

1. Housekeeping

Symbol explanation:



- A task that you need to perform. Please ask for support if you are stuck.



- The time that you need to work individually on a given task.

2. HTML and CSS

In this part we will cover HTML and CSS basics. In the end of this section, you would be able to build a static blog website using HTML and CSS.



Figure 1: The rendered blog website using HTML and CSS

We will go step by step and create the web site in *Figure 1*.

HTML

Stage 1: Set up the basic structure of the blog website.

Steps:

1. Create a new HTML file and open it in a code editor.
2. Add the HTML doctype declaration at the beginning: `<!DOCTYPE html>`.
3. Inside the `<body>` tag, add a `<header>` element to represent the website header.
4. Within the `<header>` element, add an `<h1>` element with the blog website title.
5. Below the `<header>`, add a `<main>` element to contain the main content of the website.

HTML code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Blog Website</title>
</head>
<body>
  <header>
    <h1>My Blog</h1>
  </header>
  <main>
    <!-- Content goes here -->
  </main>
</body>
</html>
```

Stage 2: Add a navigation menu.

Steps:

1. Inside the <header> element, create a <nav> element.
2. Within the <nav>, add an unordered list with several list items .
3. Each list item should contain an anchor <a> element with a meaningful link text.
4. Add appropriate class or id attributes to the elements for styling and identification.

HTML code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Blog Website</title>
</head>
<body>
  <header>
    <h1>My Blog</h1>
    <nav>
      <ul>
        <li><a href="#">Home</a></li>
```

```

        <li><a href="#">About</a></li>
        <li><a href="#">Blog</a></li>
        <li><a href="#">Contact</a></li>
    </ul>
</nav>
</header>
<main>
    <!-- Content goes here -->
</main>
</body>
</html>

```

Stage 3: Create the layout for blog posts.

Steps:

1. Inside the <main> element, add a container <div> to hold the blog posts.
2. Within the container <div>, create individual blog post sections using <article> elements.
3. Each <article> should contain a heading <h2> for the post title, and a paragraph <p> for the post content.

HTML code:

```

<!DOCTYPE html>
<html>
<head>
    <title>Blog Website</title>
</head>
<body>
    <header>
        <h1>My Blog</h1>
        <nav>
            <ul>
                <li><a href="#">Home</a></li>
                <li><a href="#">About</a></li>
                <li><a href="#">Blog</a></li>
                <li><a href="#">Contact</a></li>
            </ul>

```

```

    </nav>
</header>
<main>
  <div class="container">
    <article>
      <h2>First Blog Post</h2>
      <p>This is the content of the first blog post.</p>
    </article>
    <article>
      <h2>Second Blog Post</h2>
      <p>This is the content of the second blog post.</p>
    </article>
  </div>
</main>
</body>
</html>

```

Stage 4: Add a footer.

Steps:

1. Outside the <header> and <main>, create a <footer> element.
2. Within the <footer>, add relevant information such as copyright notice, contact details, or social media links.

HTML code:

```

<!DOCTYPE html>
<html>
<head>
  <title>Blog Website</title>
</head>
<body>
  <header>
    <h1>My Blog</h1>
    <nav>
      <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">About</a></li>

```

```

    <li><a href="#">Blog</a></li>
    <li><a href="#">Contact</a></li>
  </ul>
</nav>
</header>
<main>
  <div class="container">
    <article>
      <h2>First Blog Post</h2>
      <p>This is the content of the first blog post.</p>
    </article>
    <article>
      <h2>Second Blog Post</h2>
      <p>This is the content of the second blog post.</p>
    </article>
  </div>
</main>
<footer>
  <p>&copy; 2023 My Blog. All rights reserved.</p>
  <p>Contact: info@example.com</p>
</footer>
</body>
</html>

```

Stage 5: Your turn

Create a new page for the "About" section. The new page should have a heading, a paragraph describing the website, and an image related to the website's topic. Add a link to the "About" section in the navigation bar of the index.html, and make sure it navigates to the newly created "About" page when clicked.



Working for 5 min.

CSS

CSS (Cascading Style Sheets) is a style sheet language used to describe the look and formatting of a document written in HTML. It allows you to control the visual presentation of your web page, including layout, colors, fonts, and more. CSS works by applying styles to HTML elements using selectors and declarations.

Selectors in CSS determine which elements on the web page the styles should be applied to. Here are some commonly used CSS selectors:

Element Selector:

1. Syntax: `elementname { }`
2. Example: `h1 { }`
3. Targets all elements of a specific HTML tag name, such as `<h1>`, `<p>`, or `<div>`.

Class Selector:

1. Syntax: `.classname { }`
2. Example: `.container { }`
3. Targets all elements with a specific class attribute, specified using the class attribute in HTML. Multiple elements can share the same class.

ID Selector:

1. Syntax: `#idname { }`
2. Example: `#header { }`
3. Targets a specific element with a unique ID attribute, specified using the id attribute in HTML. Each ID should be unique within the document.

Descendant Selector:

1. Syntax: `parent descendant { }`
2. Example: `article h2 { }`
3. Targets elements that are descendants of another element. In this example, it targets `<h2>` elements that are descendants of `<article>` elements.

Stage 1: Add a CSS file to the HTML file.

Steps:

1. Create a CSS file:
 - a. Open a code editor and create a new file.
 - b. Save the file with a `.css` extension, such as `style.css`.
 - c. Note the location or directory where you saved the CSS file.
 - d. You can keep the file empty at this stage.
2. Link the CSS file to the HTML file:
 - a. Open the HTML file in a code editor.
 - b. In the `<head>` section of the HTML file, add the following code: `<link rel="stylesheet" href="style.css">`
 - c. We assume that you created a `style.css` file in the same directory.
3. Save the HTML file.

