- **Back:**
  - Rutas:
    - Definir el prefix para el sistema al que se refiere {prefix sistema} en el RouteServiceProvider
    - Agrupadas por controlador {pantalla/módulo}
    - Función a la que apunta {nombre_acción}
    - Función del controlador lowerCamelCase
    - Nombre de la ruta dot.notation
    - Auth singular
    - Modulos plural
  - Modelos: Nombre de la tabla en singular. Utilizar ide-helper para la documentación de los modelos.
  - Controladores: Nombre de la tabla singular se le agrega Controller al final UserController
  - Servicios: Seguir las convenciones de los controladores pero colocando los archivos en la carpeta Http/Services/ {NombreDelServicioService} StripeService, UploadFilesService
  - Variables de entorno:
    - Todo en mayúsculas con SNAKE_CASE: MAIL_USER, MAIL_DRIVER,
    - uso de variables de entorno para react STRIPE_KEY:MIX_STRIPE_PUBLIC_KEY="${STRIPE_PUBLIC_KEY}"
  - Mailable: Usar la plantilla creada para correos. Crear las clases de css y evitar el uso de bootstrap, tener en cuenta el maquetado en formato de tabla
  - Tablas: plural users
  - Nombre pivote: orden alfabético singular nombre de las dos tablas role_user
  - Nombre de relaciones simples: tabla se agrega _id tabla_id
  - Nombre de funciones generales: lowerCamelCase findServices
  - Nombre de variables: lowerCamelCase $pendingPayments

- **Front**:
    - **Vue**

      Estructura: Pages y components; en pages van las páginas que son referentes a index o tablas de contenido, dentro del componente la recomendación es solo hacer llamadas a componentes, y en componentes todo lo referente a componentes que no necesitan una vista completa, por ejemplo, formularios, modales de información.

      Nombre de index: PascalCase, NombreMóduloIndex (UsersIndex, ClientsForm)
      Agregar el componenteHome:  El componente home contiene todo lo que se muestra en la vista principal, para que el componente Index almacene solamente las vistas y nada de templates ni métodos.

      Nombre de componentes: PascalCase, NombreMóduloFunción (UsersIndex, UserForm). Tiende al singular, pero hay que identificar los casos donde se muestre información para más de un registro.

      Orden de las opciones para los componentes de Vue:

      1. Nombre (UsersIndex)
      2. Components
      3. Mixin
      4. Props
      5. Data
      6. Computed
      7. Filters
      8. BeforeCreated
      9. Created
      10. BeforeMount
      11. Mounted
      12. BeforeUpdate
      13. Update
      14. Activated
      15. Deactivated
      16. beforeDestroy
      17. destroyed
      18. errorCaptured
      19. Watch
      20. Methods

      Nombre de los datos, variables, props… : camelCase
      Nombre de las funciones: camelCase

Nota: Empezar a utilizar TypeScript

Limitar el uso de los style scooped en los componentes de Vue, utilizar preferentemente los archivos .less.

Para VueX, vue Router, vuetify: Las carpetas para cada instancia se colocan a nivel del apartado del sistema de js (js/admin , js/web)

En el vue Router la constante “prefix” será ‘/admin’

Nota: Investigar la creación de un solo archivo para importar todos los componentes al vue Router.

Dentro de las rutas para el vue router, en el arreglo routes:

- path: `${prefix}/nombre-módulo
- name: kebab-case, nombre-módulo-(submódulo)-index (users-index, users-schoolarSchedule-index)
- redirect: String, location, function

```
redirect: to => {
      // the function receives the target route as the argument
      // we return a redirect path/location here.
      return { path: '/search', query: { q: to.params.searchText } }
    },


redirect: to => {
      // the function receives the target route as the argument
      // a relative location doesn't start with `/`
      // or { path: 'profile'}
      return 'profile'
    },
```

- props: booleano, objeto, función
- children: Puede recibir otro objeto de ruta
- beforeEnter: función
- Meta: recibe un objeto con las propiedades de data como un componente de vue
- caseSensitive: booleano
- pathToRegexOptions: Compilar páginas dependiendo de la expresión regular

- An alias of / as /home means when the user visits /home, the URL remains /home, but it will be matched as if the user is visiting /.
  `const routes = [{ path: '/', component: Homepage, alias: '/home' }]`

Nota: Consultar los siguientes métodos para el vue router:

```
router.beforeEach((to, from, next) => {
  /* must call `next` */
})

router.beforeResolve((to, from, next) => {
  /* must call `next` */
})

router.afterEach((to, from) => {})
```

Si usamos vuetify limitar el uso de <router-link>, utilizar la propiedad “to:” de vuetify

