Front-end workbook

This workbook will guide you through setting up your environment, building a simple webpage, and using AI tools to create a more advanced application. Let’s dive in and start building together!

## Prerequisites

GitHub account, google account (for accessing AI accounts)

# Setting up the environment

## Step 1: Install Visual Studio Code (VS Code)

* Download Visual Studio Code:
  + [Windows](https://code.visualstudio.com/Download" \t "_new)
  + [Mac](https://code.visualstudio.com/Download)
* Install VS Code on your computer following the instructions for your operating system. (Optional if you got other suite or already installed VS Code)
* Install useful extensions (optional):
  + **ES7+ React/Redux/React-Native snippets**: For fast React development.
  + **Prettier**: Prettier automatically formats code for better readability.
  + **Code Runner**: For running small JavaScript snippets.
  + **Live Server**: For viewing HTML files during development.
  + **Code Spell Checker:** To avoid spelling mistakes (believe me, that causes errors in code)
  + **Tailwind CSS IntelliSense:** Autocompletes Tailwind CSS, which is very useful
  + **vscode-pets:** cute companions (no functional use other than **cute**)

## Step 2: Install Node.js

* Download Node.js:
  + [Node.js Official Website](https://nodejs.org/)
* Install the **LTS version** for compatibility. This is needed for the React project later.
* Node.js provides the runtime for JavaScript and includes **npm** (Node Package Manager), which is used to install project dependencies.

## Step 3: Verify Your Installation

1. Open a terminal:
   1. Windows: Open Command Prompt.
   2. Mac: Open Terminal.
2. Run the following commands:
   1. node -v → This should return the Node.js version.
   2. npm -v → This should return the npm version.

# Build Your First Webpage

## Step 1: Create the Project Folder

1. Create a folder on your desktop named Hackathon\_Webpage (or something more creative if you wish)
2. Open VS Code and choose **File > Open Folder** to open this folder (or drag and drop on mack into the VS Code)

## Step 2: Create Your HTML File

1. Inside Hackathon\_Webpage, create a file called index.html.
2. Add the following code:

1. <!DOCTYPE html>

2. <!-- This declaration defines the document type and version of HTML (HTML5 in this case). -->

3.

4. <html lang="en">

5. <!-- The root element of the HTML document. The 'lang' attribute specifies the language (English). -->

6.

7. <head>

8. <!-- The <head> element contains metadata about the document, such as its title and character set. -->

9.

10. <meta charset="UTF-8">

11. <!-- Specifies the character encoding for the document. 'UTF-8' is a universal character set. -->

12.

13. <meta name="viewport" content="width=device-width, initial-scale=1.0">

14. <!-- Ensures the webpage is responsive by adjusting the width to match the device's screen. -->

15.

16. <title>Hackathon Webpage</title>

17. <!-- Sets the title of the webpage, which appears in the browser tab. -->

18. </head>

19.

20. <body>

21. <!-- The <body> element contains the visible content of the webpage. -->

22.

23. <header>

24. <!-- The <header> element represents introductory content, such as a page title or logo. -->

25. <h1>Welcome to the Hackathon!</h1>

26. <!-- The <h1> element is the main heading of the page. -->

27. </header>

28.

29. <main>

30. <!-- The <main> element contains the central content of the page, unique to this document. -->

31.

32. <p>Fill out the form below and click the button:</p>

33. <!-- The <p> element is used for a paragraph of text. -->

34.

35. <form>

36. <!-- The <form> element is used to collect user input. -->

37.

38. <input type="text" placeholder="Enter your name" id="nameField">

39. <!-- The <input> element creates a text field where users can type.

40. 'placeholder' provides a hint to the user, and 'id' gives it a unique identifier. -->

41.

42. <button type="button" id="submitButton">Submit</button>

43. <!-- The <button> element creates a clickable button.

44. 'type="button"' specifies that it doesn't submit the form automatically.

45. 'id' provides a unique identifier for scripting. -->

46. </form>

47. </main>

48. </body>

49. </html>

50.

## Step 3: Add Simple CSS

1. Create a new file called style.css in the same folder.
2. Add the following code:

body {

font-family: Arial, sans-serif;

text-align: center;

margin: 20px;

}

header h1 {

color: #4CAF50;

}

form {

margin-top: 20px;

}

input {

padding: 10px;

margin-right: 10px;

border: 1px solid #ccc;

border-radius: 5px;

}

button {

padding: 10px 15px;

background-color: #4CAF50;

color: white;

border: none;

border-radius: 5px;

cursor: pointer;

}

## Step 4: Add Simple JavaScript

1. Create a new file called script.js in the same folder.
2. Add the following code:

document.getElementById("submitButton").addEventListener("click", function() {

const name = document.getElementById("nameField").value;

alert(`Hello, ${name}! Welcome to the Hackathon!`);

});

1. Link this JavaScript file in index.html before the closing </body> tag:

<script src="script.js"></script>

## Run Your Webpage

1. Open index.html in your browser (drag and drop the file into your browser window).
2. Test the form and button to see how it works!

