Estructura

Objectiu tema

Comprensió de l'estructura de projectes: adquirir coneixements sobre com estructurar correctament un projecte, incloent la descàrrega, configuració i organització dels components essencials.

Configuració inicial

Al tema anterior hem descarregat el projecte en format zip. Encara que l'habitual, en un entorn laboral, és usar un gestor de versions com pot ser git. En tal cas hauríem de fer un fork i clonar el projecte.

Com es va comentar el README inclou les instruccións per a la configuració inicial i la seva execució. En concret soó:

1. **.gitlab-ci.example.yml -> .gitlab-ci.yml**: Es pot obviar en aquest curs
2. **src/config/***: Els fitxers dins aquesta carpeta contenen la configuració pròpia del projecte. Com a configuració inicial, cal tocar el index.ts i canviar el nom i codi del projecte.
3. **package.json -> name**: Únicament s'ha de canviar el codi del projecte
4. **README.md**: Aquest fitxer és el primer que es veu en els gestors de versions, cal que es personalitzi per a que una persona nova pugui comprendre de que va el projecte i les instruccions per executar-ho.
5. **vite.config.ts -> base**: S'ha de configurar el codi del projecte que repercutirà en el path de la url.

Estructura de carpetes

Aquest esquelet té predefinida una estructura de carpetes que s'ha de respectar per a mantenir un ordre. Partirem del contingut que es troba dins la carpeta `src`, ja que tot el que hi ha fora és configuració base.

- assets
  - img
  - scss
- config
- core
- locales
- models
- modules
  - shared
  - modul[1-n]
    - components
    - composables
    - routes
    - views
- store

Assets

Dins la carpeta assets podem pujar imatges del nostre projecte i afegir o completar els css, si és necessari.

La carpeta scss ja disposa d'una sèrie de fitxers, per un costat temin la configuració de quasar que va dins el fitxer `quasar-variables` i personalitza la visualització d'aquest framework.

Per altra banda, tenim l'`index.scss` on s'importen la resta de fitxers. El `base` sobreescriu una sèrie de propietats per unificar la visualització als distints navegadors. El `imas` té una sèrie d'utilitats pròpies de l'imas i el `utils` diverses utilitats.

En el cas que el nostre projecte necessiti personalitzar el css podem afegir regles dins el fitxer adequat, o en el seu cas, crear un nou fitxer i importar-ho dins l'índex. Al igual que la resta de codi és important separar i estructurar el codi en fitxers que representin el seu contingut.

Si algú vol aprofundir en la creació de css de forma estructurada es recomana la guia css { guide: lines; }

config

index.ts: conté la configuració base de l'aplicació, a l'iniciar una nova aplicació haurem d'ajustar aquesta configuració al nostre projecte.

menu.ts: Els elements d'aquest arxiu defineixen el menú autogenerat de l'aplicació.

```
{
  title: tc('inici'), // títol del menú
  to: { name: 'Home' }, // View que obrirà
  icon: 'fas fa-home', // Icona
  children: [...] // En cas de tenir subnivells. Si s'emplena aquest no
emplenar el to
  active: () => {...} // Si volem mostrar un element en funció d'una
condició. Per exemple, rols distints.
},
```

routes.ts: Aquí definírem les rutes que disposarà la nostra aplicació. Com veurem més endavant, als mòduls es definiran les rutes pròpies que s'hauran d'importar aquí:

```
{
  path: '',
  component: Home,
  name: 'Home',
},
...demoRoutes,
```

core

No s'hauria de tocar res d'aquesta carpeta, conté la configuració que arranca el projecte. En cas d'un error, comentar-ho abans de modificar res.

locales

Un fitxer json per a cada idioma disponible a l'aplicació. L'entrada `errors` tradueix automàticament els codis d'errors retornats del back, per exemple l'entrada `errors.imas_no_permisos` traduirà l'error imas_no_permisos retornat pel back. Intentar estructurar mínimament les entrades, per exemple, per entitats o mòduls.

> Cas pràctic

Suposem que tenim una aplicació amb expedients, podem donar un codi a aquesta entitat i definir les seves propietats a dins, de forma estructurada.

```
{
  ...
  "exp": {
    "expedient": "Expedient | Expedients", // pluralització
    "lexpedient": "L'expedient | Els expedients", // pluralització
    "numero": "Número",
    "interessats": "Interessats",
    "data": "Data d'alta"
    ...
  }
}
```

Apart de la pluralització també es poden fer altres coses com format de números o referenciar a altres traduccions. Podeu ampliar la informació a Vue i18n

models

La carpeta models conté els models de l'aplicació en format Pinia ORM i un index.ts amb el llistat de repositoris

```
// Exemple de model
export enum Especialitat {
  ...
}

export class Familia extends Model {
  static entity = 'Familia'

  @Attr(null) declare id: number
  @Str('') declare nom_familia: string
  @Num(null) declare idMunicipi: number

  @Attr(null) declare especialitat: Especialitat

  @Cast(() => DateCast)
  @Attr(null)
  declare created_at: Date

  @Cast(() => BooleanCast)
  @Bool(false)
```

```
  declare actiu: boolean

  @BelongsTo(() => Municipi, 'idMunicipi') declare municipi: Municipi

  @HasMany(() => Expedient, 'idFamilia') declare expedients: Expedient[]

  // Mètode per a crear una nova instancia detached
  static from(from: Familia | null): Familia {
    const to = new Familia()

    if (from) {
      Object.assign(to, {...from})
    }

    return to
  }
```

El index.ts simplement llistarà els repositoris disponibles per a no haver d'instanciar-ho cada vegada

```
export const FamiliaRepo = useRepo(Familia)
....
```

S'ampliarà la informació en el corresponent tema.

modules

Carpeta més important del projecte on s'estructurarà el contingut de la nostra aplicació. Per a cada part diferenciada de la nostra aplicació es crearà un mòdul amb un nom identificatiu, per exemple, l'aplicació d'adopcions es separa en els mòduls de "familia", "expedient" i "menor". Addicionalment disposarem d'un mòdul shared per a contingut genèric i comú a tota l'aplicació.

Cada mòdul s'estructurarà en vàries carpetes:

- **components**: Els components són fragments visuals reutilitzable.
- **composables**: Fragments de lògica o codi no visual reutilitzable. (Es detallarà en un altre apartat)
- **routes**: Rutes pròpies del mòdul que s'han d'importar (com a conjunt) dins les rutes de configuració.
- **views**: Vistes o pàgines del mòdul. No són reutilizables i es corresponen 1 a 1 amb les rutes.

La diferència entre un component i una vista és lògica. Mentre que una vista té una correspondència amb una ruta i no es crida com a component fill, els components es reutilitzen i no s'hi accedeix directament des d'una ruta.

store

Com el seu nom indica, dins aquesta carpeta, es posaran les stores que siguin necessàries (es detallarà en un altre tema).