Checklist pour démarrer un projet

1. Je crée un nouveau dossier pour mon projet

2. Est-ce que je dois créer un backend ?

    Non. Aller à l'Étape 3.

    Oui :
    - Je crée un dossier « backend »
    - Je rentre dans le dossier « backend » avec le terminal `cd backend`
    - Je génère le backend avec Express `express --no-view --git ./`
    - J'initialise mon dépôt local `git init`
    - J'ajoute les fichiers à mon prochain commit `git add .`
    - Je crées mon commit initial `git commit -m «initial commit»`
    - Je commence le travail :
        - J'ajoute CORS à mon projet
            - J'installe cors `yarn add cors`
            - J'ajoute CORS dans le fichier app.js
                - const cors = require('cors')
                - app.use(cors())
        - Ai-je besoin d'une connexion à la DB ?
            - Non, je continue.
            - Oui,
                - j'installe mongoose `yarn add mongoose`
                - Je crées un dossier « models »
                - Je crée un fichier ./models/connection.js
                    - J'importe mongoose const mongoose = require('mongoose')
                    - Je crées une variable CONNECTION_STRING
                    - Je crées ma connexion (mongoose.connect(…,{…}).then(…).catch(…))
                    - J'importe ma connexion dans app.js
                - Je crée mes Schémas/Models
                    - Je crée un fichier qui portera le nom de ma collection.js
                    - J'importe mongoose
                    - Je définis mon schéma avec mongoose.Schema({})
                    - Je crées mon modèle qui liera ma collection à mon schéma (ex: mongoose.model('users', userSchema))
                    - J'exporte mon modèle (module.exports = User)
        - Je crée mes routes
            - Dans le dossier « routes » je crée un fichier qui aura le nom de ma famille routes.js
                - J'importe express const express = require('express')
                - Je crée mon router const router = express.router()
                - J'exporte mon router module.exports = router
                - Si mes routes doivent faire appel à la DB, j'importe les modèles nécessaires ex: const User = require('../models/users)
                - Dans le code avant mon export, je crée mes routes
                    - Si c'est pour récupérer de l'information, je crée une route GET router.get('/xxx', (req, res) => {}
                    - Si c'est pour envoyer de l'information, je crée une route POST router.post('/xxx', (req, res) => {}
                    - Si c'est pour modifier de l'information, je crée une route PUT router.put('/xxx', (req, res) => {}
                    - Si c'est pour supprimer de l'information, je crée une route DELETE router.delete('/xxx', (req, res) => {}
                    - Je m'assure que toutes mes routes renvoient bien une information en JSON res.json(MON_OBJET_JSON)
            - Dans mon fichier app.js, je crée les préfixes de mes familles de routes

- J’importe mon router const monRouter = require (‘../routes/monrouter.js’)
- Je crée le préfixe app.use(‘/famille’, monRouter)

- Je continue à l’étape 3

3. Est-ce que je dois créer un frontend ?

Non

- Je n’ai pas créé de backend. Aller à l’étape 8.
- J’ai créé un backend. Aller à l’étape 4.

Oui, en HTML/CSS/JS « classique ». Aller à l’étape 5.

Oui, en React. Aller à l’étape 6.

Oui, en React Native. Aller à l’étape 7.

4. Ai-je déployé mon backend sur vercel ?

Non. Je fais tourner mon backend avec `yarn start` et j’utilise l’URL locale dans mon frontend (http://localhost:3000)

Oui. Je récupère l’URL de mon déploiement pour l’utiliser dans mon frontend.

5. Je crée un dossier « frontend » à côté de mon dossier « backend »

- Je crée un fichier index.html (la racine de mon site)
- Je crée un fichier script.js que j’importe dans mon index.html à la fin de mon <body>
- Je crée un fichier styles.css que j’importe dans le <head> de mon index.html
- Si je dois créer plusieurs pages, je crée autant de fichiers .html, .js et .css que nécessaire.

6. Je me positionne à la racine de mon projet

- Je crée mon projet Next avec la commande `yarn create next-app`
- Je développe mes composants et mes pages
- Ai-je besoin de transférer de la donnée entre des composants non-reliés ?
  - Non, je n'ai pas besoin d'installer Redux
  - Oui, j'installe Redux
    - J'importe les modules nécessaires `yarn add react-redux @reduxjs/toolkit`
    - Je configure mon store et mon provider dans pages/_app.js
      - Ex :

```
import { Provider } from 'react-redux';
import { configureStore } from '@reduxjs/toolkit';

const store = configureStore({
 reducer: {},
});

function App({ Component, pageProps }) {
 return (
   <Provider store={store}>
     <Component {...pageProps} />
   </Provider>
 );

export default App;
```

    - Je crée un dossier reducers pour héberger mes slices
    - Je crée mes slices sans oublier d'exporter mes actions et ma slice
      - Ex :

```
import { createSlice } from '@reduxjs/toolkit';

const initialState = {
 value: [],
};

export const friendsSlice = createSlice({
 name: 'friends',
 initialState,
 reducers: {
   addFriendToStore: (state, action) => {
     state.value.push(action.payload);
   },
 },
});

export const { addFriendToStore } = friendsSlice.actions;
export default friendsSlice.reducer;
```

    - J'importe mes slices dans mon point d'entrée (pages/_app.js)
    - Ai-je besoin de faire persister mes données stockées dans mon store ?
      - Non, je ne change rien
      - Oui, j'installe les modules de persistance `yarn add redux-persist`
        - J'importe Storage : import storage from 'redux-persist/lib/storage';
        - Je modifie mon store et je mets en place mon Persistgate
- Je lance mon application en utilisant la commande `yarn dev`

7. Je me positionne à la racine de mon projet

- Je crée mon projet Expo avec la commande `yarn create expo-app`
- Je développe mes composants et mes pages
- Ai-je besoin de transférer de la donnée entre des composants non-reliés ?
  - Non, je n'ai pas besoin d'installer Redux
  - Oui, j'installe Redux
    - J'importe les modules nécessaires `yarn add react-redux @reduxjs/toolkit`
    - Je configure mon store et mon provider dans App.js
      - Ex :

```
import { Provider } from 'react-redux';
import { configureStore } from '@reduxjs/toolkit';

const store = configureStore({
 reducer: {},
});

export default function App() {
 return (
   <Provider store={store}>
     …
   </Provider>
 );
```

    - Je crée un dossier reducers pour héberger mes slices
    - Je crée mes slices sans oublier d'exporter mes actions et ma slice
      - Ex :

```
import { createSlice } from '@reduxjs/toolkit';

const initialState = {
 value: [],
};

export const friendsSlice = createSlice({
 name: 'friends',
 initialState,
 reducers: {
   addFriendToStore: (state, action) => {
     state.value.push(action.payload);
   },
 },
});

export const { addFriendToStore } = friendsSlice.actions;
export default friendsSlice.reducer;
```

    - J'importe mes slices dans mon point d'entrée (pages/_app.js)
    - Ai-je besoin de faire persister mes données stockées dans mon store ?
      - Non, je ne change rien
      - Oui, j'installe les modules de persistance `yarn add redux-persist @react-native-async-storage/async-storage`
        - J'importe AsyncStorage : import AsyncStorage from '@react-native-async-storage/async-storage';
        - Je modifie mon store et je mets en place mon Persistgate
- Je lance mon application en utilisant la commande `yarn start`

8. Pas de frontend, pas de backend, que se passe-t-il ici ?
    - Retourne à l’étape 1 ou ferme VS Code et repose-toi, tu en as besoin.