- Routing
- Link vs <a>
- Rendering
    - Passing Data From URL
        - Note : We can only pass the data from url to server in page file only for that particular url (rember how we implemented sorting by name and email.)
- Special Files
    - Layout.tsx
- Programatic Navigation
- Suspence loading.txs
- Handling Route-Which Does not exist not-found.tsx
- Handling UnExpected Error Occur in our Program error.tsx
    - Steps to configure database
        - Important commands
        - Steps
    - Start
        - Configure the providers
        - Something about tokens
        - Middleware
        - Database Adapters
        - Custom Credential Matching
            - Additional Reading
            - Exercises
-
-
    - 1. Images
    - 2. Adding third party Libraries : eg. Google Analytics
    - 3. Using Fonts
    - 4. Serach Engine Optimizations
    - 5. Lazy Loading
-

Node js help us to create component which are rendered at server side.

[Pros]

- Smaller Bundles
- Resource Efficinet

- SEO
- More Secure

[Cons - Server Component Cannot use below things]

- Listen to browser events
- Acces browser APIs
- Maintain State
- Use effects

`create-project` : npx create-next-app@latest

`Conclusion` : in realworl we often use mixture of server and client components , we should default to server components and use client compoent only we absolutely need them.

`app-directory :` all components by default in this folder is server component.So will rendered on the server only.

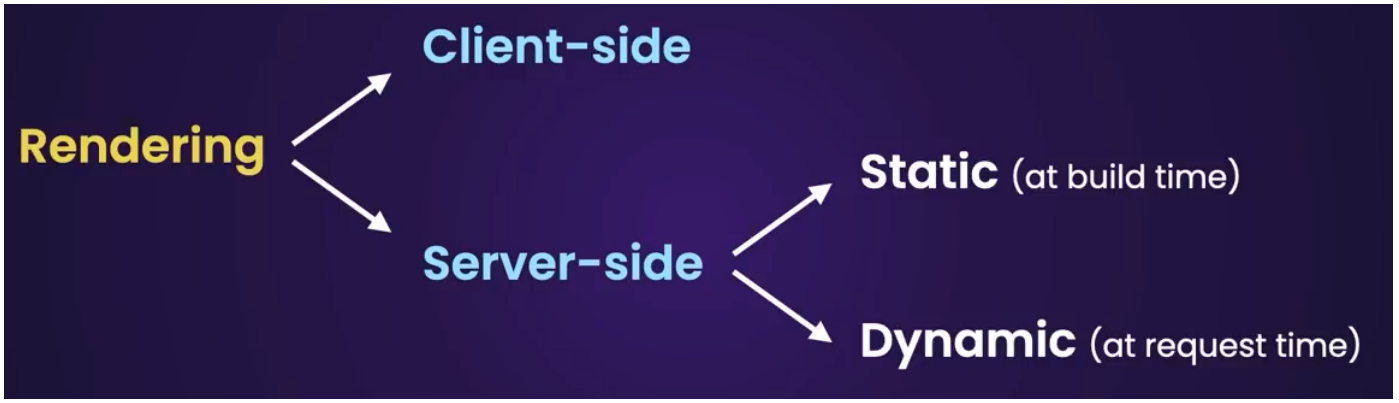
'`use client` -> add this first line of file : to make this client compoent now we can add onClick event to button and all the compoents this compoent depends upon will automatically becomes client component, so now this won't be pass as rendered html page , it will pass as js file in bundle.

`To get fake data :` https://jsonplaceholder.typicode.com/

`API-Calls` : In react we have to use usestate()+useeffect() for fetching data and maitaing the state and this is done on client side which is very slow, we can do it directly in next-js and also on server only (which do not exposes our api to client which makes our website more secure).

`Data-Souce` : Memory(fastest) , File System (Slower) , Network(Slowest).To help in that we can use cacheing.So whenever we fetch data using fetch function next-js will automatically store those data in cache (which is based on file system , means those caches will be stored in files.)So next time when we go for fetching data from same url , then next js will fetch those data from cache.(ofcourse we have full control over that cache behaviour). `To disable cache` : fetch(url, {cache: 'no-store'}) for that url chache will be disabled.