Final code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Blog Website</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <header>
    <h1>My Blog</h1>
    <nav>
      <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">About</a></li>
        <li><a href="#">Blog</a></li>
        <li><a href="#">Contact</a></li>
      </ul>
    </nav>
  </header>
  <main>
    <div class="container">
      <article>
        <h2>First Blog Post</h2>
        <p>This is the content of the first blog post.</p>
      </article>
      <article>
        <h2>Second Blog Post</h2>
        <p>This is the content of the second blog post.</p>
      </article>
    </div>
  </main>
  <footer>
    <p>&copy; 2023 My Blog. All rights reserved.</p>
    <p>Contact: info@example.com</p>
  </footer>
</body>
</html>
```

Stage 2: Set a width and center the website.

Steps:

1. Select the body element using its element selector.
2. Add the following properties:
 - a. Set the border property to 1px solid #ccc to create a border around the body.
 - b. Set the width property to 80% to make the body occupy 80% of the available width.
 - c. Set the margin property to 0 auto to center the body horizontally.

CSS code:

```
body {  
  border: 1px solid #ccc;  
  width: 80%;  
  margin: 0 auto;  
}
```

Stage 3: Style the header.

Steps:

1. Open your CSS file.
2. Select the <header> element using its element selector.
3. Add the background-color property and set it to a color of your choice, such as lightblue.
4. Select the <h1> element under <header> using descended selector
5. Add padding property and set it to 10px.
6. Add margin property and set it to 0.

CSS code:

```
header {  
  background-color: lightblue;  
}  
  
header h1 {  
  padding: 10px;  
  margin: 0;  
}
```


Stage 4: Style the navigation menu.

Steps:

1. Open your CSS file.
2. Select the <nav> element using its element selector.
3. Add the following properties:
 - a. Set the background-color to a color of your choice.
 - b. Set the padding to create some space around the menu items.
4. Select the element within the <nav> using the element selector.
5. Add the following properties:
 - a. Set the list-style-type to none to remove the bullet points.
 - b. Set the display property to flex to enable flexible layout.
 - c. Set the justify-content property to space-between to evenly distribute the menu items horizontally.
6. Select the elements within the using the element selector.
7. Add the following properties:
 - a. Set the margin property to create some space between the menu items.

CSS code:

```
nav {  
  background-color: lightgray;  
  padding: 10px;  
}  
  
nav ul {  
  list-style-type: none;  
  display: flex;  
  justify-content: space-between;  
}  
  
nav ul li {  
  margin: 0 10px;  
}
```

Stage 5: Change the font style of the blog post titles.

Steps:

1. Select the <h2> elements inside the <article> using the descendant selector.
2. Add the font-family property and set it to a font name or font stack of your choice.
3. Add the color property and set it to a color of your choice.

CSS code:

```
article h2 {  
  font-family: Arial, sans-serif;  
  color: #333;  
}
```

Stage 6: Add a border around the container of blog posts.

Steps:

1. Select the .container class using the class selector.
2. Add the border property and set it to 1px solid #ccc to create a solid border with a light gray color.

CSS code:

```
.container {  
  border: 1px solid #ccc;  
}
```

Stage 7: Style the footer.

Steps:

1. Select the <footer> element using its element selector.
2. Add the following properties:
 - a. Set the background-color to a color of your choice.
 - b. Set the padding to create some space around the content.
 - c. Set the text-align to center value to center the text.

CSS code:

```
footer {  
  background-color: lightgray;  
  padding: 10px;  
  text-align: center;  
}
```

The final CSS code:

```
body {
  border: 1px solid #ccc;
  width: 80%;
  margin: 0 auto;
}

header {
  background-color: lightblue;
}

header h1 {
  padding: 10px;
  margin: 0;
}

nav {
  background-color: lightgray;
  padding: 10px;
}

nav ul {
  list-style-type: none;
  display: flex;
  justify-content: space-between;
}

nav ul li {
  margin: 0 10px;
}

article h2 {
  font-family: Arial, sans-serif;
  color: #333;
}

.container {
  border: 1px solid #ccc;
  padding: 10px;
}
```

```

}

footer {
  background-color: lightgray;
  padding: 10px;
  text-align: center;
}

```

Stage 8: Your turn

Add author information to each article at the end of the article with smaller font size, gray colored font and right aligned.



Working for 5 min.

Bootstrap CSS

Bootstrap CSS is a popular front-end framework that helps developers quickly build responsive and visually appealing websites. It provides a set of pre-designed CSS and JavaScript components that can be easily incorporated into your web projects. Whether you're a beginner or an experienced developer, Bootstrap can significantly streamline your workflow and make your development process more efficient. More info:

<https://getbootstrap.com/docs/4.3/getting-started/introduction/>

Stage 1: Add Bootstrap CSS style file

```

<head>
  <title>Blog Website</title>
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.3.1/dist/css/bootstrap
.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x
9JvoRxT2MZw1T" crossorigin="anonymous">
</head>

```

Stage 2: Create a navigation bar

Steps:

1. Start with the basic HTML structure for a navigation bar.

2. Add the necessary Bootstrap classes to the HTML elements to create a responsive navigation bar.
3. Customize the navigation bar by adding your own brand name and navigation links.

```
<header class="container">
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <a class="navbar-brand" href="#">My Blog</a>
    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav">
        <li class="nav-item active">
          <a class="nav-link" href="#">Home<span
class="sr-only">(current)</span></a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">About</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Blog</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Contact</a>
        </li>
      </ul>
    </div>
  </nav>
</header>
```

Stage 3: Create a responsive grid layout

Steps:

1. Start with a container element to hold the grid.
2. Divide the container into rows using the `<div class="row">` element.
3. Inside each row, create columns using the `<div class="col-md-6">` element.
4. Add article content to the columns

