible classes, and an advanced grid system.

**Components:** Bootstrap includes a lot of reusable components for creating navigation, menus, cards, pop-up boxes and more.

**JavaScript plugins:** Bootstrap includes custom jQuery plugins. We can use all the plugins directly, or we can use them one by one.

**Customisation:** we can customise Bootstrap components to get our own version.







# Bootstrap Basics

Here are the steps:

For simplicity, use the CDN

CSS:

<https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css>

JS:

<https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js>

Add CSS CDN link to the head tag of the html file.

Add JS CDN link to the bottom of the body tag of the html file.

Add the viewport meta tag to the head of the page to make a Bootstrap website mobile-friendly and ensure screen zooming and proper rendering.

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

```
<!doctype html>
<html lang="en">
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <!-- Bootstrap CSS -->
  <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>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
        integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
        crossorigin="anonymous"></script>
  <nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container-fluid">
      <a class="navbar-brand" href="#">Bootstap Cards</a>
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav"
            aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarNav">
        <ul class="navbar-nav">
          <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="#">Features</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="#">Pricing</a>
          </li>
          <li class="nav-item">
            <a class="nav-link disabled">Disabled</a>
          </li>
        </ul>
      </div>
    </div>
  </nav>
  <main>
  </main>
</body>
</html>
```



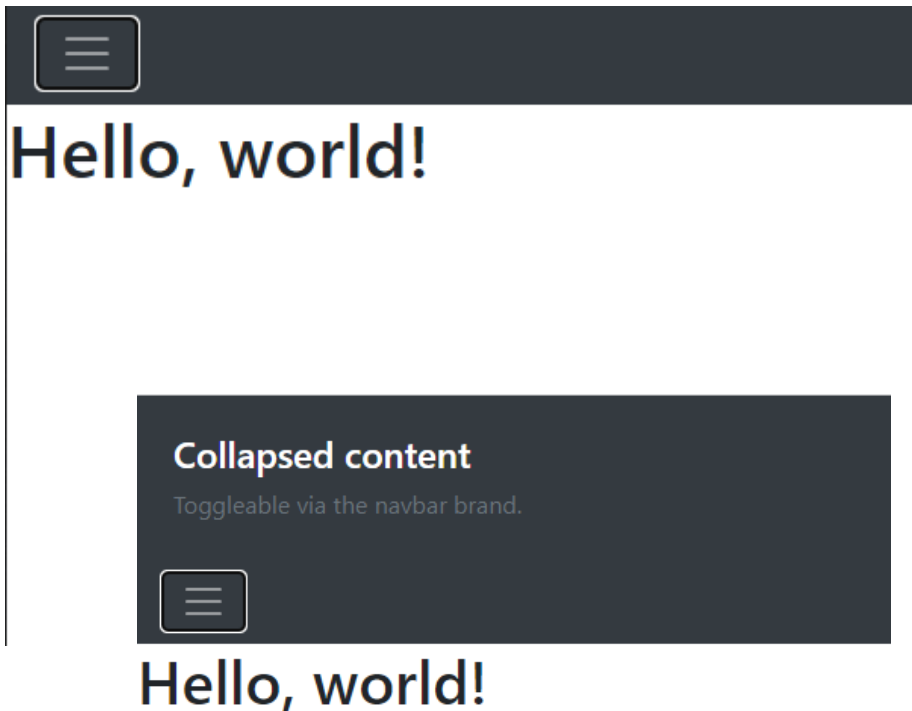
## Simple but effective

Without any effort you create great looking pages using the standard components. Imagine doing a mix and match.

Bootstrap is perfect for beginners, but can become as complex as you want it to be.

Explore more components and play around with them.

<https://getbootstrap.com/docs/4.4/components>





# Bootstrap CSS

Once we import the CSS library to a page, we can use the default CSS style from bootstrap. Remember, these are default, they can still be customised using normal CSS rules.

For example:

```
<div class="bg-primary text-white">hello</div>
```

It will display a blue background and white text, on the right is a standard html button, you can see the improvement.

Hello

Click Me!



# Bootstrap Layouts

The grid system is built with a flexbox that uses containers, rows, and columns to lay out and align content, and is fully responsive.

The grid system divides a row into **12 columns**.

So we can display a row like this:

```
<div class="row">
  <div class="col-12">This is a whole row</div>
</div>
```

We can also divide the rows into two columns, one has 2 columns and one has 10 columns.

```
<div class="row">
  <div class="col-2">
    2 columns
  </div>
  <div class="col-10">
    10 columns
  </div>
</div>
```

This is a whole row

2 columns

10 columns



# Bootstrap layout

## Equal width

If we are not set how many columns are in a “col” class, it will actually divide as same wide.

an example of dividing the same width columns:

```
<div class="row">
  <div class="col bg-primary">column </div>
  <div class="col bg-info">column 2</div>
  <div class="col bg-primary">column </div>
</div>
```

column 1

column 2

column 3



# Responsive Design in Bootstrap

Bootstrap has default breakpoints. It defaults to different widths for different screen sizes (we can customise it as well).

**X-small screen(\*-xs)** :less than 576px

**Small screen(\*-sm)** : 576px and up

**Medium(\*-md)** : 768px and up

**Large(\*-lg)** : 992px and up

**X-large(\*-xl)** : 1200px and up

**Xx-large(\*-xxl)** :1400px and up

## Container Responsive

```
<div class="container-fluid bg-danger"> alway 100% wide
</div>

<div class="container-sm bg-primary">100% wide until
small breakpoint</div>

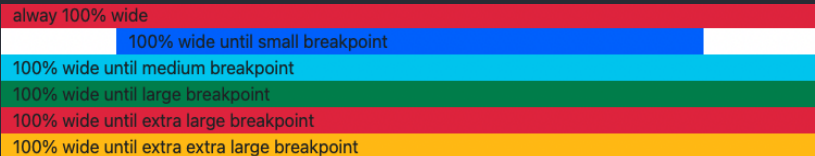
<div class="container-md bg-info">100% wide until medium
breakpoint</div>

<div class="container-lg bg-success">100% wide until
large breakpoint</div>

<div class="container-xl bg-danger">100% wide until extra
large breakpoint</div>

<div class="container-xxl bg-warning">100% wide until
extra extra large breakpoint</div>
```

When we open the browser as medium size, it will display:





# Responsive Design in Bootstrap

## Columns Responsive

For Columns Responsive we can add “col-md-\*/col-sm-\*” etc.

For example, if we want to display a column as a whole row when the screen is small, otherwise it will display as 2 columns.

```
<div class="row">
  <div class="col-12 col-sm-6 bg-info">column</div>
  <div class="col-12 col-sm-6 bg-danger">column</div>
</div>
```

On the small screen:



On width more than the small screen:





# Bootstrap components

Bootstrap has HTML/CSS components and JS components.

HTML/CSS components like button / basic card can directly use by add default CSS styles. otherwise, most of the components have to work with Javascript like Modal, Alert, Carousel.

