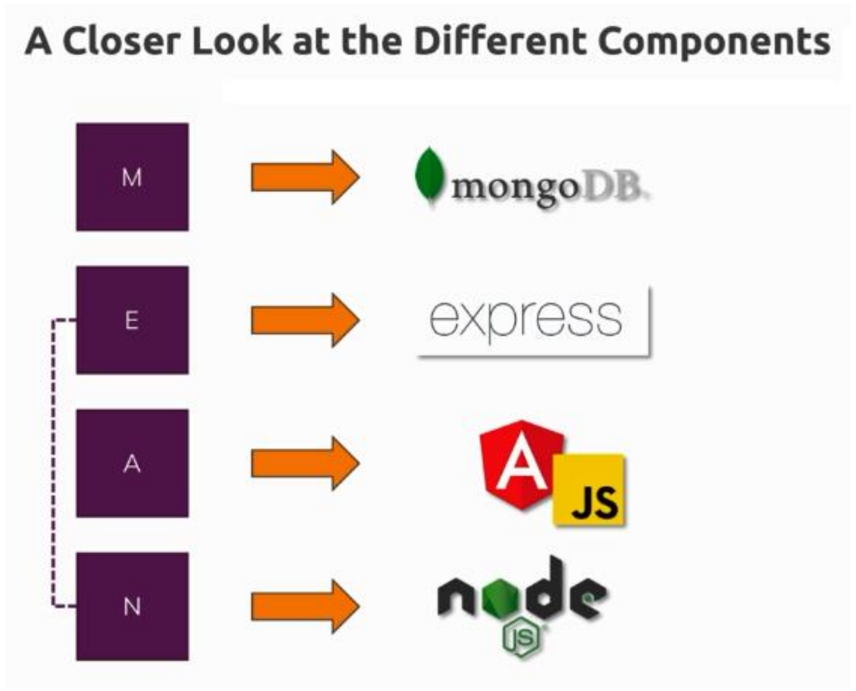
MEAN Stack

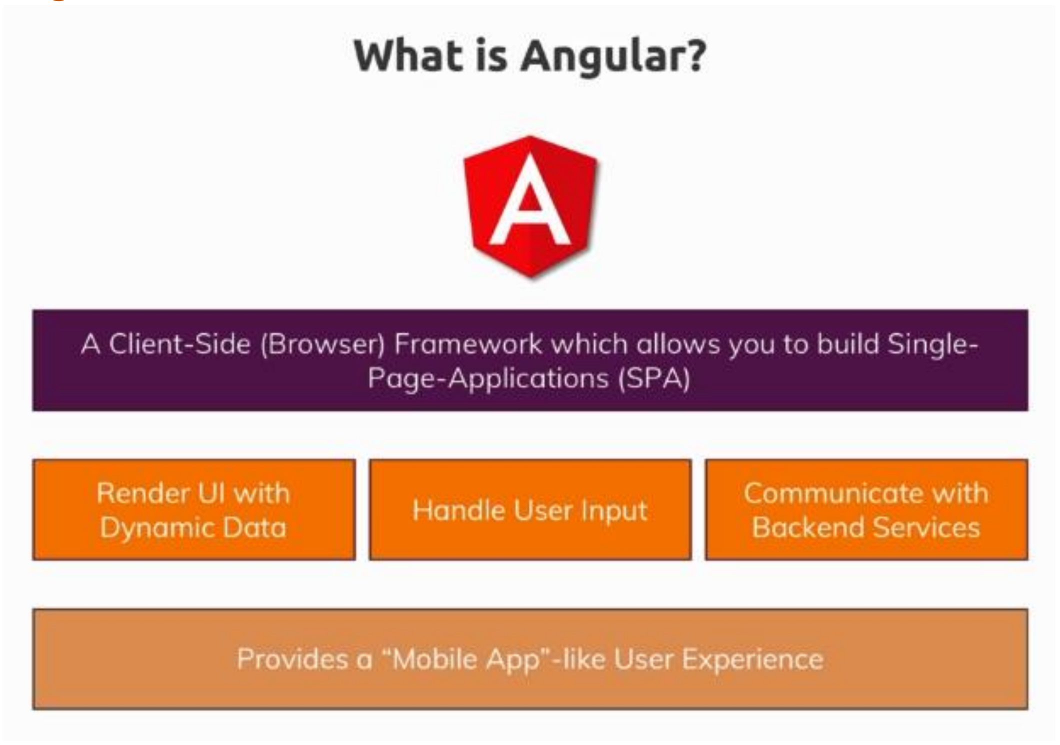
Contents

Getting Started

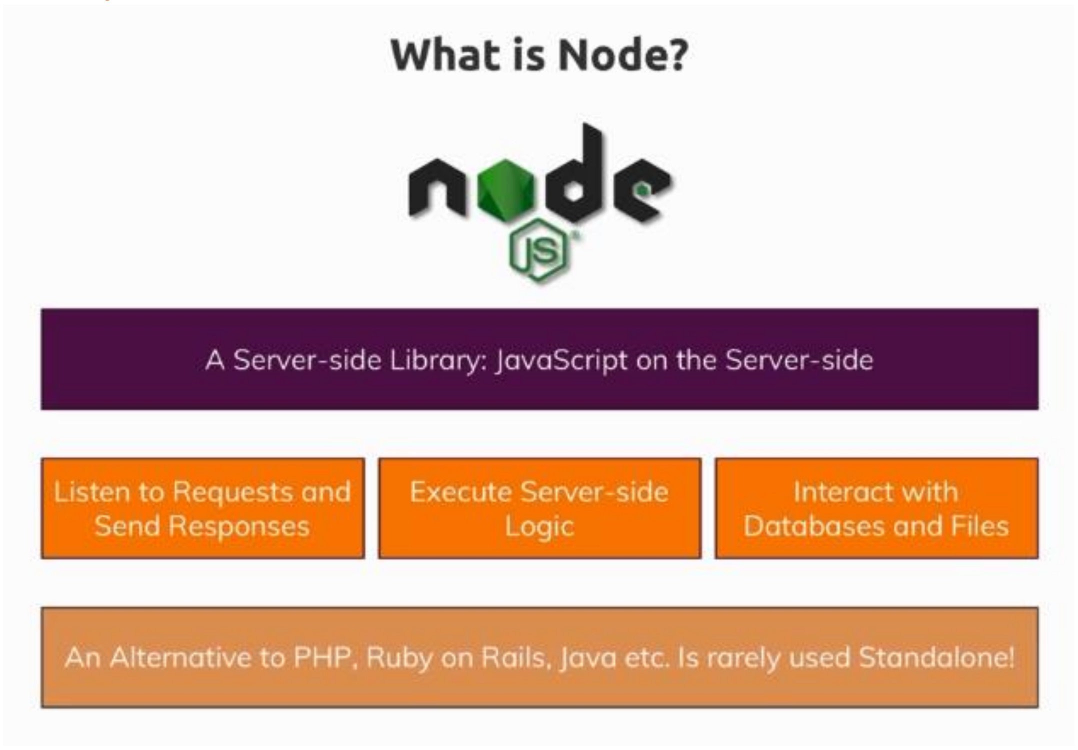
What is MEAN



Angular


What is Angular?
A Client-Side (Browser) Framework which allows you to build Single-Page-Applications (SPA)
Render UI with Dynamic Data
Handle User Input
Communicate with Backend Services
Provides a "Mobile App"-like User Experience

Node.js


What is Node?

