




**TEMPLATE DI PROJECT WORK = 3 CFU****min 12 pagine - max 20 pagine***\*da compilare e caricare in formato pdf*

<b>Cognome e Nome:</b>	Brescia Gianluca
<b>Numero di Matricola:</b>	0312300248
<b>Corso di Studio:</b>  L-5 Filosofia ed Etica  L-22 Scienze Motorie  L-31 Informatica per le Aziende Digitali	Barrare la casella riferita al proprio corso di studio
<b>Tema n:</b>	<b>Tema n.1</b>
<b>Titolo del tema:</b>	La digitalizzazione dell'impresa
<b>Traccia del PW n:</b>	<b>Traccia n.1.4</b>
<b>Titolo della traccia:</b>	Sviluppo di una pagina web per un servizio di prenotazione online di un'impresa del settore terziario
<b>Titolo dell'elaborato:</b>	L.A.B.: Laboratorio Analisi dr. Brescia

**PARTE PRIMA – DESCRIZIONE DEL PROCESSO**

<b>Utilizzo delle conoscenze e abilità derivate dal percorso di studio:</b>	Le conoscenze derivanti dal percorso di studio di <i>Programmazione Distribuita e Cloud Computing</i> hanno permesso di suddividere il progetto in layers e concentrarsi sulle funzionalità di ognuno limitando il più possibile la dipendenza funzionale da altri layers, secondo il principio di <i>loose coupling</i> delle componenti.
<b>Fasi di lavoro e relativi tempi di implementazione per la predisposizione dell'elaborato</b>	<p>Da un punto di vista progettuale sono stati definiti quattro layers (Presentation, Application, Business e Data) che rappresentano macroscopicamente come le diverse componenti interagiscono tra loro definendo il <i>round-trip</i>, ovvero rappresentando il flusso di esecuzione di una prenotazione dalla sua generazione fino alla persistenza; questo seguirà il flusso logico partendo dagli elementi esposti tramite l'interfaccia grafica in HTML, passando per un Web Server che espone le API che manipolano una prenotazione, fino ad arrivare alla memorizzazione dei dati.</p> <p>Per ciò che concerne i tempi relativi alla realizzazione del progetto, l'analisi ha permesso di stabilire che la tempistica per realizzare il prodotto finito è di <b>6 settimane</b> così suddivise:</p> <ul style="list-style-type: none"><li>• <i>Settimana 1</i>: analisi approfondita dei requisiti evidenziandone vincoli progettuali, eventuali criticità e possibili soluzioni ad alto livello;</li><li>• <i>Settimana 2</i>: realizzazione di uno schema progettuale per suddividere, ad alto livello, i diversi layer che compongono l'applicazione; individuazione delle componenti Front-End e Back-End in base al risultato della progettazione;</li><li>• <i>Settimana 3</i>: studio delle caratteristiche proprie del linguaggio di programmazione Python, delle librerie di terze parti per la gestione del Web Server locale e test di unità per i servizi realizzati;</li><li>• <i>Settimana 4</i>: studio delle caratteristiche proprie del linguaggio di markup indicato HTML, delle librerie di terze parti per la gestione grafica e comportamentale delle pagine Web; esecuzione di test sull'usabilità delle pagine e sulla responsività ai diversi dispositivi;</li></ul>

	<ul style="list-style-type: none"> <li>• <i>Settimana 5:</i> unione degli sviluppi Front-End e Back-End con test di integrazione e ulteriori migliorie apportate all'aspetto grafico e funzionale; documentazione delle funzioni tramite commenti e ottimizzazione del codice;</li> <li>• <i>Settimana 6:</i> redazione del documento Project Work e raccolta di materiale fotografico da inserire nella documentazione dell'elaborato.</li> </ul>
Risorse e strumenti impiegati:	<p>Durante la fase di analisi è emersa la necessità di comprendere il funzionamento di una prenotazione per i servizi sanitari, individuando il Codice Fiscale e il Codice Tessera Sanitaria come elementi obbligatori sia per l'aggiunta sia per la ricerca delle prenotazioni.</p> <p>A tal fine sono state sfruttate delle risorse bibliografiche, facilmente reperibili online, per stabilire le regole di formato e di validazione dei codici menzionati.</p> <p><b>Regole di formato e validazione di un codice fiscale:</b>  <a href="https://www.agenziaentrate.gov.it/portale/web/guest/schede/istanze/richesta-ts_cf/informazioni-codificazione-pf">https://www.agenziaentrate.gov.it/portale/web/guest/schede/istanze/richesta-ts_cf/informazioni-codificazione-pf</a></p> <p><b>Regole di formato e validazione di una tessera sanitaria:</b>  <a href="https://sistemats1.sanita.finanze.it/portale/documents/20182/34254/allegato%2Btecnico%2BTS-CNS%2Bex%2BDL%2B78-2010_v22-06-12.pdf/2ef2b969-879c-64f5-2b0a-8bce9877c08f">https://sistemats1.sanita.finanze.it/portale/documents/20182/34254/allegato%2Btecnico%2BTS-CNS%2Bex%2BDL%2B78-2010_v22-06-12.pdf/2ef2b969-879c-64f5-2b0a-8bce9877c08f</a></p> <p>In aggiunta alle precedenti e in mancanza di un esperto di dominio nel campo delle analisi cliniche, è stato utilizzato un modello IA (Chat GPT 4) per generare tre categorie di esami da includere nell'applicativo (Sangue, Urine e Nutrizionali) e cinque campioni per ogni categoria accompagnati da una breve descrizione.</p> <p><b>Motivi che hanno orientato la scelta delle risorse e degli strumenti</b>  <b>Modalità di individuazione e reperimento delle risorse e degli strumenti.</b>  Le risorse bibliografiche rappresentano documenti ufficiali emessi da siti istituzionali reperiti con lo scopo di creare una piattaforma realistica, in cui le informazioni inserite vengano controllate per stabilire la correttezza sintattica e logica secondo i criteri legati ai codici menzionati.  L'utilizzo dell'IA, inoltre, ha permesso di gestire velocemente i casi di test e le informazioni strettamente legate al contesto di un laboratorio di analisi, come la generazione di certificazioni ISO specifici che hanno fornito un background sufficientemente realistico all'azienda L.A.B.</p> <p><b>Le eventuali difficoltà affrontate ed il modo in cui sono state superate.</b>  Le principali difficoltà hanno coinvolto la realizzazione di validatori (i cui dettagli saranno approfonditi nel <a href="#">capitolo dei validatori</a>) e la loro integrazione sia lato Front-End sia Back-End a causa di due linguaggi differenti (JavaScript e Python) e comportamenti congrui da assumere onde evitare di inviare informazioni non coerenti.  Per gestire correttamente i validatori e, soprattutto, i test relativi a Data/Ora, Codice Fiscale, Codice Tessera Sanitaria e E-mail sono stati applicati degli approcci differenti lato Front-End e Back-End.  Nel primo caso sono state create delle funzioni ad-hoc lato JavaScript fornendo, come risposta del validatore, un oggetto contenente un flag di controllo superato o meno e una stringa di errore nel caso negativo. Questo ha permesso la gestione di test massivi tramite la console del browser utilizzando un set di campioni per ogni casistica fornito dall'IA; inoltre, il messaggio di risposta, è stato gestito tramite le notifiche del componente Pnotify per segnalare all'utente l'anomalia.  Lato Back-End, invece, i validatori sono stati progettati come parte integrante del modello di Prenotazione (Reservation) in quanto strettamente legati alle proprietà <i>setter</i> presenti nella classe stessa; in questo modo prima di assegnare un valore viene sottoposto a validazioni e, in caso negativo, si lancia un'eccezione che restituirà una risposta con codice HTTP 500 (Internal Server Error).</p>
PARTE SECONDA – PREDISPOSIZIONE DELL'ELABORATO	

