CARDIOAI

DOCUMENTAZIONE

KOSTANTINO ABATE

SOMMARIO

DIPENDENZE E DOCUMENTAZIONI

Laravel 12

Il framework centrale è Laravel 12. Lo stack per il funzionamento del frontend è Tailwind + Vite + Sass. Per quanto riguarda Sass è utilizzato esclusivamente per facilitare alcuni override del css.

Frontend

La libreria frontend in uso è Preline.

Autenticazione

Per l'autenticazione la libreria in uso è Fortify. Si tratta di una libreria front-end agnostic, quindi priva di stili.

Spatie Permissions

Per gestire i ruoli (guest e operator) è in uso Spatie Permission. Si tratta di una libreria che permette di aggiungere ruoli e permessi a modelli tramite Eloquen ORM.

Deployment su Aruba

Per il deployment su Aruba la guida per la soluzione della maggior parte dei problemi e degli errori è questa.

Altre tecnologie

Sono in uso anche:

- jQuery
- Vanilla Calendar Pro

SETUP

Per lavorare sul progetto è necessario eseguire alcuni passaggi di setup.

COMPOSER E NODE

Se non è presente sul sistema è necessario scaricare Node.js ed installarlo globalmente.

Dopodichè, da terminal, navigare fino alla root del progetto (solitamente `cd mioprogetto`)

Per installare tutte le dipendenze composer e node occorre lanciare i seguenti comandi dal terminale:

```
composer update
npm install
```

.ENV

Se non è presente il file `.env`, utilizzare i modelli `.env.dev` per l'ambiente di sviluppo e `.env.production` per l'ambiente di produzione.

I file `.env` possono essere criptati/decriptati. Per farlo seguire la documentazione di Laravel.

MIGRAZIONI

Per eseguire il setup del database è necessario modificare il file `.env` con le credenziali di accesso al database di sviluppo. Dopodiché basterà lanciare il seguente comando:

```
php artisan migrate:fresh --seed
```

L'opzione `--seed` serve per lanciare i **Seeder** e popolare il database in base alle impostazioni definite nei file **config**.

SYNC E RESET DEI CONFIG

In fase di setup, ma anche in altri momenti dello sviluppo potrebbe essere necessario azzerare la cache del progetto con il comando

```
php artisan optimize:clear
```

Oppure ricostruirla con il comando

```
php artisan optimize
```

START

Per lanciare il progetto è necessario eseguire in due prompt diversi del terminale i seguenti comandi:

```
npm dun dev
```

e

```
php artisan serve
```

Verrà servito un URL che punta al server localhost dove è possibile visualizzare il sito (aggiornato in tempo reale con Vite).

Se si usa Laragon o altri provider/emulatori localhost seguire la documentazione dei rispettivi. Ad esempio per quanto riguarda Laragon non è necessario eseguire il comando `serve`, visto che Laragon esegue autonomamente il deployment su un record DNS fittizio. Basterà quindi visitare `http://{nome dell'app}.test`.

CONFIG

Oltre ai file config stock di Laravel 12, sono presenti file config custom che gestiscono vari aspetti dell'applicazione.

CAI-AUTH.PHP

Questo file config raccoglie alcuni dati statici utili per la fase di registrazione degli utenti, come: l'associazione dei prefissi (dott./dott.ssa) al genere dell'utente, oppure la lista delle professioni e la lista delle specializzazioni sanitarie.

CAI-CODEX.PHP

Questo file config raccoglie alcuni dati statici utili per il rendering in formato leggibile (label9 di alcuni valori provenienti dai **Casts**.

CAI-BLUEPRINTS.PHP

Questo file config raccoglie l'indice delle sezioni dello screening/prima-visita per elaborare il rendering. Il formato generale è il seguente:

```
`[page][index][code]`
```

- `[page]` (int) è il numero della pagina in cui renderizzare il contenuto.
- `[index]` (int) è l'ordine di renderizzazione delle sezioni/accordion.
- `[code]` (string) è il codice del contenuto. È necessario per il filtraggio nella vista per selezionare il contenuto da renderizzare.

CAI-ITEMS/...

Questa cartella contiene tutti i config delle singole sezioni/accordion, uno per ogni `[code]` in uso. Tutti i config contenuti in questa cartella sono registrati e gestiti tramite alias nel provider `app/Providers/AppServiceProvider.php`.

La logica generale è la seguente, tenendo presente che gli unici array obbligatori sono `general[]`, e `items[]`:

```
'general' => [
        'system' => //Riporta il sistema di cast dei dati
        'title' => //Titolo della sezione
        'code' => //Codice della sezione
        'hasGate' => //Presenta o meno un input iniziale in cui si chiede se approfondire o meno (quindi aprire o meno tutti i sottoelementi)
```

```
        'statement' => //Allega una frase all'input gate
        'subStatement' => //Allega una frase dopo l'apertura dei
sottoelementi (se hasGate è false viene mostrato di default, quindi usare
questo per impostare la frase principale se hasGate è false),
        'short' => //Codice della sezione in formato Short
        'hasAI' => //Aggiunge un Alert per indicare che nella sezione sono
presenti input o campi che vengono gestiti autonomamente dall'applicazione
(ad esempio calcolo automatico di dati condizionali con JavaScript)
    ],
    //Accordions genera un accordion interno all'accordion di sezione
    'accordions' => [
        '{titolo accordion}' => [
            'label' => //Label dell'accordion
            'subLabel' => //Testo aggiunto dell'accordion
        ],
    ],
    //Groups genera una label tipo fieldset attorno agli items contenuti
al suo interno
    'groups' => [
        '{titolo del gruppo}' => [
            //Se esistono accordions
            'accordion' => '{titolo accordion}', //serve per collegare il
gruppo all'accordion
            //Altrimenti
            'label' => //Label del gruppo
            'short' => //Label Short del gruppo
        ],
    ],
    //Mains genera un'ulteriore livello di divisione indipendente da
groups
    'mains' => [
        '{titolo del main}' => [
            //Se esistono accordions
            'accordion' => '{titolo accordion}', //serve per collegare il
main all'accordion
            //Altrimenti
            'label' => //Label del gruppo
            'input' => //Input del main
            'short' => //Label Short del gruppo
        ],
    ],
    //Bypasses genera un input di tipo item, senza però necessariamente
aprire tutta la lista degli elementi, si tratta quindi di un item
indipendente
    'bypasses' => [
        '{titolo del bypass}' => [
            //Se esistono accordions
            'accordion' => '{titolo accordion}', //serve per collegare il
bypass all'accordion
```

```
            //Altrimenti
            'label' => //Label del bypass
            'input' => //Input del bypass
            'short' => //Label Short del bypass
            //Se prevede un array di valori (select)
            'values' => [
                [
                    'value' => //Attributo value del valore
                    'label' => //Label del valore
                ],
                ...
            ]
        ],
    ],
    //Items genera un input di tipo item allegato ai sottoelementi. Se
sono presenti gruppi, ogni item sarà vincolato al gruppo
    'items' => [
        '{titolo item}' => [
            //Se esistono groups
            'group' => '{titolo group}', //serve per collegare l'item al
group
            //Altrimenti
            //Se necessita di maggiori dettagli sulla natura dell'input
            'scope' => [
                'tag' => //tag dell'input (es: input o textarea)
                'type' => //type dell'input (es: checkbox o email)
                'unit' => //Se l'input ha un'unità di misura si indica qui
(es: %)
            ],
            'hidden' => //(boolean) al boot della pagina l'elemento è
mostrato o meno (solitamente hidden viene messo true su elementi che
devono essere mostrati in maniera condizionale es: se l'item ha in values
un valore open)
            'isSub' => //(boolean) è o non è un sottoelemento
            'type' => //è un tipo specifico di input (es: select)
            'input' => //Attributo name dell'input
            'label' => //Label dell'input
            'subLabel' => //Label secondaria dell'input
            'short' => //Label Short dell'input
            'sub' => //(string / array) sottoelemento o lista di
sottoelementi
            'needsIdentifier' => //valore da inserire in specifici
attributi, solo se richiesto nel codice
            //Se prevede un array di valori (select)
            'values' => [
                [
                    'value' => //Attributo value del valore
                    'label' => //Label del valore
```

```
                    'open' => //Se la selezione di questo valore prevede
l'apertura condizionale di un altro item si indica qui il nome o una lista
di nomi degli items
                ],
                ...
            ],
            //Se prevede una validazione tramite lo script
'resources/js/libs/validator.js'
            'validator' => [
                'case' => //Endpoint della validazione (es: codice
fiscale)
                'error' => //input del div con il testo dell'errore da
mostrare (es: '#input-fat_sub_1-error')
            ],
        ],
    ],
```