```
<div class="container">
  <div class="row">
    <div class="col-md-6">
```

```

<article>
  <h2>First Blog Post 1</h2>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Sed consequat nisi ut quam congue consequat. Mauris quis erat ac
justo consectetur commodo at id nisi. Pellentesque nec est elit.
Fusce pharetra gravida massa ut sollicitudin. Nulla facilisi. Nulla
laoreet eleifend augue vel convallis. Nulla sed ex tellus.</p>
  <p>Integer in urna dapibus, vulputate neque id, rutrum lorem.
Sed interdum, justo sed rutrum pharetra, arcu nisi fringilla neque,
sed commodo justo est eget nisl. Nulla sed ligula purus. Morbi vitae
mauris vel nisl fermentum posuere a nec velit. Nam mollis odio ut leo
congue dapibus.</p>
</article>
</div>
<div class="col-md-6">
  <article>
    <h2>Second Blog Post</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Sed consequat nisi ut quam congue consequat. Mauris quis erat ac
justo consectetur commodo at id nisi. Pellentesque nec est elit.
Fusce pharetra gravida massa ut sollicitudin. Nulla facilisi. Nulla
laoreet eleifend augue vel convallis. Nulla sed ex tellus.</p>
    <p>Integer in urna dapibus, vulputate neque id, rutrum lorem.
Sed interdum, justo sed rutrum pharetra, arcu nisi fringilla neque,
sed commodo justo est eget nisl. Nulla sed ligula purus. Morbi vitae
mauris vel nisl fermentum posuere a nec velit. Nam mollis odio ut leo
congue dapibus.</p>
  </article>
</div>
</div>
</div>

```

Stage 4: Create a footer

Steps:

1. Start with the HTML structure by creating a <footer> element with the classes footer, bg-dark, and text-light.

2. Inside the footer, add a `<div>` element with the class `container` to create a centered container for the footer content.
3. Within the container, create a `<div>` element with the class `row` to divide the content into two columns.
4. Inside the row, create two columns using `<div>` elements with the class `col-md-6` to create a responsive layout for the columns.
5. Inside the first column, add a `<p>` element with the content `© 2023 My Blog. All rights reserved..`
6. Inside the second column, add a `<p>` element with the content `Contact: info@example.com.`

```
<footer class="footer bg-dark text-light">
  <div class="container">
    <div class="row">
      <div class="col-md-6">
        <p>&copy; 2023 My Blog. All rights reserved.</p>
      </div>
      <div class="col-md-6">
        <p>Contact: info@example.com</p>
      </div>
    </div>
  </div>
</footer>
```

The final HTML Code

```
<!DOCTYPE html>
<html>
<head>
  <title>Blog Website</title>
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.3.1/dist/css/bootstrap
.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x
9JvoRxT2MZw1T" crossorigin="anonymous">
</head>
<body>
  <header class="container">
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
```

```
<a class="navbar-brand" href="#">My Blog</a>
<div class="collapse navbar-collapse" id="navbarNav">
  <ul class="navbar-nav">
    <li class="nav-item active">
      <a class="nav-link" href="#">Home <span
class="sr-only">(current)</span></a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">About</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Blog</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Contact</a>
    </li>
  </ul>
</div>
</nav>
</header>
<div class="container">
  <div class="row">
    <div class="col-md-6">
      <article>
        <h2>First Blog Post 1</h2>
        <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Sed consequat nisi ut quam congue consequat. Mauris quis erat ac
justo consectetur commodo at id nisi. Pellentesque nec est elit.
Fusce pharetra gravida massa ut sollicitudin. Nulla facilisi. Nulla
laoreet eleifend augue vel convallis. Nulla sed ex tellus.</p>
        <p>Integer in urna dapibus, vulputate neque id, rutrum
lorem. Sed interdum, justo sed rutrum pharetra, arcu nisi fringilla
neque, sed commodo justo est eget nisl. Nulla sed ligula purus. Morbi
vitae mauris vel nisl fermentum posuere a nec velit. Nam mollis odio
ut leo congue dapibus.</p>
      </article>
    </div>
    <div class="col-md-6">
```



```

<article>
  <h2>Second Blog Post</h2>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed consequat nisi ut quam congue consequat. Mauris quis erat ac justo consectetur commodo at id nisi. Pellentesque nec est elit. Fusce pharetra gravida massa ut sollicitudin. Nulla facilisi. Nulla laoreet eleifend augue vel convallis. Nulla sed ex tellus.</p>
  <p>Integer in urna dapibus, vulputate neque id, rutrum lorem. Sed interdum, justo sed rutrum pharetra, arcu nisi fringilla neque, sed commodo justo est eget nisl. Nulla sed ligula purus. Morbi vitae mauris vel nisl fermentum posuere a nec velit. Nam mollis odio ut leo congue dapibus.</p>
</article>
</div>
</div>
</div>
<footer class="footer bg-dark text-light">
  <div class="container">
    <div class="row">
      <div class="col-md-6">
        <p>&copy; 2023 My Blog. All rights reserved.</p>
      </div>
      <div class="col-md-6">
        <p>Contact: info@example.com</p>
      </div>
    </div>
  </div>
</footer>
</body>
</html>

```

Stage 5: Your turn

Change the layout of the blog website and make each article extend to the whole width of the page.

Hint: <https://getbootstrap.com/docs/4.3/layout/grid/>



Working for 3 min.

Stage 6: Your turn

Add an alert box on top of blog posts to advertise the latest news.

Hint: <https://getbootstrap.com/docs/4.0/components/alerts/>



Working for 3 min.

3. Javascript

JavaScript is a high-level, interpreted programming language primarily used for creating dynamic and interactive web content. It is supported by all modern web browsers and enables developers to enhance the functionality and behavior of web pages.

Task 1: Edit the content of an HTML element

Steps:

1. Create an HTML file.
2. Add a <button> element with an id attribute set to "printButton" and display the text "Print Message".
3. Create an empty <p> element with an id attribute set to "output".
4. Add <script> tag to the end of <body> tag to link script.js file.
5. Create a javascript file (script.js)
6. In script.js, use the addEventListener() method to add a click event listener to the "printButton".
7. In the event listener function, use the document.getElementById() method to get a reference to the "output" element.
8. Use the innerHTML property of the element to set its content to "Hello, world!".
9. Save the HTML file and open it in a web browser.
10. Click the "Print Message" button and verify that the message "Hello, world!" appears on the web page.

HTML Code

