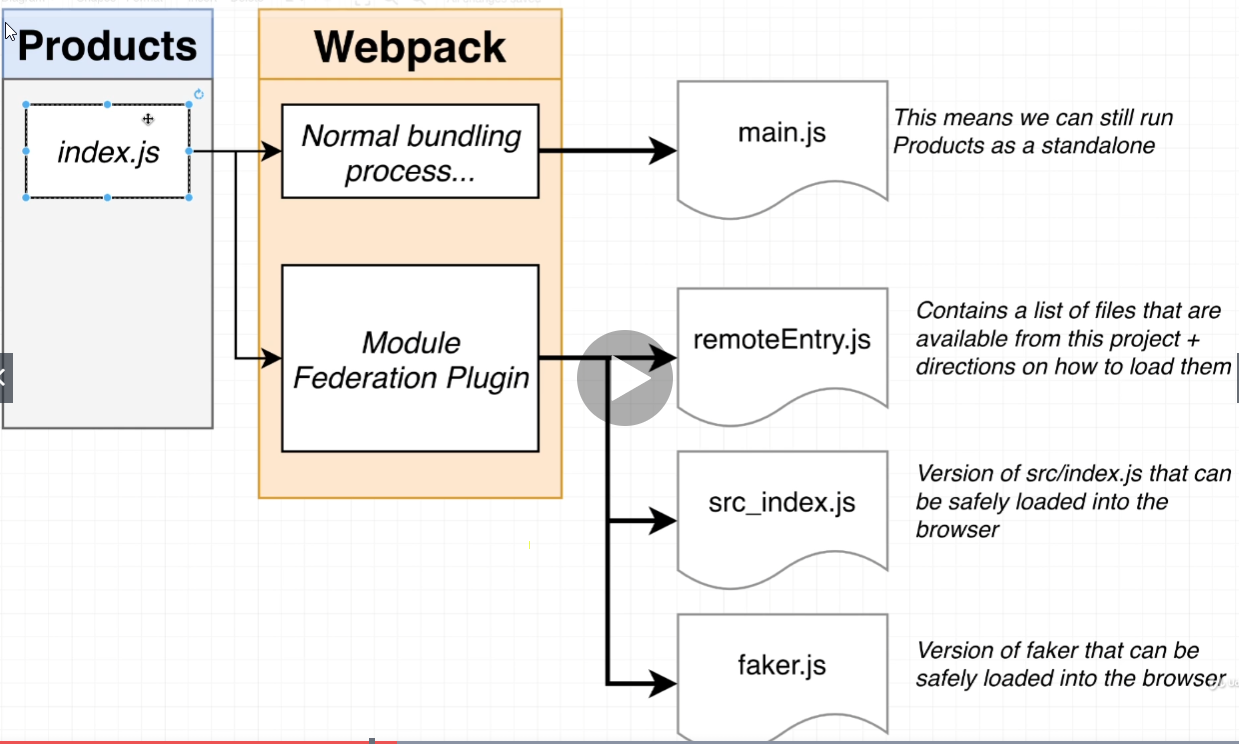
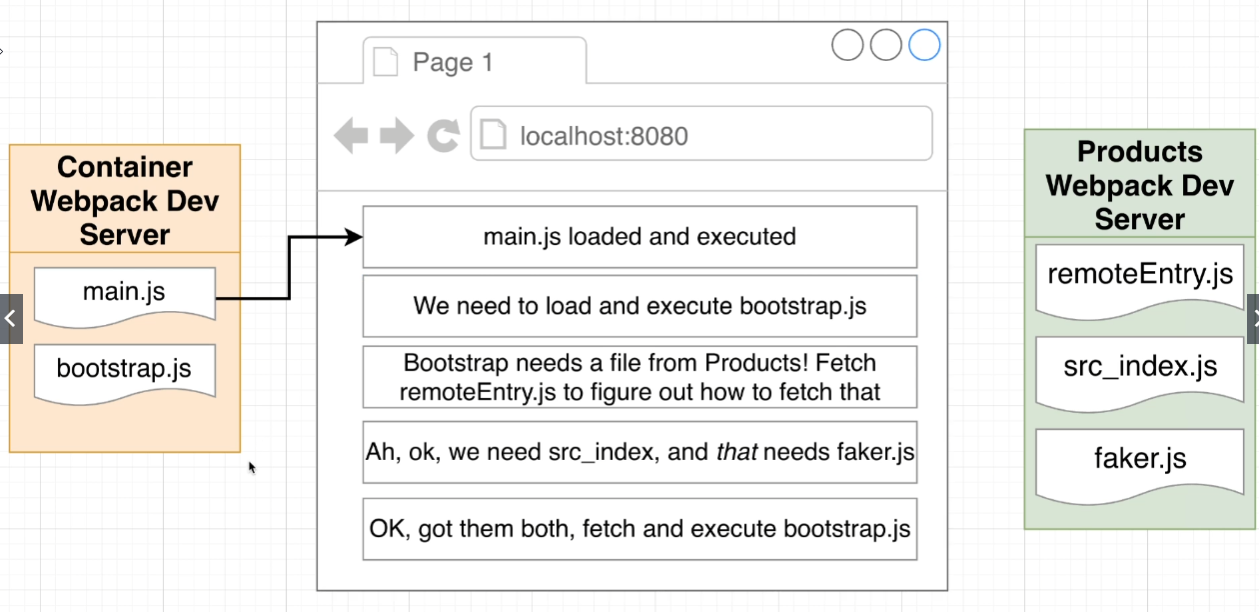
Micro Front Ends