## The major components of bootstrap

Buttons, Card, Carousel, Dropdowns, Forms, Input Group, List Group, NavBar, Modal, Navs and tabs, Pagination

### Example of Modal:

We can simply add a trigger to the button to display the corresponding modal box.

```
<button type="button" class="btn btn-primary" data-bs-  
toggle="modal" data-bs-target="#exampleModal">  
    Launch demo modal  
</button>  
  
<div class="modal fade" id="exampleModal" tabindex="-1"  
aria-labelledby="exampleModalLabel" aria-hidden="true">  
    <div class="modal-dialog">  
        <div class="modal-content">  
            <div class="modal-body">  
                hello modal  
            </div>  
        </div>  
    </div>  
</div>
```

Launch demo modal

hello modal





# Commonly used components

The navigation bar is one of the most important pieces for your application, this will drive the user.

Making the right decision will allow users to find things quickly. Try to balance functionality with appearance. You don't want to create complex nested menus, but also, you don't want your user to spend too much time looking for things.

## Navigation Bar - Nav

```
<nav class="navbar navbar-expand-lg 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="#navbarNav" aria-controls="navbarNav"
aria-expanded="false"
  aria-label="Toggle navigation"><span class="navbar-toggler-icon"></span></button>
  <div class="collapse navbar-collapse" id="navbarNav">
    <ul class="navbar-nav">
      <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="#">Features</a> </li>
      <li class="nav-item"><a class="nav-link" href="#">Pricing</a> </li>
      <li class="nav-item"> <a class="nav-link disabled">Disabled</a> </li>
    </ul>
  </div>
</div>
```

Navbar Home Features Pricing Disabled



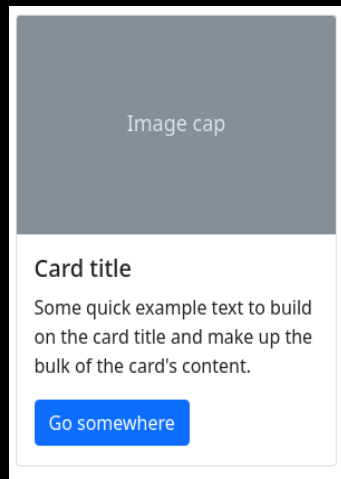
# Commonly used components

Cards are great components that truly give you the sense of data.

Normally, we cards are Leaves objects. Take for example an online shop, some data, passed through an array, is then iteratively rendered as cards.

## Card

```
<div class="card" style="width: 18rem;">
  
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <p class="card-text">Some quick example text to build
on the card title and make up the bulk of the card's
content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```





# Commonly used components

As data becomes more complex, you will need to also find better ways to present it to the users. The accordion is a good example for data-intensive explorations. The accordion is a data view structure that allows for closing and opening.

Accordion Item #1

**This is the first item's accordion body.** It is shown by default, until the collapse plugin adds the appropriate classes that we use to style each element. These classes control the overall appearance, as well as the showing and hiding via CSS transitions. You can modify any of this with custom CSS or overriding our default variables. It's also worth noting that just about any HTML can go within the `.accordion-body`, though the transition does limit overflow.

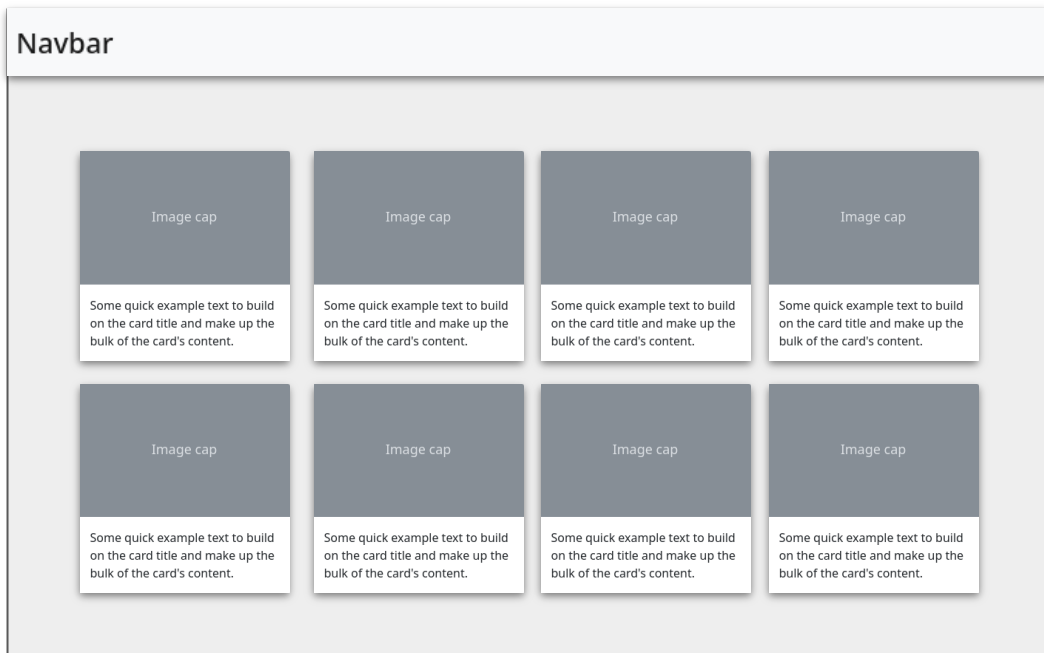
Accordion Item #2

Accordion Item #3

```
<div class="accordion" id="accordionExample">
  <div class="accordion-item">
    <h2 class="accordion-header" id="headingOne"> <button class="accordion-button"
type="button"
  data-bs-toggle="collapse" data-bs-target="#collapseOne" aria-expanded="true" aria-
controls="collapseOne">
      Accordion Item #1 </button> </h2>
    <div id="collapseOne" class="accordion-collapse collapse show" aria-
labelledby="headingOne"
      data-bs-parent="#accordionExample">
      <div class="accordion-body"> <strong>This is the first item's accordion
body.</strong> It is shown by default,
      until the collapse plugin adds the appropriate classes that we use to style each
element. These classes
      control the overall appearance, as well as the showing and hiding via CSS
transitions. <code>.accordion-body</code>, though the transition does limit overflow. </div>
    </div>
  </div>
  <div class="accordion-item">
    <h2 class="accordion-header" id="headingTwo"> <button class="accordion-button
collapsed" type="button"
  data-bs-toggle="collapse" data-bs-target="#collapseTwo" aria-expanded="false"
aria-controls="collapseTwo">
      Accordion Item #2</button></h2>
    <div id="collapseTwo" class="accordion-collapse collapse" aria-labelledby="headingTwo"
      data-bs-parent="#accordionExample">
    <div class="accordion-body"> <strong>This is the second item's accordion
body.</strong> </div>
    </div>
  </div>
  <div class="accordion-item">
    <h2 class="accordion-header" id="headingThree"> <button class="accordion-button
collapsed" type="button"
  data-bs-toggle="collapse" data-bs-target="#collapseThree" aria-expanded="false"
aria-controls="collapseThree">
      Accordion Item #3</button></h2>
    <div id="collapseThree" class="accordion-collapse collapse" aria-
labelledby="headingThree"
      data-bs-parent="#accordionExample">
    <div class="accordion-body"><strong></div>
    </div>
  </div>
</div>
```



# Lab – Display Cards Bootstrap



Using only bootstrap elements, create this web application, making it responsive, 4 cards when large, 2 when medium, and 1 when small.



# What are themes

Themes help you bootstrap applications quickly.

Themes are built as an extension to bootstrap, we can write our own extension plugins, set our custom theme colours to make our own custom theme. It is also able to download and use the official bootstrap theme directly with some settings.

This is an example

<https://startbootstrap.com/theme/sb-admin-2>



Institute of  
Data

DATA



# Dynamic Data

We live in a data-centric world, and everything can be represented as data. In the world wide web, we interpret data and we display it on screen. The same data can be displayed in hundreds of ways, everyone can make a different page to describe it.

We access data through an asynchronous function. Here is an example, we have an array of cars and we return it to the user. We use a timeout to give the illusion of being a real function online.

We can use the below script to add the cards from the carData array to the list.