<p><b>Obiettivi dell'elaborato/progetto/artefatto:</b></p>	<p>Durante la fase di analisi sono emersi vincoli ed entità che l'applicativo dovrà rispettare. Il <b>core</b> è legato al concetto di <b>prenotazione</b> in ambito di un'azienda del settore terziario, nello specifico un laboratorio di analisi.</p> <p>In questo contesto una <b>prenotazione</b> è un atto di richiesta, da parte di un utente, a sottoporsi ad uno o più esami clinici che avverranno ad una data, ora e locazione scelti dall'utente stesso.</p> <p>L'utente, tramite la piattaforma oggetto di sviluppo, potrà <i>inserire, modificare o cancellare</i> una prenotazione.</p> <p>Ogni istanza di entità di tipo <b>prenotazione</b> è univocamente individuata da tre elementi: un ID univoco, il codice fiscale e il codice tessera sanitaria del paziente. La cardinalità legata alla relazione tra Utente-Prenotazione è identificata come <i>1 a molti</i>: un utente può inviare molteplici prenotazioni ed ogni istanza di prenotazione è assegnata in modo univoco e non ambiguo ad un utente.</p> <p>La cardinalità ha guidato ulteriori vincoli sul concetto di <i>modifica</i> di una prenotazione: per i vincoli descritti poc'anzi è necessario che, in nessun modo, i valori chiave per una prenotazione (ID, codice fiscale e codice tessera sanitaria) siano alterati, altrimenti è considerabile come una nuova prenotazione a fronte del medesimo utente.</p> <p>Un ulteriore vincolo risiede nella <b>data di prenotazione</b>; questa dovrà rispettare i vincoli logici per cui una prenotazione richiede una data cronologicamente successiva alla data odierna e rientri nel range di giorni e orari compatibili con quelli del laboratorio (solo giorni feriali dalle ore 08:00 alle 18:00).</p> <p>Il laboratorio ha fornito, inoltre, tre possibili sedi in cui poter prenotare la prestazione; tutte le sedi sono abilitate all'esercizio di tutti gli esami disponibili.</p> <p>Gli <b>esami</b> rappresentano un'entità figlia della <b>prenotazione</b> e, nel contesto, sono prestazioni sanitarie suddivise in tre categorie: esami del sangue, esami delle urine ed esami dei valori nutrizionali.</p> <p>L'utente, per rendere concreta una prenotazione, dovrà scegliere almeno un esame; ciò non esclude la selezione di multipli esami anche di differenti categorie.</p> <p>In base alle risorse tecnologiche da utilizzare per lo sviluppo dell'applicazione è necessario che la piattaforma risultante sia facilmente utilizzabile a prescindere dalla natura del dispositivo utilizzato.</p>
<p><b>Contestualizzazione:</b></p>	<p>L.A.B. (acronimo di Laboratorio Analisi dr. Brescia) è un'azienda operante nel settore terziario dei servizi di analisi cliniche.</p> <p>Fondata nel 2016 a Lecce dal dott. Brescia Gianluca, un pioniere nel campo delle analisi cliniche, inizialmente il laboratorio era una piccola struttura che forniva servizi di base per la comunità. Grazie alla visione innovativa del dott. Brescia e al suo impegno per la qualità, il laboratorio ha rapidamente guadagnato una reputazione per l'affidabilità e la precisione dei suoi risultati.</p> <p>Negli anni successivi, il laboratorio ha ampliato i suoi servizi, includendo analisi più complesse e specialistiche. Questo ampliamento è stato supportato da investimenti significativi in tecnologie all'avanguardia e dalla formazione continua del personale. Oggi, L.A.B. può vantare tre centri di eccellenza sul territorio leccese che servono una vasta gamma di pazienti, dalle famiglie locali ai professionisti della sanità.</p> <p><b>Vision</b> Essere il punto di riferimento leader per le analisi cliniche in tutta la regione, riconosciuto per l'innovazione, l'affidabilità e l'impegno verso la salute e il benessere della comunità.</p> <p><b>Mission</b> Offrire servizi di analisi cliniche di alta qualità, utilizzando tecnologie avanzate e garantendo risultati precisi e tempestivi. Forte è la dedizione a migliorare continuamente i processi per soddisfare e superare le aspettative dei pazienti e dei professionisti sanitari. È imperativo promuovere un ambiente di lavoro stimolante e collaborativo, dove ogni membro del team può crescere professionalmente e contribuire al successo dell'azienda.</p>

	<p><b>Contesto in cui Opera</b></p> <p>L.A.B. opera in un contesto altamente competitivo dove la domanda di servizi diagnostici accurati e tempestivi è in costante aumento. L'azienda si trova a fronteggiare la concorrenza di altri laboratori e la necessità di conformarsi a normative stringenti e standard di qualità internazionali. Tuttavia, la dedizione all'eccellenza e l'approccio innovativo hanno permesso a L.A.B. di distinguersi e di crescere costantemente.</p> <p><b>Punti di Forza</b></p> <ol style="list-style-type: none"> <li>1. <b>Tecnologia Avanzata:</b> Il laboratorio è dotato delle più moderne apparecchiature di analisi, che permettono di effettuare test complessi con grande precisione.</li> <li>2. <b>Personale Altamente Qualificato:</b> Ogni membro del team è selezionato per la sua competenza e dedizione, e riceve formazione continua per rimanere aggiornato sulle ultime novità nel campo delle analisi cliniche.</li> <li>3. <b>Qualità del Servizio:</b> Il paziente è sempre al centro del lavoro, garantendo un servizio attento, rispettoso e professionale.</li> <li>4. <b>Affidabilità e Tempestività:</b> La capacità di fornire risultati accurati in tempi rapidi è un fattore chiave del successo maturato da L.A.B.</li> <li>5. <b>Innovazione Continua:</b> Impegno nella ricerca e nello sviluppo di nuove metodologie di analisi per migliorare continuamente i servizi offerti.</li> <li>6. <b>Collaborazioni e Partnership:</b> L.A.B. collabora con ospedali, cliniche, medici e altri laboratori per offrire un servizio integrato e completo. Le partnership permettono di accedere a risorse e competenze aggiuntive, migliorando ulteriormente la qualità dei servizi.</li> <li>7. <b>Certificazioni e Riconoscimenti:</b> Il laboratorio è certificato secondo gli standard di qualità <i>ISO 15189:2012</i> e <i>ISO 17025:2017</i>, che attestano l'impegno per la qualità e la sicurezza.</li> </ol>
Descrizione dei principali aspetti progettuali:	<p><b>Repository GitHub sorgente:</b>  <a href="https://github.com/gbrescia96/Pegaso_Project_Work">https://github.com/gbrescia96/Pegaso_Project_Work</a>  Il file <b>README</b> presente sul repository contiene la guida sui requisiti necessari per avviare ed utilizzare l'applicativo in locale.</p> <p><b>L.A.B.</b> è una piattaforma Web rappresentabile, da un punto di vista architetturale, come combinazione di quattro layers.</p>

## Presentation Layer

Fornisce interfacce utente interattive (form) per l'invio e la ricezione di richieste al Web Server; analizza le risposte gestendo la visibilità dei contenuti e le notifiche in base all'esito.



## Application Layer

Espone un Web Server mappando gli endpoint su cui instradare le richieste alle relative API REST; analizza gli oggetti in entrata e fornisce un modello di risposta generico contenente lo stato della richiesta tramite codice HTTP, un messaggio d'errore per richieste diverse da HTTP 200 e un payload fornito dal servizio che implementa l'API invocata (se previsto dal servizio stesso).



## Business Layer

Implementa i servizi che realizzano le funzionalità esposte; effettua validazioni sui modelli e manipolazione dei dati in input e output; comunica con il livello dei dati per delegare le operazioni CRUD (Create, Read, Update, Delete).



## Data Layer

Gestisce la persistenza dei dati e realizza le operazioni CRUD in base al tipo specifico di storage scelto.



Diagramma generato tramite Draw.io

## Sommario Layers

Presentation Layer .....	6
Librerie Front-End .....	6
Jquery .....	6
Bootstrap .....	6
DataTable .....	6
FontAwesome .....	7
Pnotify .....	7
Flusso logico delle pagine .....	7
Pagina Homepage .....	7
Pagina Prenotazione .....	8
Sezione 1: Anagrafica .....	9
Sezione 2: Esami .....	9
Sezione 3: Appuntamento .....	9
Sezione 4: Controllo e invio .....	9

Pagina Ricerca.....	12
Pagina About .....	12
Script condivisi .....	12
Chiamate API.....	12
Inject di Navbar e Footer.....	13
Validatori Codice Fiscale, Tessera Sanitaria e E-mail .....	14
Modal di conferma .....	16
Application Layer.....	17
Modello Response – Risposta HTTP .....	17
Business Layer.....	17
Modello Reservation – Classe Prenotazione .....	18
Metodi Privati .....	18
Data Layer.....	19

## Presentation Layer

Espono l'interfaccia UI/UX con la quale l'utente interagisce attraverso pagine Web che integrano form per l'invio e la visualizzazione dei dati.

