

Gestione Campo Estivo 2.0

Sommario

1	Introduzione	4
1.1	Informazioni sul progetto	4
1.2	Abstract	4
1.3	Scopo	5
2	Analisi	6
2.1	Analisi del dominio	6
2.2	Analisi e specifica dei requisiti	6
2.3	Use case	11
2.4	Pianificazione	12
2.5	Analisi dei mezzi	13
2.5.1	Software	13
2.5.2	Hardware	13
3	Progettazione	14
3.1	Design dell'architettura del sistema	14
3.2	Design dei dati e database	14
3.3	Design delle interfacce	17
3.3.1	Interfacce pubbliche	17
3.3.2	Interfacce gestione	19
3.3.3	Interfacce amministrazione	20
3.4	Design procedurale	22
3.4.1	Diagramma di flusso ospite	22
3.4.2	Diagramma di flusso volontari ed infermieri	23
3.4.3	Diagramma di flusso amministratori	24
4	Implementazione	25
4.1	Il Database	25
4.1.1	Tabella Person	25
4.1.2	Entità legate a ospiti	25
4.1.3	Entità legate a volontari	27
4.1.4	Entità legate al campo	28
4.2	Sito web	29
4.2.1	Creazione delle views	29
4.3	Laravel	30
4.3.1	Setup del progetto Laravel	30
4.3.2	Trasferimento delle views	31
4.3.3	Login in Laravel	32
4.4	Realizzazione del progetto	34
4.4.1	Home principale	34
4.4.2	Home campo	34
4.4.3	Registrazione e login	37
4.4.4	Registrazione al campo	37
4.4.5	User page	39
4.4.6	Pagina esplorativa	44
4.4.7	Pagine esplorativa utenti	50
4.4.8	Gestione disponibilità	53
4.4.9	Gestione alloggi	59
4.4.10	Iscrizione volontari	61
5	Test	62
5.1	Protocollo di test	62
5.2	Risultati test	67
5.3	Mancanze/limitazioni conosciute	68
6	Consuntivo	69
7	Conclusioni	70
7.1	Sviluppi futuri	70
8	Bibliografia	71
8.1	Sitografia	71
9	Glossario	72
10	Allegati	73

Indice delle figure

Figura 1 Use Case	11
Figura 2 Gantt preventivo	12
Figura 3 ER - Versione 1	14
Figura 4 ER - Versione 2	16
Figura 5 <i>Home page</i> sito.....	17
Figura 6 <i>Home page</i> campo	18
Figura 7 Pagina utente personale.....	18
Figura 8 Pagina esplorativa	19
Figura 9 Pagina utente per infermiere	19
Figura 10 Pagina utente personale admin.....	20
Figura 11 Pagina modifica campo	20
Figura 12 Pagina disponibilità.....	21
Figura 13 Pagina utente amministratore.....	21
Figura 14 Diagramma di flusso ospiti.....	22
Figura 15 Diagramma di flusso volontari e infermieri.....	23
Figura 16 Diagramma di flusso amministratori	24
Figura 17 Struttura codice.....	30
Figura 18 Struttura sito	31
Figura 19 Home - 1	34
Figura 20 Home - 2	34
Figura 21 Home - 3	34
Figura 22 Home - 4	35
Figura 23 Home - 5	35
Figura 24 Scelta ruolo.....	37
Figura 25 User Page - 1.....	40
Figura 26 Modifica dati.....	40
Figura 27 Esplora.....	44
Figura 28 Alloggi	59

1 Introduzione

1.1 Informazioni sul progetto

In questo capitolo raccogliere le informazioni relative al progetto, ad esempio:

- Allievi coinvolti nel progetto: Michea Colautti
- Classe: I4AA Scuola Arti e Mestieri Trevano
- Docenti responsabili: Guido Montalbetti
- Data inizio: 12 dicembre 2022.
- Data di fine: 06 aprile 2023
- Linguaggio: PHP
- Framework: Laravel 9

1.2 Abstract

Questo progetto è un'evoluzione di un progetto già svolto nel primo semestre, ma l'obiettivo e i requisiti rimangono gli stessi: creare un sito facile ed efficiente per la gestione di un campo estivo dedicato alle persone anziane o con disabilità, offrendo loro un'esperienza sicura e accogliente. Il sito consentirà di inserire i dati personali degli utenti insieme a certificati medici e dichiarazioni di salute, semplificando il lavoro del personale del campo. Oltre all'accoglienza, il sito si occuperà anche della gestione delle cucine, con la possibilità per gli ospiti di segnalare allergie e intolleranze, e del servizio di lavanderia e trasporto da e per il campo. Il personale medico del campo potrà accedere ad un'interfaccia per consultare i dati sanitari degli utenti, tra cui malattie, prescrizioni mediche e allergie.