* Container Application
  + Decides when/where to show each microfrontend
* Integration
  + How and when does the container get access to the source code in MFE #1 & MFE #2
* Build-time integration
  + Compile-time integration
    - Before container get loaded in the browser, it gets access to ProductsList source code
* Run-time integration
  + - After container get loaded in the browser, it gets access to productsList source code
* Server integration
  + - While sending down JS to load up container, a server decides on whether or not to include ProductsLIst source code
* Build time integration
  + Engineering team develops ProductList
  + Time to deploy
  + Publish ProductsList as an NPM package
    - NPM registry
      * ProductsList
  + Team in charge of container installs ProductsList as a dependency
  + Container team builds their app
  + Output bundle that includes all the code for ProductsList
  + Cons
    - Container has to be redeployed every time Productslist is updated
    - Tempting to tightly couple the Container and Productslist together
* Run Time Integration
  + Engineering team develops ProductsList
  + Time to deploy
  + ProductsList code deployed at static url. Htpps://productsList.js
    - Javascript file has all the source code
  + User navigates to my-app.com, container app is loaded
  + Container app fetches productsList.js and executes it
  + Pros
    - Independently deployed
    - Different versions of productslist can be deployed and container can decide which one to use
  + Cons
    - Tooling and setup is complicated
* Course is focused on run-time integration using webpack Module federation
  + Spend lots of time focusing on webpack and how it works

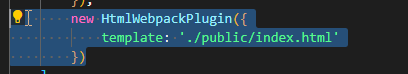
Webpack

* Takes JS files from node\_modules and combines them into 1 file
* Integration
  + Designate one app as the host (CONTAINER) and one as the remote (PRODUCTS)
  + In the remote, decide which modules (files) you want to make available to other projects
  + Set up Module Federation plugin to expose those files
  + In the host, refactor the entry point to load asynchronously
  + In the host, import whatever files you need from the remote