# Make AI work for you

Now that we have investigated the building blocks of the font-end and have insight on how it all connects and works, lets leverage AI to take this on to the next level and have some fun.

## Bolt.new

For this we will be using [bolt.new](https://bolt.new/). It is an AI-powered web development agent that allows you to prompt, run, edit, and deploy full-stack applications directly from your browser—no local setup required.

1. Head to [bolt.new](https://bolt.new/)
2. Create a free account (it will give you approx. 100k tokens[[1]](#endnote-1) a day; the bigger the project the more tokens it will use) – needs GitHub account
3. Using Bolt, we will build the foundation for our app, and see how much acceleration we can get by leveraging AI

## Time for a prompt

Now that we created account with bold.new let’s get it to work. For this session we will be building a simple Joke of the day app. If you wish to work on your own ideas, you can. Just adjust the prompt to have the details of your idea. You will need the Joke of the day for the Backend session, but you can get that from the zip file that will be provided.

1. Type the prompt into the text area:
   1. Build me a joke of the day app. It needs to have a **"Get Joke of the Day"** button that fetches a random joke and displays it clearly in an h1 tag on the page. A **text input field** for users to submit their own jokes, accompanied by a **"Submit" button**. Make sure the app is responsive and follows modern web requirements [[2]](#endnote-2)with clean and user-friendly UI.
   2. Optional: bolt provides with enhance prompt functionality (the sparkle icon next to the upload files button). Result for the above prompt:
      1. Build a joke app with:
         1. "Get Joke" button that fetches and displays a random joke
         2. Text input for user submissions with a submit button
         3. Clean, responsive UI using Tailwind CSS
         4. Error handling and loading states
         5. Local storage to persist user-submitted jokes
         6. Ability to toggle between random jokes and user submissions
         7. Basic animations for UI interactions
2. If content with the prompt, enter the prompt and let the AI do its magic.

A screenshot of a computer

Description automatically generated

1. Now we have a working app. What is good about this, that it comes with all the required dependencies and file structure that could be a potential pain point for those who are just starting out. A screenshot of a computer

   Description automatically generated
2. Structure maters. The beauty of react is creating reusable components. While creating a small application you might not feel much of a need for that, when working on a large apps and systems, you will see how great it is to keep reusing the logic and code you got.
   1. /src – it stands for source, this is the folder where app components, hooks, utility files and all other files go. Something where you will spend most of the time while creating your app.
      1. /components – this is a folder for react components. In small apps it can just contain all your components in this directory, but when projects become bigger and bigger, we add folders within this folder, for specific bigger components. Components are like the tables in the restaurant, the chairs, lights.
      2. /hooks - Hooks in programming, specifically in **React**, are like **tools** or **shortcuts** that make it easier to manage and update the "stuff" your app needs to remember and respond to. Going back to the restaurant example, it is a way to remember what each table have ordered and quickly update their bill. So, the hooks are like notebooks and pens for the staff.
      3. There can be more folders under /src such as, but not limited to:
         1. /types, for keeping the project neat and maintainable
         2. /utils to store **helper functions** that don’t fit neatly into any single component but are used across multiple parts of the app
         3. /services to store **code that interacts with backends** (API calls), third-party services (like payment gateways), or handles complex business logic that doesn’t belong in UI components.
      4. App.tsx – this is where the main app structure lives. It is the root component of React application. Here you often set up the routing, global state and render child components
      5. Index.css - file is where you typically define your global **styles** for the entire application. These styles will apply to all components unless overridden. You can set the default global styles for your app here.
      6. main.tsx - The **entry point** for the app. It is where the root React component (<App />) is rendered into the DOM. This file tells React to attach to a specific element in the HTML file (usually index.html).
      7. types.ts - TypeScript types or interfaces that describe the structure of your data (used throughout the app for safety). In bigger projects the /types folder mentioned earlier appears in the src/
   2. .gitignore - This file tells **Git** (a version control system) which files or folders it should **ignore** when tracking changes to your project. You don’t want Git to track certain files like temporary files, build outputs, or sensitive information (like passwords).
   3. eslint.config.js - This file is used by **ESLint**, a tool that checks your code for mistakes and helps ensure your code follows a consistent style. The file contains rules that tell ESLint what to look for and how to fix issues. It could define rules like “always use semicolons” or “ensure variable names are camelCase.”
   4. index.html – Oh we seen this one! This is the **main HTML file** that is used to load the app in the browser.
   5. package-lock.json - This file locks down the specific **versions** of the libraries (dependencies) your app uses. It helps ensure that when you or someone else installs the project, you get the same version of every library. Very useful
   6. package.json - This is the **manifest file** for your project. It lists the libraries your app needs, scripts to run the app (like starting it or building it), and general information about the project. Using this we can also add specific commands to run on the terminal.
   7. postcss.config.js - PostCSS is a tool used to process your CSS (stylesheets). This file contains configuration settings for how to **transform** your CSS code, like adding features that aren't natively supported by all browsers.
   8. tailwind.config.js - This is the configuration file for **Tailwind CSS.** It defines custom settings for how Tailwind should behave in your project, like the colors, fonts, and breakpoints you want to use. Like index.css but for tailwind.
   9. tsconfig.app.json - It helps TypeScript understand how to handle your TypeScript files, like which directories to include or exclude during compilation.
   10. tsconfig.json - It defines rules and settings for the TypeScript compiler, ensuring that your app is checked for type errors
   11. tsconfig.node.json - This configuration file is similar to **tsconfig.json**, but it’s specifically used for the **Node.js** backend. It might define how TypeScript compiles code that runs on the server side[[3]](#endnote-3).
   12. vite.config.ts - This is the **configuration file** for **Vite**, the build tool used to bundle and run your app. Vite handles the development server, optimizing the app for production, and compiling modern JavaScript and TypeScript.
3. We will focus here what’s inside the /src folder. The files outside ensure that everything runs smoothly, is properly styled, and can be deployed. They can be adjusted, but only in specific scenarios.
4. Are we happy with the result Bolt gave us? If no, what we are not happy with? Can we fix it, or should AI help us. If so, enter another prompt, properly defining the problem and the solution you wish for. Do it until satisfied or run out of AI tokens