```
<script>
const carData = [
  {title: 'Audi', description: 'Audi AG is a German automotive manufacturer of luxury vehicles headquartered in Ingolstadt, Bavaria, Germany.'},
  {title: 'Mercedes-Benz', description: 'Mercedes-Benz, commonly referred to as Mercedes, is a German luxury automotive brand.'},
  {title: 'BMW', description: 'Bayerische Motoren Werke AG, commonly referred to as BMW, is a German multinational corporate manufacturer of luxury vehicles and motorcycles headquartered in Munich, Bavaria, Germany.'}
];

function getCars() {
  setTimeout(function() {
    return carData;
  }, 1000)
}
</script>
```



# Dynamic Data and Iterations

We can use what we have explored before before, we iterate through data, so that we don't have to write it one by one.

Given an array of cars - display them all on the screen using html <template> and bootstrap.

Example Code : [sanchitd5/bootstrap\\_example \(github.com\)](https://sanchitd5.github.io/bootstrap_example/)

```
<script>
const carData = [
  {title: 'Audi', description: 'Audi AG is a German automotive manufacturer of luxury vehicles headquartered in Ingolstadt, Bavaria, Germany.'},
  {title: 'Mercedes-Benz', description: 'Mercedes-Benz, commonly referred to as Mercedes, is a German luxury automotive brand.'},
  {title: 'BMW', description: 'Bayerische Motoren Werke AG, commonly referred to as BMW, is a German multinational corporate manufacturer of luxury vehicles and motorcycles headquartered in Munich, Bavaria, Germany.'}
];

function getCars() {
  return carData;
}

function addCards() {
  getCars().forEach((car) => {
    const template = document
      .getElementById("car-card-template")
      .content.cloneNode(true);
    template.querySelector('.card-title').innerText = car.title;
    template.querySelector('.card-text').innerText = car.description;
    document.querySelector('#car-list').appendChild(template);
  });
}

if ('content' in document.createElement('template')) {
  addCards();
}
```





## Lab: Manage Data 1

In this page lab you will show your understanding of using very simple templates.

We are syncing what is on the screen with what is in the data. In the future we will make it much more complex, but for now, we are really just after the concepts.

Part 1 - Use the following array to populate a web page which contains news. When the page loads up, it will display the news in the array.

Use a interval function to read the array every 5 seconds. Every time the array is read, empty the container fill it in with the latest news – so it is always in sync.

```
let news=[  
  { id: 1, title: 'News1', content:"bla"},  
  { id: 2, title: 'News2', content:"ble"},  
  { id: 3, title: 'News3', content:"blu"}  
];
```



## Lab 2

This time you will need to add news to the previous array. So when the interval function come through, it will add your latest post.

To update the array, create a form somewhere in your page, you will pass the title of the news and the content.

There is a trick here, if you use a form and submit, it will trigger a page reload. There are two ways of solving this.

- 1) You can research the prevent default behavior, which stops the form from doing a normal post.
- 2) You can simply recreate the form without using an actual html form. Use Text areas and normal buttons instead.



Institute of  
Data

# Manage Data



# Intro: Data through HTTP

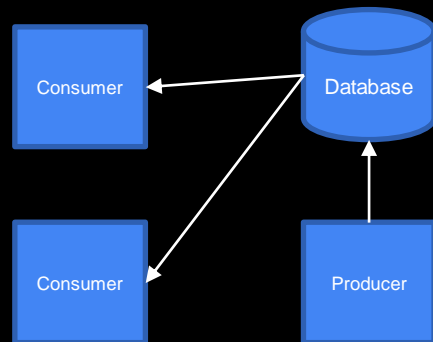
When we talk about data centric applications, we normally refer to how we manage the data.

We normally use the Producer / Consumer model. Normally, a producer will put data into a database, and a consumer will take data from the database and **consume** it by displaying it to the user.

Generally, we produce once, and consume many.

In today's world, we use HTTP methods and Rest API to make this clear.

You will learn more about this in module 5, for now, you should stick with the basic understanding that data is accessed through a Rest interface using a **Producer/Consumer** model.





# Rest APIs

Our stack is based on RESTful interfaces.

A REST API (also known as RESTful API) is an application programming interface (API or web API) that satisfies the constraints of REST architectural style and allows for interaction with RESTful web services. REST stands for representational state transfer and was created by computer scientist Roy Fielding.

What makes it restful:

- A client-server architecture of clients, servers, and resources, with requests managed through HTTP.
- It is Stateless, with no client information stored
- Cacheable data that streamlines client-server interactions.



# HTTP Methods

HTTP defines a set of **request methods** indicating the desired action to be performed for a given resource. Although they can also be nouns, these request methods are sometimes referred to as *HTTP verbs*.

In this course, we will mainly focus on the four main methods, but there are more, which offer more complex interactions.

These are the most 4 most common methods.

## GET

The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.

## POST

The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server.

## PUT

The PUT method replaces all current representations of the target resource with the request payload.

## DELETE

The DELETE method deletes the specified resource.



# Examples

Here are some examples. We normally refer to methods through their resources. For example, we do not refer to a function for creating a user as “createUser” but it will be a POST on a /users.

## HTTP action /URL

## Meaning

GET /things	->	Retrieve <i>list</i> of <b>things</b>
GET /things/23	->	Retrieve <i>one</i> <b>thing</b> identified with 23
POST /things	->	Create a new <b>thing</b>
PUT /things/23	->	Replace thing 23 (or create <b>thing</b> 23)*
DELETE /things/23	->	Delete <b>thing</b> 23



# JSON Placeholder

In this module, we are not touching real backends yet, but we will do the next best thing, using JSON Placeholder, it emulates perfectly how a real server behaves.

- We can simply consume the API from JSON Placeholder to get some dummy data for our front end.
- We will use their “/posts” API which will give us a list (an array) of objects consisting of:
  - userId
  - id
  - title
  - Body

*GET api URL for getting posts from JSON Placeholder*

`https://jsonplaceholder.typicode.com/posts?_limit=10`

*POST api URL for creating posts to JSON Placeholder*

`https://jsonplaceholder.typicode.com/posts`

This ***\_limit*** query parameter tells JSON Placeholder to give us only a specific number of posts (10 in this case).





## Get Data

We use the inbuilt fetch function of the browser to request data. The fetch makes use of **then** keyword, because the data is async, you are telling the application “when you receive a response, do this.” These are all changed, for example we need to parse them from JSON before we can actually use them.

This will display in the console, the last 10 posts . You can change the limit, try to put 1, or 100.