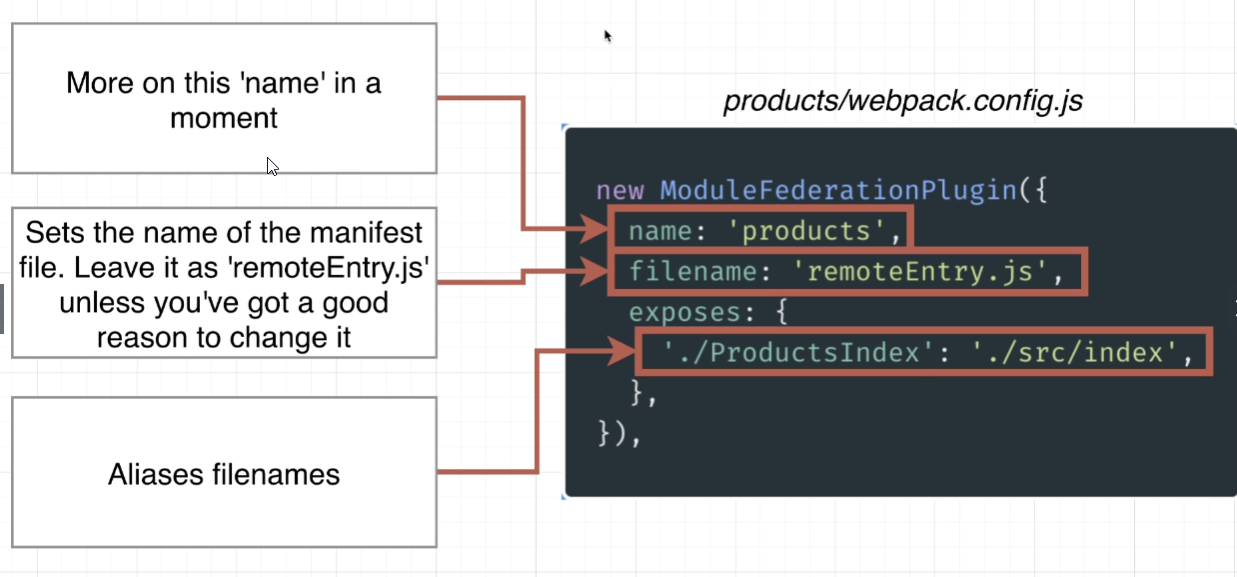




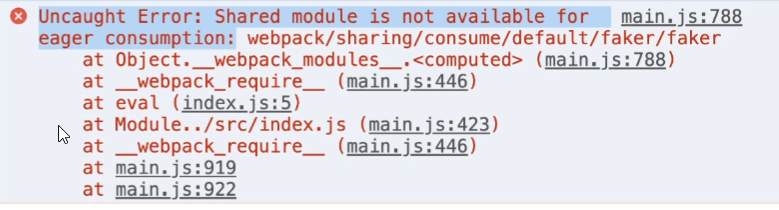
Tells word where to put JS files



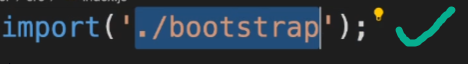
To export file, if anyone wants to import,