A Server-side Library: JavaScript on the Server-side

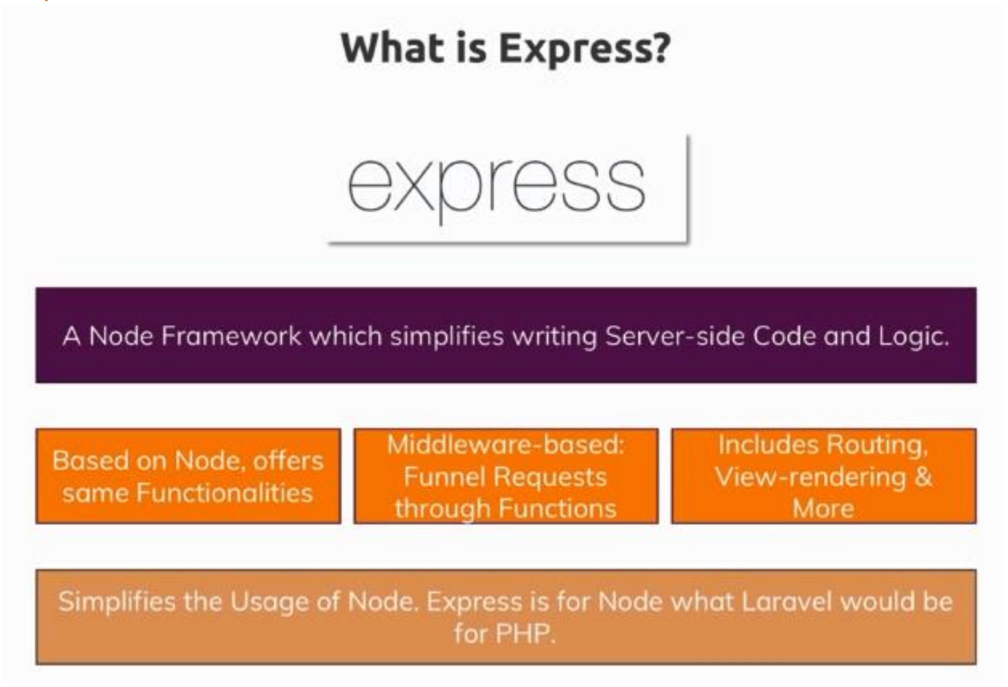
Listen to Requests and Send Responses

Execute Server-side Logic

Interact with Databases and Files

An Alternative to PHP, Ruby on Rails, Java etc. Is rarely used Standalone!

Express


What is Express?

express

A Node Framework which simplifies writing Server-side Code and Logic.

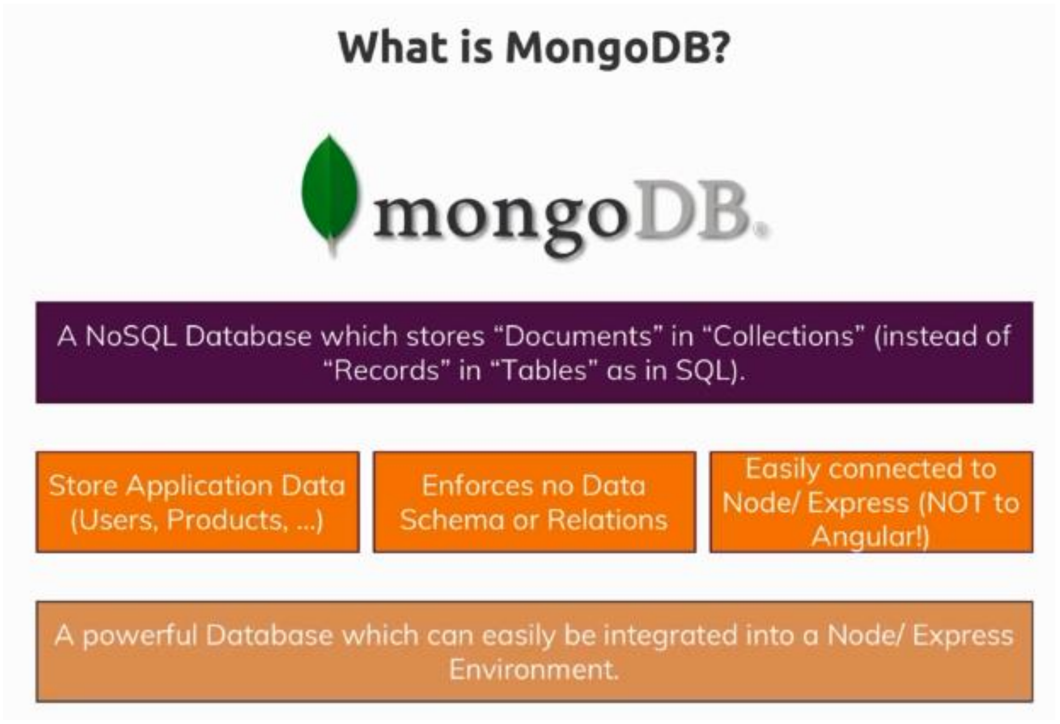
Based on Node, offers same Functionalities

Middleware-based: Funnel Requests through Functions

Includes Routing, View-rendering & More

Simplifies the Usage of Node. Express is for Node what Laravel would be for PHP.

MongoDB


What is MongoDB?

mongoDB.

A NoSQL Database which stores “Documents” in “Collections” (instead of “Records” in “Tables” as in SQL).

Store Application Data (Users, Products, ...)