```
<!DOCTYPE html>
<html>
  <head>
    <title>Printing a Message</title>
  </head>
  <body>
    <button id="printButton">Print Message</button>
    <p id="output"></p>
    <script src="script.js"></script>
```

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

Javascript Code

```
var printButton = document.getElementById("printButton");
var outputElement = document.getElementById("output");

printButton.addEventListener("click", function () {
    outputElement.innerHTML = "Hello, world!";
});
```

Task 2: Edit the HTML of the page

Steps:

1. Create an HTML file with an empty `` element, an empty `<input>` element, and two `<button>` elements inside the `<body>` tag.
2. Set the text of the first button to "Add Item" and the text of the second button to "Remove Item".
3. Create a javascript file: script.js
4. Inside the javascript file, use the `addEventListener()` method to add click event listeners to both buttons.
5. In the event listener function for the "Add Item" button, use the `document.getElementById()` method to get a reference to the `` element and the `<input>` element.
6. Retrieve the value entered in the `<input>` element using the `value` property.
7. Create a new `` element and set its content to the entered value.
8. Append the `` element to the `` element using the `appendChild()` method.
9. In the event listener function for the "Remove Item" button, use the `document.querySelector()` method to select the last `` element in the `` element.
10. Use the `remove()` method to remove the selected `` element from the DOM.
11. Save the HTML and javascript file and open the HTML file in a web browser.
12. Enter a value in the input field and click the "Add Item" button to add a new list item to the `` element.
13. Click the "Remove Item" button to remove the item that matches with content of `itemInput` from the `` element.

HTML Code

```
<!DOCTYPE html>
<html>
```

```
<head>
  <title>Manipulating HTML DOM (Adding and Removing
Elements)</title>
</head>
<body>
  <input type="text" id="itemInput" placeholder="Enter item" />
  <button id="addItemButton">Add Item</button>
  <button id="removeItemButton">Remove Item</button>
  <ul id="itemList"></ul>
  <script src="script.js"></script>
</body>
</html>
```

Javascript Code

```
var itemList = document.getElementById("itemList");
var itemInput = document.getElementById("itemInput");
var addItemButton = document.getElementById("addItemButton");
var removeItemButton = document.getElementById("removeItemButton");

addItemButton.addEventListener("click", function () {
  var itemValue = itemInput.value;
  var listItem = document.createElement("li");
  listItem.innerHTML = itemValue;
  itemList.appendChild(listItem);
});

removeItemButton.addEventListener("click", function () {
  var itemValue = itemInput.value;
  var listItems = itemList.getElementsByTagName("li");
  for (var i = listItems.length - 1; i >= 0; i--) {
    if (listItems[i].innerHTML === itemValue) {
      listItems[i].remove();
    }
  }
});
```

Task 3: Create a blog post form

Steps:

1. Use the HTML file from bootstrap section.
2. Create modal dialog from <https://getbootstrap.com/docs/4.0/components/modal/>
3. In the modal, add <input> tag for post title, add <textarea> tag for the post content.
4. Add required javascript script links to the html as stated in the bootstrap document js section: <https://getbootstrap.com/docs/4.0/getting-started/introduction/#js>

HTML Code

```
<!-- Add between navigation link -->
<li>
  <button type="button" class="btn btn-primary" data-toggle="modal"
  data-target="#newPostModal">
    Write new post
  </button>
</li>

<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/Gp
GFF93hXpG5KkN" crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.mi
n.js"
integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFP
skvXusvfa0b4Q" crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.m
in.js"
integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpilMquVdAyj
Uar5+76PVCmYl" crossorigin="anonymous"></script>

<!-- Bootstrap Modal -->
<div class="modal fade" id="newPostModal" tabindex="-1" role="dialog"
aria-labelledby="newPostModalLabel" aria-hidden="true">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
```

```

        <h5 class="modal-title" id="newPostModalLabel">Write New
Post</h5>

        <button type="button" class="close" data-dismiss="modal"
aria-label="Close">

            <span aria-hidden="true">&times;</span>

        </button>

    </div>

    <div class="modal-body">

        <form>

            <div class="form-group">

                <label for="postTitle">Title</label>

                <input type="text" class="form-control" id="postTitle">

            </div>

            <div class="form-group">

                <label for="postContent">Content</label>

                <textarea class="form-control" id="postContent"
rows="5"></textarea>

            </div>

        </form>

    </div>

    <div class="modal-footer">

        <button type="button" class="btn btn-secondary"
data-dismiss="modal">Close</button>

        <button type="button" class="btn btn-primary">Save</button>

    </div>

</div>

</div>

```

Task 4: Read data from modal dialog

Steps:

1. We continue to build on the existing HTML page.
2. Add createPost function to onclick event of the Save button
3. Link javascript file script.js to the HTML page
4. In javascript file script.js, createPost function
5. In the createPost function, read title from postTitle element and content from postContent element.
6. Print data to the console.

7. Close the modal dialog.

HTML Code

```
<script src="script.js" />
<button type="button" class="btn btn-primary"
onclick="createPost()">Save Entry</button>
```

Javascript Code

```
function createEntry() {
    var title = document.getElementById("postTitle").value;
    var content = document.getElementById("postContent").value;
    document.getElementById("postTitle").value = "";
    document.getElementById("postContent").value = "";
    console.log(title + ": " + content);
    $("#newPostModal").modal("hide");
}
```

Task 4: Add new post to the HTML page

Steps:

1. We continue to build on the existing HTML page.
2. We add an id to the div which has the row className to identify the element we will add new articles.
3. Update createPost function
 - a. Get posts div element by id
 - b. Create div, article, h2 and p elements to create the HTML of an article.
 - c. Set the innerHTML of the h2 to the title variable.
 - d. Set the innerHTML of the p to the content variable.
 - e. Append article column div to the posts element
- 4.

HTML Code