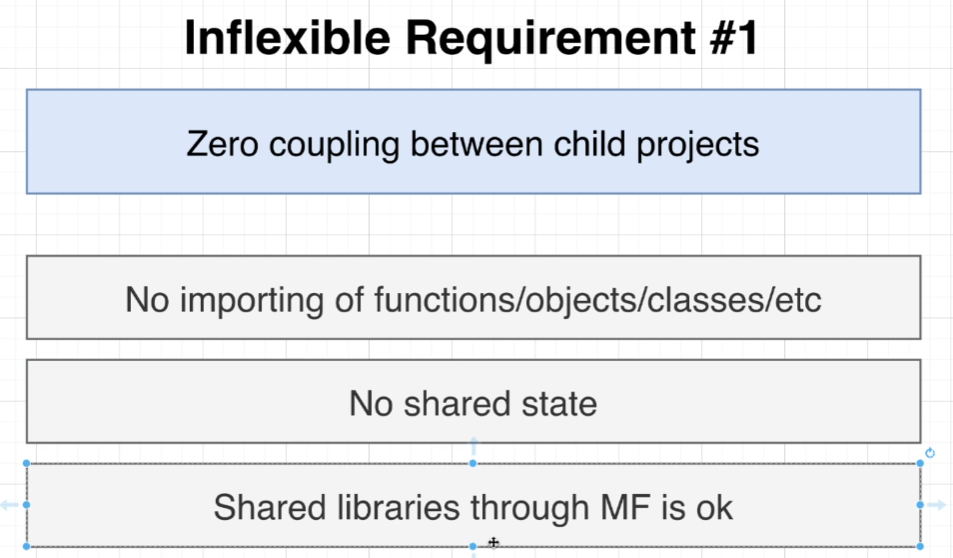
Async Script loading

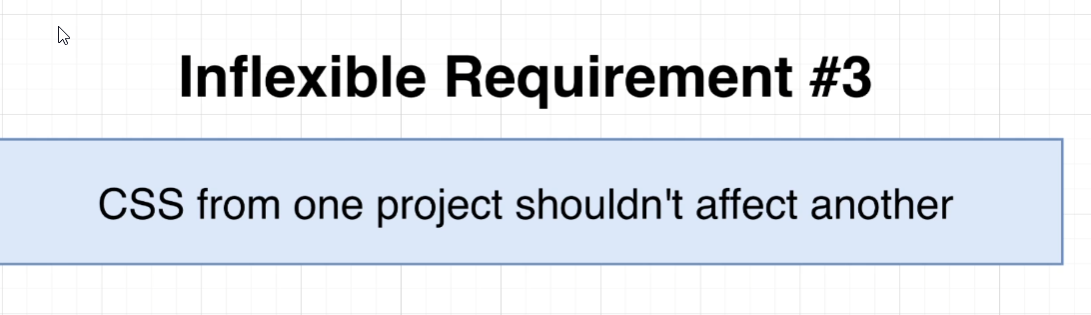


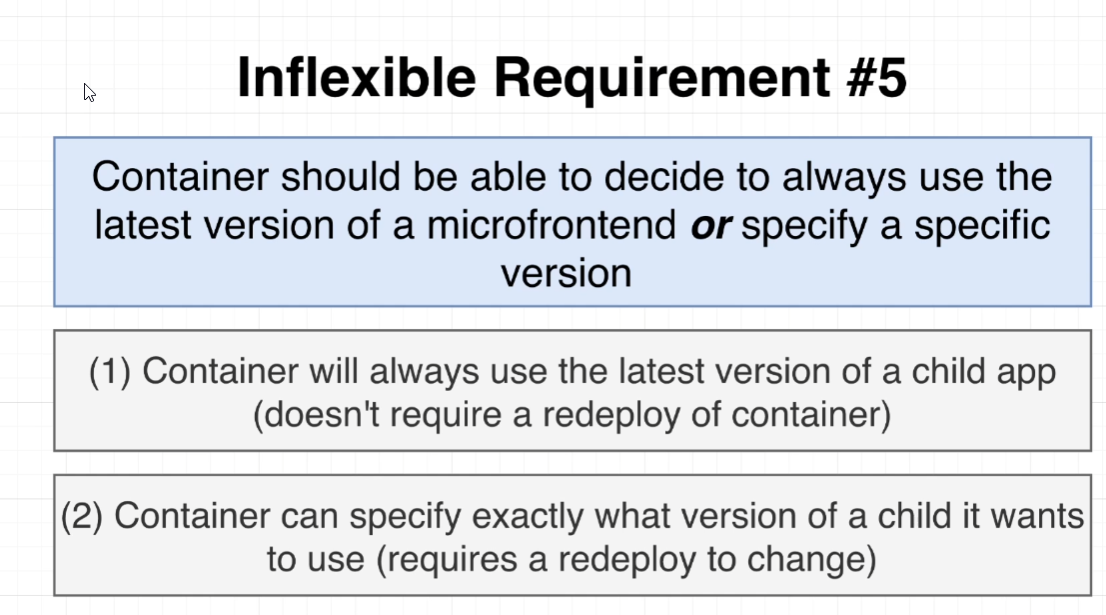
Use import as a function rather than a statement