Inoltre, rispetto al primo progetto offre anche nuove funzionalità, come la gestione degli eventi organizzati dal campo e la gestione/pianificazione degli alloggi per gli ospiti e i volontari. In questo modo, il sito web diventa un punto di riferimento completo per la gestione del campo estivo, in grado di offrire una piacevole e sicura esperienza di campagna a tutti gli ospiti.

This project is an evolution of a project already carried out in the first semester, but the objective and requirements remain the same: to create an easy and efficient site for the management of a summer camp dedicated to elderly or disabled persons, offering them a safe and welcoming experience. The site will allow users' personal data to be entered along with medical certificates and health declarations, simplifying the work of the camp staff. In addition to reception, the site will also manage the kitchens, with the possibility for guests to report allergies and intolerances, and the laundry and transport service to and from the camp. The camp's medical staff will be able to access an interface to consult users' health data, including illnesses, prescriptions and allergies.

Compared to the first project, it also offers new functionalities, such as the management of events organised by the camp and the management/planning of accommodation for guests and volunteers. In this way, the website becomes a complete reference point for the management of the summer camp, offering a pleasant and safe country experience to all guests.

1.3 Scopo

Lo scopo del lavoro rimane pressoché invariato a quello del primo progetto: creare un sistema gestionale per un campo estivo dedicato a persone disabili e/o anziane. Il gestionale sarà costituito da un sito web che sarà disponibile online entro due mesi dall'avvio del progetto. Il sito sarà utile sia per gli amministratori del campo che per gli utenti stessi.

Una volta raggiunto il sito, gli utenti troveranno una homepage con le informazioni sul campo e un collegamento al "Centro Diurno" della Fondazione Vita Serena, l'organizzazione che organizza il campo. Sarà possibile registrarsi o effettuare il login al sito.

Una volta effettuato l'accesso, gli utenti potranno procedere con l'iscrizione al campo estivo. Gli utenti già iscritti avranno accesso alle informazioni del campo e ad una lista degli utenti già registrati. La lista degli utenti sarà filtrabile e ordinabile tramite una serie di attributi come nome, cognome, mansione, ecc.

Le informazioni alle quali gli utenti potranno accedere dipenderanno dal loro ruolo:

- Gli ospiti potranno vedere solo i dati anagrafici degli altri utenti;
- I volontari potranno vedere i dati anagrafici di tutti gli utenti e le intolleranze alimentari, questo tornerà utile per poter meglio gestire la cucina;
- I volontari con conoscenze infermieristiche potranno vedere e modificare tutti i dati degli utenti, compresi quelli medici;
- Gli amministratori potranno vedere e modificare tutti i dati degli utenti, ad eccezione dei dati medici che saranno riservati al personale medico o ai volontari abilitati.

Se gli utenti non sono ancora iscritti al campo, sul sito sarà presente un link al form di iscrizione. Qui sarà possibile selezionare se si vuole partecipare come ospite o come volontario. Gli ospiti dovranno compilare tutti i campi richiesti, compresi i formulari relativi allo stato di salute e ai farmaci assunti.

Per i volontari minorenni, sarà obbligatorio compilare un certificato medico sullo stato di salute. Per i volontari maggiorenni, il certificato medico sarà facoltativo ma consigliato. In seguito alla compilazione del form, i volontari potranno scegliere la mansione preferita o lasciare questo campo in bianco e successivamente discutere con l'amministratore per l'assegnazione della mansione.

Gli account degli amministratori saranno creati da altri amministratori e questi potranno anche modificare le informazioni riguardanti i campi precedenti e quello attuale. Potranno inoltre copiare la documentazione degli scorsi campi nella pagina del nuovo campo o apportare eventuali modifiche.