```
<html>
  <head>
    <title>Hello</title>
  </head>
  <body>
    <h1>Hello world</h1>
  </body>
  <script>
    fetch('https://jsonplaceholder.typic
ode.com/posts?_limit=10')
    .then((response) => response.json())
    .then((json) => console.log(json));
  </script>
</html>
```



## Create Data

This time instead, we are creating resources, imagine that you are on Facebook and you are posting your latest update. This is exactly that, you will take the data from the UI and you pass it to the function, which will then update it in the database. In this case, you are a **producer**.

This time instead, we post data. Post is the action to create a resource. This is not being saved on the server — **it is a fake server after all**

```
<html>
  <head>
    <title>Hello</title>
  </head>
  <body>
    <h1>Hello world</h1>
  </body>
  <script>
    fetch('https://jsonplaceholder.typicode.com/posts', {
      method: 'POST',
      body: JSON.stringify({
        title: 'The Studio',
        body: 'Something funny',
        userId: 1,
      }),
      headers: {
        'Content-type': 'application/json; charset=UTF-8',
      },
    })
      .then((response) => response.json())
      .then((json) => console.log(json));
  </script>
</html>
```

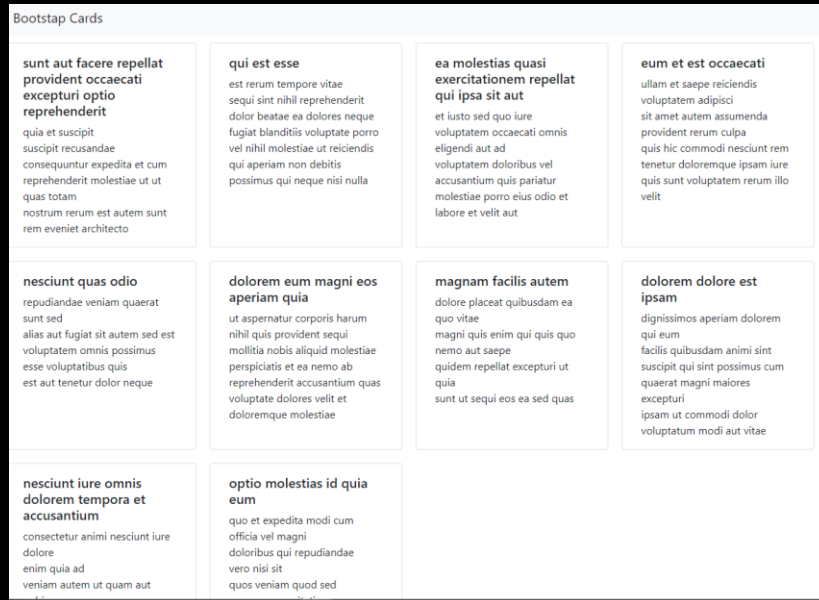


# Lab: Fetch Data

Create a web page that will read the posts from JSONPlaceholder.

Similarly to the previous lab, arrange them on the web page.

Part 1) Make use of the fetch API to retrieve data online and display it. Set the limit default to 10. When the page loads up, it will use the default value.





# Using Axios instead

**Fetch** is a built-in functionality. You can further explore industry standards such as Axios.

**Axios** is an http client to call APIs.

- It is a promise-based HTTP client for the browser and node.js.
- It is used to make XMLHttpRequests from the browser.
- You can use **Axios** to simplify our API calls.

Example use:

```
axios({  
  method: 'get',  
  url: 'https://example.com'  
})  
  .then((data) => {  
    console.log(data)  
  })
```

The method describes what kind of action we want to do on a resource.

[https://axios-http.com/docs/api\\_intro](https://axios-http.com/docs/api_intro)



Institute of  
Data

ENHANCE THE APPLICATION



# CSS Animations

CSS allows animation of HTML elements without using JavaScript or Flash!

To create a CSS animation, we need to use the animation property or its sub-properties, which allows you to configure the animation time, duration and other animation details, but this property does not configure the actual representation of the animation, which is implemented by the [@keyframes](#) rule.

Let's start by looking at the animation sub-properties.



# Animation sub-properties

**animation-name** (The name of the keyframe described by @keyframes.)

**animation-delay** (Sets the delay, which is the time between when the element is loaded and when the animation sequence begins.)

**animation-direction** (Sets whether the animation will run in reverse after each run, or whether it will return to the start position and run again.)

**animation-duration** (Sets the duration of the animation for one cycle.)

**animation-iteration-count** (Sets the number of times the animation will repeat, you can specify infinite to repeat the animation.)

**animation-play-state** (Allows animation to be paused and resumed.)

**animation-timing-function** (Sets the animation speed.)

**animation-fill-mode** (Specifies how to apply a style to the target element before and after the animation is executed.)

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>CSS Animation</title>
  <style type="text/css">
    p {
      animation-duration: 3s;
    }
  </style>
</head>
<body>
  <p>Hello World!</p>
</body>
</html>
```

We can add animation to the `<p>` element, but nothing will happen because we didn't add a keyframe here.

**Next, we will create an Animation by using keyframes.**



# Animation keyframes

Once the timing of the animation has been set, it is time to define how the animation will behave.

This is achieved by creating two or more keyframes using `@keyframes`. Each keyframe describes how the animated element should be rendered at a given point in time.

keyframes use a percentage to specify the point at which the animation occurs.

0% indicates the first moment of the animation and 100% the final moment of the animation.

Because of the importance of these two points in time, there are special aliases: `from` and `to`, both of which are optional; if `from/0%` or `to/100%` is not specified, the browser uses the calculated value to start or end the animation.

Let's add another keyframe and making the animation repeat.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>CSS Animation</title>
  <style type="text/css">
    p {
      animation-duration: 3s;
      animation-name: slidein;
    }
    @keyframes slidein {
      from {
        margin-left: 100%;
        width: 100%;
      }

      to {
        margin-left: 0%;
        width: 100%;
      }
    }
  </style>
</head>
<body>
  <p>Hello World!</p>
</body>
</html>
```

1. We create a `@keyframes` and name it as 'slidein'
2. Then add that animation-name to the style

In this example the `<p>` element will slides from the right to the left of the browser window.





## Animation keyframes (continued)

We add a new keyframe.

When the animation is running at 50% we set the font size and change the width; we also set the distance from the left to 40% to make the text appear to be in the middle.

We also set `animation-iteration-count` to 'infinite' which means it will repeat forever.

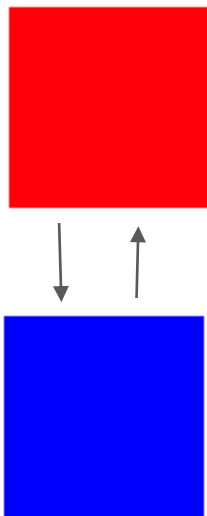
We can also change `animation-iteration-count` to a number. If we change it to '3', it will repeat 3 times.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>CSS Animation</title>
  <style type="text/css">
    p {
      animation-duration: 3s;
      animation-name: slidein;
      animation-iteration-count: infinite;
    }
    @keyframes slidein {
      from {
        margin-left: 100%;
        width: 100%;
      }
      50% {
        font-size: 300%;
        margin-left: 40%;
        width: 150%;
      }
      to {
        margin-left: 0%;
        width: 100%;
      }
    }
  </style>
</head>
<body>
  <p>Hello World!</p>
</body>
</html>
```