- **Blade**
  Folder Sistema: nombre en singular y lowercase (admin, client).
  - Folder template:
    - Archivos global_css: Estos deberán de llevar como cabecera el nombre y versión (si existe más de una versión) de los css que estemos usando.
      1. Link de los css externos
      2. Link de nuestros css
    - Archivo global_js: Estos deberán de llevar como cabecera el nombre y versión (si existe más de una versión) de los js que estemos usando.
      1. Script de los js externos
      2. Script de nuestros js

- Archivo main: Vista inicial para todo el sistema incorporado.
    - Head:
        - Incorporación de la etiqueta title.
        - Incorporación de nuestros estilos (global_css) con @include.
        - Incorporación de @stack('css') para agregar css de otras vistas.
    - Body:
        - Incorporación de @yield('content') (en caso de usar otras vistas en blade) o  (en caso de utilizar componentes de vue)
        - Incorporación de @routes (si usamos vue).
        - Incorporación de nuestros js (global_js) con @include.
        - Incorporación de @stack('scripts') para agregar js de otras vistas.
- Folder Sección: lowerCamelCase (inventoryMovements).
    - Archivos blade de vistas: nombre en singular y lowerCamelCase (index, upsertInventory).
    - Archivos de contenido de formularios o dinámicos de blade: inicia con guión bajo (_) continúa con nombre en singular y lowercase (_form, _upsertform).
- Agregar en los comentarios de la vista las variables que se reciben del servidor en caso de ocuparlas, para tener listadas todas las variables que se usan en la vista.
- Evitar hacer llamadas al modelo desde la vista, toda información requerida deberá de ser mandada desde el controlador.

## ● React

Estructura: Pages y components; en pages van las páginas que son referentes a index o tablas de contenido, y en componentes todo lo referente a componentes que no necesitan una vista completa, por ejemplo, formularios, modales de información.

Nombre de index: PascalCase, NombreMóduloIndex (UsersIndex, ClientsForm)

Nombre de componentes: PascalCase, NombreMóduloFunción (UsersIndex, UserForm).
Tiende al singular, pero hay que identificar los casos donde se muestre información para más de un registro.
Orden de las opciones para los componentes de React:

1. Importación de componentes
2. Nombre de la función y componente (UsersIndex)
3. Props / parámetros de la función principal
4. useState, variables de estado para el componente
5. Hooks personalizados, llamada a los hooks que se creen para el componente específico

6. useEffect carga de datos provenientes de API
7. Métodos, funciones que ejecutan alguna tarea dentro del componente
8. Return: HTML que contendrá el cuerpo DOM virtual de nuestra página o componente.
9. PropTypes, definición de los nombres y tipo de datos que deberán contener las props, así como su uso obligado u opcional

- Nombre de los datos, variables, props… : lowerCamelCase
- Nombre de las funciones: lowerCamalCase
- Nombre de las funciones provenientes de un evento: lowerCamelCase empezando con handle: handleClickSubmit
- Nombre de los módulos de estilo scss:
    - Nombre de archivo: CamelCase finalizando con Styles seguido de las extensiones .module.scss: ClientFormStyles.module.scss.
    - Nombre de las clases: loweCamelCase, los nombres de las clases de estilo se nombran como a las propiedades de un objeto ya que así son tratadas.

Nota: Empezar a utilizar TypeScript
Utilizar preferentemente los archivos .module.scss

React Router
<Routes> Componente HOC a ser colocado que englobará las rutas a utilizar.
<Route> Componente el cual renderizará el componente deseado dependiendo la ruta que se le coloca:
path: ruta en plural que tendrá que coincidir con la ruta del navegador: '/clients'
component: <Componente> que será renderizado al coincidir la ruta
Cambio si este componente es un template:
index se le coloca la palabra index para indicarle que el componente a renderizar sobre la ruta es el señalado en component

La navegación se implementa con to al igual que Vue para poder navegar entre los componentes.

# Anexos:

- Ejemplo de creación de rutas:

```
Route::prefix('auth/')->name('auth.')
      ->group(function () {
            Route::post('/register', [LoginController::class,'register'])
                ->name('register');

            Route::get('/logout', [LoginController::class,'logout'])
                ->name('logout');
      });

Route::prefix('users/')->name('users.')
      ->group(function () {
            Route::post('index', [UserController::class,'indexContent'])
                ->name('index_content');

            Route::get('upsert', [UserController::class,'upsert'])
                ->name('upsert');
      });
```

```
Route::prefix( prefix: 'truck/')
    ->name('truck.')
    ->group(function () {

        Route::get( uri: 'index',
            [TruckController::class, 'indexTrucks'])
            ->name('index_content');

        Route::post( uri: 'create',
            [TruckController::class, 'indexTrucks'])
            ->name('create');

    });

route('admin.truck.index_content');
```