2 Analisi

2.1 Analisi del dominio

Ad oggi esistono molti campi estivi. Tuttavia, poter beneficiare un'esperienza "personalizzata" come quella che viene offerta nel campo estivo di questo progetto appare importante. Trovo infatti che persone anziane o disabili spesso non possano fare tutto quello che una persona in salute può fare, spesso per motivi puramente pratici: a volte capita infatti che per semplici difficoltà organizzative si chiudano molte porte a persone più fragili per ragioni anagrafiche o per disabilità. Tramite quest'evoluzione del mio progetto spero che esse avranno la possibilità di partecipare ad un campo estivo attento ai loro problemi e alle loro necessità. I formulari di iscrizione e la presenza online di una pagina dedicata riguardante lo stato di salute (che sarà visibile al personale medico) permetteranno un'esperienza sicura e piacevole, che idealmente li metterà al riparo da cattive sorprese. Oltre alla parte riguardante l'aspetto medico, il mio progetto permetterà anche di visualizzare una pagina in cui saranno presenti tutti gli ospiti e, per ognuno di essi, eventuali intolleranze o esigenze alimentari. Gli utenti saranno quindi infermieri, ospiti, e volontari: questi ultimi aiuteranno nella gestione interna del campo, ma anch'essi potranno segnalare eventuali allergie nel processo di iscrizione, se fosse necessario. Nel sito da me creato ovviamente anche gli amministratori potranno interfacciarsi. Il sito permetterà agli amministratori di evitare di avere a che con certificati cartacei e formulari riempiti in modo errato, permettendo una gestione molto più comoda e veloce. La presenza di funzioni come ad esempio potere ordinare gli ospiti in base a più criteri, sarà un passo avanti rispetto al passato.

2.2 Analisi e specifica dei requisiti

ID: REQ-001	
Nome	Creazione pagina base
Priorità	1
Versione	1.0
Note	Creazione della pagina home con collegamento a fondazione.
Sotto requisiti	
001	Mostrare anteprima informazioni campo estivo corrente

ID: REQ-002	
Nome	Creazione maschera registrazione
Priorità	1
Versione	1.0
Note	Collegamento in schermata home

ID: REQ-003	
Nome	Creazione maschera <i>login</i>
Priorità	1
Versione	1.0
Note	Collegamento in schermata home

ID: REQ-004	
Nome	Creazione Form iscrizione al campo.
Priorità	1
Versione	1.0
Note	Un ospite o un volontario, specificando il loro ruolo, si iscrivono al campo fornendo tutti i dati.
Sotto requisiti	
001	L'account rimane in sospeso fino all'approvazione da parte di un amministratore.

ID: REQ-005	
Nome	Creazione pagina personale utente loggato iscritto al campo
Priorità	1
Versione	1.0
Note	Una pagina dove l'utente ha la possibilità di vedere e modificare i suoi dati inerenti al campo.

ID: REQ-006	
Nome	Creazione pagina esplorativa.
Priorità	1
Versione	1.0
Note	Un utente può vedere, oltre ai dati sul campo, i dati anagrafici delle altre persone al campo. Con la loro mansione.
Sotto requisiti	
001	Lista con utenti, filtrabili per nome, cognome, funzione.
002	Ogni utente ha una sua pagina, cliccando sull'utente viene aperta. Contiene tutte le informazioni che l'ospite ha il permesso di vedere.

ID: REQ-007	
Nome	Creazione pagine di modifica utenti.
Priorità	1
Versione	1.0
Note	<p>Un utente, selezionando un utente dalla lista, ne visualizza i dati.</p> <p>Se è un utente "infermiere" visualizza i dati sanitari</p> <p>Se è un utente "admin" può modificare i dati anagrafici.</p>

ID: REQ-008	
Nome	Creazione pagine di modifica dati campo per amministratore.
Priorità	1
Versione	1.0
Note	Un amministratore, può modificare i dati sul campo corrente o su quelli passati. Inoltre può creare una nuova edizione del campo, inserendo tutte le informazioni necessarie.

ID: REQ-009	
Nome	Creazione pagina per la creazione di un utente da parte di un amministratore.
Priorità	1
Versione	1.0
Note	Un amministratore può autonomamente creare un account per un ospite o un infermiere.

ID: REQ-010	
Nome	Creazione funzione di approvazione di un utente.
Priorità	1
Versione	1.0
Note	Un amministratore può attivare o eliminare l'account di un utente.