```
<div class="row" id="posts">
```

Javascript Code

```
function createEntry() {
    const title = document.getElementById("entryTitle").value;
```

```

const content = document.getElementById("entryContent").value;
document.getElementById("entryTitle").value = "";
document.getElementById("entryContent").value = "";

const postsElement = document.getElementById("posts");

const colDivElement = document.createElement("div");
colDivElement.classList.add("col-md-6");

const articleElement = document.createElement("article");
const titleElement = document.createElement("h2");
titleElement.innerHTML = title;

const contentElement = document.createElement("p");
contentElement.innerHTML = content;

articleElement.appendChild(titleElement);
articleElement.appendChild(contentElement);

colDivElement.appendChild(articleElement);

posts.appendChild(colDivElement);

$("#newEntryModal").modal("hide");
}

```

4. Typescript

TypeScript is a powerful programming language that builds upon JavaScript by adding static typing and enhanced tooling. TypeScript provides a more robust development experience, catching errors at compile-time and enabling better code organization and scalability.

Task 1: Installation

Steps:

1. Install Node.js from the official website: <https://nodejs.org>
2. Create a new project directory ts-project
3. Run the following command in project directory to init Node.js project (and create package.json file): `npm init -y`

4. Run the following command in project directory to install typescript: `npm install typescript --save-dev`
5. Create a `tsconfig.json` file in the project directory to configure TypeScript. Run the following command to generate a basic configuration file: `npx tsc --init`

Task 2: Typescript Hello World

Steps:

1. Create a new TypeScript file, for example `index.ts`, in the project directory and write your TypeScript code.
2. Compile Typescript to Javascript with: `npx tsc`

Typescript Code (`index.ts`):

```
// index.ts
function greet(name: string) {
    console.log(`Hello, ${name}!`);
}

greet("John");
```

Compile Javascript Code (`index.js`):

```
"use strict";
// index.ts
function greet(name) {
    console.log("Hello, " + name + "!");
}

greet("John");
```

Task 3: Typescript type

Steps:

1. Create a new TypeScript file, for example `index.ts`, in the project directory and write your TypeScript code.
2. Compile Typescript to Javascript with: `npx tsc`
3. Verify compiler gives type error.

Typescript Code (`index.ts`):

```
// index.ts
function greet(name: string) {
    console.log(`Hello, ${name}!`);
}
```

```
}  
greet(42);
```

Compile Error message:

```
index.ts:6:7 - error TS2345: Argument of type '42' is not assignable  
to parameter of type 'string'.
```

Task 4: Typescript custom types using interface

Steps:

1. Create a new TypeScript file, for example index.ts, in the project directory.
2. Define an interface with members having different types including a function type.
3. Create an instance of the interface.
4. Call the function member on the instance.
5. Compile Typescript to Javascript with: `npx tsc`
6. Verify the console message.

Typescript Code (index.ts):

```
type VoidFunc = () => void;  
// Define an interface for a person  
interface Person {  
  name: string;  
  age: number;  
  greet: VoidFunc;  
}  
// Implement the interface for a person  
const person: Person = {  
  name: "John Doe",  
  age: 25,  
  greet: () => {  
    console.log(`Hello, my name is ${person.name} and I'm  
${person.age} years old.`);  
  },  
};  
// Invoke the greet method  
person.greet();
```

Task 5: Typescript composing types

Steps:

1. Create a new TypeScript file, for example index.ts, in the project directory.
2. Define a union type using two different variable types
3. Define a function with an argument of union type.
4. Call the function with two different types.
5. Compile Typescript to Javascript with: `npx tsc`
6. Verify the console message by calling `node index.js` (in the command line)

Typescript Code (index.ts):

```
// Define a union type
type ID = string | number;
// Create a function that accepts a union type parameter
function displayID(id: ID) {
  console.log(`ID: ${id}`);
}
// Use the function with different types
displayID("ABC123");
displayID(123456);
```

5. Creating a react application

In the next sections you will learn how to build a simple React application.

6. Create the react skeleton

The creation can take more than 5 minutes, so it is worth it to do this step before learning about React.

Prerequisites to install

Please install NPX, if you don't have it already.

<https://www.npmjs.com/package/npx>

Generate skeleton

Navigate to your home/work folder and generate the react app skeleton:

```
npx create-react-app blog-app --template typescript
```

7. React in nutshell

Please read this article to understand the basic concepts of React.

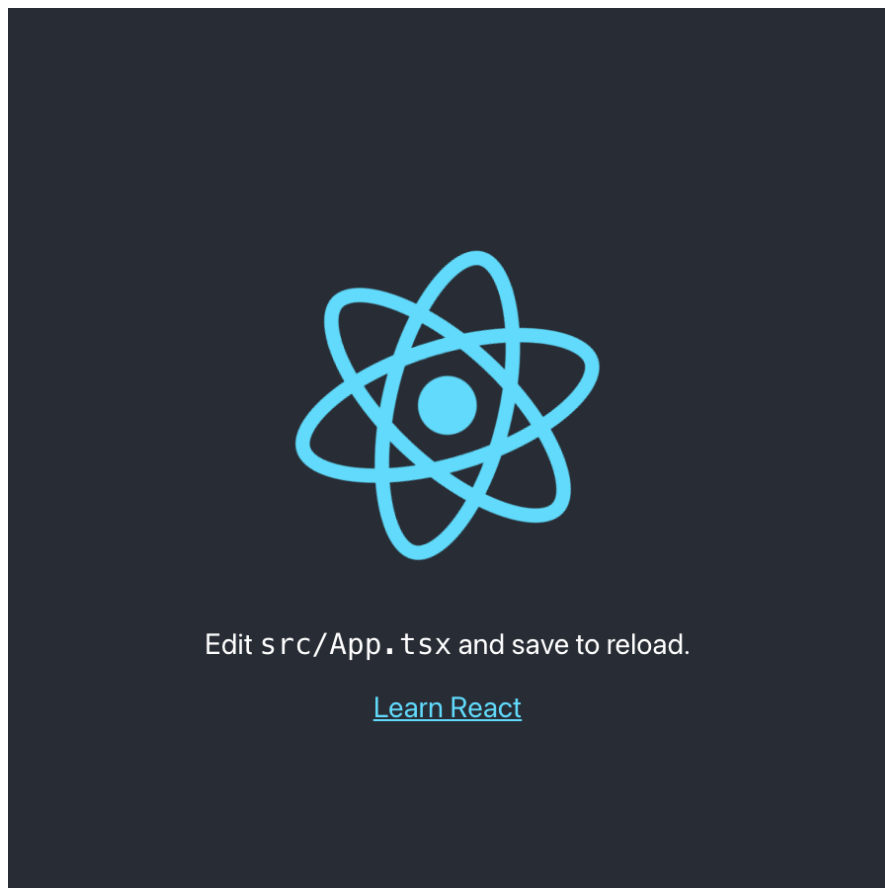
[React 101: A Quick Intro - DEV Community](#) - 7 mins to read

8. React skeleton

Run the skeleton and open your browser.

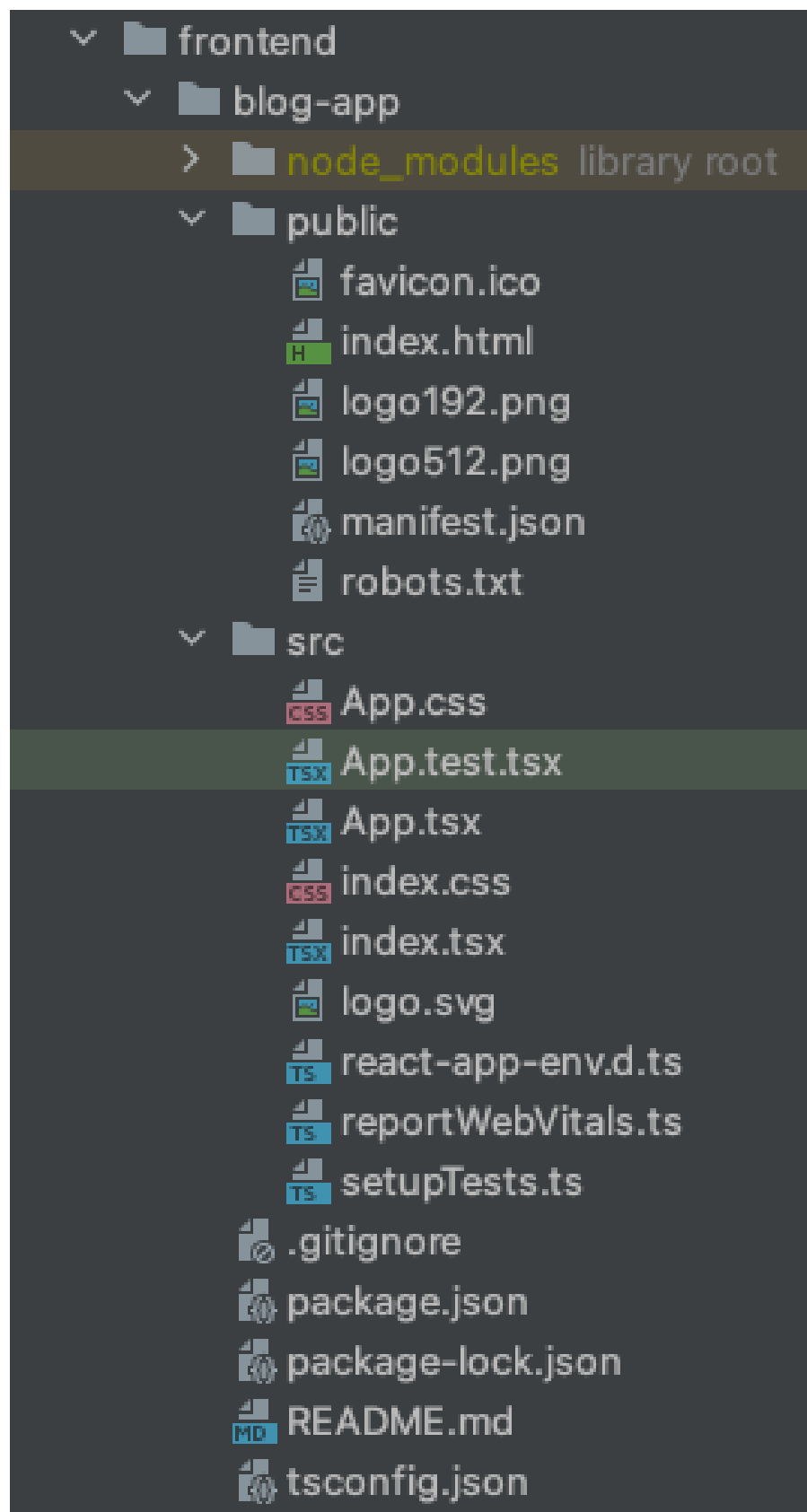
```
npm run start
```

A new tab should appear in your browser opening the <http://localhost:3000/> page with the following content:



If it doesn't open, please copy-paste the above address into your browser. From now on you can keep the command running as it will automatically recompile and render if an update happens in the source files.

The project view should look like this:



[Folder Structure | Create React App](#)

9. Let's Build a Blog



Now that we know the basics of front end technologies and React we will build our own personal blog.

Showing the blog posts

Create the DTO (Data Transfer Object) in the UI project as well.

Previously during the backend workshop you have created a DTO object for the blog post. Let's create a folder named `api/models/` and put the `PostDTO.ts` inside that.

```
export interface PostDTO {
  id: string;
  text: string;
  author: string;
  title: string;
  readmoreUrl?: string;
  pictureUrl?: string;
}
```

Let's create the `PostQueryResponse.ts` in the same folder as well.