AUTENTICAZIONE

Per l'autenticazione è in uso Fortify, una libreria front-end agnostic che fornisce tutta una serie di classi e metodi già predisposti per la registrazione, l'autenticazione e la gestione degli users.

REGISTRAZIONE

La registrazione viene gestita in base alle indicazioni contenute in `app/Actions/Fortify/CreateNewUser.php`.

Questo sistema utilizza la struttura base per la registrazione fornita da Fortify, ma aggiunge i campi `surname`, `username` e `codice_fiscale`.

1. Valida l'input dell'utente.
2. Genera l'`username`, sommando `name` e `surname`. Ciclando tra quelli esistenti, se esiste già aggiunge un numero.
3. Crea le preferenze tramite la chiamata al metodo `userPreference()` per creare un record associato.
4. Crea l'estensione (operator/guest) in base all'input `role` (hidden e determinato in fase di registrazione in base al link cliccato).
5. Infine sincronizza il ruolo (operator/guest).

LOGIN

Il login viene gestito in base alle indicazioni contenute in `app/Provider/FortifyServiceProvider.php`.

La pipeline di autenticazione è customizzata:

1. Verifica l'esistenza dell'user recuperandolo con `email`, `username` o `codice_fiscale`. Utilizza attivamente solo l'input `login` (definito anche nel file config di Fortify).
2. Verifica che l'utente abbia un ruolo (guest/operator), altrimenti lancia una Exception.

ALTRI TWEAK

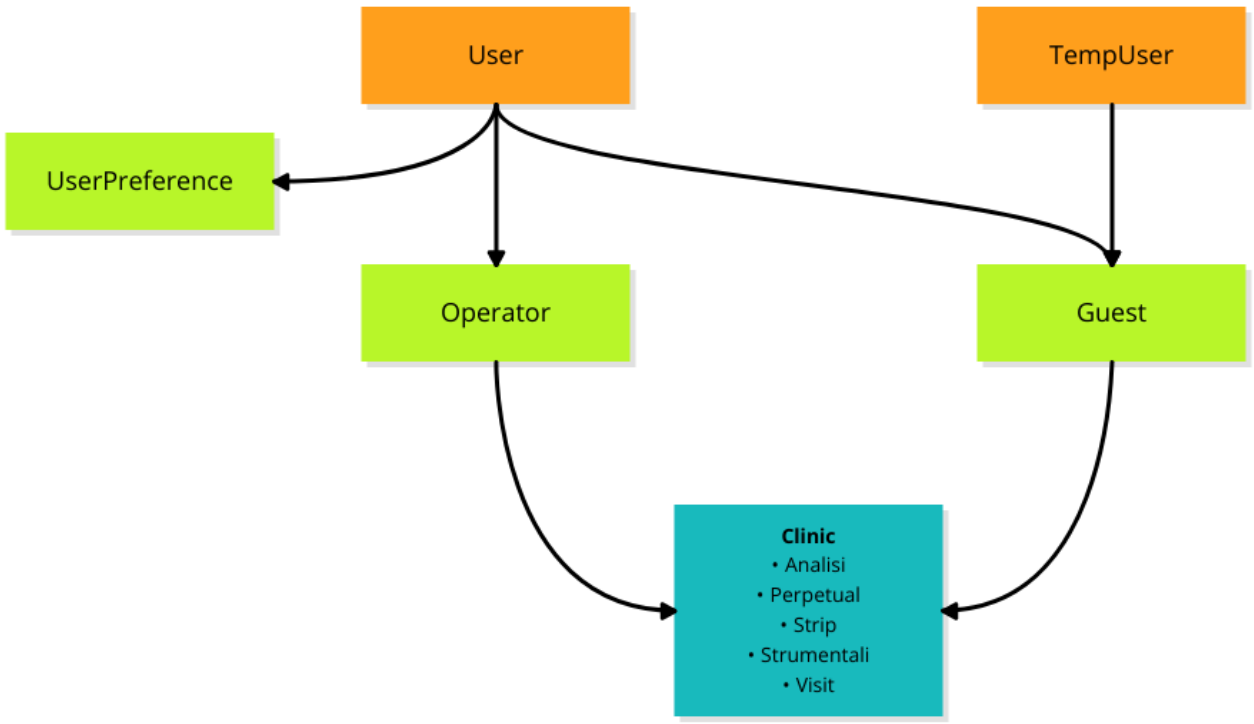
Nel metodo `boot()` è presente `Fortify::ignoreRoutes()` che bypassa la registrazione di base delle routes. È stato fatto per cambiare nome alla rotta di registrazione. Le rotte sono registrate manualmente con `require __DIR__ . '/fortify.php'` in `routes/web.php`.

MODELLI

Di seguito tutto ciò che riguarda i modelli in uso.