ID: REQ-011	
Nome	Pagina per visualizzare disponibilità volontari
Priorità	1
Versione	1.0
Note	Un amministratore può visualizzare la disponibilità dei suoi volontari

ID: REQ-012	
Nome	Pagina per gestione eventi
Priorità	1
Versione	1.0
Note	Un amministratore può pianificare degli eventi. Su un calendario saranno mostrati

ID: REQ-013	
Nome	Pagina per gestione alloggi
Priorità	1
Versione	1.0
Note	Un amministratore può gestire gli alloggi di ospiti e volontari

ID: REQ-014	
Nome	Sito responsive
Priorità	2
Versione	1.0
Note	-

ID: REQ-014	
Nome	Log
Priorità	2
Versione	1.0
Note	Usati i log di Laravel in fase di debug. Possibile implementazione di log ma non attuata

2.3 Use case

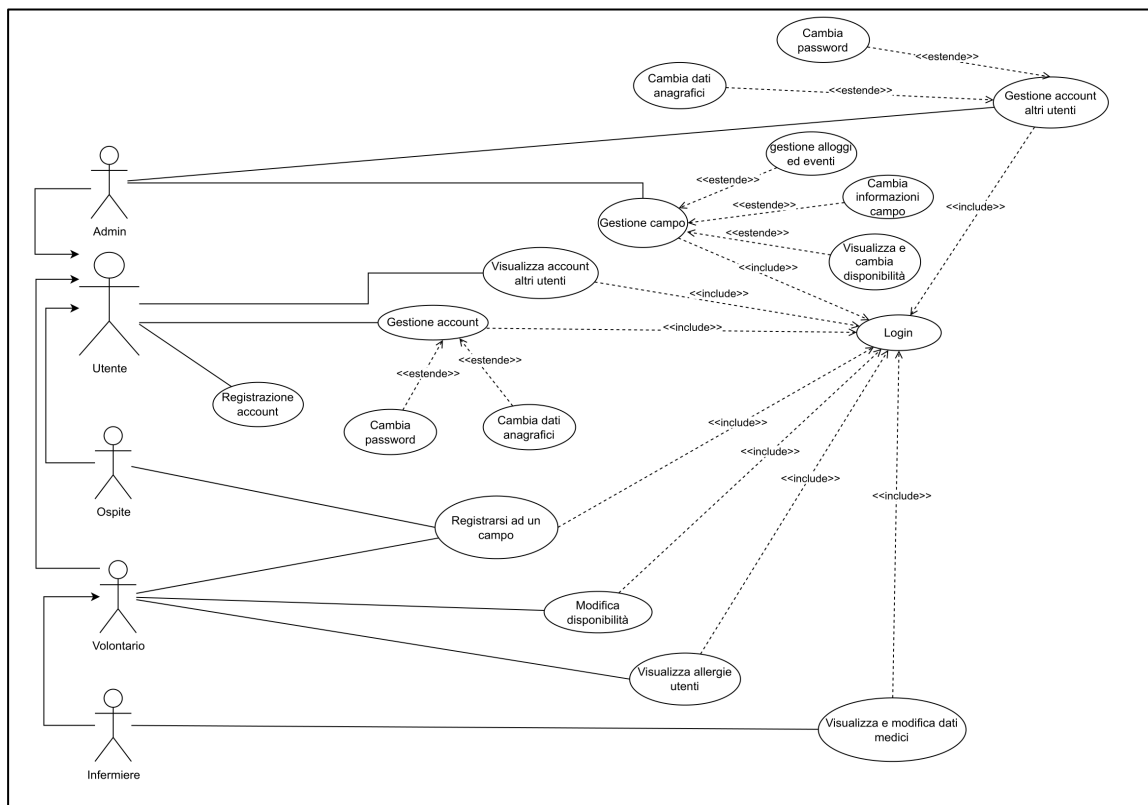


Figura 1 Use Case

Questo *Use Case* rappresenta tutte le funzioni della mia applicazione: l'utente base rappresentato "Utente", è soltanto una base per meglio rappresentare il tutto, non è dunque da considerare un utente finale. La prerogativa per buona parte delle funzioni è, logicamente, essere autenticati.

Essenzialmente tutti gli utenti del campo estivo possono fare tutto ciò che è collegato all'Utente: Con "gestione account" si intende una gestione completa, che permette di cambiare i dati anagrafici e la password. La registrazione account avviene una volta sola e differisce dalla registrazione al campo.

Sia l'ospite che il volontario -una volta che si sono autenticati- possono registrarsi al campo estivo.

Un volontario può modificare la sua disponibilità e visualizzare le allergie degli utenti.

Un infermiere, in aggiunta a tutte le funzioni del volontario, può gestire i dati medici di tutti gli utenti nel campo.

Un amministratore può gestire tutti gli account del sito e cambiare le informazioni del campo.

Può inoltre gestire gli alloggi e gli eventi.