```
import {PostDTO} from "../PostDTO";

export interface PostQueryResponse {
  posts: PostDTO[];
}
```

If we are ready with creating the “contract” interfaces between the frontend and backend, let's add our first React component. Create a file named `BlogPost.tsx` in the `src` folder. The extension of this file is not TS, but TSX. TypeScript needs the x to know that it should generate [JSX](#) instead of normal JavaScript.

```
import React from 'react';
import {PostDTO} from "../api/models/PostDTO";
```

```

type Props = {
  post: PostDTO;
}

function BlogPost(props: Props) {
  const {post} = props;

  return <article>
    <h1>{post.title}</h1>
    <p>{post.text}</p>
    <footer>Written by: {post.author}</footer>
  </article>
}

export default BlogPost;

```

This component gets a single post as an input and passes back its content to the parent as HTML. The {brackets} let's you write TypeScript logic within the HTML. In this example the fields of the post constant object are used.

Adding the BlogPosts.tsx file leaves us with our second component:

```

import React from 'react';
import {PostDTO} from '../api/models/PostDTO';
import BlogPost from './BlogPost';

type Props = {
  blogPosts: PostDTO[];
}

function BlogPosts(props: Props) {
  const {blogPosts} = props;

  const articles = blogPosts.map(post => <BlogPost post={post} />)

  return <div>{articles}</div>;
}

export default BlogPosts;

```

As you can see we already used the BlogPost within another component. In this case the BlogPosts is the parent and BlogPost is the child. With the for of loop we reuse the BlogPost multiple times.

Let's put all together by opening the App.tsx file and modify it and add your own posts into the blogPosts list.

```

import React from 'react';
import './App.css';
import BlogPosts from './BlogPosts';
import {PostDTO} from '../api/models/PostDTO';

function App() {
  // Please fill this with your own blog posts. 😊

```

```

const blogPosts: PostDTO[] = [];

return (
  <div className="App">
    <header className="App-header">
      { "[Your Name Here]'s personal blog." }
    </header>
    <body>
      <BlogPosts blogPosts={blogPosts.reverse()} />
    </body>
  </div>
);
}

export default App;

```

Example blogPosts content:

```

const blogPosts: PostDTO[] = [{
  id: "1",
  title: "Trip to Beirut",
  text: "Last summer I went to Beirut, a beautiful city in the middle of the Mediterranean Sea.",
  author: "your name",
}];

```

The App.tsx is the entry point to our React app. Please open the index.tsx and have a look at its content:

```

const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement
);
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

```

As you can see it embeds our root component the App and renders it. Back to the App.tsx, we can see that it has a single child, the BlogPosts.

If you run it (npm start) and check the result you will see only the header is shown, so it is time to do styling.

The App.tsx already using the App.css for styling. Let's extend it with style for the HTML body.

Add a new class called .App-body and change the min-height setup of the .App-header.

```

.App-header {
  background-color: #282c34;
  min-height: 20vh; /* https://www.w3schools.com/cssref/css_units.php */
  display: flex;
  flex-direction: column;
  align-items: center;

```



```

justify-content: center;
font-size: calc(10px + 2vmin);
color: white;
}

.App-body {
background-color: white;
margin: 2vh;
min-height: 80vh; /* https://www.w3schools.com/cssref/css_units.php */
display: flex;
flex-direction: column;
text-align: left;
align-items: center;
justify-content: start;
font-size: calc(10px + 1vmin);
color: black;
}

```

vh: Relative to 1% of the height of the viewport. Viewport = the browser window size. If the viewport is 50cm wide, 1vw = 0.5cm.



Have a look on the header in the App.tsx and add the .App-body style to the body. If you are done it should look like this:



- 3 mins

[Your Name Here]'s personal blog.

Trip to Beirut

Last summer I went to Beirut, a beautiful city in the middle of the Mediterranean Sea.

Written by: your name

Feel free to play with the values in the css and check how the look differs with each change.



- 3 mins

Adding bootstrap



It would take a while if we want to make it stylish, so let's cheat a bit and use bootstrap.

Bootstrap

<https://react-bootstrap.github.io/getting-started/introduction/>

Amazon also has its own UI toolkit called [Cloudscape Design System](#). However for simplicity we are going with bootstrap this time.

Make sure to navigate into the root folder of your project, where your package.json is.

```
npm install react-bootstrap bootstrap
```

We will use the [Cards component from React bootstrap](#) to render a blog post. Let's modify the BlogPost component.

```
import React from 'react';
import {PostDTO} from "../api/models/PostDTO";
import {Card} from "react-bootstrap";

type Props = {
  post: PostDTO;
}

function BlogPost(props: Props) {
  const {post} = props;

  return <Card
    className="mb-2"
```

```

    border="dark" /* Try to change this:
https://react-bootstrap.github.io/components/cards/#card-styles */
    bg="light" /* Try to change this:
https://react-bootstrap.github.io/components/cards/#card-styles*/
    style={{whiteSpace: "pre-wrap"}} /* Making sure that \n and \t treated
as new line and tab in our text. */
  >
    <Card.Body>
      <Card.Title>{post.title}</Card.Title>
      <Card.Text>{post.text}</Card.Text>
    </Card.Body>
    <Card.Footer>
      <small className="text-muted"> Author: {post.author}</small>
    </Card.Footer>
  </Card>
}

export default BlogPost;

```

However, as you might have seen, it didn't change the look much. Include this import in the index.tsx to see the magic.