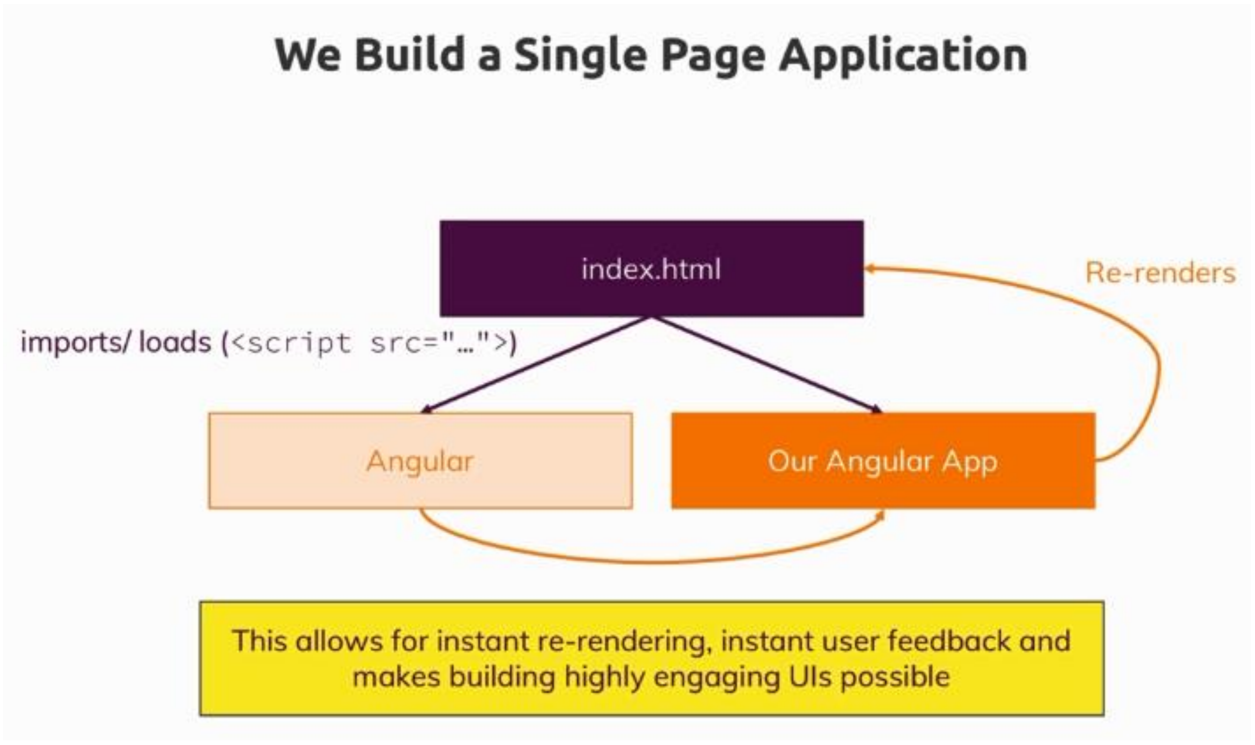
Enforces no Data Schema or Relations

Easily connected to Node/ Express (NOT to Angular!)

A powerful Database which can easily be integrated into a Node/ Express Environment.

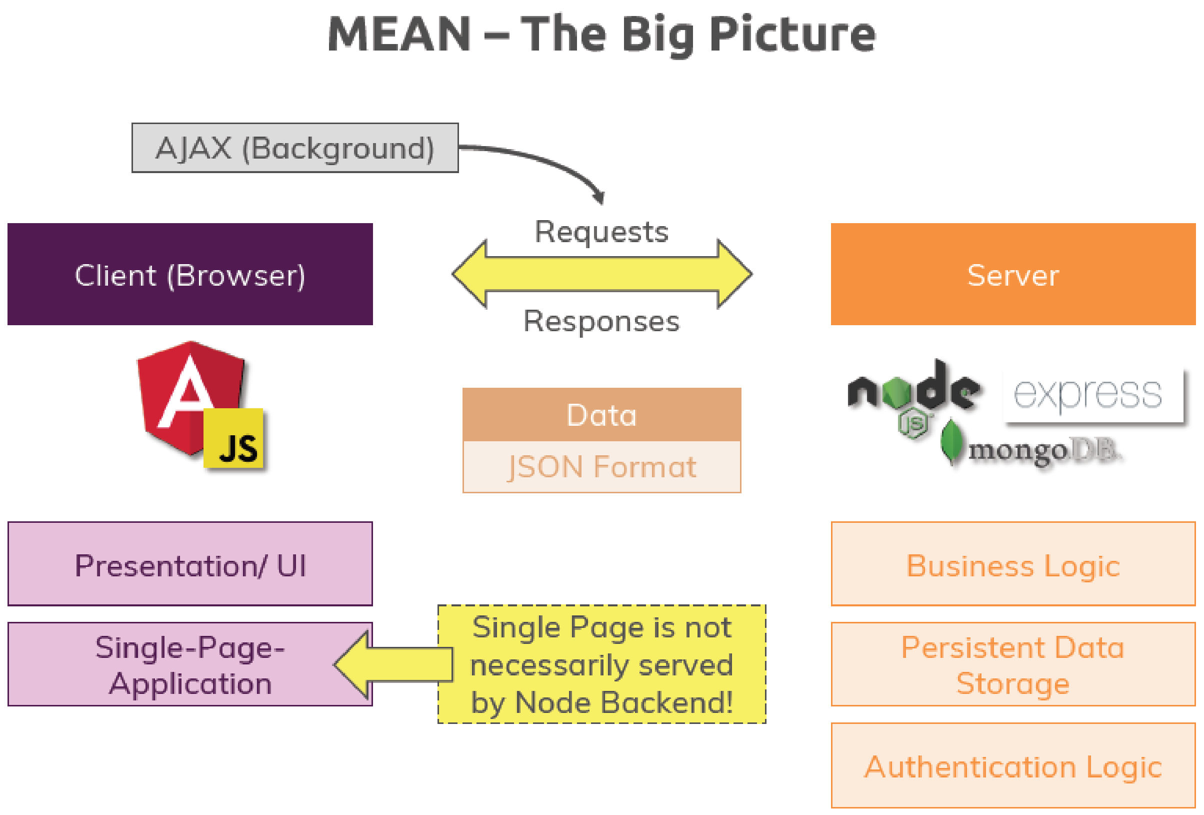
- You could theoretically replace MongoDB with a different database.
- You could use a SQL database in a Node/Express/Angular application too.
- D on the application and the type of data you are storing, if it's a data with a lot of relations, maybe a SQL database might be better.