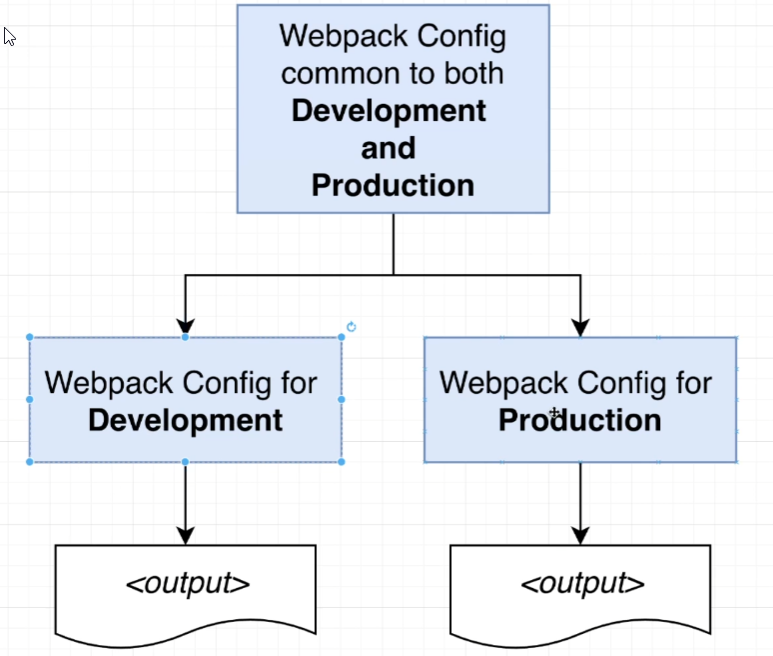


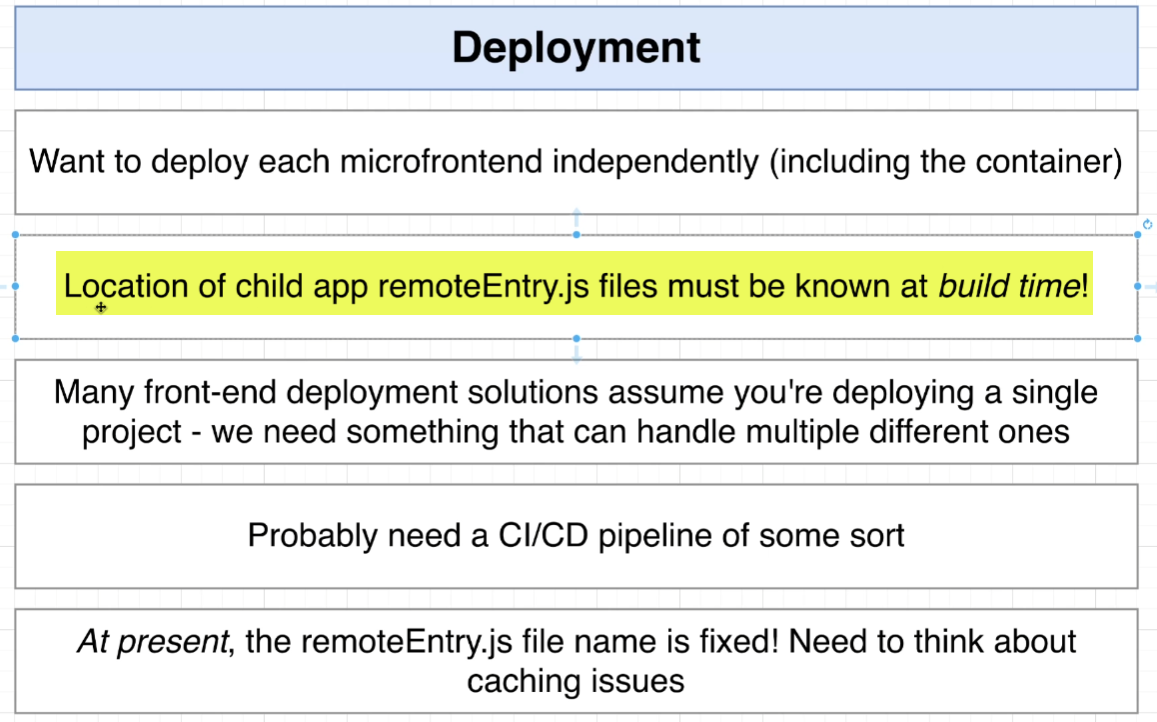
Project 2 requirements