`To Revalidate Cache` : fetch(url, {next : { revalidate : 10}}) this will make cache to revalidate this data every 10 sec (means refresh it.)



`Tailwind-UI-Library` : https://daisyui.com/components/

`Shared-cdn` : https://ui.shadcn.com/

`Next UI` : https://nextui.org/

Routing

- `Static` : this type of routing does not accept any parameters they just define folder name and page file in it.
- `Dynamic` : this accepts the parameters inside it remember how we define routing for users and [id] in it.
- `Generic` : This is my term it's basically universal king of routing , remember how we defined the [[...slug]] one. See name can be any but it's conventions so we are using it.

Link vs `<a>`

`note`: Never use `<a>` tag for links use `<Link>` provided by nextjs so that for each request it should not download all the files on require file for that page.

- Link will only downloads the content of the target page.
- It Pre-fetches links that are in the viewport.(to see in action start app in prodution)
- As we navigate between pages Link will cache pages on the client.So next time we visit the same page next js won't do any request will pull out page from client cache.And this client cache only exist for one session and clear when we do full page reload(refresh button in web-browser).

Rendering

- `Client-Side` : This is like react works where html file is created(rendered) in the client side all the api call everyting done on client side , which leads to the round trip to server (like bust the bundle of scripts and then for api calls) which makes it very slow.
- `Server-Side Rendering` : In this files are renderd at server side (html will be created on server side it self which makes it very fast and secure because all api request will be made before client get's the pages.)
- `Hybrid -Rendering` : In this some compoent are rendered on server side and some on client (as mentioned above server side render pages have some limitation on interactivity for that we put the interactive part such a way that will render on client side , Remember how we created the add to cart button on start.)

Passing Data From URL

`Note : We can only pass the data from url to server in page file only for that particular url (rember how we implemented sorting by name and email.)`

Special Files

`page.tsx` : to define the route (only this is publically accessible in form of url.)

`layout.tsx` : defining common layout for pages.

`loading.tsx` : for showing loading UI's

`route.tsx` : for creating api's

`not-found.tsx` : for showing custom errors

`error.tsx'` : for showing general custom error page.

`Note` : any file that we are creating can have extension : .js, .jsx, or tsx

Layout.tsx

- This is file where we define the common layout for our pages , now this follows like, the one inside app (direct child) will apply to all the page files.

`Note` By default everything will be unstyled in when using tailwind, remember how h1 were acting like the normal one.

Programatic Navigation

- if we want to navigate to page click on button (like submit a form) use below code.

```
import { useRouter } from "next/navigation";

const NewUserPage = () => {
  const router = useRouter();
  return (
    <button
      className="btn btn-primary"
      onClick={() => router.push('/users')}>Create</button>
  )
}
```

Suspence `loading.txs`

- This is fallback UI that we show to user , while our page is loading.
- `How it works` : First server will send the html having suspence , but wait there request response cycle won't end here(which generally do in normal pages) , server will keep it open until it will send the main content which user was waiting(in our example table) and this process is called streaming (same as video and audio streaming).
- `ways to implement it` : 1. Wrap component around the suspence component. 2. Define the loading file. (in both creating loading file is better.)

Handling Route-Which Does not exist `not-found.tsx`

- How to handle case when user goes to page that does not exist.
- `not-found.tsx:` just create this file and define the components you want to show.

- once you define above `tsx` file then want to programitically show it call `notFound()`
- `notFound()` : will try to runder that not-found file which is closer to route.

Handling UnExpected Error Occur in our Program `error.tsx`

- Define `error.tsx` file which will show the custom page in case of error and it must have `use client` .
- It is same as not-found.tsx just like it we can define for all routes and closest will be called.
- and it will not detect error if occurs in `root-layout` file layout.tsx which is in app.For detecting error for this we have to define file `global-error.tsx`
- If some error occur in our produciton app it;s good to log somether eg `senerty website` so we can see what happend, now sometime we want to give users to retry in that case use parameter `reset` in propos and provide button to call it.Don't use retry techinque generally other wise it will bolt our error log just in certain part of our application.
- We simulated error my breacking the url endpoint of `json-place-holder`.