In questo contesto le tecnologie utilizzate riguardano la combinazione del linguaggio HTML supportato da CSS per la gestione dell'aspetto grafico e JavaScript per il comportamento dinamico della pagina in base all'interazione dell'utente con i form.

### Librerie Front-End

La tripla HTML/CSS/JS viene estesa da librerie di terze parti che mirano a semplificare e, in molti casi, migliorare gli strumenti che HTML/CSS/JS mettono a disposizione.

#### Jquery

Jquery basa la sua esistenza su JavaScript, fornendo un linguaggio per estenderne le funzionalità impiegando meno codice e istruzioni più semplici.

#### Bootstrap

Bootstrap rappresenta una collezione di classi CSS pre-costruite, già dotate di un'ottima responsività per la maggior parte dei dispositivi (PC, tablet e smartphone) e funzionalità che tramite JavaScript realizzano comportamenti specifici tramite utilizzo degli attributi dei tag.

#### DataTable

DataTable JS è una libreria che combina funzionalità offerte da Jquery e Bootstrap per offrire la rappresentazione tabellare supportata da filtri, riordinamento e rendering di elementi HTML basati sui dati direttamente sulla tabella risultante, senza dover rigenerare i dati con nuove caratteristiche.

Filtro:					
Data/Ora	Stato	Laboratorio	Esami	Modifica	Elimina
30/07/2023 - 12:02	Scaduta	Lecce - Via Rossi 10			
30/07/2024 - 14:31	Valida	Lecce - Via Rossi 10			

Risulta molto utile la possibilità di generare un dato in forme differenti base alla tipologia di rendering che DataTable mette a disposizione. Di base sono tre i rendering più importanti: *display*, *filtering* e *ordering*.

Il primo genera il dato che verrà visualizzato nella pagina HTML, il secondo è il dato che verrà utilizzato come filtro di ricerca e il terzo è il dato utilizzato per l'ordinamento.

Nell'esempio visualizzato i pulsanti *modifica* ed *elimina* hanno un rendering nullo in caso di *filtering* e *ordering*, mentre un rendering come pulsanti HTML in caso di *display*.

Allo stesso modo la colonna *Stato* esegue un controllo sulla data odierna rispetto alla data di prenotazione: per il *display* verrà visualizzata una stringa derivante dall'esito del controllo, per *filtering* e *ordering* verrà mantenuta la data originale della prenotazione.

Questo meccanismo di generazione dei dati in base al rendering è notevolmente importante per poter visualizzare i dati in una forma differente pur mantenendo la possibilità di riordinare e filtrare.

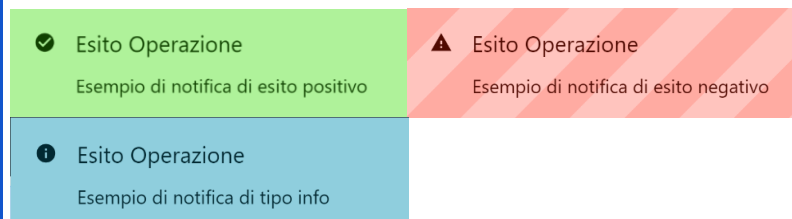
Il campo *Filtra*, infatti, legge da DataTable il rendering *filtering* e tramite esso può generare dei filtri anche quando i dati hanno una natura completamente diversa da quella rappresentata nel record.

#### FontAwesome

Viene utilizzata per generare icone da modelli SVG senza doverle gestire come file separati su disco. L'utilizzo si basa sulla creazione di un tag HTML che renderizza una classe CSS generando l'icona scelta.

#### Pnotify

Il feedback delle azioni dell'utente è in gran parte gestito con le notifiche in tempo reale offerte da Pnotify, una libreria basata su JQuery; in questo modo l'utente visualizzerà le notifiche prendendo atto di un'operazione andata a buon fine, di un fallimento con motivazione o di un messaggio di informazione generica.



#### Flusso logico delle pagine

L.A.B. espone quattro pagine Web:

- Homepage
- Prenotazione
- Ricerca
- About

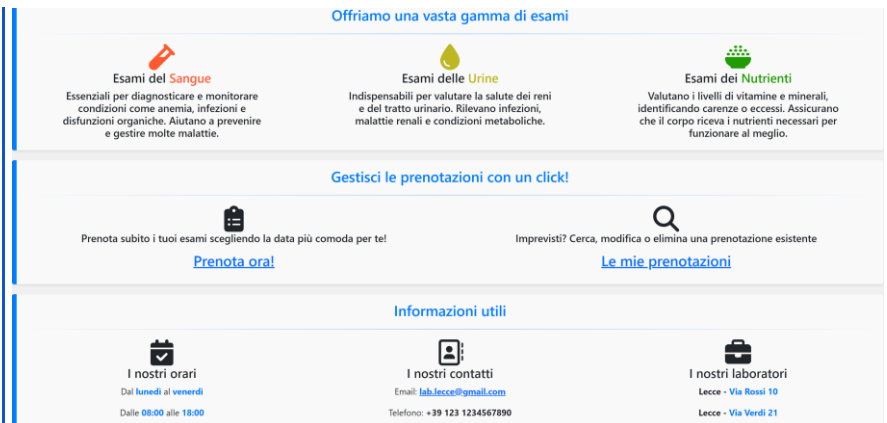
#### Pagina Homepage

È la pagina root del progetto e rappresenta il punto di accesso alla piattaforma Web. Contiene un'immagine che pubblicizza il laboratorio e una sezione sottostante con le informazioni utili sugli esami svolti, i giorni e gli orari utili, i contatti e il riferimento alle pagine di Prenotazione e Ricerca.



Cliccando su "Scopri di più" la pagina sposterà gradualmente il focus sulla sezione sottostante. Lo stesso comportamento si può ottenere facendo scrolling verticale.





## Pagina Prenotazione

Rappresenta il cuore dell'applicativo in quanto contiene i campi necessari per creare o aggiornare una prenotazione.

Data la complessità e lunghezza del form è stato stabilito, in fase progettuale, di suddividere la pagina in quattro sezioni; questo permette di esporre solo una parte del form, nascondendo le altre, visualizzando un sottoinsieme di campi per creare un percorso logico dall'inizio della compilazione fino all'invio finale.

Tutte le sezioni condividono tre elementi comuni: un *breadcrumb* che visualizza il blocco attivo, illuminato con un carattere blu sottolineato e posto in cima alla pagina. È possibile cambiare sezione premendo sulla voce specifica.

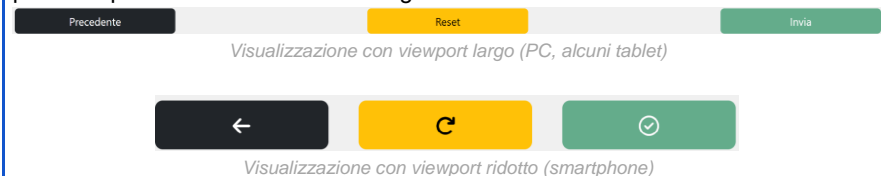
**ANAGRAFICA** • ESAMI • APPUNTAMENTO • CONTROLLA E INVIA

Successivamente vi è una sezione di pulsanti che permettono la navigazione verso la precedente e la successiva sezione.

Questi sono rappresentati da pulsanti con le label estese qualora il dispositivo abbia abbastanza spazio, visualizzando "Precedente" e "Successivo", altrimenti vi saranno solo le frecce avanti e indietro.



Fanno eccezione la prima sezione (Anagrafica) in quanto non vi è il tasto per una sezione precedente e l'ultima sezione (Controlla e Invia) in cui sarà presente un pulsante per l'invio dei dati e il reset globale.



Infine vi è una porzione di pagina rappresentata da un box azzurro, visualizzabile o collassabile con un click, in cui vi è una breve descrizione sui campi visualizzati, le regole da seguire per la compilazione degli stessi e qualsiasi altra informazione che possa aiutare l'utente a comprendere la natura dei campi da compilare. Il messaggio che invita l'utente al click è sempre visualizzabile, a prescindere dallo stato del blocco.

