1. npm install
2. npm run build
3. npx next export

This website is built with NextJS with is a framework for ReactJS. This means the markup, unlike from pure HTML in **.html** files – is component-based and the extension is **.js**.

Read more on the differences [here](https://blog.maisie.ink/jsx-html-differences/).

**How to update the website:**

**First method:**

This method is preferred if you wish to add new/more components to pages. It is done by editing the development files which are located in the ‘pages’ and/or ‘components’ folders. For this you need NodeJS installed and access to its command prompt commands. Make the desired changes to the website.

1. Open the command prompt / terminal, navigate to the root directory (where package.json is located) and run “**npm run dev**” to start the site in development mode on your local machine.
   1. The site will run on: <http://localhost:3002>
   2. To change the port (3002), edit *package.json* inside the root directory.
2. If you would like to keep track of the website’s version:
   1. Open layout.js (located inside the ‘*pages’* folder).
   2. Look for **const appVersion = "x.y.z;**
   3. Increment the value (e.g. 1.1.3, 1.1.4, ….1.1.9, 1.2.0, …).
3. In the command prompt, run the following commands in order:
   1. **npm run build**
   2. **npx next export**
4. Once the second command finishes, look for the new *‘out’* folder inside the root directory
5. Copy the contents of this folder to your FTP.
6. Ready.

Example:

The components inside of the downloadpage (DownloadPage.js) depend on *downloadData.json* which is located inside the *‘public’* folder. The content of the components inside of the download page (DownloadPage.js) depends on the content of downloadData.json (located inside the ‘public’ folder). If you decided to update the website through the development files then you shall update downloadData.json to make changes to the download page. downloadData.json is an object that has 2 objects and one array.

The two objects are 'windows' and 'mac', whereas the array is 'changelog'.

The contents of the objects 'windows' and 'mac' are similar. Both have 4 properties all of whose values are strings:

|  |  |
| --- | --- |
| Version | Version of the app (required) |
| windowsOS/macOS | The intended OS (optional) |
| fileName | Name of the file (required – will be used for link) |
| fileSize | File size (optional) |

The array 'changelog' is an array of strings. Each element in the array represents a separate bullet-point / paragraph in the changelog list on the website.

This concludes updating downloadData.json.

Now on how to update tutorial displayer.

The data for it comes from tutorialsData.js (located in the *public* folder).

The structure of tutorialsData.js is as follows:

* An array of objects, each object represents a category in the tutorial selector.
* The object has 2 properties, categoryName and tutorials.
* Tutorials is an array of objects as well.
  + Each object has 2 properties:
    - Name – the name of the tutorial (within that category)
    - The other property can be either images or verbalInstruction
      * verbalInstruction – if you do not want images but just a text-based tutorial, use this property.
      * Images – if you want images instead of a text-based tutorial. This one is an array of objects. Each object represents a step in the tutorial. Each object has the following properties.
        + src: this is the path to the image for the step in the tutorial (required).

Example:

* `${tutorialsImagesPath}/boards/create-a-board/boards-closed-navbar.jpg`

tutorialsImagesPath comes from the top of the file:

const tutorialsImagesPath = "./images/tutorials";

*The* ***images/tutorials*** folder is located inside the **public** folder.

* + - * + Coordinates: an object with properties *x* and *y*, represents the position of the pointer in relation to the dimensions of the image (required).
        + pointerSize: string value that can be either “***xs”****, “****s”****,* or *“****m”***. Shrinks the size of the pointer to focus on smaller elements in the photo (optional)
        + instruction: string. The text instruction next to the pointer (required).
        + rightSide: Boolean. If true, the instruction text will be on the right side of the pointer (optional – use this if you believe the pointer instruction is too close to the left perimeter of the tutorial viewer, I have implemented mobile responsiveness, the tutorial viewer will adjust the side automatically and even reverse this property if necessary).
        + brighterPointer: Boolean. If true, the shadow around the pointer circle will be brighter (optional – use this if the pointer is over a white background, it reduces contrast and makes it easier on the eyes).

The structure of the images/tutorials folder is as follows:

* The direct child folders of the *tutorials* folder represent the categories.
  + The children folders of those category folders represent a separate tutorial.

If you wish to add tutorials from development files, do it through this *tutorialsData.js*.

This concludes adding tutorials.

If you wish to add a YouTube video to the videos section of the demo page: First acquire the embed link, which should look like this:

**https://www.youtube-nocookie.com/embed/44ANYZP5484.**

In VideosContainer.js (located inside ***components/VideosContainer***) there’s an array **videos**.

Create a new object like so:

{

url: “link to the video here”

title: “Type the title of the video here”

}

Insert the embed link from youtube into the ***url*** property.

If you wish to remove or add components to the website, it can be done with usual html.

Components are rendered inside the *return()* function of each component/page. Therefore, to add add/edit/remove components you add markup inside the return() function. All components must be wrapped inside a single element (i.e. div).

|  |  |
| --- | --- |
| return(  </div>  <h1>Hi</h1>  <h1>Hello</h1>  <div>  ) | This is good |
| return(  </div>  <h1>Hi</h1>  <div>  <h1>Hello</h1>  ) | This is wrong |
| return(  <h1>Hi</h1>  <h1>Hello</h1>  ) | Also wrong |

If you want to add a class to an element, add a “className” property within the opening tag, like so <div className=”myClass”></div>.

Add your styles to globals.css (located inside the ***styles*** folder).

**Second method**:

If you do not wish to add more components, but instead only to edit something then this can be done by editing the production files (located in the ‘*out’* folder).

For example, to update the contents of the download page (e.g. version, filename, link, etc.) go to the download.html file. This is the case only with download.html because the data for it came from an external json file.

To edit other pages post-production, you will have to navigate to out\\_next\static\chunks\pages

In there are two .js files for each page of the website. Open the file that contains the page name with a hash code. The content is going to be minimized, so you will have to use a formatter. Then, look for the content you wish to edit. To check if the updates are working, open the corresponding .html file in the ‘*out’* folder (or refresh if it is already open).

For example, suppose you want to edit the text of the header “Ardi - The Organizer. A Gmail Client and Notes Manager.” on the home page. Open *index-randomhashnumber.js* and search for “ardi – the organizer” or any keywords from that header.

You may find multiple instances. Look for the one that looks the most similar to the development file. In the development file the header comes from ParagraphWithText.js component which includes the *headerText* property. Therefore, you want to edit the line in the production file that looks like this:  
*headerText: "Ardi - The Organizer. A Gmail Client and Notes Manager."*

Object(n.jsx)(a.a, {

           forceNoBackground: !0,

           noUnderline: !0,

          extraLarge: !0,

           color: "white",

           headerText: "Ardi - The Organizer. A Gmail Client and Notes Manager"

           paragraphText: "Manage notes, emails, contacts, tasks and images.",

           visibilitySensorReveal: !0,

            }),

Note: When opening .html files on local machine (At least on Windows OS) the page might load without CSS. This can be fixed with the following steps:

1. Open the .html file in your IDE.
2. Format the minimized code.
3. Select all the instances that contain “/\_next" and then add a dot before, like so: "./\_next”
4. Reload the page.

The CSS should load on FTP without applying the fix above.

If you wish to add tutorials from production files, access the *demo-hashcode.js* file and look for the array contents of *tutorialsData.js* inside that file.