# CSS Animation Lab

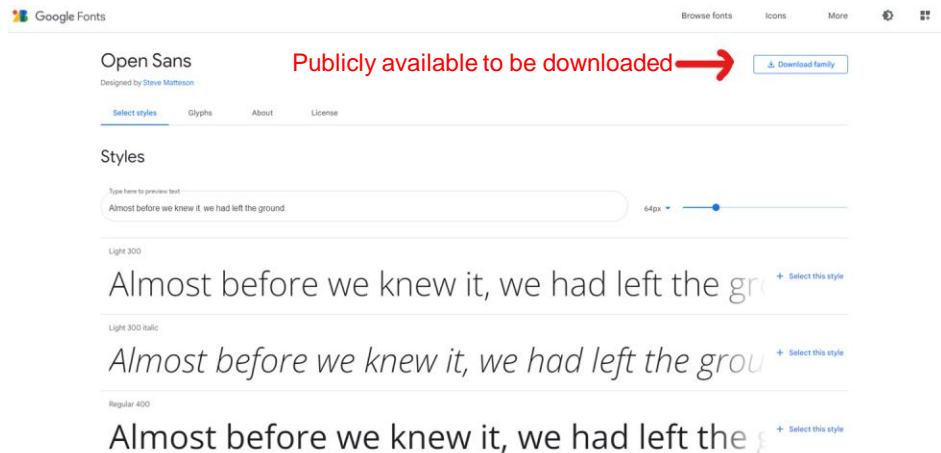
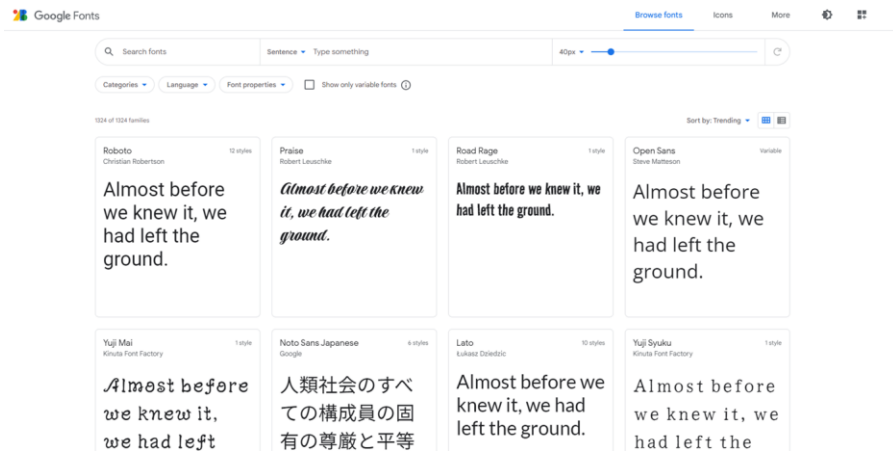
Make a square slider so that it slides from the top to the bottom and returns to its original position, changing background colour in the process.





# Font Libraries

- **Google Fonts:** Google Fonts is a font embedding service library which contains a vast library of publicly available fonts.

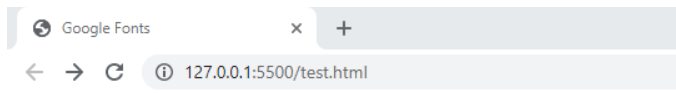




# Google Fonts example

The code on the right will allow you to generate this example. Overall, all you have to do is:

- Reference the font you want to use from Google Fonts.
- Add CSS rules to specify families.



Google Fonts - Roboto Mono

Hello World!!!

```
<!DOCTYPE html>
<html>
<head>
  <title>Google Fonts</title>
  <!-- Reference Google Font -->
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com"
crossorigin>
  <link
href="https://fonts.googleapis.com/css2?family=Roboto+Mono:wght@100&display=swap" rel="stylesheet">

  <!-- Add CSS rules to specify families -->
  <style>
    body {
      font-family: 'Roboto Mono', monospace;
    }
  </style>
</head>
<body>

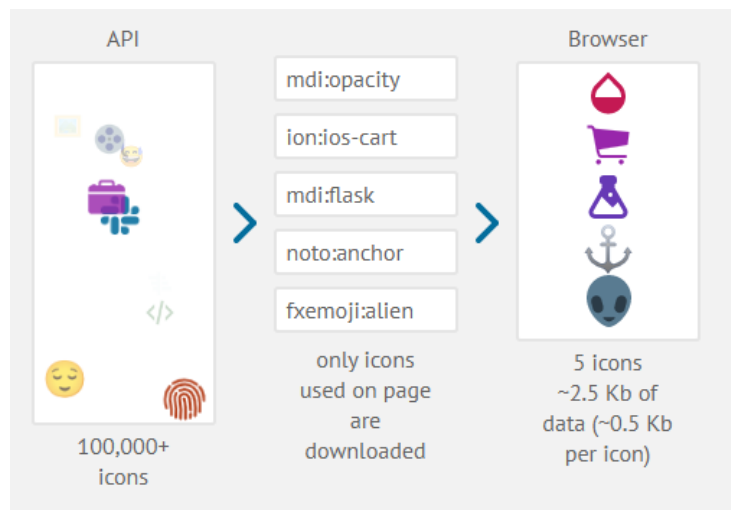
  <div>
    <h1>Google Fonts - Roboto Mono</h1>
    <p>
      Hello World!!!
    </p>

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



# Icon Libraries

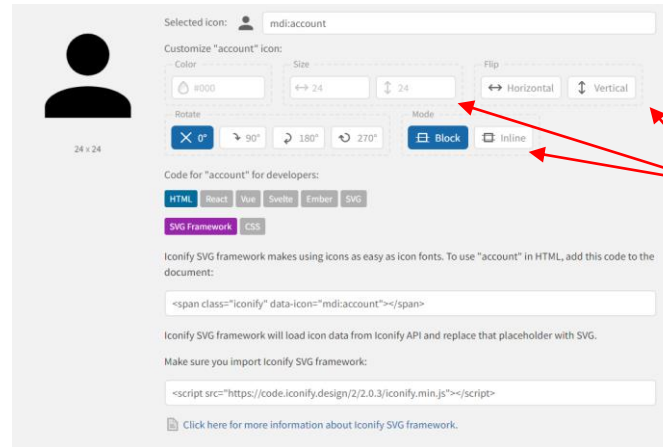
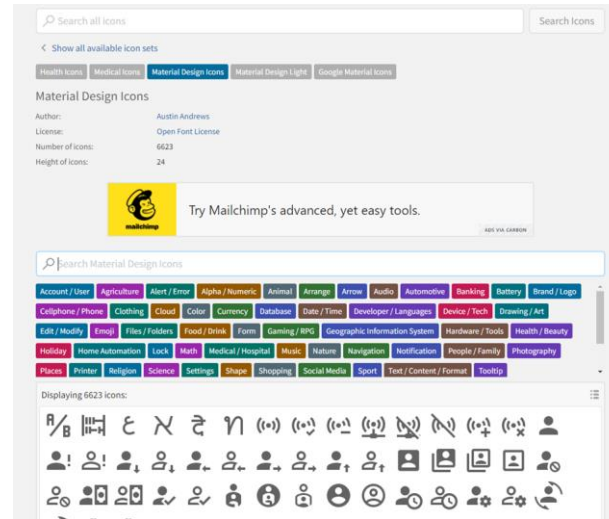
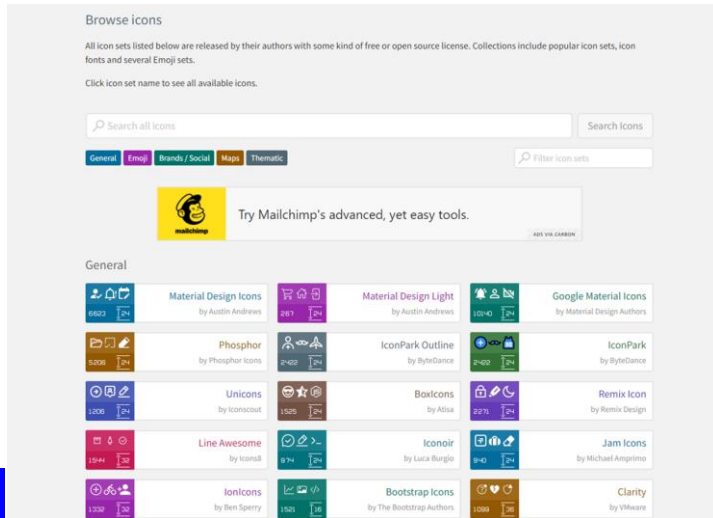
- **Iconify.design:** Iconify.design is a unified icons framework which consists of over 100,000 vector icons.
  - Various collections to choose from.
  - Easy to use.





# Icon Libraries

Numbers of icons with different designs to be used by simply copy and paste from **Iconify** website.



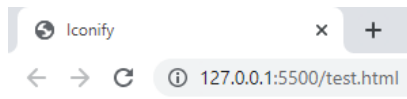
Customisable on the website before copy and paste in your code



# Iconify example

The code on the right will allow you to generate this example. Overall, all you have to do is:

- Import the Iconify SVG framework.
- Add the icon you want to use using “iconify” class