BUSINESS LOGIC

La business logic di CardioAI segue un sistema a cascata che estende man mano il modello **User/TempUser**.



Flow Chart di **User**:

1. Ogni **User** ha una **UserPreference** (one to one)
2. Ogni **User** può essere **Operator** o **Guest** (one to one)
3. Ogni **Operator** o **Guest** interagisce con i **Clinic**
    a. Se è **Operator** può eseguire azioni CRUD sui **Clinic**
    b. Se è **Guest** è relazionato ma può leggere i suoi **Clinic**

Flow Chart di **TempUser**:

1. Ogni **TempUser** può essere unicamente **Guest** (one to one)
2. Ogni **Guest** è relazionato con **Clinic**

La presenza di **TempUser** serve a garantire la possibilità all'operatore di creare un'utenza che in seguito può essere riscattata dal paziente. Quando viene riscattata il **TempUser** viene convertito in **User** e viene completata la registrazione con i campi mancanti.

Operator e **Guest** sono relazionati tramite una tabella pivot (operator_guest). Questo garantisce un'associazione temporanea tra operatori sanitari e pazienti. L'associazione è protetta da un sistema di autorizzazione tramite OTP.

API

Di seguito la lista di tutte le funzioni utilizzabili sui modelli in uso.

USER

Il modello **User** identifica l'utente registrato nel database e autenticato.

Metodo	Tipologia	Descrizione
userPreference()	Relazione one to one	Restituisce il modello **UserPreference** associato
operator()	Relazione one to one	Restituisce il modello **Operator** associato
guest()	Relazione one to one	Restituisce il modello **Guest** associato
getInitials()	Helper	Restituisce le iniziali maiuscole da name e surname

TEMPUSER

Il modello **TempUser** identifica l'utente registrato nel database ma mai autenticato.

Metodo	Tipologia	Descrizione
guest()	Relazione one to one	Restituisce il modello **Guest** associato
getInitials()	Helper	Restituisce le iniziali maiuscole da name e surname

GUEST

Il modello **Guest** identifica l'utente che impersona il paziente.

Metodo	Tipologia	Descrizione
user()	Relazione one to one	Restituisce il modello U**ser** associato
tempUser()	Relazione one to one	Restituisce il modello **TempUser** associato
perpetual()	Relazione one to one	Restituisce il modello **Perpetual** associato
strips()	Relazione one to many	Restituisce i modelli **Strip** associati
analisi()	Relazione one to many	Restituisce i modelli **Analisi** associati
visits()	Relazione one to many	Restituisce i modelli **Visit** associati
operators()	Relazione many to many	Restituisce i modelli **Operator** associati tramite pivot
currentOperator()	Filtro su many to many	Restituisce il modello **Operator** attivo
getFullName()	Helper	Restituisce il nome completo del **User** (bind su relazione)
getCodiceFiscale()	Helper	Restituisce il codice fisale del **User** (bind su relazione)
getEmail()	Helper	Restituisce l'email del **User** (bind su relazione)
getAge()	Helper	Restituisce l'età (int) del **User** (bind su relazione)

OPERATOR

Il modello **Operator** identifica l'utente che impersona l'operatore sanitario.

Metodo	Tipologia	Descrizione
user()	Relazione one to one	Restituisce il modello **User** associato
guests()	Relazione many to many	Restituisce i modelli **Guest** associati tramite pivot
currentGuest()	Filtro su many to many	Restituisce il modello **Guest** attivo

USERPREFERENCE

Il modello **UserPreference** identifica l'insieme delle preferenze espresse per ogni **User**.

Metodo	Tipologia	Descrizione
user()	Relazione one to one	Restituisce il modello **User** associato

ANALISI

Il modello **Analisi** identifica l'insieme dei record contenenti dati sulle analisi di laboratorio.

Metodo	Tipologia	Descrizione
guest()	Relazione one to many	Restituisce il modello **Guest** associato

PERPETUAL

Il modello **Perpetual** identifica l'insieme dei record contenenti dati sulla pre-anamnesi fornita dal paziente.

Metodo	Tipologia	Descrizione
guest()	Relazione one to many	Restituisce il modello **Guest** associato

STRIP

Il modello **Strip** identifica l'insieme dei record contenenti dati sulla strip (codice sintetico del paziente).

Metodo	Tipologia	Descrizione
guest()	Relazione one to many	Restituisce il modello **Guest** associato

STRUMENTALI

Il modello **Strumentali** identifica l'insieme dei record contenenti dati sulle analisi di laboratorio.

Metodo	Tipologia	Descrizione
guest()	Relazione one to many	Restituisce il modello **Guest** associato

VISIT

Il modello **Visit** identifica l'insieme dei record contenenti dati sulla visita effettuata sul paziente.

Metodo	Tipologia	Descrizione
guest()	Relazione one to many	Restituisce il modello **Guest** associato

CASTS

Di seguito tutti i "Casts", ossia classi che determinano il modo in cui alcuni dati vengono letti e trasformati in altro prima di essere salvati e viceversa vengono ritrasformati prima di essere distribuiti. In CardioAI queste classi gestiscono soprattutto l'inserimento dei valori per le visite. Vanno letti in questa maniera:

Get()	**Set()**
Il valore nel database che viene trasformato in valore leggibile o utilizzabile.	Il valore leggibile o utilizzabile che viene trasformato in valore numerico/booleano più facilmente immagazzinabile nel database.

BSSD

La sigla indica "Buono, Sufficiente, Scarso, Dettaglia".

GET()

$value	**return**
null	Null
1 (*int*)	buono (*string*)
2 (*int*)	sufficiente (*string*)
3 (*int*)	scarso (*string*)
4 (*int*)	dettaglia (*string*)
default	$value

Es: nel db è segnato 3, nella vista verrà segnato "scarso"

SET()

$value	**return**
null	Null
buono (*string*)	1 (*int*)
sufficiente (*string*)	2 (*int*)
scarso (*string*)	3 (*int*)
dettaglia (*string*)	4 (*int*)
default	error

Es: nella vista è segnato "sufficiente", nel db verrà segnato 2

Default -> error garantisce un comportamento di tipo enum senza le limitazioni legate alla logica degli enum. Non è necessario lanciare un errore anche nella casistica default del metodo get() poiché basta costringere la formattazione dei dati in salvataggio (constraint apriori).

N1

La sigla indica "Null, true".

GET()

$value	return
null	Null
1 (*int*)	true (*boolean*)
default	$value

Es: nel db è segnato 1, nella vista verrà segnato “true”

SET()

$value	return
null	Null
true (*boolean*)	1 (*int*)
default	error

Es: nella vista è segnato “true”, nel db verrà segnato 1

Default -> error garantisce un comportamento di tipo enum senza le limitazioni legate alla logica degli enum. Non è necessario lanciare un errore anche nella casistica default del metodo get() poiché basta costringere la formattazione dei dati in salvataggio (constraint apriori).