  Creating APIs

Will be creating APIs for

- Getting Objects
- Creating Objects
- Updating Objects
- Deleting Objects
- Validating requests with Zod

`Note` : every folder can have any of thes files `page` or `route` file , if we want to handle markup request then use `page` file but if we want to handle use `route` file.

Few HTTP methods

- GET : getting data

- POST: creating data
- PUT: updating data (technically to replace full entry / but use `patch` for just updating property of entry)
- For more : `https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods`
- About seciont s of HTTP request : `https://www.ibm.com/docs/en/cics-ts/5.3?topic=protocol-http-requests`
- We used validaiton library ZOD : `https://zod.dev/` (npm i zod) please read the doc of it for more information.

`note` Please refer file in dir : `app/admin/route.tsx` and `app/admin/[id]/route.tsx` to see the implementation of the apis.

`note` : serach few status code like , 200,201,400,404

```
Database Integration with Prisma
```

- `what is it` : This is ORM library for databases.

Steps to configure database

- Download sql community version
- Download jetbrains dataGrip to see our database.
- npm i prisma
- npx prisma init : to initialize prisma in our project(this time it will ask for the database name .)
- change connection string according to database you are using in .env file and also add that file in .gitignore.
- Define your model(eg.User : name will always start wil first letter capital.) in the `schema.prisma` file .
- To format the code we define in our above file type : `npx prisma format`
- `how to define more complex models` : `https://prisma.io/docs/concepts/components/prisma-schema/data-model`
- So as we define or change our model we have to create migrations this will help to database schema to be in sync with prisma schema. `npx prisma migrate dev` for mong : `npx prisma db push`

- Go to datagrip : connect to my sql (by providing creds and database name) and below click on test-connection to validate whether everything is okay or not.
- To work with our database first we have to create a prisma client, go in prisma folder and add file `client.ts` . Best practices to instantiate prisma client `https://www.prisma.io/docs/guides/other/troubleshooting-orm/help-articles/nextjs-prisma-client-dev-practices`

Important commands

```
# Setting up prisma
npx prisma init

#Formatting Prisma schema file so it look good in tabulor form
npx prisma format

#creating and running a migration
npx prisma migrate dev

#Quick Fix : When I created the product model it was sync with database as soon as migration but scripts does not recognized it for that I have used this command before that I have stopped the server:
        npx prisma generate
        npx prisma db push
```

```
Uploading Files
```

`note` : for uploading files we would need cloud providers eg. Amazon S3, Google Cloud,Microsoft Azure, Cloudinary, we would go with last one `Cloudinary` since it provides many react compoenet to wrok easier. `https://console.cloudinary.com/` , `npm i next-cloudinary`

- To know more about : `https://next.cloudinary.dev/`

Steps

- register to cloudnary if don't have account
- install npm package : `npm i next-cloudinary`
- follow the installation instruction from : `https://next.cloudinary.dev/`

- to get upload preset info got to => `https://console.cloudinary.com/` => Settings(bottom left small icon) => Product environment settings `Upload` => click on add upload preset => copy the name and change sign mode to unsign (makes easier to use while testing). In the end put the name of upload preset in the parameter of the CldUploadWidget component.
    - Once we upload any image then go to console=>Media Library => assests.

  `note` : cloudnary provides options to fully customized it's upload page please read the documentation for more. to do it go to `demo.cloudinary.com/uw/#/` and customize it there , and then go to source (now source is in js , but the property you see there will apply to component we can copy those and use it.)

  Authentication with Next Auth

- Setting Up Next Auth
- Configuring the Google Provider
- Authentication Sessions
- Protecting routes
- Database adapters
- Configuring the credentials Provider

Start

- To setup authentication go to : `next-auth.js.org` in future it is going to be `auth.js`
- Go to it's Getting started page to find how to setup which module to setup
- Create two environment variable `NEXTAUTH_URL=http://localhost:3000` `NEXTAUTH_SECRET=4MbFj82RXAyu7tVhMLyG2Zhw1uUPyOBiWUq5Q1pl5zo=`
- To generate the `NEXTAUTH_SECRET` I have used the `random.js` in `scratch` directory.