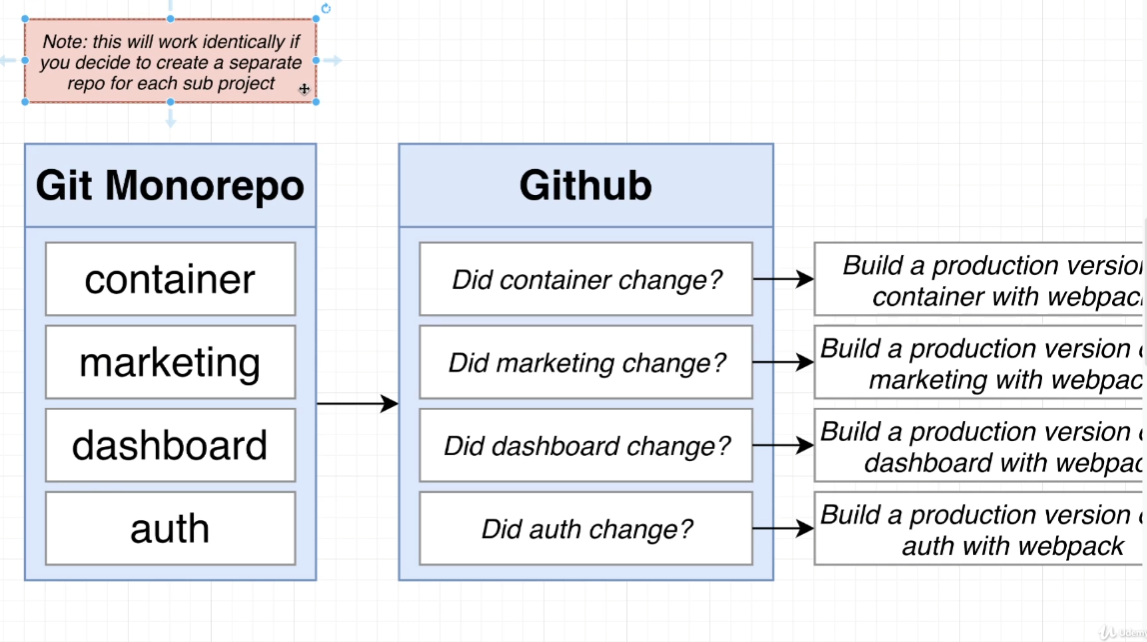


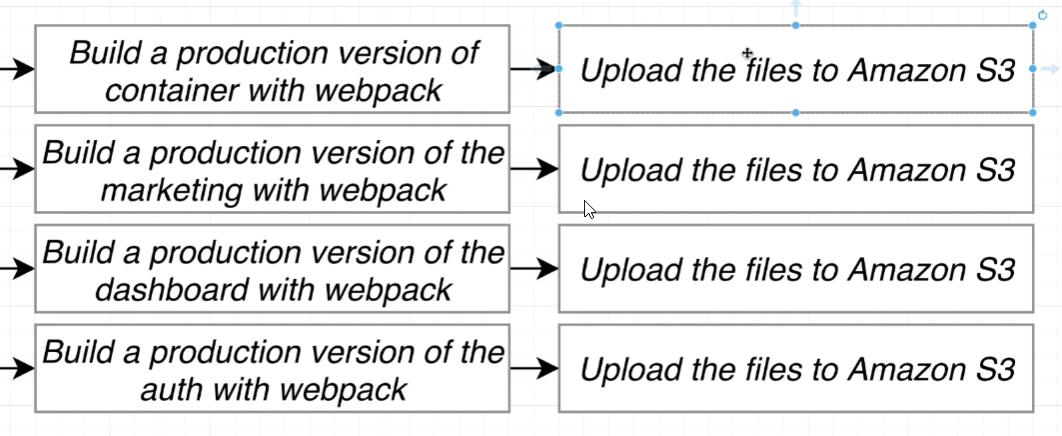




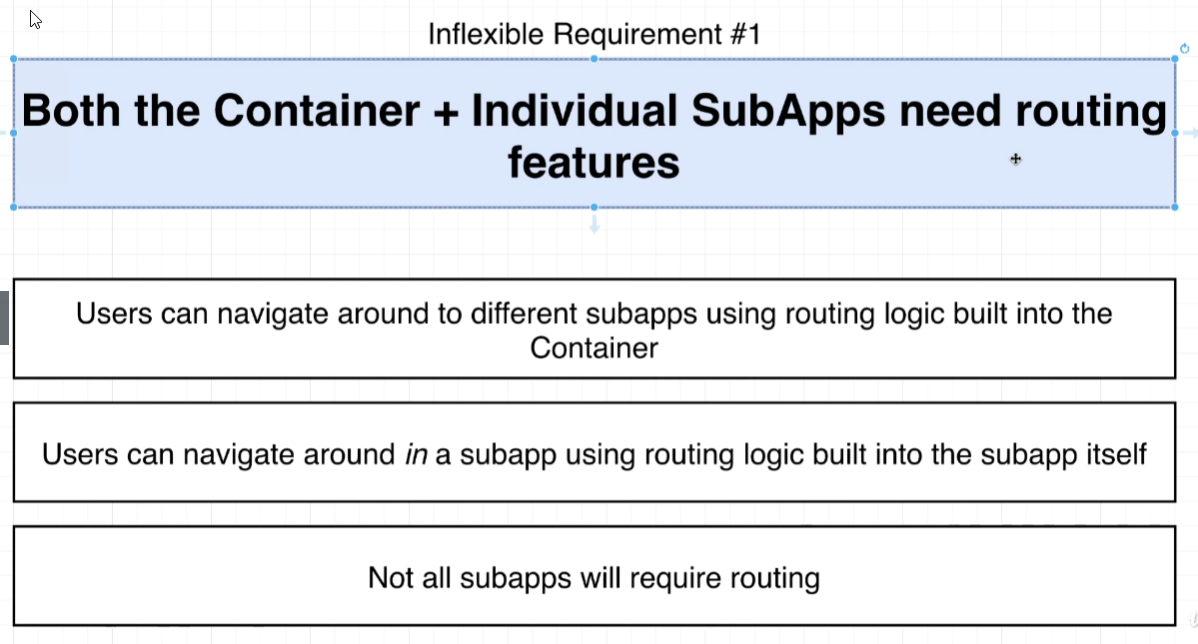