N01

La sigla indica “Null, false, true”.

GET()

$value	return
null	Null
0 (*int*)	false (*boolean*)
1 (*int*)	true (*boolean*)
default	$value

Es: nel db è segnato 0, nella vista verrà segnato “false”

SET()

$value	return
null	Null
false (*boolean*)	0 (*int*)
true (*boolean*)	1 (*int*)
default	error

Es: nella vista è segnato “false”, nel db verrà segnato 0

Default -> error garantisce un comportamento di tipo enum senza le limitazioni legate alla logica degli enum. Non è necessario lanciare un errore anche nella casistica default del metodo get() poiché basta costringere la formattazione dei dati in salvataggio (constraint apriori).

N012

La sigla indica “Null, false, true, codex_da”. “codex_da” si rifersice al file config `cai-codex.php` – invocato tramite `config()` restituisce il seguente array:

```
'codex_da' => [
     'short' => 'D/A',
     'full' => 'da approfondire',
],
```

GET()

$value	**return**
null	Null
0 (*int*)	false (*boolean*)
1 (*int*)	true (*boolean*)
2 (*int*)	codex_da (*string*)
default	$value

Es: nel db è segnato 2, nella vista verrà segnato “codex_da”

SET()

$value	**return**
null	Null
false (*boolean*)	0 (*int*)
true (*boolean*)	1 (*int*)
codex_da (*string*)	2 (*int*)
default	error

Es: nella vista è segnato “true”, nel db verrà segnato 1

Default -> error garantisce un comportamento di tipo enum senza le limitazioni legate alla logica degli enum. Non è necessario lanciare un errore anche nella casistica default del metodo get() poiché basta costringere la formattazione dei dati in salvataggio (constraint apriori).

N012A

Variante di **N012**. La sigla indica “Null, false, true, codex_dtl”. “codex_dtl” si rifersice al file config `cai-codex.php` – invocato tramite `config()` restituisce il seguente array:

```
'codex_dtl' => [
     'short' => 'DTL',
     'full' => 'dettaglia',
],
```

GET()

$value	return
null	Null
0 (*int*)	false (*boolean*)
1 (*int*)	true (*boolean*)
2 (*int*)	codex_dtl (*string*)
default	$value

Es: nel db è segnato 2, nella vista verrà segnato “codex_dtl”

SET()

$value	return
null	Null
false (*boolean*)	0 (*int*)
true (*boolean*)	1 (*int*)
codex_dtl (*string*)	2 (*int*)
default	error

Es: nella vista è segnato “true”, nel db verrà segnato 1

Default -> error garantisce un comportamento di tipo enum senza le limitazioni legate alla logica degli enum. Non è necessario lanciare un errore anche nella casistica default del metodo get() poiché basta costringere la formattazione dei dati in salvataggio (constraint apriori).

N012P

Variante di **N012**. La sigla indica “Null, codex_pat, codex_non_pat, codex_da”. “codex_pat”, “codex_non_pat” e “codex_da” si rifersicono al file config `cai-codex.php` – invocati tramite `config()` restituiscono il seguenti array:

```
'codex_pat' => [
     'short' => 'PT',
     'full' => 'patologico',
],

'codex_non_pat' => [
     'short' => 'NPT',
     'full' => 'non patologico',
],

'codex_da' => [
     'short' => 'D/A',
     'full' => 'da approfondire',
],
```

GET()

$value	return
null	Null
0 (*int*)	codex_pat (*string*)
1 (*int*)	codex_non_pat (*string*)
2 (*int*)	codex_da (*string*)
default	$value

Es: nel db è segnato 1, nella vista verrà segnato “codex_non_pat”

SET()

$value	return
null	Null
codex_pat (*string*)	0 (*int*)
codex_non_pat (*string*)	1 (*int*)
codex_da (*string*)	2 (*int*)
default	error

Es: nella vista è segnato “codex_pat”, nel db verrà segnato 0

Default -> error garantisce un comportamento di tipo enum senza le limitazioni legate alla logica degli enum. Non è necessario lanciare un errore anche nella casistica default del metodo get() poiché basta costringere la formattazione dei dati in salvataggio (constraint apriori).

N0123S

Variante estesa di **N012**. La sigla indica “Null, false, codex_cln, codex_sbc, codex_da”. “codex_cln”, “codex_sbc” e “codex_da” si rifersicono al file config `cai-codex.php` – invocati tramite `config()` restituiscono il seguenti array:

```
'codex_cln' => [
     'short' => 'CLN',
     'full' => 'clinico',
],

'codex_sbc' => [
     'short' => 'SBC',
     'full' => 'subclinico',
],

'codex_da' => [
     'short' => 'D/A',
     'full' => 'da approfondire',
],
```

GET()

$value	return
null	Null
0 (*int*)	false (*boolean*)
1 (*int*)	codex_cln (*string*)
2 (*int*)	codex_sbc (*string*)
3 (*int*)	codex_da (*string*)
default	$value

Es: nel db è segnato 2, nella vista verrà segnato “codex_sbc”

SET()

$value	return
null	null
false (*boolean*)	0 (*int*)
codex_cln (*string*)	1 (*int*)
codex_sbc (*string*)	2 (*int*)
codex_da (*string*)	3 (*int*)
default	error

Es: nella vista è segnato “codex_cln”, nel db verrà segnato 1

Default -> error garantisce un comportamento di tipo enum senza le limitazioni legate alla logica degli enum. Non è necessario lanciare un errore anche nella casistica default del metodo get() poiché basta costringere la formattazione dei dati in salvataggio (constraint apriori).

N0123T

Variante estesa di **N012**. La sigla indica “Null, false, codex_prg, codex_att, codex_prg_att”. “codex_prg”, “codex_att” e “codex_prg_att” si riferiscono al file config `cai-codex.php` – invocati tramite `config()` restituiscono il seguenti array:

```
'codex_prg' => [
     'short' => 'PRG',
     'full' => 'pregresso',
],

'codex_att' => [
     'short' => 'ATT',
     'full' => 'attuale',
],

'codex_prg_att' => [
     'short' => 'PRG/ATT',
     'full' => 'pregresso/attuale',
],
```

GET()

$value	**return**
null	Null
0 (*int*)	false (*boolean*)
1 (*int*)	codex_prg (*string*)
2 (*int*)	codex_att (*string*)
3 (*int*)	codex_prg_att (*string*)
default	$value

Es: nel db è segnato 2, nella vista verrà segnato “codex_att”

SET()

$value	**return**
null	null
false (*boolean*)	0 (*int*)
codex_prg (*string*)	1 (*int*)
codex_att (*string*)	2 (*int*)
codex_prg_att (*string*)	3 (*int*)
default	error

Es: nella vista è segnato “codex_att”, nel db verrà segnato 2

Default -> error garantisce un comportamento di tipo enum senza le limitazioni legate alla logica degli enum. Non è necessario lanciare un errore anche nella casistica default del metodo get() poiché basta costringere la formattazione dei dati in salvataggio (constraint apriori).