## Open the editor

Now that we have some code, and a bit of an idea of the code structure lets open the online code editor that this AI so gracefully provides to us.

1. On the bold.new page, top right corner, there are 3 buttons. Click the Open in StackBlitz. This action will take you to your online Editor!
2. Here from left to right you will see all your project files, the code editor with a terminal, and on the right, you will see your app! Above your files you might notice a button that says, ‘Create a repository’ and that, if you allow StackBlitz to connect with your GitHub, will give you an option to instantly put your project into your GitHub, for the version control and safekeeping of your project. Isn’t that nice. ( you can create a repository from this, but we will not go into the whole GitHub topic as it is a whole ecosystem that we would need an extra 30min, but here is a cheat sheet for it <https://education.github.com/git-cheat-sheet-education.pdf> )
3. Let us look at the code now.
4. Let us ask AI to explain the code for us. For this we will be using [Claude.ai](https://claude.ai/). Claude is a great companion for debugging and code explanation. Why are we using Clause instead of bolt? Well, we want to save up the precious AI tokens on one tool, so we can make as much progress as we can, as cheap as we can. (we can also use ChatGPT for this as well, basically, use all those free options online)

## Let’s add more features

Now that we investigated our code, let us add a few more features, give a few more asks to Bolt, and see how our code changes, depending on the prompts we give.

1. Go back to bolt.new where you have your app.
2. Let’s add a few more features for our joke bot (add your own if you are working on your own app)
   1. History and date – observe the changes, what are they? Any problems you notice?
   2. Delete joke using the id of the joke – observe the changes, what are they? Any problems you notice?
   3. Add the author – observe the changes, what are they? Any problems you notice?
3. Address any issues we have noticed

## Let’s run it locally

While bolt.new provides a whole suite to work online, with everything you would need to develop your application, it is in our best interest to see how to run it in local environment on our devices. This not only provides us with more insight, but also makes us sure, our code is saved locally (there are known bugs for bolt.new to change your code without a way to go back to previous version, and some you have already noticed). In short: running locally ensures we can work offline and have full control of our project files.

1. Press download button, a .zip file will be downloaded to your device
2. Extract the files
3. Drag and drop on your Visual Studio or Open Visual Studio and follow the steps:
   1. Hover over ‘File’
   2. Click on ‘Open Folder’
   3. Navigate and select your project
4. Wait for the project to open
5. Open terminal:
   1. Hover over ‘Terminal’ at the top of VS
   2. Click on ‘New Terminal…’
6. Do `npm install` to install the required packages for your project. This will create a node\_modules folder that contains all the dependencies your app needs to run.
7. To run your app, on the terminal run ` npm run dev`
8. Navigate to <http://localhost:5173/>
9. Now you have your code working locally for you as well. Good job

1. Tokens represent a measure of computational usage when interacting with AI tools like **Bolt.new**. Each action, such as running a prompt or deploying an app, consumes tokens based on the complexity and resources required. The larger the task or application, the more tokens are used. [↑](#endnote-ref-1)
2. **Why Modern Web Requirements Are Essential:**

   **Responsiveness** and **accessibility** are crucial for creating inclusive, user-friendly websites:

   **Responsiveness** ensures that websites function well on devices of all sizes, from smartphones to desktops. This adaptability enhances user experience and is increasingly important as mobile devices dominate web usage.

   **Accessibility** ensures that websites are usable by people with disabilities, such as those relying on screen readers or keyboard navigation. Compliance with accessibility standards (e.g., WCAG) not only broadens the audience but also reflects ethical design practices and may be legally required in some regions.

   Neglecting these requirements can alienate users, reduce engagement, and harm a brand's reputation. [↑](#endnote-ref-2)
3. **Server-Side** is what happens on a remote server, managing data, processing requests, and ensuring the app works correctly. When you go to check out, your order details are sent to the **server**. The server then checks if the items are available, calculates the total price, and processes the payment. It then sends back a confirmation of your purchase. [↑](#endnote-ref-3)