Configure the providers

- more info at : `https://next-auth.js.org/configuration/initialization#route-handlers-app` go to

providers section
- Refer any Video on how to setup .
- Now we have to add env variables `GOOGLE_CLIENT_ID` and `GOOGLE_CLIENT_SECRET` and which we will get from google.
- Now setup the handler file and make sure to end `!` in the end of variable so that typescript sure that we are providing those variables otherwise it will show us error.

Something about tokens

- When the user logs in next-auth creates a authentication session for that user , by default it represt that token as json web token , which you can see by going in browser->application->cookies. default expiry of that token is 30days.We also call this token JWT.
- Cookies : this is piece of information that is transefered between user and server each time user makes request.
- We just created the test api just to see the content encoded in token you don't have to do that : `http://localhost:3000/api/auth/token note` : we have created the directory for auth as `api/auth/[...nextauth]` so whever we find the `api/auth/signout` or signin ,it will hadle by this auth itself
- We can always customize the way signIn or signOut page looks.

Middleware

- This will help us to protect our api routes ,with middleware we can run the code before the request is completed.

- Make sure to use the name : middleware.ts and put in same leve as your app directory is.

```
export { default } from 'next-auth/middleware';
export const config = {
    // * : zero or more
    // + :  one or more
    // ?  : zero or one
    matcher: ['/users/:id*']
}
```

Database Adapters

- Since we are able to login via google , now we have to store the users in our database so we can know which are our users. For more `https://next-auth.js.org/adapters`
- Install the prisma adapter (don't use the way mention in website it mentioned in way of auth.js we are using next-auth) : `npm i @next-auth/prisma-adapter`
- We have to delete old table we created while testing and copy the tables schema from website prisma section.
- As soon as we include the database adapters our Next-auth will change authentication from JWT to database , for that we have to provide extra parameter , after the providers parameter.

  ```
  session: {
          strategy: "jwt"
      }
  ```

  `note` : if we don't use social login like google in our case then we have to handle . 1. store passwords in encrypted way in our database , we have to implement functionality to user to register,to change the pasword and reset the password and so , luckily this all is handled by social login or oauth providers.

now if you really want to do those step then go to this link how we can do that : this is Providers->credentials `https://next-auth.js.org/providers/credentials`

`Advice` : go with the social providers that will help us to deal with extra burden and security risk of managing the credentials.

Custom Credential Matching

- install package to encrypt our password : `npm i bcrypt`
- And as dev dependencies install : `npm i -D @types/bcrypt` so we can have suggestion while typing
- We have added `CredentaialsProvider` in `authOptions`.
- We have added optional password field in the user table
- Also we can customize the look and feel of the credential page we see.

Additional Reading

- We can replace autogenrated login and logout pages with our custom ones. https://next-auth.js.org/configuration/pages
- NextAuth.js provides a number of events (eg. signIn, signOut,createUser etc) : https://next-auth.js.org/configuration/events
- We can also provide handlers for these events as part of our NextAuth.js setup L https://next-auth.js.org/configuration/options#events

Exercises

- Configure another OAuth provider, such as GitHub or Twitter.
- Create a custom registeration form that captures user's name , email and password.Make sure these values are stored in the database.
- Create a change password page.Make sure it's only accessible to logged in users.

```
Sending Emails
```

- Use this libray : https://react.email/ which will help us to create , see email easily.
- npm i react-email @react-email/components
- Add this to .gitignore : `.react-email/` because while preview this will create many junk files which are use less and can be regenrated using `npm run preview-email`.
- Add this to package.json script section `"preview-email": "email dev -p 3030`
- after we have created the welcome template file in emails folder then run : above command and go to port 3030 to see result.
- We know two to style our component for the email , one with normal style other is tailwind
- Now we opt for service which will help to sent mail in our behalf. https://resend.com/ and create api key and paste in .env file with name `RESEND_API_KEY`
- Install this package : `npm i resend@1.0.0`
- Final step create-api end point for sending emails (it just for testing how to test it , in real world no api end point should be exposed to sent emails.)