```
import 'bootstrap/dist/css/bootstrap.min.css';
```



With the help of the [card documentation](#) add a link in the Card.Body which redirects to post.readmoreUrl.



- 3 mins

One trick to show the link only if the post.readmoreUrl is present:

```

<Card.Body>
  <Card.Title>{post.title}</Card.Title>
  <Card.Text>{post.text}</Card.Text>
  {
    // This line make sure that the link is only displayed if it exists.
    post.readmoreUrl && <Card.Link href={post.readmoreUrl}>Read
more...</Card.Link>
  }
</Card.Body>

```

Make it possible to post



It would be inconvenient to add the post into the code so let's create a [Form](#) where we can submit them.

Create a file called BlogPostForm.tsx and fill it with this content:

```
import React from 'react';
import {Button, Form} from "react-bootstrap";

type Props = {
};

function BlogPostForm(props: Props) {

  // This is the event handler function, which gets executed once the Save
  button is pressed.
  const handleSubmit = (event: React.FormEvent<HTMLFormElement>) => {
    const form = event.currentTarget;

    // To cancel the native behaviour of the submit button,
    // you need to use React's event.preventDefault() function
    // https://sebastian.com/react-preventdefault/
    // We need to keep it here until the API call is added,
    // after that we can move into the if block.
    event.preventDefault();

    // Checking if the form has any invalid input. If not it stops the
    form to be saved.
```

```

        if (!form.checkValidity()) {
            // The stopPropagation() method of the Event interface prevents
            further propagation
            // of the current event in the capturing and bubbling phases.
            //
https://developer.mozilla.org/en-US/docs/Web/API/Event/stopPropagation
            event.stopPropagation();
        }
    };

    return (
        <Form className="mb-3" style={{width: "60vw"}}
        onSubmit={handleSubmit}>
            <Form.Group className="mb-3" controlId="formTitle">
                <Form.Label>Title</Form.Label>
                <Form.Control required type="text" placeholder="Enter the
title" size="lg"/>
            </Form.Group>
            <Form.Group className="mb-3" controlId="formText">
                <Form.Label>Text</Form.Label>
                <Form.Control required as="textarea" rows={3}/>
            </Form.Group>
            <Form.Group className={"mb-3"} controlId="formAuthor">
                <Form.Label>Author</Form.Label>
                <Form.Control type="text" placeholder="Enter the author
name"/>
            </Form.Group>
            <Button variant="primary" type="submit">
                Save
            </Button>
        </Form>
    );
}

export default BlogPostForm;

```

Have a deep look at this code.



If you are done, let's make it work by adding the BlogPostForm component to the App.tsx. As a reference use the BlogPosts component which is already used within the App.tsx.



- 2 mins

🎉 Tada 🎉

[Your Name Here]'s personal blog.

Title

Text

Link name

Save

Trip to Beirut

Last summer I went to Beirut, a beautiful city in the middle of the Mediterranean Sea.

Author: your name

Handle the event properly

To be able to store the data in the fields on the form, we need to learn about State - a component's memory in React. The problem with local variables is they don't persist between render and won't trigger a render when their value changes. This is addressed by the State.

Using it:

```
const [author, setAuthor] = useState("nobody");
```

author is a state variable and setAuthor is the setter function. [More on state](#).

Saving the form fields in the components



Extend the form fields so they get the value from the state variable and execute the `set<Variable>` when a change happens in the given form field (control). Here is an example to work with.



- 5 mins

```
const [title, setTitle] = useState("");
```

```
...
```

```

<Form.Label>Title</Form.Label>
<Form.Control
  required type="text"
  placeholder="Enter the title"
  size="lg"
  value={title}
  onChange={e => setTitle(e.target.value)}
/>

```

Adding the link



Add the link form control. Use any of the Form.Group as an example. What type should it be?

Do you need the required flag?



- 2 mins

```

<Form.Group ...
  // TODO
</Form.Group>

```

Time to handle the event

First we will pass an event handler from the App to the child BlogPostForm component.

BlogPostForm.tsx

```

type Props = {
  onSubmit: (blogPost: PostDTO) => void;
};

function BlogPostForm(props: Props) {

  const {onSubmit} = props;

  ...

}

```

We are passing it as part of the props.

App.tsx:

```

const onSubmit = (post: PostDTO) => {
  // TODO
}


return (
  <div className="App">
    <header className="App-header">

```

```

    { "[Your Name Here]'s personal blog." }
  </header>
  <body className="App-body">
    <BlogPostForm onPostSubmit={onFormSubmit}/>
    <BlogPosts blogPosts={blogPosts}/>
  </body>
</div>
);


```

 Call the onPostSubmit function in the right place and pass the blogPost as an input. - you can use the crypto.randomUUID() to generate the id for the blogPost.



- 3 mins

 Tada 

 Refresh the page, what happens?

To be able to persist between refreshes in React we have two options: integrate with the backend or use [Redux persist](#). Let's go with option one as we have the backend ready and it will persist our blog even if we delete the storage of our browser.

10. Integrate with the backend

Start the backend server

Run npm run start in the folder of the backend app.



If you are lucky it will let you know that the React app and the backend service are trying to use the same port, which is causing a port collision. (In some cases the backend service starts and doesn't inform you about the port collision.)

To fix that add the .env file next to the package.json in the frontend app and restart it. The content is:

```
PORT=3008
```

This is the recommended way to solve it on your dev environment, however it is a different story when you deploy it to production.

Posting the post 😊

In the App.ts change the onFormSubmit logic to this:

```
const [blogPosts, setBlogPosts] = React.useState<PostDTO[]>([]);

useEffect(() => {
  fetchBlogPosts();
});

const onFormSubmit = (post: PostDTO) => {
  fetch("http://localhost:3000/post", {
    method: "POST",
    headers: {
      "Content-Type": "application/json"
    },
    body: JSON.stringify(post)
  })
  .then(() => fetchBlogPosts())
  .catch(error => console.log(error));
}
```

The useEffect is a [React Hook that lets you synchronize a component with an external system](#).

The fetch command is executing an HTTP POST call. Press F12 in your browser which opens the developer console. Go to the network tab. Post something and look for the request.

11. 🎉 Celebrate 🎉



Congrats you have successfully completed the workshop.

12. Extra features as homework



Store a pictureUrl for each post on the backend and render it in the top of the Card item.

Each person who shares a screenshot, how the added picture looks like on [#homework](#) slack channel will get an ACS sticker.



Style the blog, so it looks better.

Please share screenshots and upload the code to github/gitlab and link it in your message on [#homework](#) slack channel.

The students creating the best 3 designs win a water bottle.

Deadline: Sunday end of the day. (CEST)