Single Page Application (SPA)


We Build a Single Page Application
index.html
Re-renders
imports/ loads (<script src="...">)
Angular
Our Angular App
This allows for instant re-rendering, instant user feedback and makes building highly engaging UIs possible

- In an Angular app, we'll have one root HTML file, a so-called index.html file and we will serve that from our Node server or from a different server. And this HTML page basically includes some script imports that houses our Angular app, so the Angular framework and our own code and we use this application to dynamically re-render what the user sees, without ever requesting a second page to be rendered by the server.
- Because by having this pattern, we never need to reload the page.
- This provides a highly-interactive, mobile-app-like feeling, a very responsive and fast web page where we never have to wait, where things always happen and that of course is a great user experience.

MEAN – The Big Picture



- We exchange requests and responses and these requests and responses are sent behind the scenes, so-called AJAX requests. We use exactly the same pattern in Angular.
- These are requests which can be sent without us needing to reload the page which is of course exactly what we want.
- The type of data we exchange is not HTML because we never want HTML code, we do all that presentation and re-rendering logic with Angular. Instead what we get is so-called JSON data, that's a data format that's really efficient for encoding data like a list of posts.

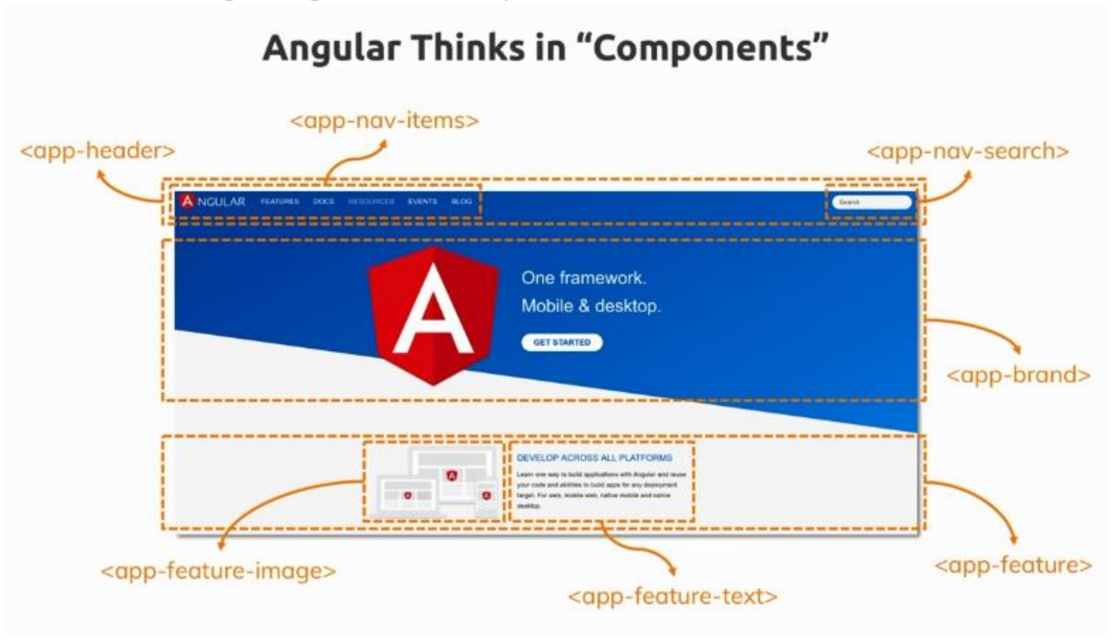
The Angular Frontend – Understanding the Basics

- Typically we start with frontend part so that we can see something and we can always work with some dummy data until we have the real backend added but we get a beautiful UI right from the start.

Understanding the Folder Structure

- Angular thinks in components.
- The Angular cli in the build process (ng server, etc.) takes our code, bundles it up, adds all the angular logic from the angular framework to it and therefore creates a bunch of script files which we don't see (in case of ng serve) because it's loaded in memory for development and injects these script file imports into the index.html file. And in the end is this actually then detects this <app-root> element and swaps it with the content of our component and content is not just the visual part (html part), it would also be any logic we have in corresponding component .ts file.
- We only have one root component in a typical angular application and all other components would be somehow nested in that root component.
- platformBrowserDynamic() is a function, we execute that, it essentially starts the angular application in the browser.