```
<html>
  <head>
    <meta charset="utf-8" />
    <title>Iconify</title>
    <!-- Import Iconify SVG framework -->
    <script src="https://code.iconify.design/2/2.1.0/iconify.min.js">
    </script>
  </head>
  <body>
    <!-- Add icon using iconify class -->
    <span class="iconify" data-icon="mdi-light:account" data-
width="150" data-height="150"></span>
  </body>
</html>
```




# Icon Libraries

Other good icon libraries:

- Ant.design
- Font Awesome

**Font Awesome**



malchimp

Grow sales with a smart marketing platform. Try Malchimp today. ads via Carbon

All 1,608 Awesome Icons

Free

☒ Free

☐ Pro Only

☐ Solid

☐ Regular

☐ Light

☐ Duotone

☐ Brands

☐ Latest Release

500px

accessible-icon

accusoft

acquisitions-incorporated

ad

address-book

address-book

address-card

address-card

adjust

adn

adversal

affiliatetheme

air-freshener

airbnb

algolia

align-center

align-justify

align-left

align-right

alipay

allergies

amazon

amazon-pay

ambulance

american-sign-language-interpreting

amilia

**Icon**Semantic vector graphics. Before use icons, you need to install `@ant-design/icons` package:

```
npm install --save @ant-design/icons
```

**Ant Design**

Basic

Two-tone icon ...

Custom Icon

Use iconfont.cn

Multiple resour...

API

List of icons

☒ Outlined

☐ Filled

☐ Two Tone



Directional Icons

  
StepBackwardOutlined

  
StepForwardOutlined

  
FastBackwardOutlined

  
FastForwardOutlined

  
ShrinkOutlined

  
ArrowsAltOutlined

  
DownOutlined

  
UpOutlined

  
LeftOutlined

  
RightOutlined

  
CaretUpOutlined

  
CaretDownOutlined

  
CaretLeftOutlined

  
CaretRightOutlined

  
UpCircleOutlined

  
DownCircleOutlined

  
LeftCircleOutlined

  
RightCircleOutlined





# DateTime and Internationalisation libraries



- **MomentJS**: One of the most used JavaScript date libraries among software engineers. However, over the past few years, other alternatives challenged its existence.

The screenshot shows the Moment.js website with a dark background and a subtle hexagonal pattern. At the top, there's a navigation bar with logos for Moment, Moment Timezone, and Luxon, along with links to Home, Docs, Guides, Tests, and GitHub. The main content area features a large clock icon in the center. Below it, a prominent banner reads "Black Lives Matter" in white text, followed by the quote "It is not our differences that divide us. It is our inability to recognize, accept, and celebrate those differences. - Audre Lorde" and links to "Donate", "Wikipedia", "Read", "Watch", and "Get Involved". To the left of the clock, text describes Moment.js 2.29.1 as a library to "Parse, validate, manipulate, and display dates and times in JavaScript." To the right, a message suggests considering Moment in a project but notes that "There may be better modern alternatives" and refers to the "Project Status" in the docs. At the bottom, there are two columns: "Download" with links for "moment.js" (18.2k gz) and "moment-with-locales.js" (73.5k gz), and "Install" with a code block showing various installation commands for npm, yarn, nuget, spm, meteor, and bower.

Moment.js 2.29.1  
Parse, validate, manipulate,  
and display dates and times in JavaScript.

Considering using Moment in your project?  
*There may be better modern alternatives.*  
For more details and recommendations,  
please see [Project Status](#) in the docs.

## Black Lives Matter

*It is not our differences that divide us. It is our inability to recognize, accept,  
and celebrate those differences. - Audre Lorde*

[Donate](#) | [Wikipedia](#) | [Read](#) | [Watch](#) | [Get Involved](#)

Download

**moment.js**  
moment.min.js 18.2k gz

**moment-with-locales.js**  
moment-with-locales.min.js 73.5k gz

Install

```
npm install moment --save # npm
yarn add moment # Yarn
Install-Package Moment.js # NuGet
spm install moment --save # spm
meteor add momentjs:moment # meteor
bower install moment --save # bower (deprecated)
```



# DateTime and Internationalisation libraries



- Supports a wide range of languages and libraries.
- **MomentJS** is a legacy project now focusing on **stability**. It is **not dead** but is indeed **done**.

## Using Moment

Moment was designed to work both in the browser and in Node.js.

All code should work in both of these environments, and all unit tests are run in both of these environments.

Currently, the following browsers are used for the ci system: Chrome on Windows XP, IE 8, 9, and 10 on Windows 7, IE 11 on Windows 10, latest Firefox on Linux, and latest Safari on OSX 10.8 and 10.11.

If you want to try the sample codes below, just open your browser's console and enter them.

### # Node.js

```
npm install moment
```

```
var moment = require('moment'); // require
moment().format();
```

Or in ES6 syntax:

```
import moment from 'moment';
moment().format();
```

**Note:** if you want to work with a particular variation of moment timezone, for example using only data from 2012-2022, you will need to import it from the [builds](#) directory like so:

```
import moment from 'moment-timezone/builds/moment-timezone-with-data-2012-2022';
```

**Note:** In **2.4.0**, the globally exported moment object was **deprecated**. It will be removed in next major release.

Project Status

Using Moment

Node.js

Browser

Bower

Require.js

NuGet

meteor

Browserify

Webpack

Typescript

System.js

Other

Troubleshooting

Parse

Get + Set

Manipulate

Display

Query

i18n

Customize

Durations

Utilities

Plugins



# DateTime and Internationalisation libraries

- **Date-FNS:** Date-FNS is similar to Moment JS but is written with a more modular approach.

The screenshot shows the homepage of the date-fns library. The header is dark red with a geometric pattern. It features the library's logo, a description as a 'Modern JavaScript date utility library', and a list of features. A 'Documentation' button is prominent. Below the header, there are links to GitHub, the community, and Twitter. The 'Examples' section is on a light yellow background, showing three tabs: 'Format date' (selected), 'I18n', and 'Composition & FP'. The 'Format date' tab displays a code block with JavaScript code for formatting dates.

(📅) date-fns

Modern JavaScript date utility library

date-fns provides the most comprehensive, yet simple and consistent toolset for manipulating JavaScript dates in a browser & Node.js.

[Documentation](#)

[Star on GitHub](#) [Join the community](#) [Follow on Twitter](#)

Examples

[Format date](#) [I18n](#) [Composition & FP](#)