2.4 Pianificazione

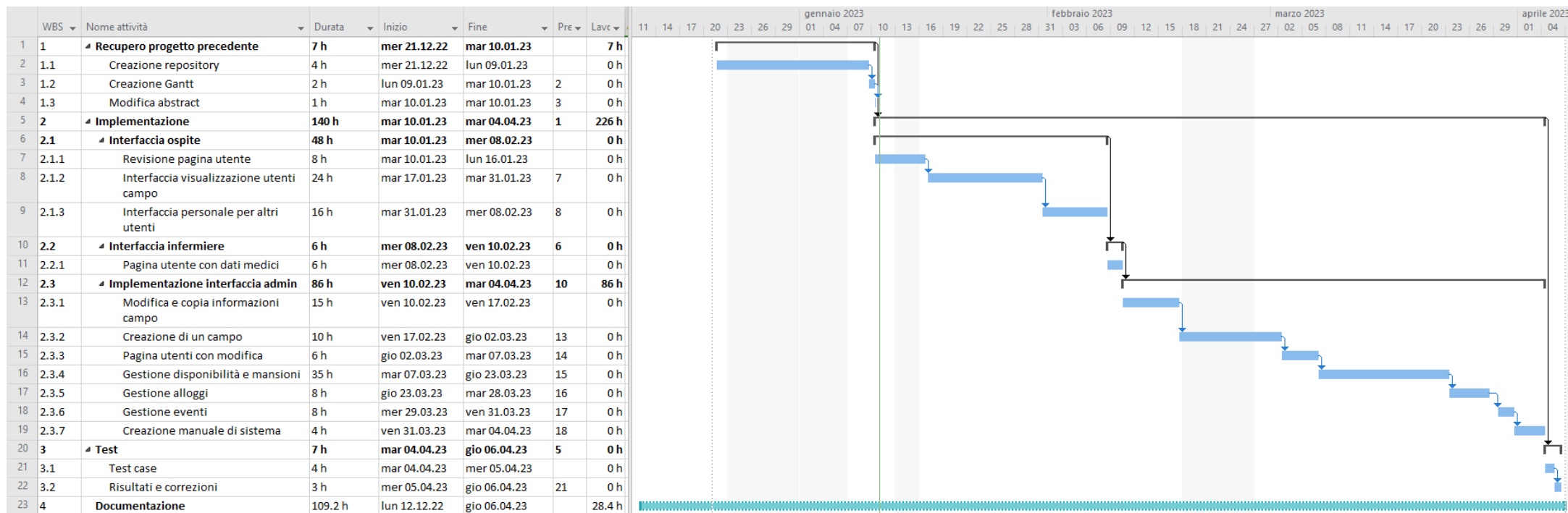


Figura 2 Gantt preventivo

2.5 Analisi dei mezzi

2.5.1 Software

- MockFlow 1.4.7
- Draw.io
- GitLab
- GitHub Desktop 2.9.12
- Laravel 9
- VS Code 1.7
- Nicepage desktop
- nicepage.com
- MS Office
- Fullcalendar 6
- MacDown 0.7.3

2.5.2 Hardware

- PC scolastico
 - Processore: Intel I7
 - Scheda grafica: Nvidia
 - RAM: 32GB
 - SW: Windows 10 Enterprise
- MacBook Air personale da remoto
 - Processore: 1.1 GHz Intel Core i3 dual-core
 - Scheda grafica Intel Iris Plus Graphics 1536 MB
 - RAM: 8 GB 3733 MHz LPDDR4X
 - SW: macOS Ventura 13.2.1

3 Progettazione

Il progetto parte come evoluzione del precedente, nell'architettura non è cambiato molto. Tuttavia ci sono dei dettagli che è opportuno riportare, oltre alla pianificazione completa.

3.1 Design dell'architettura del sistema

L'architettura del mio sito è relativamente semplice. Alla base c'è Laravel: un framework web per lo sviluppo in PHP, basato sul modello di sviluppo MVC. Laravel è stato creato per facilitare l'avvio e la gestione di applicazioni in PHP.