Understanding Angular Components

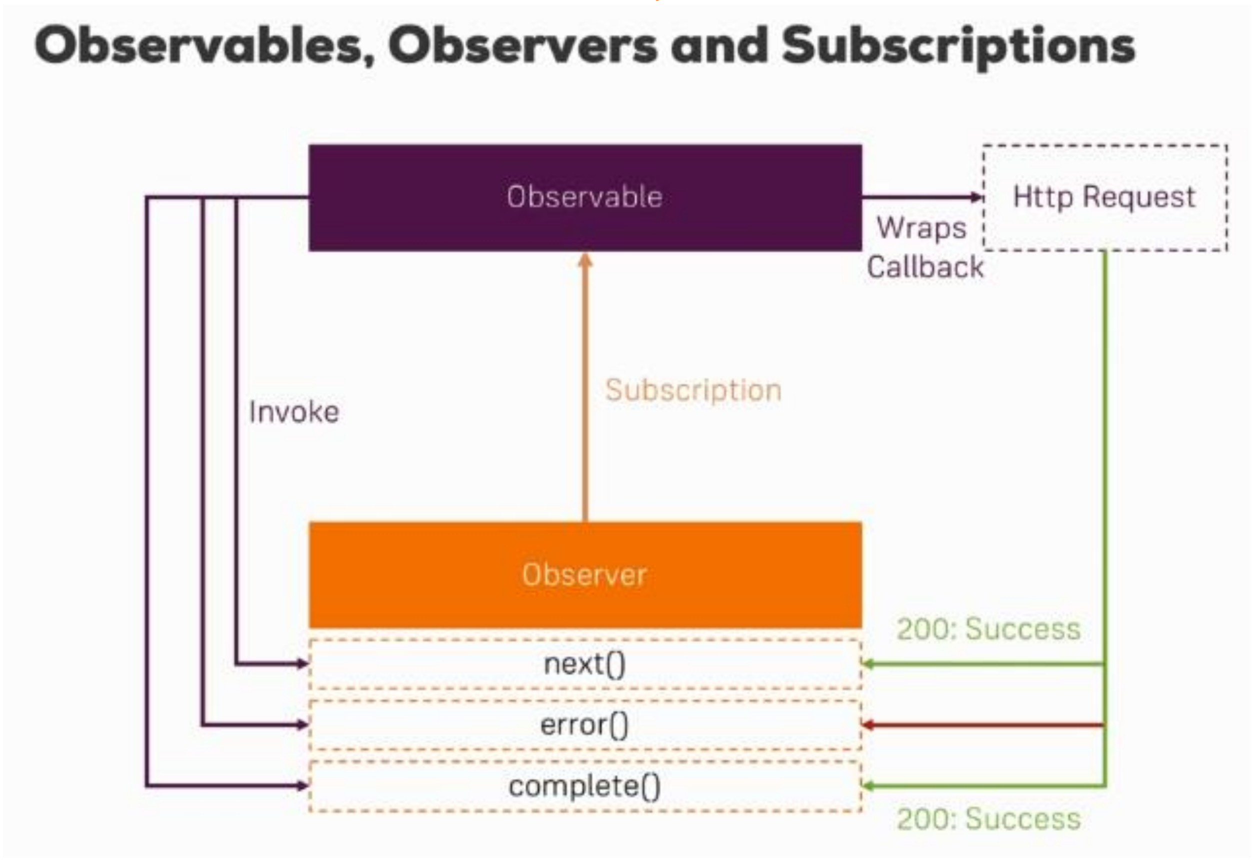


- Angular thinks in components.
- Essentially you compose an entire page of these components, you build it with these components because the advantage of this is that you have small, easily to maintain and

manage building blocks for your UI which you can even reuse because some components appear more than once on a page.

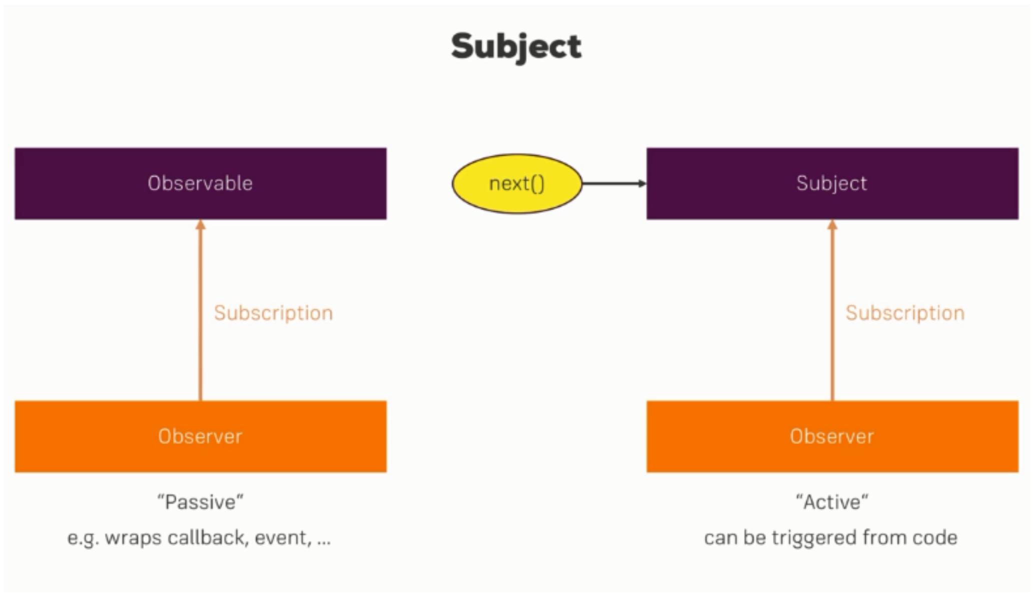
- If you got a certain part of your UI that really is decoupled from other elements on the UI and that probably also contains some logic, like the search here, it definitely has some logic like type ahead or filtering or showing some preview, so if you get that, then you want to put it into its own component so that you can easily manage the code and possibly reuse it, that search could be used in other parts of the application too.

Observable, Observer and Subscription



Subject

- A subject is really just a special kind of observable.
- A **normal observable** is kind of **passive**, you wrap a callback or an event source like a click listener with it. So you don't actively trigger when a new data package is emitted, that happens when your http requests gets a response or when the user clicks something, instead you just set up this listener and then you can subscribe to it.
- A **subject** is more **active**. We also subscribe to a subject but there we can manually call next() and that makes it a perfect event emitter because we cannot just subscribe and wait for something but we can directly influence when a new data package is emitted.



Stream

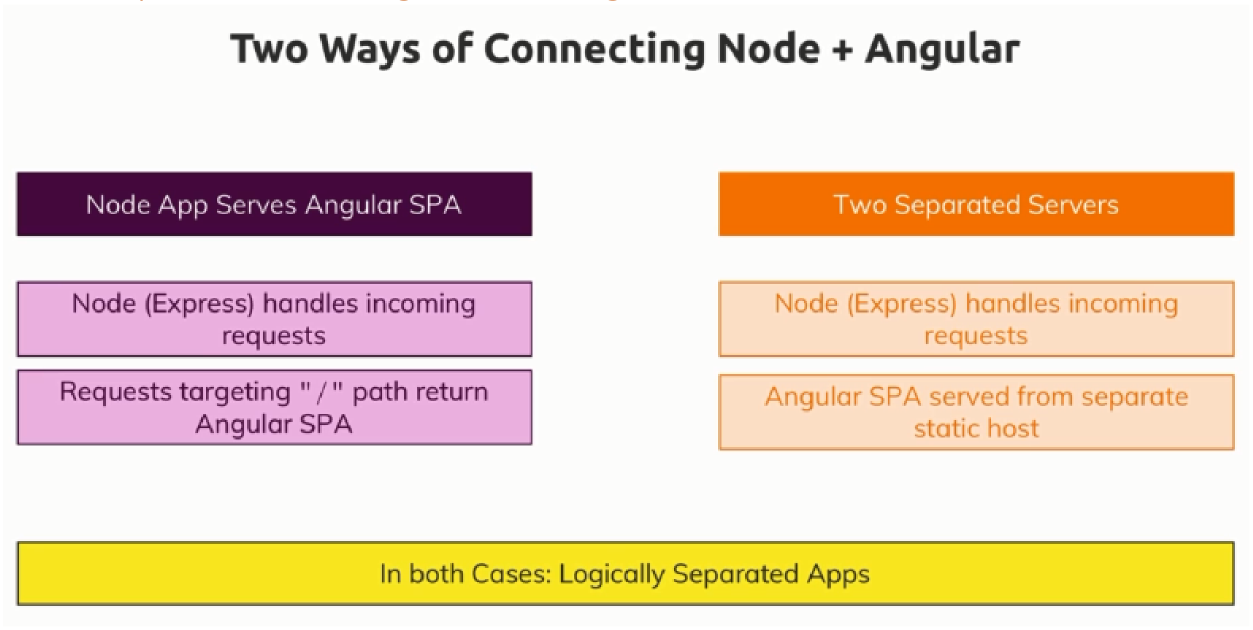
- In general, you can think of observables and therefore also subjects as streams of data or of values, so we've got one value and we can have more values which are emitted over time depending on the observable and the data source it covers.