```
import { format, formatDistance, formatRelative, subDays } from 'date-fns'

format(new Date(), "'Today is a' eeee")
//> "Today is a Monday"

formatDistance(subDays(new Date(), 3), new Date(), { addSuffix: true })
//> "3 days ago"

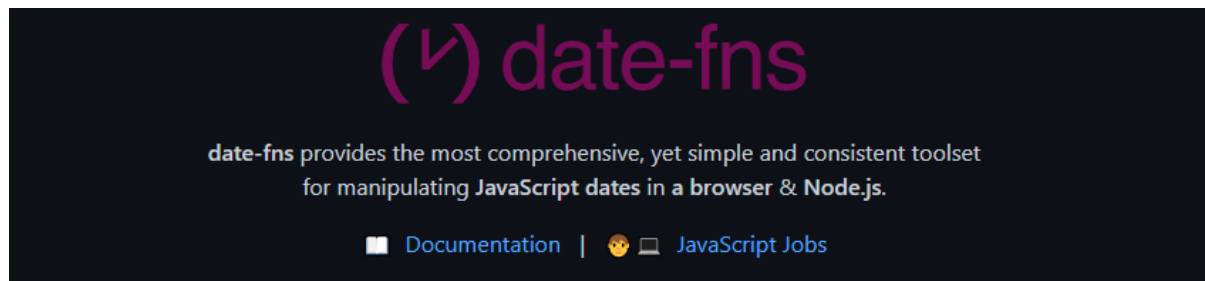
formatRelative(subDays(new Date(), 3), new Date())
//> "last Friday at 7:26 p.m."
```



# DateTime and Internationalisation libraries



- A modern JavaScript date utility library.
- Well maintained and developed.
- Multiple benefits:
  - Modular
  - Fast.
  - Consistent.
  - I18n.
  - Immutable & pure
  - etc.

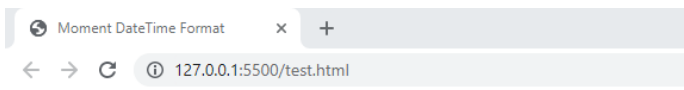




# Moment.js example

The code on the right will allow you to generate this example. Overall, all you have to do is:

- Import Moment library.
- Create the function to format the date/time.
- Place the result into the place you want to render the result.



## Moment Date

Sun Nov 28 2021 22:32:46 GMT+1100

## JavaScript Date

Sun Nov 28 2021 22:32:46 GMT+1100 (Australian Eastern Daylight Time)

```
<html>
<head>
  <title>Moment DateTime Format</title>
  <!-- reference Moment.js library -->
  <script
src="//cdnjs.cloudflare.com/ajax/libs/moment.js/2.7.0/moment.min.js"
type="text/javascript"></script>
</head>
<body>
  <h2>Moment Date</h2>
  <!-- container for Moment.js output -->
  <div id="displayMoment"></div>

  <h2>JavaScript Date</h2>
  <!-- container for JavaScript Date output -->
  <div id="displayJsDate"></div>

  <script type="text/javascript">
(function() {
  // instantiate a moment object
  var NowMoment = moment();

  // instantiate a JavaScript Date object
  var NowDate = new Date();

  // display value of moment object in #displayMoment div
  var eDisplayMoment = document.getElementById('displayMoment');
  eDisplayMoment.innerHTML = NowMoment;

