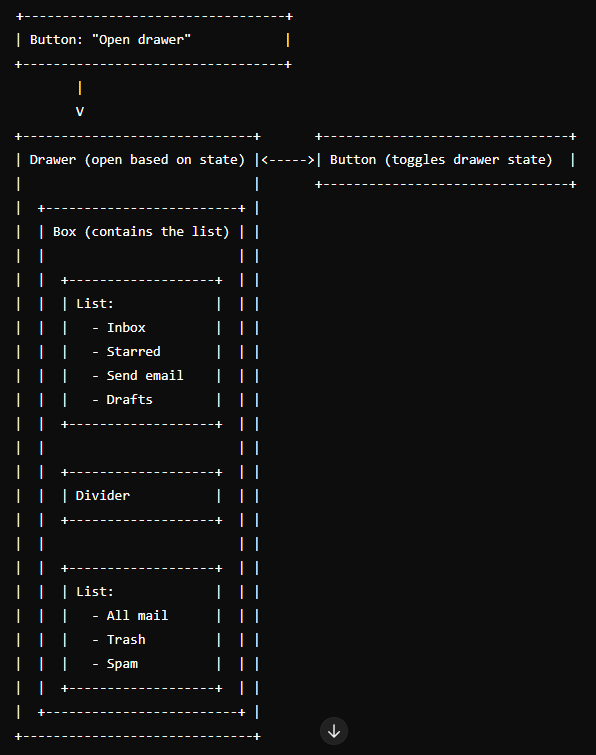
**General Points**

1. When we click on a website. The web server asks the **Application Server** to send the asked data. The Sent Data might be Static and dynamic type.
2. **Static** is when everything is sent in one go.
3. **Dynamic** is when everything is sent when we ask for it.
4. **Caching** 🡺 Saving a copy of the requested page, so that when it’s called again. It will be fast to response.
5. So when a dynamic content is requested it first check in **cache memory**, then goes to application server if not there.
6. **Single Page Websites** means only one HTML page that will be received from the server to browser, then as the request is made content is received and updated in that HTML page. SPA have two ways to deliver the requested content Bundling and Lazy Loading.
7. **Bundling** is sending all data in go as bundle for browser to show.
8. **Lazy loading** is sending one by one data as requested.
9. **React Fiber Architecture 🡺** Think of it as a priority system. The highest priority changes, the elements visible to the user, are updated first. While lower priority changes, the elements not currently displayed, are updated later.
10. **window.scrollTo()** 🡺 is a method available on the window object in the browser. It is used to scroll the window to a specific set of coordinates.
    1. window.scrollTo(x-coord, y-coord) 🡺 window.scrollTo(0, 0);
       1. **x-coord:** The pixel along the horizontal axis of the document that you want displayed in the upper left. 0 for the x-coord means no horizontal scrolling; the left edge of the content is aligned with the left edge of the viewport.
       2. **y-coord:** The pixel along the vertical axis of the document that you want displayed in the upper left. 0 for the y-coord means no vertical scrolling; the top edge of the content is aligned with the top edge of the view.
11. Webpack is responsible for resolving file extensions when you import a file in JavaScript.
    1. Example: import { file1 } from "module1";
    2. Without Webpack:
       1. If you're not using Webpack (or another bundler), the browser requires the exact file path with the correct extension (e.g., "module1.js" or "module1.ts") to load the module.This is because browsers do not perform extension resolution automatically.
    3. With Webpack:
       1. Webpack resolves file extensions automatically based on the configuration in the resolve.extensions array (default includes [".js", ".json", ".ts", etc.]).
       2. If you omit the extension in your import (e.g., import { file1 } from "module1"), Webpack will try to resolve the file by appending extensions from the resolve.extensions array in order.
       3. Important: If you explicitly mention an extension in your import (e.g., import { file1 } from "module1.js"), Webpack does not add another extension. For instance, import { file1 } from "module1.js" will not result in "module1.js.js". Webpack simply uses the provided extension as-is.
12. MUI Drawer Chronology:  
    
13. Some **JavaScript libraries**, such as Lodash, **are not inherently compatible with TypeScript** because they do not have built-in type definitions. To use these libraries in a TypeScript project, **you need to install the @types package** for the library (e.g., @types/lodash), which provides the necessary type definitions.
    1. If you install only the @types package without the actual library, you will encounter errors because the @types package is just for TypeScript type checking and doesn't include the library's functionality. Both the library and its @types package are needed to use the library effectively in a TypeScript project
14. The primary purpose of the class-transformer package is to transform plain JavaScript objects (raw data) into class instances. This is especially useful when working with APIs or raw data where you want to create instances of a specific class without manually mapping the data.
    1. Using the plainToClass(<Class>, <data>) method, you can directly transform the raw data into instances of the specified class.
15. The class-validator library is used to validate the data in JavaScript or TypeScript projects, particularly when working with class-based objects.
    1. By **using decorators**, class-validator allows you to define validation rules directly in the class definition. When you validate an instance of the class, the library checks if the data meets the specified rules. This eliminates the need for manual validation logic.
    2. Decorators like @IsString(), @IsEmail(), @IsNotEmpty(), and many more allow you to specify validation rules on each property.
    3. You can validate objects within objects using the @ValidateNested() decorator.
    4. Validate arrays of data using @IsArray() and additional constraints for each element.
    5. Validator() returns promises to handle errors. Ask for code snippet in ChatGPT