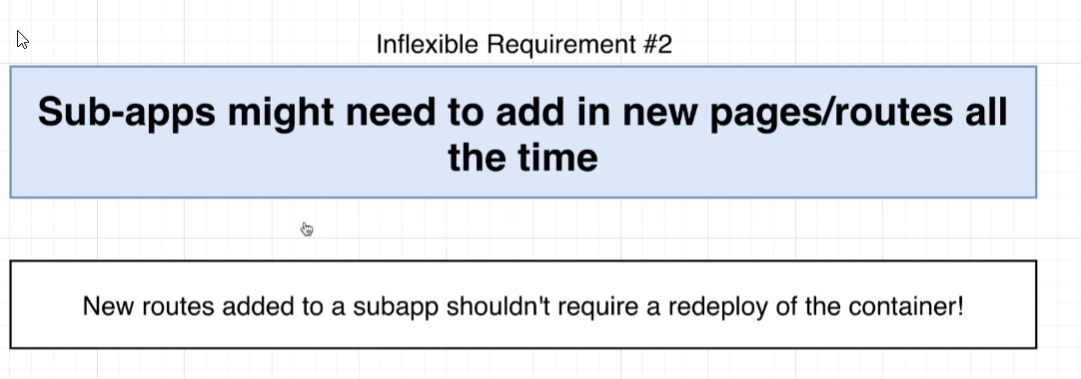


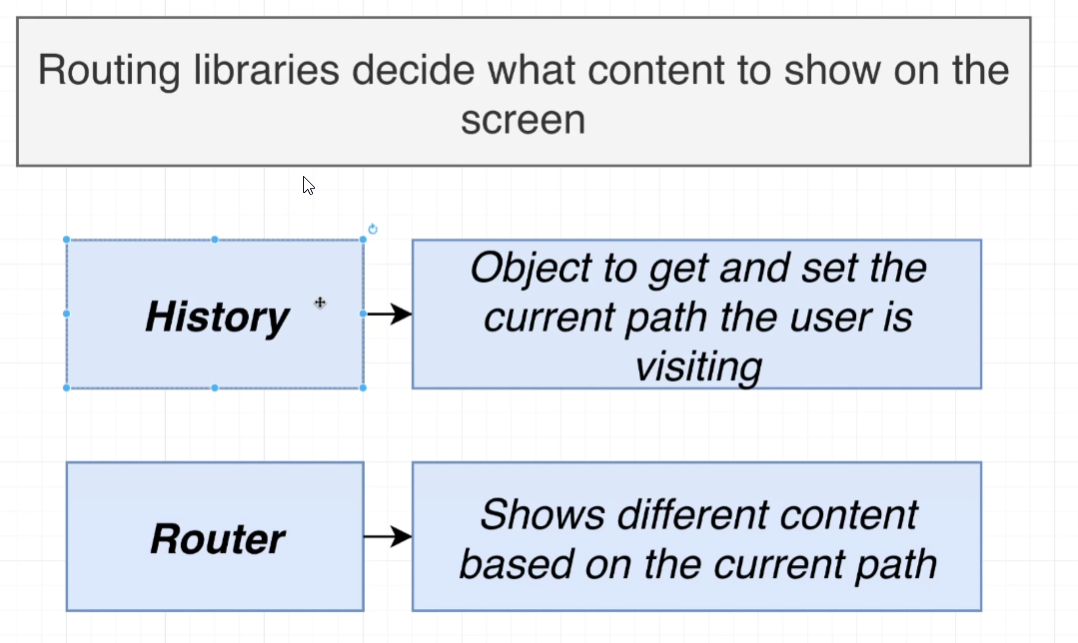


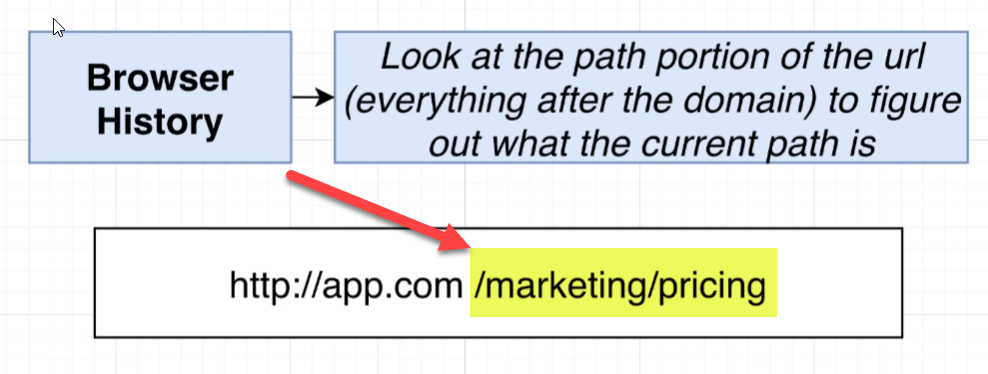