Adding Node.js to the project

Connecting Node and Angular

- ng serve behind the scenes actually uses a Node.js server, really just because it's very simple to set up such a node server but ng serve only gives us a development server.
- A development server simply means it's a server aimed at angular development, it's not a production ready server and it certainly doesn't contain any of the logic we want to add to our server side. It also doesn't give us an entry point to add such logic. It's really just a server that returns the angular app and that also has useful features like auto-restart whenever we have a new angular app version.
- So ng serve is nice for developing our angular application, it's not the server we will use as a backend.

Two ways of connecting Node + Angular


Two Ways of Connecting Node + Angular
Node App Serves Angular SPA
Node (Express) handles incoming requests
Requests targeting " / " path return Angular SPA
Two Separated Servers
Node (Express) handles incoming requests
Angular SPA served from separate static host
In both Cases: Logically Separated Apps

One Single Server

- The first approach we can take is that we have a node application, a node express application that serves the angular single page application as part of it. It contains our other server side logic but it also has a certain path so a url endpoint to which we can send a request where it will return that angular single page application.
- So in the first approach here, we have our node express back which handles incoming requests and that's not just the request for the angular app but also requests sent by the angular app because angular sends background http requests to store data, to fetch data and these requests need to be handled by the node express backend.

Two Separate Servers

- The alternative is that we have two separate servers. So we have our node express server for our business logic, for the authentication, for the data storage and then we have a separate static host which only returns our angular single page application.
- For the separate server solution, we still have our node express app handling incoming requests because we have these background requests sent by angular but additionally, we also serve our angular single page application from a totally separate static host.
- A **static host** is really just a simple server, could be a node server but could be any server, an apache, an nginx server, so any server that doesn't run any server side logic or any server side code but simply returns html, javascript and css files and that's all our angular app consists of.

Understanding CORS (Cross Origin Resource Sharing)

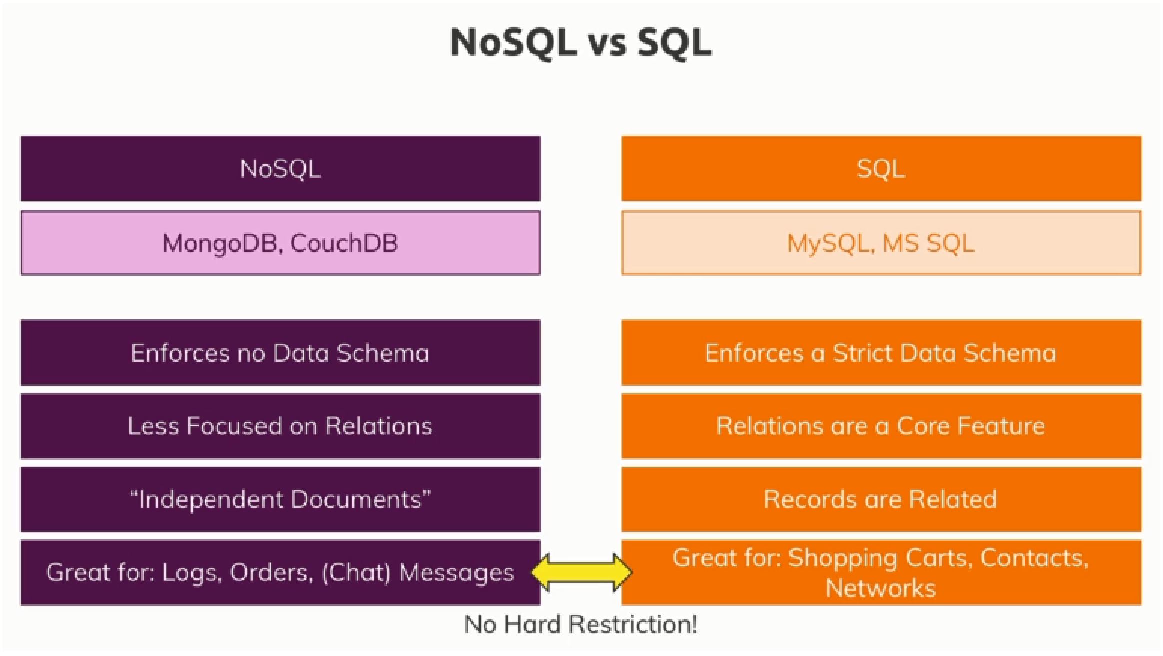
- If client and server want to talk to each other and they're on the same host (IP + Port), we could communicate without any issues.
- If client and server want to talk to each other and they're NOT on the same host (IP + Port), then communicating with such background requests will fail and that's a security mechanism.
- You should not be able to access the data on a server or its resources in general if you're not running on the same server. So if the request is coming from a different address, this will give us a so-called CORS error.
- The problem just is for our setup and for many modern web apps, we want to allow this. It's not a security issue here it's a wanted behavior.
- We want to expose our server API to all possible clients and naturally, they will not run on our server. So we need to disable this default mechanism. And this is done by setting the right headers on the server side response.
- To allow the Cross-Origin requests, we need to set few headers.
    - '**Access-Control-Allow-Origin**' to * which means no matter which domain the app which is sending the request is running on, it's allowed to access our resources.
    - '**Access-Control-Allow-Headers**' – restrict this resource to domains sending requests with a certain set of headers besides the default headers. So we want to allow the Origin, X-Requested-With, Content-Type, Accept headers.
        - You can add more, you can also remove some of them.
        - This essentially means the incoming request may have these extra headers, it doesn't have to have them but it may have them and it will still be allowed.