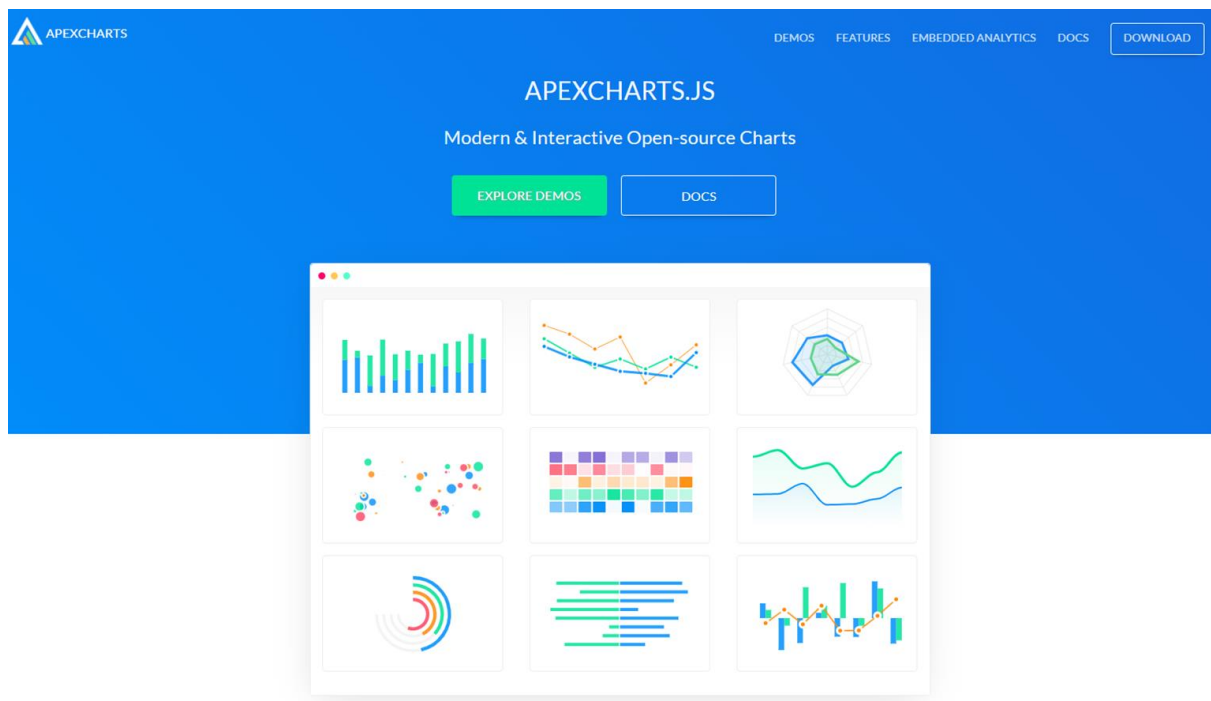
  // display value of Date object in #displayJsDate div
  var eDisplayDate = document.getElementById('displayJsDate');
  eDisplayDate.innerHTML = NowDate;
})();
</script>
</body>
</html>
```



# Charts libraries



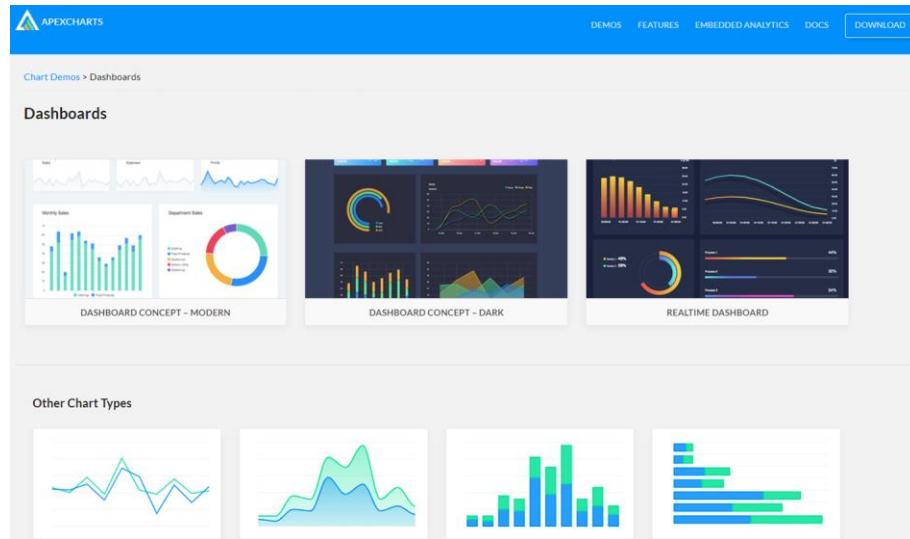
- **Apex Charts:** A modern and interactive open-source charts library.





# Charts libraries

- **Apex Charts** look great on different devices, and the library allows for customisation and comes with comprehensive documentation.
- **Features:**
  - Responsive.
  - Interactive.
  - Dynamic
  - High Performance.
- However, it can be laggy with larger datasets





# Charts libraries

- **Apache Echarts**: is a huge open-source JavaScript library for data visualisation developed by Apache.

## Apache ECharts

An Open Source JavaScript Visualization Library

 Get Started

 Demo

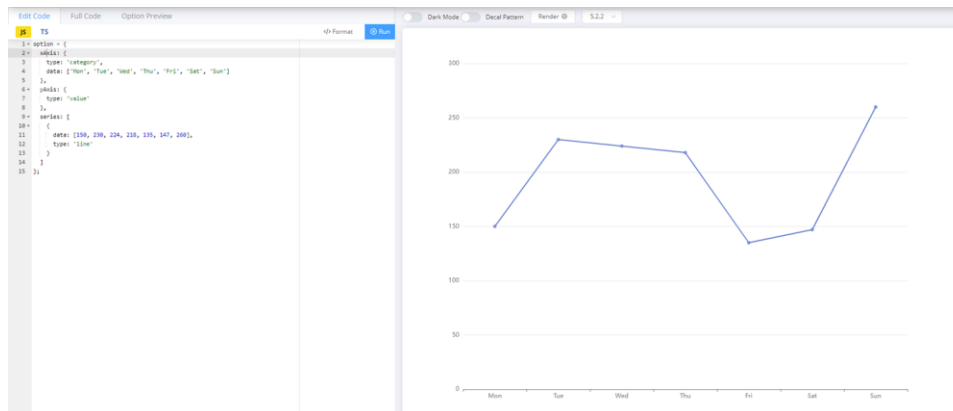
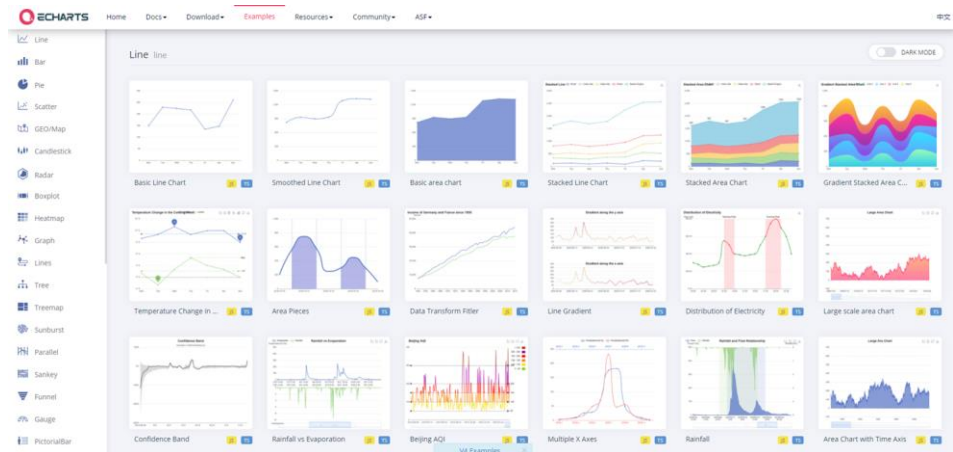






# Charts libraries

- **Apache Echarts** is super useful for JavaScript data visualisations intended for the Web.
- Works great with big datasets.
- It also supports both SVG and Canvas rendering.
- It also provides comprehensive interactable documents with visual examples.

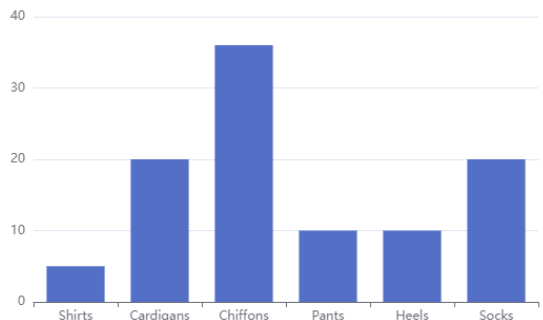




# Echart example

The code on the right will allow you to generate this example. Overall, all you have to do is

- Select a Div where to place the chart
- Create a chart object
- Pass the data to the chart object and render it using the setOption



© 2022 Institute of Data

```
<html>
  <head>
    <meta charset="utf-8" />
    <title>ECharts</title>
    <!-- Include the ECharts file you just downloaded -->
    <script src="echarts.js"></script>
  </head>
  <body>
    <!-- Prepare a DOM with a defined width and height for ECharts -->
    <div id="main" style="width: 600px;height:400px;"></div>
    <script type="text/javascript">
      // Initialise the echarts instance based on the prepared dom
      var myChart = echarts.init(document.getElementById('main'));

      // Specify the configuration items and data for the chart
      var option = {
        title: {
          text: 'ECharts Getting Started Example'
        },
        tooltip: {},
        legend: {
          data: ['sales']
        },
        xAxis: {
          data: ['Shirts', 'Cardigans', 'Chiffons', 'Pants', 'Heels', 'Socks']
        },
        yAxis: {},
        series: [
          {
            name: 'sales',
            type: 'bar',
            data: [5, 20, 36, 10, 10, 20]
          }
        ]
      };
      // Display the chart using the configuration items and data just
      // specified.
      myChart.setOption(option);
    </script>
  </body>
</html>
```



## Lab: Create social posts page

- Utilise what you have learnt here to create a rich application.
  - Start with what you want to show, for example, a hobby of yours.
  - Create requirements, do some research, perhaps Pinterest to get some ideas.
  - Prepare your lo-fi / sketches
  - Prepare your application on Figma
  - Get Coding and remember to use responsive design.
- Add things such as data, animations, charts.
- Discuss the application with your lead trainer, in order to get feedback.
- Upload everything to GitHub, potentially, make a GitHub page. You may want to start showcasing your knowledge.