Clicca su questo box per visualizzare/nascondere la guida

Blocco di aiuto non espanso



Clicca su questo box per visualizzare/nascondere la guida

In quest'area inserirai i dati anagrafici relativi alla tua persona. Per favore osserva i seguenti vincoli:

- i campi marcati con il simbolo \* sono obbligatori
- il codice della tessera sanitaria è lungo 20 caratteri ed è presente sul retro della tessera stessa, in fondo
- puoi inserire un'e-mail se vuoi ricevere il referto direttamente sulla tua casella di posta elettronica
- dopo aver inviato la prenotazione i tuoi dati anagrafici non potranno variare eccetto l'e-mail

Blocco di aiuto espanso (Anagrafica)

### Sezione 1: Anagrafica

La sezione Anagrafica contiene i dati fondamentali della persona fisica, identificabile da Nome, Cognome, Codice Fiscale e Codice Tessera Sanitaria. Questi dati sono obbligatori, mentre l'unico dato opzionale è rappresentato dal campo E-mail che ha il solo scopo di stabilire se l'utente preferisce ricevere il referto sulla propria casella di posta oppure recarsi in laboratorio per il ritiro; la presenza di un'e-mail in questo campo implicherà una delle due opzioni per il ritiro del referto.

Sig.	Nome*	Cognome*	Codice Fiscale*	Codice Tessera Sanitaria*	E-mail
▼	Gianluca	Brescia	BR5GLC9680985060	08038000110345678900	gianluca.brescia@gmail.com

### Sezione 2: Esami

La sezione Esami contiene tre sezioni che distinguono gli esami per categoria:

#### Sangue, Urine e Nutrizionali.

Ogni categoria comprende cinque esami corredati da una breve descrizione che ne identificano la natura.

È obbligatorio selezionare almeno un esame; vi è comunque la possibilità di selezionarne più di uno per ogni categoria e, di conseguenza, anche più categorie contemporaneamente.

Esami del sangue				
<input checked="" type="checkbox"/> Emocromo completo Esame dei valori del sangue, compresi globuli rossi, bianchi e piastrine.	<input type="checkbox"/> Profilo lipidico (colesterolo, trigliceridi) Misura dei livelli di colesterolo totale, HDL, LDL e trigliceridi nel sangue.	<input type="checkbox"/> Glicemia Misura del livello di glucosio nel sangue, importante per il controllo del diabete.	<input type="checkbox"/> Funzionalità epatica (AST, ALT) Valutazione degli enzimi epatici per monitorare la salute del fegato.	<input type="checkbox"/> Test della tiroide (TSH, T3, T4) Valutazione della funzionalità tiroidea attraverso i livelli di TSH, T3 e T4.
Esami delle urine				
<input type="checkbox"/> Clearance della creatinina Misura della quantità di creatinina eliminata nelle urine per valutare la funzionalità renale.	<input checked="" type="checkbox"/> Urinocoltura Esame delle urine per rilevare e identificare batteri e altri microrganismi.	<input checked="" type="checkbox"/> Proteinuria Misura della quantità di proteine nelle urine, indicativa di problemi renali.	<input type="checkbox"/> Funzionalità epatica (AST, ALT) Valutazione degli enzimi epatici per monitorare la salute del fegato.	<input type="checkbox"/> Microalbuminuria Misura di piccole quantità di albumina nelle urine, importante per il monitoraggio del diabete e malattie renali.
Esami nutrizionali				
<input type="checkbox"/> Test della vitamina D Misura i livelli di vitamina D nel sangue, essenziale per la salute delle ossa e del sistema immunitario.	<input type="checkbox"/> Test della vitamina B12 Determina i livelli di vitamina B12 nel sangue, importante per la produzione di globuli rossi e il funzionamento del sistema nervoso.	<input checked="" type="checkbox"/> Test dei folati Misura i livelli di folati nel sangue, fondamentali per la sintesi del DNA e la salute del sistema nervoso.	<input type="checkbox"/> Test del ferro e della ferritina Valuta i livelli di ferro e ferritina nel sangue, cruciali per il trasporto dell'ossigeno e la salute generale.	<input type="checkbox"/> Test degli elettroliti Misura i livelli di elettroliti (sodio, potassio, calcio, magnesio) nel sangue, importanti per l'equilibrio idrico e il funzionamento di nervi e muscoli.

Categorie ed esami, con alcuni campioni selezionati

### Sezione 3: Appuntamento

La sezione Appuntamento propone all'utente data, ora e laboratorio in base alle proprie esigenze. Tutti i laboratori sono abilitati ad eseguire tutti gli esami proposti.

Data e ora	Laboratorio
09/08/2024 10:30	Lecce - Via Rossi 10

### Sezione 4: Controllo e invio

L'ultima sezione contiene un riepilogo di tutte le informazioni inserite nelle sezioni precedenti, dando all'utente la possibilità di verificare che non vi siano errori nella compilazione.

Riepilogo dei dati

Sig. **Gianluca Brescia** con codice fiscale **BR5GLC9680985060** e codice tessera sanitaria **08038000110345678900** eseguirà in data **09/08/2024** alle ore **10:30** presso il laboratorio **Lecce - Via Rossi 10** i seguenti esami:

**Sangue:** emocromo  
**Urine:** urinocoltura, proteinuria  
**Nutrizionali:** folati

Gli esiti verranno inviati all'e-mail **gianluca.brescia@gmail.com**

Il messaggio precedente contiene una variante nella frase finale, relativa all'invio dei referti, qualora non sia stata fornita un'e-mail.

Non è stata fornita alcuna e-mail; gli esiti andranno ritirati al laboratorio indicato oppure [clicca qui per aggiungere un'e-mail](#)

Infine, prima dell'invio, è necessario spuntare la casella per confermare i dati inseriti.

☐ Confermo i dati inseriti

L'invio è soggetto alla validazione dei dati tramite funzioni presenti nella pagina che controlleranno ogni sezione per verificare la correttezza logica dei dati. Qualora tutte le validazioni vadano a buon fine la prenotazione sarà inviata al Web Server che, a sua volta, effettuerà nuovamente le validazioni. Se l'esito sarà positivo una nuova prenotazione verrà aggiunta e la pagina visualizzerà il messaggio di esito positivo, concludendo il flusso della prenotazione.

## Prenotazione inviata correttamente!

[Inserisci nuova prenotazione](#)

### Validazioni pre-invio (Upsert)

Prima dell'invio dei dati di una prenotazione, noto come evento *upsertReservation* (Update or Insert), i validatori presenti nella pagina devono analizzare ogni sezione e ritornare esito positivo. In questo senso vi è una condizione posta in *or logico* su tutte le sezioni, vale a dire che se un solo validatore fallisce l'intera funzione di *upsertReservation* verrà sospesa. Sarà altresì visualizzata una notifica per documentare l'errore da parte del validatore specifico che ha riscontrato l'anomalia.

```
//Gestisce Update oppure Insert in base allo stato della prenotazione attuale.
//Questo unifica le validazioni sui campi prima di inviare i dati
async function upsertReservation() {
  var response = null;

  if (preValidateFormSection("anagrafica") == false ||
      preValidateFormSection("esami") == false ||
      preValidateFormSection("appuntamento") == false ||
      preValidateFormSection("preconferma") == false)
  {
    return false;
  }
}
```

Vi è la funzione *preValidateFormSection* che ha lo scopo di raggruppare i diversi validatori, distinguendoli per sezione (i cui rami sono stati collassati per motivi di spazio).

```
//Contiene la validazione dei campi di ogni sotto-sezione
function preValidateFormSection(section) {
  if (section == "anagrafica") { ...
  else if (section == "esami") { ...
  else if (section == "appuntamento") { ...
  else if (section == "preconferma") ...

  return true;
}
```

I controlli riguarderanno la presenza di valori non nulli nei campi di testo obbligatori, la selezione di almeno un esame, la validità della data e ora selezionati e la presenza della spunta sulla conferma dei dati inseriti.