        - If it has another non-default header which is not defined here, access would be blocked even though we do generally allow it for all domains.
    - '**Access-Control-Allow-Methods'** – here we control which http verbs may be used to send requests. Here we can set comma separated list of allowed HTTP methods.
        - Allow OPTIONS because This is an implicit request sent by the browser by default prior to post requests for example to check whether the post request is valid.
- E.g. In express, we can set it as

```
app.use((req, res, next) => {
  res.setHeader("Access-Control-Allow-Origin", "*");
  res.setHeader(
    "Access-Control-Allow-Headers",
    "Origin, X-Requested-With, Content-Type, Accept"
  );
  res.setHeader(
    "Access-Control-Allow-Methods",
    "GET, POST, OPTIONS, DELETE, PUT, PATCH"
  );
  next();
});
```

Working with MongoDB

NoSQL vs SQL

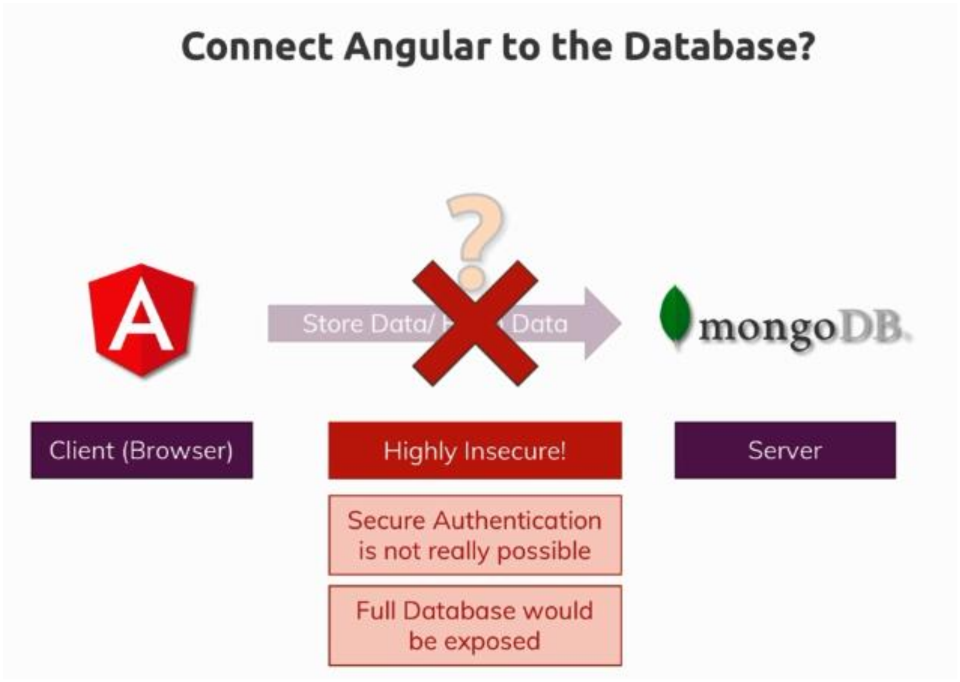


NoSQL vs SQL

NoSQL	SQL
MongoDB, CouchDB	MySQL, MS SQL
Enforces no Data Schema	Enforces a Strict Data Schema
Less Focused on Relations	Relations are a Core Feature
"Independent Documents"	Records are Related
Great for: Logs, Orders, (Chat) Messages	Great for: Shopping Carts, Contacts, Networks

No Hard Restriction!

Connecting Angular to MongoDB?



Connect Angular to the Database?

Store Data/
Data
mongoDB.
Client (Browser)
Highly Insecure!
Secure Authentication is not really possible
Full Database would be exposed
Server

Mongoose driver

- Mongoose is a third party package which builds up on the official mongodb driver but it makes accessing mongodb much easier and more convenient.
- The official mongodb driver does not use schemas but **mongoose uses schemas**, so you can define how your data should look like and that allows you to conveniently store and fetch data.
- mongoose is a great tool that could even work with unstructured data but it’s **great especially if you have structured data,** if you have a certain model you use.

Enhancing the App

Client side Routing vs Server side Routing

- The angular router is actually just a tool that is able to parse the url of our app and then render different things onto the screen through javascript.
- It's not loading different html pages, we only have one html page, the index.html page but it's re-rendering the content on that page for the different urls we're visiting.
- The routes in the angular router are only known by angular which is a client side application. This means that the server doesn't know these routes, neither our backend here nor any server that might serve our angular app.
- It's important to understand the difference between client side routing which is all about reading the url and re-rendering parts of the page and server side routing which is all about handling incoming requests and sending back something different.
- In the server side routing, you really exchange data, you are sending requests and responses, in the client side case, you're not doing that, you are reading the url and you're re-rendering the page.
- It's also **important** to note that if you were to host your angular app on the SAME host (IP + Port) as your node app, then you must NOT use routes you defined in angular in your backend too. That is, the routes across backend and angular must be unique in this case.

File Upload

- Store the uploaded file on the File system and save its path to the database.
- It is inefficient to store the file in the database.

Pagination

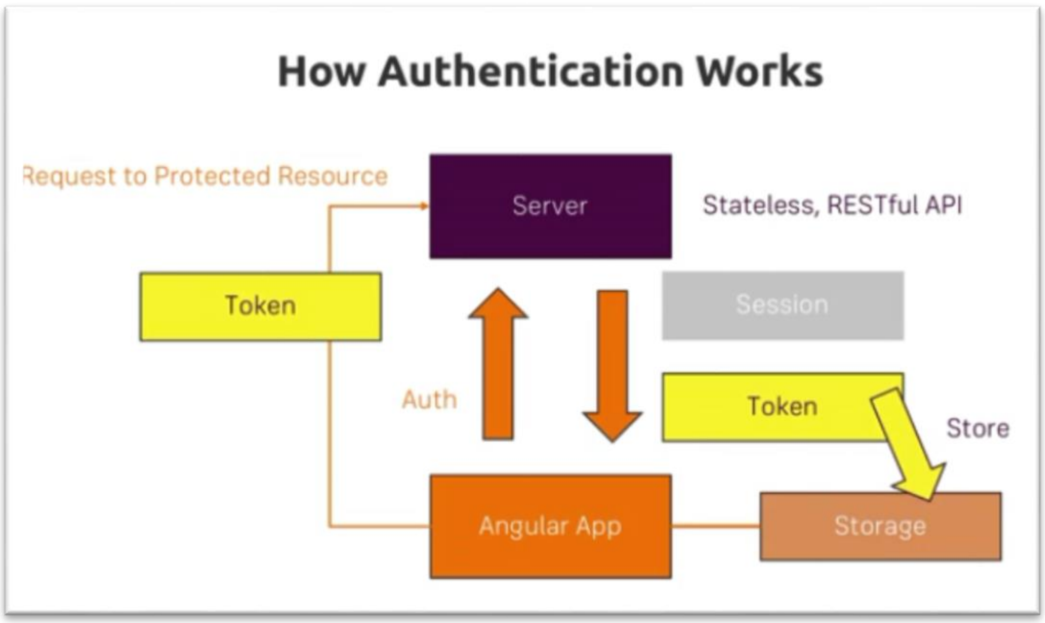
- Data should be dynamically fetched for given page number and number of records per page.