SCRIPT

La logica di divisione generale degli script è la seguente:

- **Bundle** – *Root* – Richiamano lo script `bootstrap.js` nel quale vengono inizializzate tutte le risorse necessarie al funzionamento di Laravel. Inoltre richiamano la libreria frontend e altre dipendenze necessarie al funzionamento delle pagine. Non necessitano di delucidazioni perché sono solo import builder.
- **Libs** – */libs* – Sono librerie di dipendenza, necessarie in altri script o a livello generale.
- **Modules** – */modules* – Sono librerie one-shot, ossia dedicate all'esecuzione di script su una singola pagina. Hanno infatti il nome costituito dalla formula "exec-" seguita dal nome della pagina o dello scopo della pagina.

LIBS

Di seguito una spiegazione degli script presenti nella sezione **Libs**.

ALERT.JS

Si tratta di una libreria necessaria per il funzionamento degli alert di sessione di Laravel.

API

Attributo	Posizione	Valore	Descrizione
alert-duration	Alert	Int	Un valore numerico che indica dopo quanti ms (millisecondi) l'alert verrà chiuso e rimosso dal DOM.
alert-close	Bottone close	//	Usato come identificatore sul bottone o sul div che, se cliccato, chiuderà l'alert.

LOADER.JS

Si tratta di una libreria che gestisce il funzionamento del loader della pagina (spinner).

MOBILE.JS

Si tratta di una libreria che gestisce il funzionamento del menu mobile.

API

Attributo	Posizione	Valore	Descrizione
mobile-menu	Wrapper	//	Usato come identificatore sul wrapper del menu mobile
open-mobile-menu	Bottone open	//	Usato come identificatore sul bottone o sul div che, se cliccato, aprirà il menu.
close-mobile-menu	Bottone close	//	Usato come identificatore sul bottone o sul div che, se cliccato, chiuderà il menu.
overlay-mobile-menu	Div	//	Usato come identificatore sul div che funge da blocco nero trasparente

NAVIGATION.JS

Si tratta di una libreria che gestisce il funzionamento dei bottoni di navigazione nell'interfaccia della visita.

API

Attributo	Posizione	Valore	Descrizione
cai-goto	Bottone	String (id)	Usato sul bottone o sul div che, se cliccato, mostrerà il div con ID uguale al valore impostato.
cai-goto-next	Bottone avanti	//	Usato sul bottone o sul div che, se cliccato, mostrerà il div con ID uguale al valore impostato + 1.
cai-goto-previous	Bottone indietro	//	Usato sul bottone o sul div che, se cliccato, mostrerà il div con ID uguale al valore impostato - 1.
cai-page	Div wrapper	//	Usato come identificatore sul div che funge da wrapper per la pagina da mostrare/nascondere.

THEMESELECTOR.JS

Si tratta di una libreria che gestisce il salvataggio in localStorage delle preferenze sul tema.

VALIDATOR.JS

Si tratta di una libreria che gestisce la validazione degli input.

API

Attributo	Posizione	Valore	Descrizione
data-validator	Input	String	Usato sull'input che, se compilato, mostrerà il messaggio di errore in caso di validazione fallita o si colorerà di verde in caso di validazione confermata.
data-validator-error	Input	String (id)	Usato sull'input con valore uguale all'ID del div contenente il testo di errore.
data-validator-submit	Bottone submit	//	Usato sul bottone o sul div che, se cliccato, validerà prima tutti gli input.
data-validator-confirm	input	String (id)	Usato sull'input con valore uguale all'ID dell'input contente il valore da confrontare con quello del div presente.

Valore	Descrizione
nome	Solo lettere (maiuscole/minuscole) e spazi
cognome	Solo lettere (maiuscole/minuscole) e spazi
email	Regex ottimizzata per evitare caratteri sospetti come doppio punto e dominio minimo a 2 lettere
password	Almeno una minuscola, una maiuscola, un numero, un carattere speciale e minimo 6 caratteri.
confronta valore	[Richiede data-validator-confirm] Recupera il selettore dell'elemento da confrontare dal data-validator-confirm. In seguito valida solo se entrambi i campi hanno lo stesso valore.
confronta (alias)	[Richiede data-validator-confirm] Recupera il selettore dell'elemento da confrontare dal data-validator-confirm. In seguito valida solo se entrambi i campi hanno lo stesso valore.

codice fiscale	16 caratteri in questa sequenza: 6 lettere, 2 numeri, 1 lettera, 2 numeri, 1 lettera, 3 numeri, 1 lettera.
cap	5 cifre
telefono cellulare	Inizia con 3, seguito da 9 cifre (10 numeri totali)
telefono fisso	Inizia con 0 e seguito da 8/10 cifre
altezza in cm	Numero in un range plausibile (50-250)
licenza	Formato: XXXXX-XXXXX-XXXXXXX (5-5-7, caratteri alfanumerici)
data	Accetta solo numeri, /, -, : e spazi
peso	Numero positivo inferiore a 500
circonferenza della vita	Numero in un range plausibile (30-200)
pressione sistolica	Numero in un range plausibile (90-250)
pressione diastolica	Numero in un range plausibile (50-150)
frequenza cardiaca	Numero in un range plausibile (30-220)
frequenza respiratoria	Numero in un range plausibile (5-60)
saturazione arteriosa	Numero in un range plausibile (70-100)
valore analisi	Numero

VISIBILITY.JS

Si tratta di una libreria che gestisce la visibilità dei contenuti.

API

Attributo	Posizione	Valore	Descrizione
cai-internal-open	Bottone	String (id)	Usato sul bottone con valore uguale all'ID del target che deve aprire
cai-open-group	Bottone	String (id)	Usato sul bottone con valore uguale all'ID del target che deve aprire
cai-close-group	Bottone	String (id)	Usato sul bottone con valore uguale all'ID del target che deve chiudere
cai-direct-open-dettaglia	Bottone	String (id)	Usato sul bottone con valore uguale all'ID del target che deve aprire
cai-direct-close-dettaglia	Bottone	String (id)	Usato sul bottone con valore uguale all'ID del target che deve chiudere
cai-open-dettaglia	Bottone	String (id)	Usato sul bottone con valore uguale all'ID del target che deve aprire
cai-close-dettaglia	Bottone	String (id)	Usato sul bottone con valore uguale all'ID del target che deve chiudere
cai-open	Bottone	String (id)	Usato sul bottone con valore uguale all'ID del target che deve aprire
cai-open-sub	Bottone	String (id)	Usato sul bottone con valore uguale all'ID del target che deve aprire
cai-close-sub	Bottone	String (id)	Usato sul bottone con valore uguale all'ID del target che deve chiudere
cai-open-note	Bottone	String (id)	Usato sul bottone con valore uguale all'ID del target che deve aprire

MODULES

Di seguito una spiegazione degli script presenti nella sezione **Modules**.