- **Just see how to do with google or something , because it would require custom domain, see how we configured api in `app/api/send-email/route.tsx` and how we configured the template for sending emails on `./emails/WelcomeTemplate`**

```
Optimizations
```

- Optimizing Images
- Using third-party JS libraries
- Using custom fonts
- Search engine optimization
- Lazy loading

1. Images

- Put all the images in public folder
- Always use the image compoent that provided by next js , does following things , a. Compress the images according the screen size it going to server(which next-js detect automtaically) and it also offer verious options to style it.
- By default next-js uses lazy loading , means images won't be served from server until it going to server in viewport.if don't want that use `priorit` prop in the component.
- If you want to use image from url then there is some configuration please watch the next documentation on that.

2. Adding third party Libraries : eg. Google Analytics

- watch these steps : `https://nextjs.org/docs/messages/next-script-for-ga`
- **IMPORTANT : always put google analytics as top as possible so that it can used on entire app eg. first componet of body . app/layout.tsx**

3. Using Fonts

- We experiment with Roboto in file `/app/layout` what next js will is , it will download the font with all specified weight at once while we build our web app first time.Then server that one for each request.

- That is simple but what about the local fonts. it is also simple , go in public and put your font file there and import it and define the object of it as we defined for `Roboto` font we have used.
- How about the registering custom font using tailwind : see document.

4. Serach Engine Optimizations

- There when ever we export metadata object from layout or page file , next js automatically include that in head section.And search engine look for these meta tags for index our contents.So to make our website search engine friendly make sure that every page has proper meta-tags.
- Normal metadata object is straight forward to create , but what about those where person click on shoes and we are fetching information about that , for that use below code. Make sure don't change the name `generateMetadata`

```
export async function generateMetadata(): Promise<Metadata>{
const product = await fetch('...')
return {
        title: product.title,
        description: product.description,
         .....

}

}
```

5. Lazy Loading

- This is strategy of loading client components or third party libraries in the future when we need them typically as a result of user interaction (like : user clicks the button, or scroll beyond a certain points)

- Use cases a like , there is some compoenets have some rich text editor or having lot's of code.

- Let's simulate the loading of heavy compoent from click on a button.Also it does not make sense to load small component as lazy . Because page size would not get effected any way so only prefer for large pages.

```
import dynamic form "next/dynamic";
const HeavyComponent= dynamic(()=> import('./components/HeavyComponent'),
        {
```

```
          ssr: false, // to disable pre render on server ( uscase: we use
browser api which is not available on server so that will throw an eror which
is not good.)
          loading:() => <p>Loading...</p>}
)


.....

const [visible,setVisible] = useState(false);
<button onClick={()->setVisible(true)}> Show </button>
{isVisible && <HeavyComponent/>}



...
```

- we can also dynamically import the modules , so that it won't go with our page and only when user clicks the button or something then only.

```
// install lodash and npm i -D @type/lodash also
// this library provides utility funciton to work with our collections
<button onClick = {async ()=>{
        const _ = (await import('lodash)).default;
        const users = [ {name: 'c'} , {name: 'a'} ,{name: 'b'}  ]
        const sorted = _.orderBy(users,['name']);
        console.log(sorted)
}}
```

```
Deployment
```

- Before we deploy our application first we should build it locally , so we can detect any error ahead of time. `npm run build`
- We encounter error , we have exported authOptions from route file, which is not google route file should only export get , post , ... types only , Resolution : we puth auth options in another seprate file.
- And another eror : later we deleted product table from our scheme and but in our api we are using it to retrive the data , so for that it gave us error.
- Once build succes , then push everything to github.
- Then depoly on any of these paltforms (vercel ,aws, google cloud platform , Heroku and so on.)Out of all these vercel is fastest and safest way to deploy any next js applications.it's same company which build the next-js.

- Also every time we push our code to github vercel will download it and run build on it (good right.) and if there is any build error it will show up there too.But it always adviced first to build locally then to push, on main branch.
- We would also requre the mysql hosting for our database on cloud , we can use services provided by (Digital Ocean, Google Cloud Platform , MS Azure , Hostinger , GoDaddy)
- We also used cloudinar for uploading our upload feature , we used dev name , we have to use production name just go in there and see it.
- `advice` : always have different set of keys for developement and produciton environment , and always keep production keys to a different safe place , so that if anyone get access to our system won't be able to steal it.