Risulta interessante un approfondimento relativo ai controlli sulla data, come riporta lo stesso box di informazioni della sezione *Appuntamento*, in quanto vi è la necessità che si verifichino contemporaneamente le seguenti condizioni:

- Il giorno dell'appuntamento dev'essere successivo ad oggi;

- il giorno dell'appuntamento dev'essere feriale (dal lunedì al venerdì);
- l'orario dell'appuntamento dev'essere dalle 08:00 alle 18:00,

```
else if (section == "appuntamento") {
    var dateTime = getDateTimeFromString($("#wPrDataOra").val());
    if (dateTime == null) {
        pushNotification("error", "Data e ora prenotazione non hanno un formato valido");
        return false;
    }

    if (checkReservationIsNextDay(dateTime) == false)
    {
        pushNotification("error", "La data della prenotazione dev'essere successiva ad oggi");
        return false;
    }

    if (checkReservationDayRange(dateTime) == false) {
        pushNotification("error", "La data della prenotazione deve essere compresa tra lunedì e venerdì");
        return false;
    }

    if (checkReservationTimeRange(dateTime) == false) {
        pushNotification("error", "L'orario della prenotazione deve essere compreso tra le 08:00 e le 18:00");
        return false;
    }
}
```

Controlli specifici sulla validazione della section Appuntamento

Vi sono controlli aggiuntivi sui campi di Codice Fiscale e Tessera Sanitaria in quanto soggetti a specifiche regole di formato, i cui dettagli sono reperibili nel [capitolo dei validatori](#).

Inoltre, qualora il form non sia stato completato e ci si sposti alla sezione *Controlla e Invia* sarà presente un box di errore al posto della conferma dei dati e sarà bloccata l'interazione (tramite attributo *“disabled”* sui tag) con il pulsante *Invia* e la checkbox di conferma dei dati inseriti.

Errore: non tutti i campi sono stati compilati

☐ Confermo i dati inseriti

Precedente Reset Invia

### Modifica di una prenotazione esistente

È possibile utilizzare la stessa pagina per modificare una prenotazione esistente. Ciò è reso possibile grazie ad un redirect fornito dalla pagina *Ricerca* che invocherà la pagina *Prenotazione* tramite tre parametri aggiuntivi nell'URL.

[Q](#) `prenotazione.html?id=7bbc67c9-4da6-11ef-ba0b-14cb19887d48&cf=BR5GLC96B09B506O&ts=08038000110345678900`

Questi contengono l'id univoco della prenotazione (un campo di tipo Guid generato dai servizi lato Back-End), il codice fiscale (cf) e il codice della tessera sanitaria (ts) del soggetto in questione; è necessario che siano presenti tutti e tre per innescare la gestione di modifica della prenotazione.

Qualora si entri con questi parametri è compito dell'handler di caricamento della pagina leggerli ed invocare una API che restituisce i dati della prenotazione.

I dati saranno utilizzati per compilare automaticamente tutti i campi del form, fornendo la possibilità di effettuare le variazioni necessarie.

Il flusso e i controlli descritti in fase di aggiunta prenotazione sono i medesimi anche nella modifica, ad eccezione dei campi anagrafici (esclusa l'e-mail) che saranno *readonly* in quanto non possono essere soggetti a modifica.

Sig. Nome\* Cognome\* Codice Fiscale\* Codice Tessera Sanitaria\* E-mail

La pagina visualizzerà un blocco di errore qualora la prenotazione non venga trovata rispetto ai parametri inseriti oppure sia scaduta.

**Errore: Prenotazione non trovata**

[Inserisci nuova prenotazione](#)

**Errore: La prenotazione è scaduta il 30/07/2023 e non è possibile apportare modifiche.**

[Inserisci nuova prenotazione](#)

## Pagina Ricerca

La pagina Ricerca offre la possibilità di effettuare una ricerca delle prenotazioni visualizzandone data e ora, stato (valida o scaduta) e la possibilità di effettuare modifica o eliminazione del record.

Per effettuare una ricerca sono necessari il Codice Fiscale e la Tessera Sanitaria. Il risultato di una ricerca visualizzerà una tabella come la seguente.

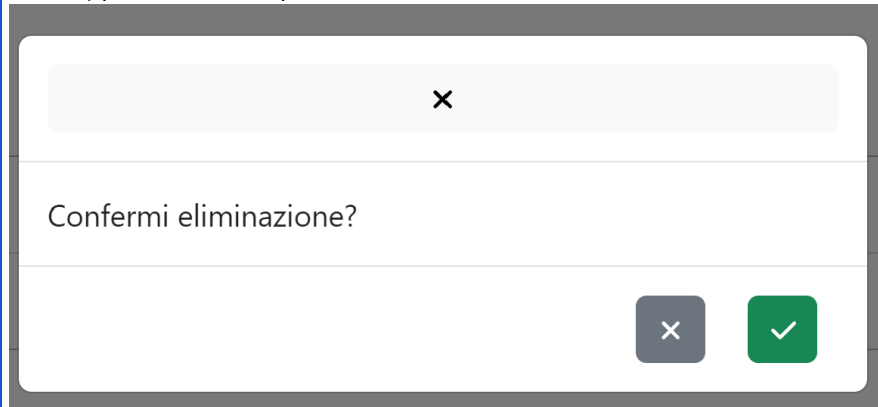
Data/Ora	Stato	Laboratorio	Esami	Modifica	Elimina
30/07/2023 - 12:02	Scaduta	Lecce - Via Rossi 10		-	-
30/07/2024 - 14:31	Valida	Lecce - Via Rossi 10			

È possibile ordinare i record cliccando sulle frecce a destra dei campi Data/Ora, Stato e Laboratorio; si potranno visualizzare gli esami prenotati e richiedere la modifica o l'eliminazione del record.

I dettagli sul rendering dei pulsanti in DataTable e la logica legata al riordinamento e ai filtri è dettagliata nel [capitolo della componente DataTable](#). Come già accennato nel capitolo precedente, la modifica produrrà un redirect alla pagina *Prenotazione* componendo l'URL tramite la seguente funzione, invocata sulla pressione del tasto "Modifica" presente in DataTable.

```
function editReservation(id, cf, ts)
{
  window.location.href = `prenotazione.html?id=${id}&cf=${cf}&ts=${ts}`;
}
```

In caso di eliminazione verrà visualizzato un *modal* con due pulsanti (conferma e annulla) per richiedere esplicitamente conferma dell'azione di eliminazione.



## Pagina About

La pagina About espone la storia di L.A.B., la vision, la mission e il background dell'azienda.

## Script condivisi

Il file *general.json* contiene una serie di funzionalità condivise da tutte le pagine del progetto.

## Chiamate API

Per fornire un metodo generico per invocare le API si fa uso di una funzione custom chiamata *executeApiCall* che ammette in ingresso la tipologia di chiamata HTTP secondo lo standard REST, l'endpoint della chiamata, eventuali parametri da passare nell'URL e l'eventuale body.

Il modello di risposta generato deriva dall'endpoint dell'API qualora il Web Server sia raggiungibile, altrimenti si emula lo stesso modello per mantenere uno standard nelle risposte.

Il modello di risposta è composto da tre campi:

- *code*: indica il codice HTTP che rappresenta la risposta;
- *error\_message*: contiene un messaggio in caso di errore HTTP;
- *payload*: contiene il body della risposta (se previsto).

Questo modello verrà discusso nuovamente nel [capitolo dell'Application Layer](#).

```
/**
 * Astrazione di chiamata API al server. Ritorna l'oggetto Response (modello di risposta presente sul server).
 *
 * @param {string} methodType - Un metodo HTTP tra: GET | POST | DELETE.
 * @param {{}} [urlParams={}] - Lista di parametri da concatenare all'url della richiesta
 * @param {string} apiEndpoint - L'endpoint dell'API da chiamare (verrà concatenato all'endpoint del server).
 * @param {Object} [body=null] - Body della richiesta se presente.
 * @returns {Promise<Object>} Oggetto della risposta API, se presente, altrimenti null
 */
async function executeApiCall(methodType, apiEndpoint, urlParams = {}, body = null) {
  //Mapping dei parametri nel formato param=value con concatenazione di multipli parametri con &
  var urlParamsString = Object.keys(urlParams)
    .map((key) => `${encodeURIComponent(key)}=${encodeURIComponent(urlParams[key])}`)
    .join("&");

  try {
    var url = API_BASE_URL + apiEndpoint;
    //All'url di base + api viene concatenata la stringa dei parametri, se presente
    if (urlParamsString != "") {
      url += "?" + urlParamsString;
    }

    const response = await fetch(url, {
      method: methodType.toUpperCase(),
      headers: {
        "Content-Type": "application/json",
      },
      body: body,
    });

    return await response.json();
  } catch (error) {
    return { code: 503, error_message: "Il server non risponde" };
  }
}
```