EXEC-ADD-GUEST.JS

Script eseguito sulla pagina/funzione di aggiunta del Guest. Ecco una descrizione dettagliata:

1. Aggregazione dei campi input multipli

- Tutti gli input con l'attributo data-hs-pin-input-item vengono letti.
- I loro valori vengono concatenati in un'unica stringa chiamata aggregate.
- Se almeno uno degli input è vuoto, una variabile booleana allFilled viene impostata su false.

2. Aggiornamento del campo nascosto

- Il valore aggregato viene inserito in un campo hidden con ID #aggregate-codice_fiscale.

3. Gestione del bottone di submit

- Il bottone submit viene abilitato solo se **tutti gli input** sono compilati.

4. Supporto per un campo unico

- Se si utilizza un input alternativo con ID #plain_cf (un campo unico per tutto il codice fiscale), viene ignorata la logica sopra:
    - Il suo valore viene copiato direttamente nel campo nascosto.
    - Il bottone di submit viene sempre abilitato.

5. Reset del form

- Quando si clicca sul bottone reset, la funzione updateAggregate viene eseguita con un leggero ritardo (setTimeout(..., 0)) per assicurarsi che il reset degli input sia completato prima dell'aggiornamento.

EXEC-COMPLETE-GUEST.JS E EXEC-COMPLETE-GUEST.TS

Script eseguiti sulla pagina/funzione di completamento del Guest. Ecco una descrizione dettagliata:

1. All'avvio:

- I select #input-province e #input-city sono **disabilitati**.
- Le regioni vengono caricate tramite una **chiamata AJAX** e popolate nel select #input-region.

2. Quando l'utente seleziona una regione:

- Viene abilitato #input-province, svuotato e ricaricato con le province relative.
- #input-city viene disabilitato e resettato.

3. Quando l'utente seleziona una provincia:

- Viene abilitato #input-city, svuotato e ricaricato con i comuni associati.
- Ogni comune è caricato con il suo codice (usato come value) e nome.

4. **Configurazione del calendario:**

- Localizzazione italiana (locale: 'it'), intervallo da 1920-01-01 fino ad oggi.
- Il calendario è collegato all'input #input-birthdate.
- Quando l'utente seleziona una data, viene:
    - Mostrata nel formato gg/mm/aaaa nell'input visibile.
    - Salvata nel campo nascosto #birthdate-iso in formato aaaa-mm-gg.

5. **Gestione dell'inserimento manuale:**

- All'uscita dal campo (blur), se l'utente ha scritto una data valida:
    - Viene convertita in formato ISO e salvata nell'input nascosto.
    - Il calendario viene aggiornato internamente.
- Se la data non è valida o il campo è vuoto:
    - L'input nascosto viene **svuotato**, evitando dati inconsistenti.

EXEC-FIRST-VISIT-AI.JS

Script eseguito sulla pagina/funzione di visita del Guest. Ecco una descrizione dettagliata:

1. Stato dei fattori di rischio (FAT)

Quando l'utente inserisce dati in certi input (esami o misurazioni), vengono eseguite delle **funzioni helper** (ipr(), obs(), mtb(), dlp(), ins()) per valutare se un determinato fattore di rischio è presente.

Se il fattore viene identificato come **presente**, allora:

- Il campo input-has_fat viene impostato a 1.
- I pulsanti/interfacce associate a input-has_fat vengono **disabilitati e stilizzati** (colore verde).
- Vengono attivati specifici campi secondari come input-fat_4, input-fat_5, ecc., marcandoli con **stili visivi appropriati** (es. verde o arancio).
- Viene eseguito l'aggiornamento dello stato con updateN012('fat').
- Viene mostrata una sezione chiamata #fat-sub.

Se il fattore **non è più rilevato**, tutti i campi legati vengono:

- **Resettati**
- **Sbloccati**
- Riportati al colore di default (bianco).

2. Score ESC (Rischio Cardiovascolare)

Quando l'utente inserisce il valore input-score_esc:

- In base al valore numerico (<1, 1-4.9, 5-9.9, ≥10), viene **attivato e disabilitato** il relativo pulsante:
    - < 1: input-rcv_14
    - 1-5: input-rcv_13
    - 5-10: input-rcv_11
    - ≥ 10: input-rcv_5
- I pulsanti a livello inferiore vengono **disabilitati in cascata**.

3. Gestione UI con attributi cai-input

L'interfaccia utilizza attributi personalizzati ([cai-input] e [cai-input-set-val]) per:

- Applicare stili dinamici
- Abilitare/disabilitare gruppi di input
- Evidenziare visivamente lo stato dei fattori clinici con colori distinti:
    - **Verde** per valori confermati
    - **Rosso/arancio** per stati patologici o in attesa
    - **Bianco** per valori neutri o non definiti

EXEC-FIRST-VISIT-FOLLOWUP.TS

Script eseguito sulla pagina/funzione di visita del Guest. Ecco una descrizione dettagliata:

- Quando l'utente seleziona una data dal calendario:

1. La data scelta (in formato ISO: yyyy-mm-dd) viene:
    - **Convertita** in formato leggibile (dd/mm/yyyy)
    - **Visualizzata** nel campo #input-followup-single
    - **Salvata** nel campo nascosto #followup-single-iso
2. Il calendario viene **chiuso** automaticamente.

- Gestione dell'inserimento manuale della data

- Quando l'utente **scrive la data a mano** nel campo visibile e esce dal campo (blur):
    1. Lo script verifica se è nel **formato gg/mm/aaaa**
    2. Se è valido:

- Viene **convertita in formato ISO** (aaaa-mm-gg)
- Aggiornato l'input nascosto #followup-single-iso
- Viene aggiornato anche internamente il calendario con quella data

3. Se non è valido:
    - L'input nascosto viene **resettato** a stringa vuota ('')

EXEC-FIRST-VISIT-OUTCOME.JS

Script eseguito sulla pagina/funzione di visita del Guest. Ecco una descrizione dettagliata:

1. Valutazione degli esiti clinici

- Usa input di tipo select, checkbox e text per valutare:
    - **Anamnesi cardiovascolare** (acv)
    - **Esame obiettivo** (esm)
    - **Fattori di rischio** (fat)
    - **Eventi cardiovascolari** (evp)
- Classifica il paziente in categorie **RCV (0–5)** e **SHF (0–4)** usando logica condizionale avanzata.