Authentication & Authorization

Authentication

How Authentication works in SPAs (like Angular)

- When we authenticate or when we login, we sent the credentials to the server and the credentials are validated and we get back a response that we are authenticated.
- Now in the traditional web app where we have multiple pages, we would use a session to store it. The session would be stored on the server and on the client we would get a cookie or sessionid.
- Now in single page applications, our back ends are stateless (like REST API). Any Angular app is stateless because we always use Ajax HTTP requests behind the scenes as only got one single page and we don't request new pages in between. So therefore the session based approach doesn't work with SPAs. So here we use token instead, to be specific JSON Web Token (JWT).
- JWT is essentially a long string that encodes (not encrypts) some data about our authentication status, data that can't be fiddled with. Because if we would fiddle, then the token would be detected as manipulated on the server and would be invalid.
- So in SPAs, when we send credentials, we get this token from the server after successful authentication. We then should store this token on the front end that is, in the browser storage place like localstorage.
- Now whenever we want to access some protected resource on the server, so let's say our database, we will attach that token to the request.
- Now as we know the token is structured in a way that the server can validate whether it's still a valid token (the one which was sent by the server) or manipulated one.
- So if the token is still the original token and we still are logged in (so the token is still valid because the token will also expire after some time), then we get access. Otherwise we don't.
- This is how authentication works in SPAs.

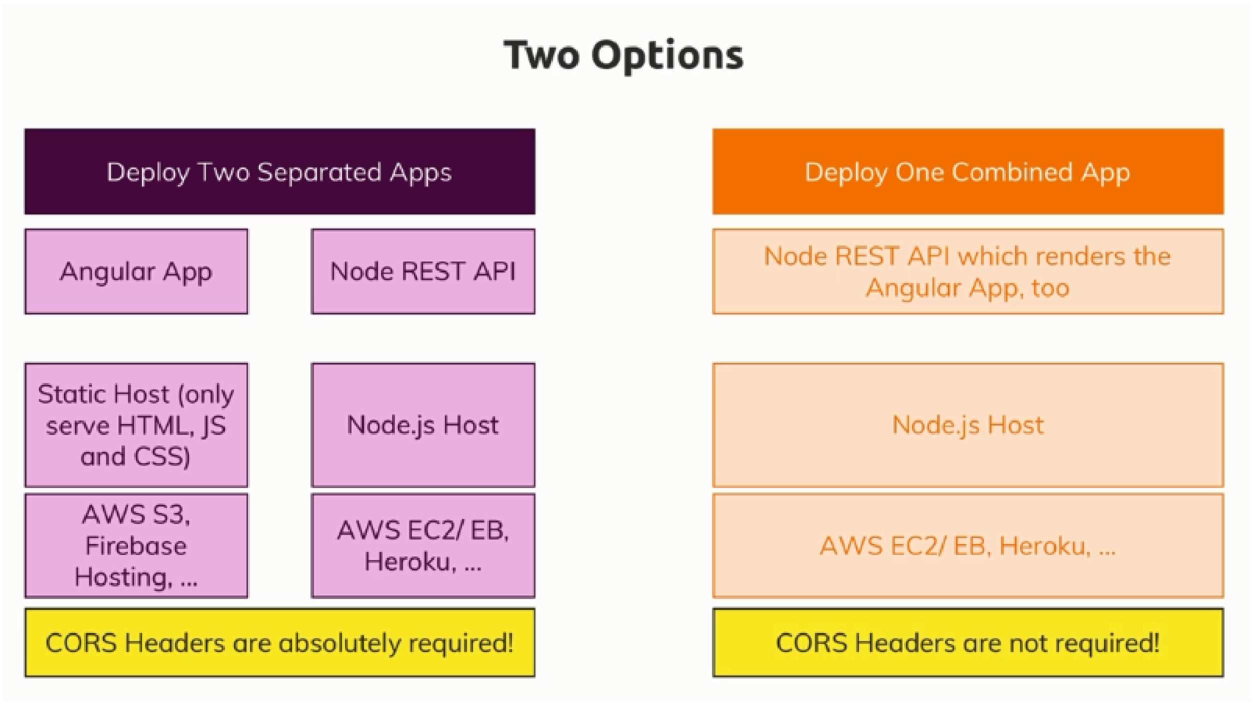


Authorization

- Perform proper Authorization at backend by protecting certain actions on certain routes.
- Use the backend data to protect UI Actions (e.g. allowing only creator of a post to edit/delete that post).
- Authorization on frontend is basically for better User Experience only. Actual authorization must be handled in the backend only.

Deployment

Deployment Options


Two Options
Deploy Two Separated Apps
Angular App
Node REST API
Static Host (only serve HTML, JS and CSS)
Node.js Host
AWS S3, Firebase Hosting, ...
AWS EC2/ EB, Heroku, ...
CORS Headers are absolutely required!
Deploy One Combined App
Node REST API which renders the Angular App, too
Node.js Host
AWS EC2/ EB, Heroku, ...
CORS Headers are not required!

- Please read this as well – Two ways of connecting Node + Angular

Deploy Two Separated Apps

- Refer: https://github.com/sameerbhilare/MEAN/tree/main/Workspace/mean-course

Deploy One Combined App

- Refer: https://github.com/sameerbhilare/MEAN/tree/main/Workspace/mean-course-SingleHost

Tips and Tricks

- In Angular project, double-clicking on the index.html file won't work because that will use the file protocol and not the HTTP protocol and therefore many features we need are not enabled.
- ng serve behind the scenes actually uses a nodejs server, really just because it's very simple to set up such a node server but ng serve only gives us a development server.
- **Global Environment Variables for Backend (Node.js)**
    - We can use with dotenv package. But this needs requiring the package in the code and reading .env configuration file.
    - OR we can easily use nodemon's nodemon.json file to inject global environment variables which we can then directly access using process.env.<env-var-name>
- Sample nodemon.json file for setting global env variables.

```
{
  "env": {
    "DATABASE": "mongodb+srv://<db-user>:<db-password>@<mongodb-host-
name>/<db-name>?retryWrites=true&w=majority",
    "JWT_SECRET": "secret-key"
  }
}
```

    - Now we can access the above variables like this – process.env.DATABASE, process.env.JWT_SECRET