Inoltre, per evitare di dover inserire esplicitamente gli endpoint delle API in ogni pagina e facilitare le eventuali attività di manutenzione, il file riporta anche le costanti che mappano gli endpoint. In questo modo ogni pagina richiamerà la costante che rappresenta la chiamata API, astraendosi dal conoscere quale sia l'endpoint specifico utilizzato.

```
const API_BASE_URL = "http://127.0.0.1:5000/api/";
const API_GET_RESERVATION = "getReservation";
const API_GET_RESERVATION_LIST = "getReservationList";
const API_DELETE_RESERVATION = "deleteReservation";
const API_ADD_RESERVATION = "addReservation";
const API_UPDATE_RESERVATION = "updateReservation";
```

Di seguito un esempio di chiamata utilizzando il metodo custom e il mapping tramite costanti.

```
var response = await executeApiCall("GET", API_GET_RESERVATION_LIST, {"cf": cf, "ts": ts});

if (hasHttpSuccessCode(response.code) == false)
{
  pushNotification("error", response.error_message);
  $("#wDivResult").hide();
  return;
}
```

*Invocazione di Get Reservation List passando Codice Fiscale e Tessera Sanitaria nel body*

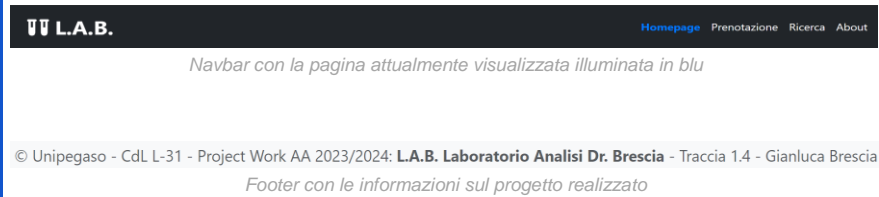
Infine, per fornire le funzionalità di controllo della risposta, è stato realizzato il metodo `hasHttpSuccessCode` (ispirato al medesimo controllo in C#) che verifica semplicemente se lo status code HTTP è considerabile come successo.

```
/**
 * Verifica se il codice HTTP indica un successo.
 *
 * @param {number} code - Codice di stato HTTP.
 * @returns {boolean} True se il codice indica successo, false altrimenti.
 */
function hasHttpSuccessCode(code) {
  return code >= 200 && code <= 299;
}
```

## Inject di Navbar e Footer

Nel progetto sono presenti una *navbar* e un *footer*.

La presenza di questi due elementi in ogni pagina ha reso necessario unificarne la generazione in modo da non dover riscrivere codice ridondante e centralizzarne eventuali modifiche.



A questo scopo è stato predisposto un handler nel file che, nel momento in cui la pagina è pronta per essere visualizzata, genera *navbar*, *footer* e illumina la voce relativa alla pagina attuale in base all'URL.

```
$(document).ready(function () {  
    //Creazione navbar  
    generateNavbar();  
  
    //Creazione footer  
    generateFooter();  
  
    var path = window.location.pathname;  
    var page = path.split("/").pop();  
  
    //Gestione link attivo sulla navbar basandosi sulla pagina attuale  
    $(".navbar-nav .nav-link").each(function () {  
        var href = $(this).attr("href");  
        if (href === page) {  
            $(this).parent().addClass("active");  
        } else {  
            $(this).parent().removeClass("active");  
        }  
    });  
});
```

### Validatori Codice Fiscale, Tessera Sanitaria e E-mail

Nel file sono presenti i validatori per il determinare se il codice fiscale, il codice della tessera sanitaria e-mail sono validi da un punto di vista strutturale e semantico.

Il risultato delle seguenti funzioni è un oggetto composto da un flag che indica lo stato dei controlli e da un messaggio di errore.

Il codice fiscale, come [documentato dall'Agenzia delle Entrate nel seguente allegato](#), è composto da 16 caratteri così suddivisi:

- tre caratteri alfabetici per il cognome;
- tre caratteri alfabetici per il nome;
- due caratteri numerici per l'anno di nascita;
- un carattere alfabetico per il mese di nascita;
- due caratteri numerici per il giorno di nascita ed il sesso;
- quattro caratteri, di cui uno alfabetico e tre numerici per il comune italiano o per lo Stato estero di nascita.
- sedicesimo carattere, alfabetico, ha funzione di controllo (checkdigit).

Di seguito la funzione JavaScript che realizza questo controllo:

```

//Validatore di codice fiscale
function validatorCF(cf) {
  //Il codice, di default, parte da uno stato di errore
  var validatorResult = { IsValid: false, Error: "il codice è errato" };

  if (cf.length !== 16) {
    validatorResult.Error = "il codice non è di 16 caratteri";
    return validatorResult;
  }

  //Verifica i primi sei caratteri alfabetici
  for (let i = 0; i < 6; i++) {
    if (!/[A-Z]/.test(cf.charAt(i))) {
      return validatorResult;
    }
  }

  //Verifica i successivi due caratteri numerici (anno nascita)
  for (let i = 6; i < 8; i++) {
    if (!/[0-9]/.test(cf.charAt(i))) {
      return validatorResult;
    }
  }

  //Verifica il carattere relativo al mese di nascita
  if (!/[A-Z]/.test(cf.charAt(8))) {
    return validatorResult;
  }

  //Verifica i due caratteri numerici del giorno di nascita
  for (let i = 9; i < 11; i++) {
    if (!/[0-9]/.test(cf.charAt(i))) {
      return validatorResult;
    }
  }

  //Verifica il carattere successivo come lettera relativa al sesso
  if (!/[A-Z]/.test(cf.charAt(11))) {
    return validatorResult;
  }

  //Verifica i successivi tre caratteri numerici
  for (let i = 12; i < 15; i++) {
    if (!/[0-9]/.test(cf.charAt(i))) {
      return validatorResult;
    }
  }

  //Verifica la presenza del check digit (non viene controllato se è corretto o meno)
  if (!/[A-Z]/.test(cf.charAt(15))) {
    return validatorResult;
  }

  validatorResult.IsValid = true;
  validatorResult.Error = null;
  return validatorResult;
}

```

La funzione non include il controllo sulla validità del checkdigit ma si limita a verificare la sua presenza.

Il codice della tessera sanitaria, rappresentato dall'ultimo campo in basso sul retro della tessera, ha una struttura che come [documentato dal Sistema Tessera Sanitaria nel seguente allegato al capitolo 1.5](#) è composto da 20 caratteri così suddivisi:

- valore 80 fisso per Assistenza Sanitaria
- codice stato di tre caratteri (380 per Italia)
- Due caratteri 00 seguiti da tre caratteri che indicano il codice Ente
- Nove caratteri numerici
- Ultimo carattere checkdigit



```
//Validatore di codice tessera sanitaria
function validatorTS(code) {
  //Il codice, di default, parte da uno stato di errore
  var validatorResult = { IsValid: false, Error: "il codice è errato" };

  if (code.length !== 20) {
    validatorResult.Error = "il codice non è di 20 caratteri";
    return validatorResult;
  }

  // Il primo carattere è 0
  if (code.charAt(0) !== "0") {
    return validatorResult;
  }

  // Codice Tipo Tessera (fisso "80" per prestazioni sanitarie)
  if (code.substring(1, 3) !== "80") {
    return validatorResult;
  }

  // Codice Stato: (fisso "380" per Italia)
  if (code.substring(3, 6) !== "380") {
    return validatorResult;
  }

  // Codice Ente: deve essere cinque cifre (primi due "00" seguiti dalle tre cifre specifiche della regione o ente)
  if (code.substring(6, 8) !== "00") {
    return validatorResult;
  }

  const validRegions = new Set([
    "010", "020", "030", "041", "042", "050", "060", "070", "080", "090", "100",
    "110", "120", "130", "140", "150", "160", "170", "180", "190", "200",
    "001", "002", "003", //enti speciali
  ]);
  let entity = code.substring(8, 11);
  if (!validRegions.has(entity)) {
    return validatorResult;
  }

  // I successivi 9 caratteri devono essere cifre
  if (!/^\d{9}$/.test(code.substring(11, 20))) {
    return validatorResult;
  }

  validatorResult.IsValid = true;
  validatorResult.Error = null;
  return validatorResult;
}
```

La funzione non include il controllo sul checkdigit ma si limita a verificare la sua presenza.

## Modal di conferma

Alcune azioni rendono necessaria l'esplicita conferma da parte dell'utente.

Bootstrap mette a disposizione una componente nota come *modal* che visualizza una finestra nella pagina.