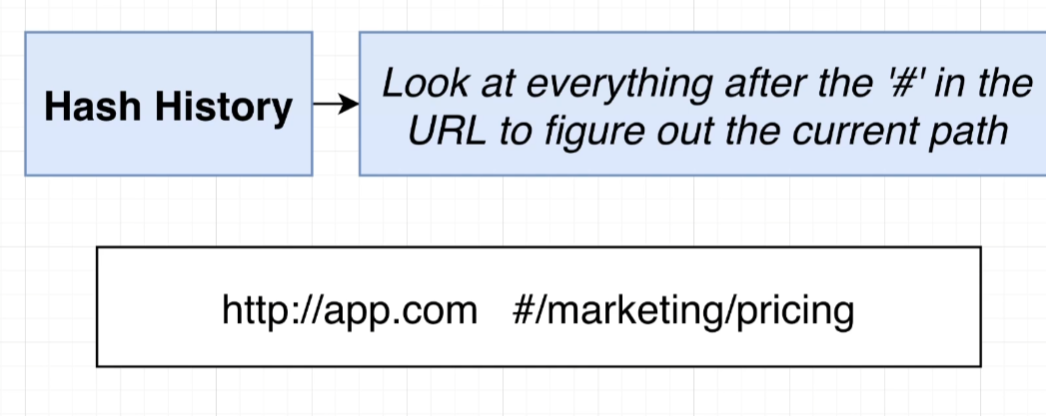
Cloudfront = (CDN) Content Delivery Network

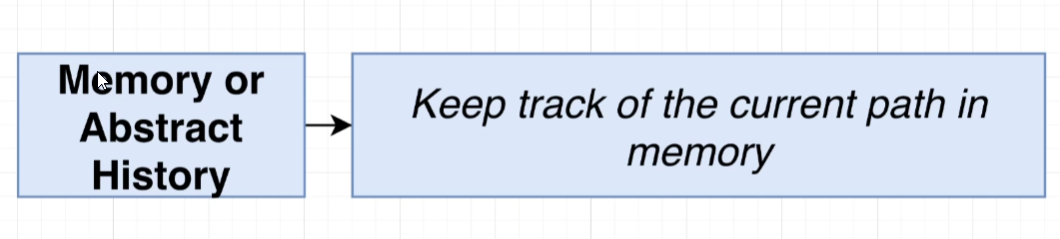










 +

