Frontend:
Html, css (cascading style sheet)
Javascrypt para interacciones
Dom (document object model) -> allows JavaScript to change web page
En vez de Dom , usamos un JavaScript framework para reutilizar código

Frameworks típicos incluyen ReactJS, Angular y VueJS
A java le faltan cosas para permitir tener varios archivos ordenados, para eso se usa un bundler webpack, que nos deja separar el código en distintos files
También se usa Transpiler Typescript qiue permite hacer enhance a javascript y luego devolverlo a javascript.

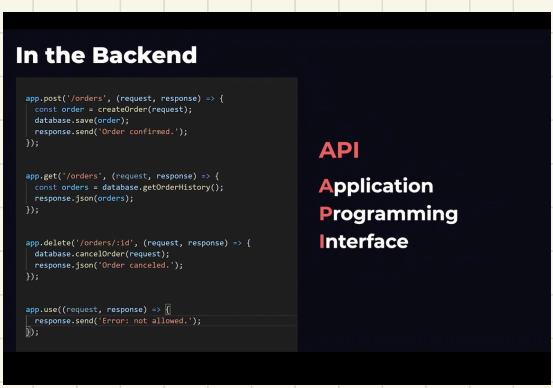
Css tiene problemas parecidos así que se usan bundler+transpiler =preprocessor, que sirve para lo mismo
css framework -> bootstrap es el más típico,
Comunicación frontend, backend -> HTTP
Usando XMLHTTP Request, backend escucha al url.

HTML in 100 seconds:
HyperTextMarkupLanguage define la estructura de del contendio en un página web. a -> anchor to a different website.
Típico es hi
ejemplos:

learn to code

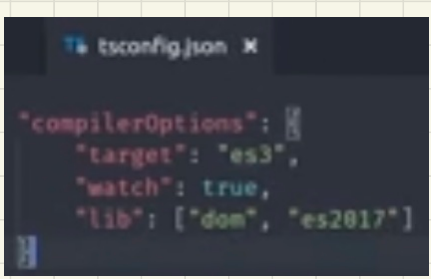
css, order matters
p { font-size}

backend programming
frameworks:
expressJS
Python Django
Ruby on rails
Java spring
package manager according to languafe
npm, pip (python), bundler (ruby).
request-response cycle.



Microservices -> divide la pega entre distintos backend

Typescript
Angular app
Para más info ver el opensource libro "TypeScript Deep Dive"
Para instalarlo hay que poner npm i -g typescript
tsc —version
index.ts -> compiler to pass it to javascript
tsc index.ts crea un archivo de javascript
doesn't support async function, so you create a touch tsconfig.json
en el touch tsconfig.json
luego corres tsc y es compatible

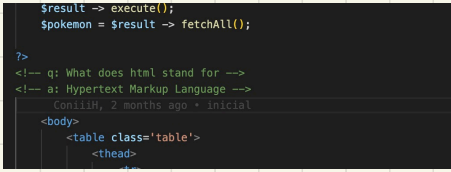


para no tener que correrlo siempre librerías

dom incluye url

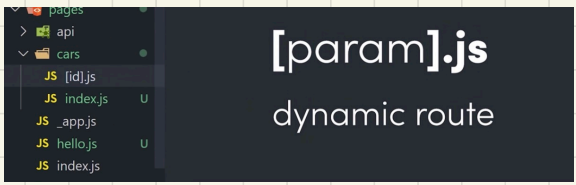
lodash library -> npm i -D @types/lodash para autocomplete
strong typing -> asignar tipos a las variables ya sea en simples, funciones, arrays
question marks si son opcionales

Github -> history book, opensource.
git init -> nuevo repositorio



Tailwind -> nombres de clases hacen que css sea difícil de mantener, bootstrap ayuda
Es más un approach práctico y de funcionalidad. Por ejemplo flex hace que sea dlex, p es padding entre otras cosas,
Hay condicionales, dark mode, code duplication es otro problema. No hay que memorizar tantos por vs-code autocomplete.