Data la sua natura, il *modal* è un tag HTML e, come tale, prevalentemente statico; per non generare codice ridondante in ogni pagina, è stata implementata una funzione che crea un template del componente e, basandosi sui parametri di input, visualizza un messaggio, le azioni in caso di pressione del pulsante "Conferma" e del pulsante "Annulla".

Data la lunghezza della funzione si riporta solo l'intestazione commentata.

```
/**
 * Genera e visualizza un modal con un messaggio custom e due handler per i tasti CONFERMA e ANNULLA
 *
 * @param {string} message -- messaggio custom
 * @param {function} [onClickYes] -- handler al click su YES/CONFERMA (di default chiude il modal)
 * @param {function} [onClickNo] -- handler al click su NO/ANNULLA (di default chiude il modal)
 * @example
 *
 * Esempio di invocazione con due handler
 * generatePromptModal('Testo custom',
 * .....function(){
 * .....alert('Invocato handler CONFERMA');
 * .....},
 * .....function(){
 * .....alert('Invocato handler NO');
 * .....});
 */
function generatePromptModal(message, onClickYes, onClickNo) {
```

I parametri opzionali *onClickYes* e *onClickNo* sono degli handler che racchiudono l'invocazione ad una o più funzioni quando si clicca il pulsante corrispondente. Nell'esempio riportato verrà generato un alert che visualizza CONFERMA oppure NO in base al pulsante cliccato.

È compito della funzione chiamante valorizzare questi handler con le funzionalità specifiche rispetto all'azione da intraprendere.

## Application Layer

Espone un Web Server locale utilizzando il framework Flask per il linguaggio Python. Come configurazione default di debug verrà utilizzare la porta 5000 per rimanere in ascolto delle chiamate provenienti dal Front-End.

Gli endpoint sono descritti nel file *endpoints.py*; questo file non espone ancora la logica dietro alle chiamate API in quanto questo layer, per sua natura, si occupa solo di verificare che la chiamata e gli eventuali requisiti siano soddisfatti, trasferisce il controllo all'API specifica e incapsula la risposta in un oggetto che rappresenta lo stato della chiamata.

### Modello Response – Risposta HTTP

Come già discusso [nell'invocazione di chiamate API lato JavaScript](#) l'oggetto di risposta sarà composto da tre campi:

- *code*: indica il codice HTTP che rappresenta la risposta;
- *error\_message*: contiene un messaggio in caso di errore HTTP;
- *payload*: contiene il body della risposta (se previsto).

Questo wrapper è noto come *Response* (e referenziato con il nome di *HttpResponse*) ed è una classe custom realizzata per stabilire uno standard (basato sui principi REST) nella comunicazione Client-Server di questo progetto.

```
class Response:
    def __init__(self, code, error_message=None, payload=None):
        self.code = code
        self.error_message = error_message
        self.payload = payload

    # Converti da classe a json
    def to_json(self):
        return {
            "code": self.code,
            "error_message": self.error_message,
            "payload": self.payload
        }
```

La classe contiene un solo metodo che converte i campi in un oggetto JSON.

Di seguito un esempio di endpoint per la chiamata *getReservation* che, ottenendo in input tramite query string i parametri di id, codice fiscale e codice tessera sanitaria restituisce una prenotazione, se presente, altrimenti un errore di risorsa non trovata.

```
@app.route('/api/getReservation', methods=['GET'])
def get_reservation():
    try:
        id = request.args.get('id', type=str)
        cf = request.args.get('cf', type=str)
        ts = request.args.get('ts', type=str)
        reservation = svc_get_reservation(id, cf, ts)
        if reservation:
            return HttpResponse(200, payload=reservation.to_json().to_json())
        else:
            return HttpResponse(404, error_message="Prenotazione non trovata").to_json()
    except Exception as ex:
        return HttpResponse(500, error_message=f"{str(ex)}").to_json()
```

Tutti gli endpoint sono simili a quello visualizzato, in quanto conterranno un blocco *try...except* per gestire qualunque errore non previsto e fornire sempre una risposta standardizzata lato server.

## Business Layer

È responsabile della logica che realizza concretamente le funzionalità espone dal Web Server mediante servizi implementati in locale, nel file *services.py*.

Esiste un servizio specifico per ogni endpoint esposto, in aggiunta ad alcune funzionalità non esposte ma utilizzate nella classe (considerabili come metodi *privati*).

### Modello Reservation – Classe Prenotazione

Una prenotazione è composta dagli stessi campi visualizzati nel form, più un ID generato automaticamente come Guid e non modificabile, in modo da garantire l'univocità.

Sono presenti i campi Nome, Cognome, E-mail, Tessera Sanitaria, Codice Fiscale, Laboratorio e Lista Esami, quest'ultimo è rappresentato da un array composto dal tipo (categoria) e nome dell'esame.

Infine sono presenti tre date:

- data di prenotazione (la stessa indicata nel form)
- data di inserimento (prodotta la prima volta che si scrive il record)
- data di modifica (prodotta ogni volta che si invoca l'aggiornamento di una prenotazione)

La stessa logica dei validatori per Codice Fiscale, Tessera Sanitaria, E-mail e Data Prenotazione presenti in Front-End sono riportati nella classe Reservation. Per uniformare i punti in cui si leggono i dati o si scrivono nella classe sono stati creati dei metodi *getter* e *setter*, per ogni campo; dove previsto verranno invocati i validatori prima di assegnare il valore ad un campo.

Il fallimento di un validatore produrrà un'eccezione che, a sua volta, genererà una risposta con un codice di errore e l'azione non andrà a buon fine, restituendo il controllo all'utente; questo preserva l'integrità dei dati allineando i controlli con quelli presenti lato Front-End.

Di seguito un esempio di funzionamento dell'API *getReservation* che recupera tutti i dati dallo storage e seleziona l'istanza di Reservation che corrisponde alla tripla ID, Codice Fiscale e Tessera Sanitaria indicati in input.

```
def svc_get_reservation(id: str, cf: str, ts: str):
    """
    Recupera una reservation specifica dal disco basandosi sull'ID, il codice fiscale (CF) e il codice della tessera sanitaria (TS).

    Args:
        id (str): L'ID della reservation
        cf (str): Il codice fiscale
        ts (str): Il codice della tessera sanitaria

    Returns:
        Reservation: L'oggetto Reservation se trovato, altrimenti None.
    """
    cf = cf.upper()
    json_files = _get_data_from_disk()

    for json in json_files:
        if json["data"].get("id") == id and json["data"].get("cf") == cf and json["data"].get("ts") == ts:
            return Reservation().from_json(json["data"])

    return None
```

### Metodi Privati

Sono presenti alcuni metodi la cui natura è da considerarsi *privata* (ovvero solo i metodi interni alla classe potranno invocarli) seppur Python, di per sé, non supporti la classica visibilità dei metodi offerta da altri linguaggi di alto livello. Per questo motivo è stato adottato lo standard per cui i nomi dei metodi privati vengono preceduti dal simbolo underscore.

```
def _get_data_from_disk(): ...

def _write_data_on_disk(reservation: Reservation, file_path: str): ...

def _delete_data_from_disk(file_path: str): ...

def _get_file_unique_id():
    """
    Genera un ID unico basato su UUID1, che utilizza l'indirizzo MAC del computer, un timestamp e un numero casuale.

    Returns:
        str: ID univoco.
    """
    return str(uuid.uuid1())
```

I primi tre realizzano le funzionalità dedicate al recupero, al salvataggio e alla rimozione dei dati dallo storage definito nel [Data Layer](#), mentre *\_get\_file\_unique\_id()* è la generazione della stringa Guid che rappresenterà l'ID univoco della Prenotazione.

## Data Layer

Questo livello si occupa di recuperare i dati memorizzati o salvarne di nuovi, permettendo di astrarre il modo con cui si manipolano i dati persistenti rispetto alle funzioni definite nel [Business Layer](#).

I dati vengono memorizzati sul File System locale della macchina che ospita i servizi (che, nel caso specifico di questo progetto, coincide con il Web Server) e si utilizza la memorizzazione su file di tipo JSON.

Un JSON permette di memorizzare e manipolare facilmente dati con una struttura complessa come può esserla quella di una classe di tipo Reservation.