2. Trigger e logica condizionata

- Al click di un pulsante con attributo cai-goto-next, lo script:
    - Valuta condizioni cliniche complesse (es. combinazioni di esami, età, tipo di diabete, pressione, LDL).
    - Chiama funzioni updateROutcome(), updateSOutcome() e updateRLdl() per aggiornare lo stato clinico.

3. Gestione visuale dei blocchi e contenuti

- Blocchi HTML con attributi personalizzati:
    - [cai-target-update-block] → mostra/nasconde intere sezioni
    - [cai-target-update-content] → popola liste <ul> con informazioni dettagliate
    - [cai-target-update-hint] → inserisce alert o messaggi di incompletezza
- Riporta dinamicamente il contenuto sotto forma di testo leggibile, utile per **verifica visiva** e **stampa/reporting**.

4. Dettagli e specificità

- L'interfaccia riconosce:

- Sottotipi clinici (acuta, subclinica, chirurgica, ecc.)
- Gravità e completezza delle informazioni
- Necessità di approfondimento con classi text-error e messaggi in rosso

5. Calcoli automatici

- Calcola **IMC (BMI)** in base a peso e altezza
- Interpreta punteggi come **SCORE ESC (%)** e li traduce in classi cliniche
- Verifica condizioni per malattie complesse come **diabete con danno d'organo**, **cardiopatie strutturali**, **nefropatia**, ecc.

EXEC-FIRST-VISIT-VALIDATION-AND-STATE.JS

Script eseguito sulla pagina/funzione di visita del Guest. Ecco una descrizione dettagliata:

1. Valutazione e Outcome RCV e SHF

- Assegna classi RCV (0–5) e SHF (A–D) secondo criteri ESC e clinici.
- Calcolo tramite array di funzioni condizionali (rcvClasses, shfClasses).
- Aggiorna il risultato in tempo reale tramite updateROutcome() e updateSOutcome().

2. Anamnesi Cardiovascolare Dinamica (ACV)

- Mostra/nasconde blocchi riepilogativi ([cai-target="acv"]) in base alla presenza di input validi.
- Selezione automatica del contenuto dettagliato: ASCVD clinico/subclinico, cardiopatie, DNN, bypass.
- Crea liste <li> dinamiche nel blocco [cai-target-update-content="acv"].

3. Fattori di Rischio (FAT)

- Analisi di 12 fattori clinici (familiarità, ipertensione, diabete, obesità...).
- Verifica interrogativi (valori = 2) → mostra hint.
- Popola dinamicamente il contenuto testuale in base ai sottocampi (es. tipo di diabete, stato del tabagismo, ecc.).

4. Esame Clinico (ESM)

- Mostra blocco ESM e costruisce riepilogo di sintomi (dispnea, angina, IMC, frequenze, saturazione...).
- Gestione NYHA e classi canadesi per sintomi pregressi/attuali.
- Supporta valori complessi (pressioni sistoliche/diastoliche, aritmie con sottotipi, saturazioni in ossigenoterapia o area ambiente).

5. Esami di Laboratorio (ANL), Empirici (EMP)

- Funzioni riutilizzabili: updateN012, updateN012P, updateN0123S per gestire logica di stato (null, pending, success).
- Controllano sia valori input (0,1,2) sia note testuali.

6. Farmaci Precedenti e Successivi (TER / NTR)

- Gestione dinamica di blocchi farmaco:
    - Aggiunta, eliminazione, numerazione dinamica ID.
    - Parsing JSON per ter_before e ter_after.
- Duplica i farmaci precedenti in un contenitore ntr-container per la revisione.
- Supporta stato null, success, pending in base a #input-has_ter o #input-has_ntr.

7. Follow-up

- Valida presenza di numero/frequenza o note.
- Mostra stato in fwp in base a #input-has_fwp e contenuti correlati.
- Aggiorna input ter_after e riepilogo a ogni modifica.

8. Gestione Dinamica Stati Visivi

- Tutte le sezioni usano:
    - cai-has-gate: abilita/disabilita input condizionati.
    - cai-target="[nome]", cai-state="success|pending|null" per indicare lo stato.
- Funzione hideTarget() per resettare lo stato visivo prima del rendering.

9. Update Completo della Toolbar

- Il pulsante [cai-goto-next] è attivato/disattivato dinamicamente in base allo stato delle sezioni principali (acv, fat, esm, anl, ter).
- Stile verde se tutto è in stato success, altrimenti è attivo ma neutro se c'è almeno un pending.

EXEC-FIRST-VISIT.JS

Script eseguito sulla pagina/funzione di visita del Guest. Ecco una descrizione dettagliata:

1. Gestione click su valutazioni N012 (classi binarie o trinarie)

- Target: pulsanti con cai-input-set-val → aggiornano input corrispondenti con id=....
- Impostano classi colore in base al valore (0 = rosso, 1 = verde, 2 = arancione).
- Supportano l'apertura/chiusura dinamica di sottosezioni tramite cai-open-item-sub e cai-close-item-sub.

2. Valutazioni funzionali BSSD (Buono/Sufficiente/Scarso/Dettaglia)

- Gestite tramite attributo data-bssd.
- Cambiano colore dei bottoni e aggiornano l'input target.
- Controllano dinamicamente la visibilità dei sottoblocchi (sub-sezioni dettagliate).

3. Valutazioni N0123P (Non patologico / Patologico / Pending)

- Usano cai-evaluation-set-val su pulsanti.
- Differenziano tra:
    - **Gruppi principali** → click disabilitato se già selezionato.
    - **Sottogruppi (sub)** → click attiva o disattiva il valore (toggle).
- Assegnano classi colore specifiche:
    - Verde = non patologico, Rosso = patologico, Arancio = pending.

4. Valutazioni cliniche N0123S

- Simile ai gruppi precedenti, ma con significati clinici:
    - 0 = Nessuna, 1 = Clinica, 2 = Subclinica, 3 = Pending.
- Classi visive coerenti con il significato clinico.
- Supporto all'espansione/chiusura delle sottosezioni come per N012.