Next.js -> fast. reactive with no configuration renderiza el contenido en avance, best of both worlds. La estructura de las carpetas de tus archivos mirrors los url.



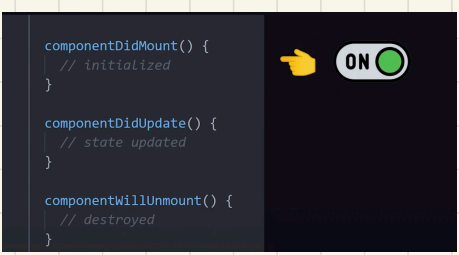
npx create-next-app <name>
package json, hay que hacer npv run dev

Data fetching es lo más valioso de next. Depende de tus necesidades si es static (por ejemplo un blog v/s ebay que requiere constante actualización).

Para que pueda renderizar todos los car ids, se hace la función getStaticPaths que se los da antes para que pueda renderizar.

React Hooks: https://es.reactjs.org/docs/hooks-intro.html

Para hacer cosas dinámicas. Se puede usar clases pero hooks es mejor al ser más ergonómicos al permitir reusar lógica sin cambiar la jerarquía.
useSuperPower() -> solo funcionan al principio y no en loops ni condicionales, excepto los que son custom . useHook por ejemplo
useState(inicial state) es el más común y útil, handle reactive data, inicial state dentro. Se cambiar con la setter function.
useEffect() implementa la lógica de las tres cosas de abajo. correra cada vez que pase alguno de los eventos de abajo creo. Para no hacer un loop hacemos dependencies



useContext() -> happy mood v/s sad mood para ciertas partes de la página.
useRef -> cuando cambia algo o fetchear un elemento del dom (ns que es eso)
useReducer() -> en vez de cambiar algo directamente como usestate, lo hace más indirectamente.
useMemo -> optimize performance but not inmediatly.
useCallback() -> prevent unnecesary re-renders
useImperativeHandle
useLayoutEffect -> va a esperar que termine el código antes de actualizar la UI
useDebugValue -> para tus own hooks, hace posible definir propias labels

ReactQuery -> library fro fetching data, infinite scroll o actualizar

Backend:
Prisma:
SQL no se usa tanto en al modernidad al perder control, objetc relational mapping, java classes.
SCHEMA para relations -> prisma lo cambia a definiciones, ve las migraciones.
npx prisma
Se define la schema, para modificarlo se cambia las tablas con migrate.

GraphQL

REST API
Single entry point
Hacemos un schema
También se puede mutar la tabla

WEB: conceptos generales

- Frontend:
  - Explicación general: https://www.youtube.com/watch?v=WG5ikvJ2TKA
  - HTML: https://www.youtube.com/watch?v=ok-plXXHlWw
  - CSS: https://www.youtube.com/watch?v=OEV8gMkCHXQ
- Backend: https://www.youtube.com/watch?v=XBu54nfzxAQ
- TypeScript: https://www.youtube.com/watch?v=ahCwqrYpIuM
- Github: https://www.youtube.com/watch?v=HkdAHXoRtos

CLEARN: Nuestro frontend

- Tailwind: https://www.youtube.com/watch?v=mr15Xzb1Ook
- Next.js: https://www.youtube.com/watch?v=Sklc_fQBmcs
- React Hooks:
  - Explicación general: https://www.youtube.com/watch?v=TNhaISOUy6Q
  - Documentación: https://es.reactjs.org/docs/hooks-intro.html
- React Query: https://www.youtube.com/watch?v=novnyCaa7To

SHAFT: Nuestro backend

-
- Prisma:
  - Explicación general: https://www.youtube.com/watch?v=rLRIB6AF2Dg
  - Documentación: https://www.prisma.io/docs/concepts/overview/what-is-prisma
-
  - Documentación de lo que más usamos: https://www.prisma.io/docs/reference/api-reference/prisma-client-reference
  -
- Grapqhl: https://www.youtube.com/watch?v=eIQh02xuVw4