La cartella di storage è mappata in una costante, presente in cima al file `services.py`, garantendo che un'eventuale modifica al nome della cartella si rifletta concretamente sui metodi di lettura e scrittura.

```
# Crea un puntamento relativo alla cartella storage partendo dalla posizione del file attualmente in esecuzione
current_directory = os.path.dirname(os.path.abspath(__file__))
LOCAL_STORAGE_FOLDER = os.path.join(current_directory, 'storage')
```

Per gestire il percorso della cartella "storage" in modo relativo vi è una funzione che recupera il percorso della cartella in cui è presente il file attualmente in esecuzione (`services.py`) e lo concatena a "storage".

Quando un servizio ha necessità di leggere un file invoca `_get_data_from_disk()` che raccoglie tutti i file con estensione `.json` presenti nella cartella di storage e restituisce un array di oggetti composti da due campi: nome del file (`file_path`) e contenuto (`data`). Questo realizza concretamente la funzionalità di recupero di una singola Prenotazione o di una lista.

```
def _get_data_from_disk():
    """
    Recupera tutti i file JSON dalla cartella storage.

    Returns:
        list: Lista di dizionari contenenti, per ogni reservation, i dati della stessa e il path completo del file.
    """
    json_files = []

    # Creazione cartella storage al primo accesso in lettura/scrittura
    if not os.path.exists(LOCAL_STORAGE_FOLDER):
        os.makedirs(LOCAL_STORAGE_FOLDER)

    for filename in os.listdir(LOCAL_STORAGE_FOLDER):
        if filename.endswith('.json'):
            file_path = os.path.join(LOCAL_STORAGE_FOLDER, filename)
            with open(file_path, 'r', encoding='utf-8') as f:
                data = json.load(f)
                json_files.append({
                    'file_path': file_path,
                    'data': data
                })

    return json_files
```

Quando un servizio ha necessità di scrivere un file (che sia in creazione o in modifica) invoca `_write_data_on_disk()` passando come argomenti di funzione un'istanza di classe Reservation e il percorso del file da scrivere.

In questo modo, utilizzando la funzione `open` con flag 'w', si istanzia un puntatore in modalità *scrittura* verso il percorso indicato; se il file nel percorso indicato non esiste verrà creato, altrimenti verrà sovrascritto.

In questo modo di realizzano concretamente le funzionalità di creazione e aggiornamento di una Reservation.

```
def _write_data_on_disk(reservation: Reservation, file_path: str):
    """
    Scrive un JSON che rappresenta una reservation.

    Args:
        reservation (Reservation): L'oggetto Reservation da scrivere.
        file_path (str): Il percorso completo del file dove scrivere i dati.
    """
    with open(file_path, 'w', encoding='utf-8') as f:
        json.dump(reservation.to_json(), f)
```

Infine, quando un servizio ha necessità di rimuovere una prenotazione invoca `_delete_data_from_disk()` passando come argomento di funzione il percorso completo del file.

	<p>Così come descritto nella funzione, è stato creato un flag per verificare che il file esista o meno in quanto <code>os.remove()</code> non fornisce direttamente un feedback sull'esito dell'azione; inoltre questo garantisce che non si sollevi un'eccezione nel caso in cui il file non esista. Il medesimo flag verrà utilizzato per gestire la risposta a seguito una richiesta di rimozione file da parte dei servizi.</p> <pre>def delete_data_from_disk(file_path: str):     """     Rimuove un file dal disco.     Nota: la os.remove() di per sé non restituisce alcun risultato, rendendo ambiguo il risultato della cancellazione.     A tale scopo vi è un flag che controlla l'esistenza del file prima di rimuoverlo e la funzione restituisce il valore del flag.      Args:         file_path (str): Il percorso completo del file da rimuovere.      Returns:         bool: flag che indica se il file esiste ed è stato cancellato.     """     is_success = False     if os.path.exists(file_path):         os.remove(file_path)         is_success = True     return is_success</pre> <p>Il formato di una classe Reservation convertito in JSON sarà il seguente:</p> <pre>{   "id": "ac073a96-4d91-11ef-9f36-14cb19887d48",   "nome": "Gianluca",   "cognome": "Brescia",   "email": "email@mail.com",   "dataOraInserimento": "2024-07-29T12:02:32",   "dataOraModifica": null,   "dataOraPrenotazione": "2023-07-30T12:02",   "cf": "BRSGLC96B09B5060",   "ts": "80380001101234567890",   "laboratorio": "Lecce - Via Rossi 10",   "listaEsami": [{"tipo": "Sangue", "nome": "profilo-lipidico"}, {"tipo": "Urine", "nome": "proteinuria"}]</pre>
Campi di applicazione:	<p>Il progetto L.A.B. mira ed essere una piattaforma online dedicata alla gestione delle prenotazioni degli esami disponibili nel catalogo e la sua presenza online è simbolo di crescita ed espansione dei servizi offerti dall'azienda.</p> <p>Il design e l'interfaccia utente offrono la possibilità di ricercare, modificare e cancellare le prenotazioni in modo semplice e veloce; inoltre la piattaforma è disponibile su qualunque dispositivo hardware dotato di accesso alla rete internet, adattandosi alle dimensioni dei differenti schermi per garantire un corretto utilizzo a prescindere dalla natura del dispositivo utilizzato.</p> <p>Da un punto di vista grafico la piattaforma mantiene un certo grado di semplicità: questo fornisce un ambiente senza eccessi di colori o effetti grafici in modo tale da enfatizzare l'usabilità anche per gli utenti che non usano regolarmente le piattaforme web.</p> <p>La palette cromatica si orienta su colori scuri tendenti al grigio e al blu, che rappresentano i colori primari dell'applicativo.</p> <p>Sono state divise le categorie di esami con ulteriori tre colori, in modo da evidenziare la natura della categoria stessa quando si è in fase di compilazione della prenotazione.</p> <p>Oltre ai benefici per gli utenti meno esperti, l'applicativo ha pubblicizzato i propri servizi direttamente nella homepage che funge da vetrina per descrivere le potenzialità dell'azienda.</p> <p>TODO: rivedi</p>
Valutazione dei risultati (potenzialità e criticità):	<p>Il progetto presenta, oggettivamente, alcuni aspetti che possono essere migliorati.</p> <p>La presenza di validatori sia Front-End (prima di inviare i dati) sia Back-End (nel momento in cui si assegnano i dati ad un oggetto Reservation) è una tecnica normalmente utilizzata nei team di sviluppo.</p> <p>Questa dualità di controlli data l'attuale realizzazione ha degli svantaggi noti:</p> <ul style="list-style-type: none"> <li>• <b>Codice duplicato e sincronizzazione:</b> il codice viene scritto due volte e in linguaggi differenti, pur mantenendo la stessa logica; se vengono introdotte nuove regole vi è la necessità di allineare entrambi;</li> <li>• <b>Aumento del rischio di errori:</b> due implementazioni separate delle stesse regole possono introdurre errori o incongruenze;</li> </ul> <p>Il problema può essere mitigato introducendo dei file condivisi che contengono le regole di validazione, facendo uso di espressioni regolari (<i>regex</i>) che sono ampiamente supportate dai linguaggi di programmazione, centralizzando le modifiche in una sola sorgente di verità.</p>

La soluzione può essere integrata tramite *middleware* tra i layer Presentation e Application, riducendo il round-trip, il numero e la dimensione delle richieste che necessitano di uno scarto a seguito di incongruenze.

Il secondo problema, dettato perlopiù dalla necessità accademica del progetto realizzato, riguarda l'utilizzo di un sistema di persistenza dei dati basato su semplici file JSON.

Introdurre un Database relazionale per il salvataggio dei dati porterebbe notevoli vantaggi in quanto la presenza di un DBMS garantirebbe (grazie alle proprietà A.C.I.D.) una persistenza dotata di controlli sull'integrità dei dati, sistemi di backup, ripristino e gestione della concorrenza.

Inoltre l'utilizzo di *query* garantirebbe un recupero di dati in maniera efficiente e basata su attributi ben noti (id, codice fiscale, tessera sanitaria) invece di dover caricare in memoria tutti i file .JSON e filtrarli sulla base delle condizioni dettate dai servizi.

Un ultimo aspetto, relativo ad una migliore user experience e ad un servizio più efficiente, riguarda l'introduzione di un sistema di pagamento online per le prestazioni erogate.

Alcuni gateway di pagamento come Paypal o Nexi sono già ampiamente supportati da librerie ufficiali per i controlli lato back-end, riducendo gli sforzi necessari per l'integrazione in un progetto già avviato.