Inoltre:

- Calcola il numero di caratteri rimanenti per i <textarea> con maxlength usando l'attributo **cai-note-character**.
- Mostra dinamicamente il conteggio residuo in uno span.
- Quando un <select> cambia, lo script:
    - Legge l'attributo **cai-open-sub-identifier** della option selezionata.
    - Interpreta il contenuto come JSON o CSV.
    - Mostra i blocchi con cai-sub-identifier="..." corrispondenti.
    - Nasconde tutti gli altri sub-blocchi nel contenitore corrente.
- Due pulsanti (#set-followup-single, #set-followup-cycle) attivano le rispettive interfacce:
    - Se cliccato, evidenzia il pulsante e mostra il container associato.
    - Se cliccato nuovamente, deseleziona e nasconde tutto.
- I contenitori #followup-single-container e #followup-cycle-container sono gestiti tramite classi Tailwind (hidden).
- In base al valore #input-followup-timerange-operator, lo script mostra il campo "al" o "ogni":
    - Se al → mostra container timerangeAlContainer.
    - Altrimenti → mostra timerangeOgniContainer.

EXEC-VERIFY-OPERATOR.JS

Script eseguito sulla pagina/funzione di completamento dell'Operator. Ecco una descrizione dettagliata:

1. Disabilitazione iniziale dei campi dipendenti

- Al caricamento, disabilita i campi:
    - #input-province
    - #input-city
    - #input-asl

Per evitare che l'utente interagisca prima che siano popolati correttamente.

2. Gestione del campo "Prefisso" in base al sesso

- Campi controllati:
    - input[name="sex"] (radio con valori m o f)
    - #input-prefix (select con le opzioni disponibili)
- Le opzioni disponibili variano in base al sesso:
    - **Maschio (m)**: dottore, professore, operatore
    - **Femmina (f)**: dottoressa, professoressa, operatrice
- Se il sesso non è selezionato o è un valore diverso: unione di tutte le opzioni.

3. Caricamento asincrono delle Regioni

- Effettuato via AJAX da:
- Popola #input-region con un elenco di regioni italiane.

---

4. Popolamento dinamico di Provincia, Comune e ASL

Regione → Province e ASL

- Al change di #input-region:
    - Abilita e svuota #input-province, #input-city, #input-asl
    - Richiede le **province** via API:

      /province/{nome_regione}
    - Carica le **ASL** da un oggetto JS predefinito aslOptions[regione]

Provincia → Comuni

- Al change di #input-province:

- Abilita e svuota #input-city
- Carica i comuni tramite API:
  /comuni/provincia/{nome_provincia}

AI.JS (HELPER)

Script usato come dipendenza in altri script. Ecco una descrizione dettagliata:

1. Ipertensione (Funzioni ipr() e iprStatus())

- Usa i valori di pressione **media** (fat_sub_1, _2) o **attuale** (esm_9, _10).
- Calcola se i valori rientrano nei range di **ipertensione (grado I–III)**, **pressione normale alta** o **isolata sistolica/diastolica**.
- Tiene conto anche dello **stadio di danno renale** e della presenza di **malattie cardiovascolari** (evp).
- Restituisce true/false (presenza ipertensione) e uno **status descrittivo HTML**.

2. Obesità (Funzioni obs() e obsStatus())

- Calcola il **BMI** con peso (esm_7) e altezza (esm_6).
- Restituisce il valore numerico e una descrizione del **grado di obesità (I, II, III)** o **sovrappeso**.

3. Tabagismo (Funzione tbgStatus())

- Legge lo stato selezionato (input-fat_sub_3).
- Restituisce:
  - "pregresso" → ex fumatore
  - "corrente" → fumatore attivo

4. Dislipidemia (Funzioni dlp() e dlpStatus())

- Usa LDL (anl_14 / ven_ldl) e rischio RCV (rcv / rcv_output).
- Determina se il valore è **in target o no** in base alle linee guida per il **grado di rischio cardiovascolare (RCV 1–5)**.
- Output:
  - dlp() → true/false
  - dlpStatus() → stringa HTML con target e soglia raggiunta o superata.

5. Trattamenti cardiotossici (Funzione tctStatus())

- Valuta il valore di neo_status:
  - "incorso", "sospesimeno", "sospesipiu"
- Restituisce una descrizione HTML dello **stato del trattamento cardiotossico**.

6. Insufficienza renale (Funzioni ins() e insStatus())

- Valuta la funzione renale tramite **eGFR** (anl_8).
- Determina la presenza di danno (eGFR < 100 → true) e lo **stadio (G1–G5)**:
    - G5 → < 15
    - G4 → 15–29
    - G3B → 30–44
    - G3A → 45–59
    - G2 → 60–89
    - G1 → ≥90

7. Sindrome metabolica (Funzione mtb())

- Valuta 6 parametri:
    - Circonferenza vita (esm_8)
    - Pressione sistolica e diastolica (esm_9, _10)
    - HDL (anl_13)
    - Trigliceridi (anl_15)
    - Glicemia (anl_5)
    - Obesità (via fat_5)
- Restituisce true se almeno 3 criteri su 6 sono soddisfatti.

UPDATESTRIP.JS (HELPER)

Script usato come dipendenza in altri script. Ecco una descrizione dettagliata:

1. **updatePOutcome(outcome)**
    a. Gestisce il blocco **Prevenzione** ([cai-update-strip="p"]).
    b. Se outcome === 'P' → **Prevenzione Primaria**
    c. Se outcome === 'S' → **Prevenzione Secondaria**
    d. Altrimenti, default su Primaria.
    e. Imposta:
        i. **Classe CSS (dyn-2, dyn-4)**
        ii. **Valore dell'input**

iii. **Testo del codice, titolo e descrizione**

2. **updateSOutcome(outcome)**
    a. Gestisce il blocco **Stadio HF** (Heart Failure) ([cai-update-strip="s"]).
    b. Valori attesi: 0, A, B, C, D + fallback.
    c. Ogni stadio modifica:
        i. Input di stato
        ii. Colore del box (dyn-1…dyn-5)
        iii. Titolo (es. “Stadio HF B”) e descrizione associata
    d. Fallback: N/D se non determinabile dai dati inseriti.
3. **updateROutcome(outcome)**
    a. Gestisce il blocco **Classe di rischio cardiovascolare** (RCV) ([cai-update-strip="r"]).
    b. Valori previsti: da '0' a '5'.
    c. Aggiorna:
        i. Codice RCV e titolo
        ii. Colore dinamico (da dyn-1 a dyn-6)
        iii. Descrizione testuale
        iv. Fallback: N/D + grigio se non calcolabile
4. **updateRLdl(current, outcome)**
    a. Calcola e mostra lo stato in target / non in target dei livelli di **LDL** in base alla classe di rischio (outcome) e al valore LDL corrente (current).
    b. Casi gestiti:
        i. RCV 0 → soglia < 130 mg/dl
        ii. RCV 1 → < 116 mg/dl
        iii. RCV 2 → < 100 mg/dl
        iv. RCV 3 → < 70 mg/dl
        v. RCV 4 → < 50 mg/dl
        vi. RCV 5 → < 40 mg/dl
    c. Colore verde/rosso del valore in base all’esito