3.2 Design dei dati e database

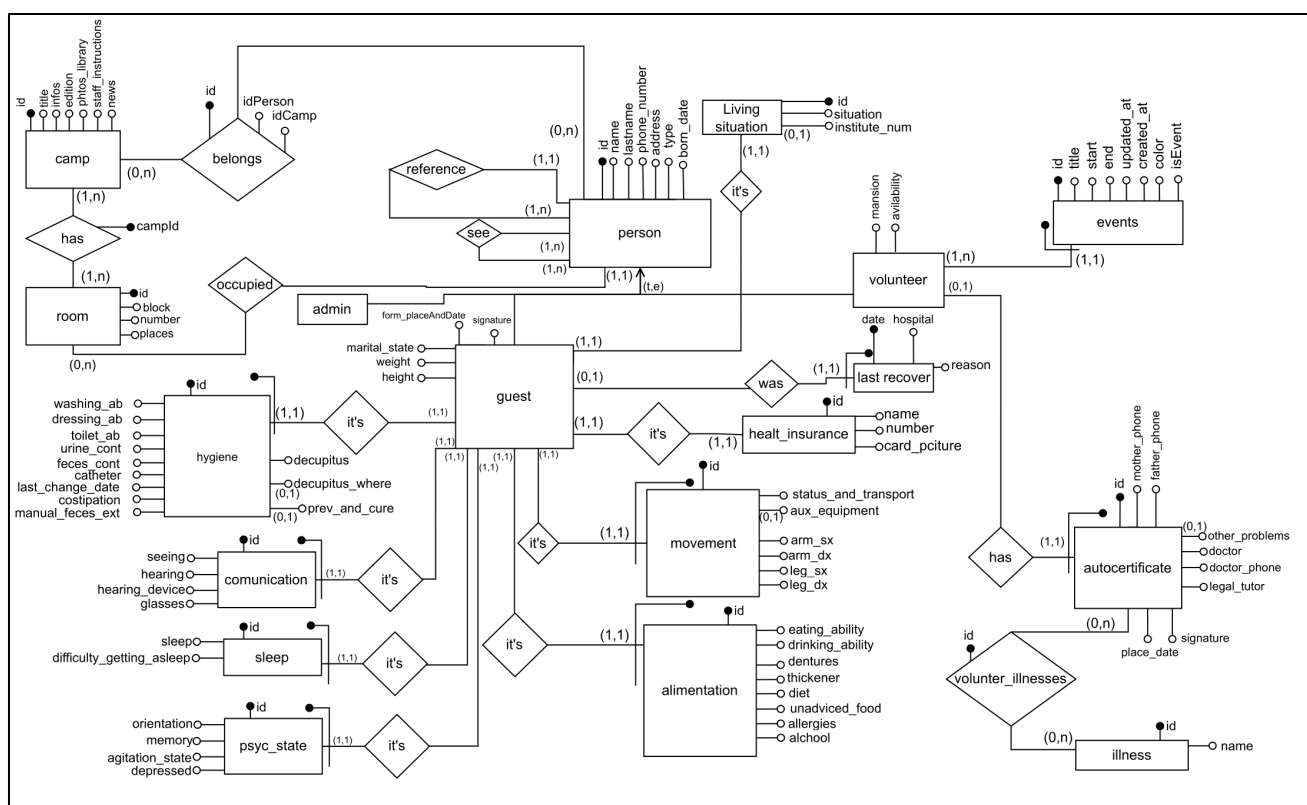


Figura 3 ER - Versione 1

Per realizzare lo schema ER ho studiato la procedura di iscrizione attuale tramite i documenti che mi sono stati forniti, e ho riflettuto su come avrei potuto implementare il salvataggio dei dati. Come si vede dallo schema, ho proceduto in maniera modulare. Inizialmente ho creato uno schema ER con le convenzioni standard della scuola. Ho riflettuto, in particolare, sul concetto di evitare di utilizzare tabelle eccessivamente grandi e con molti campi *null*. Ho quindi optato per generalizzare gli utenti in un'unica entità, **person**. La specializzazione "totale, esclusiva" indica che tutti gli utenti si devono classificare come una delle tre sotto entità elencate e che non possono essere due cose contemporaneamente.

Entità person

Questa è l'entità base, possiede alcuni campi che tutti gli utenti hanno in comune, come il nome, cognome, e id, che è la *primary key* per gli utenti nell'intero DB. Un altro campo importante è **type**, ovvero tipo. Questo dato serve a distinguere la tipologia di utente: **admin**, **guest**, **volunteer**.

Le uniche relazioni che coinvolgono l'entità **person** è **reference**, che indica la persona di riferimento, e se, che rappresenta la possibilità di tutti gli utenti di vedere qualunque altro utente.

Una persona è anche collegata ad un campo tramite la relazione **belongs**. Questo permette, a differenza della prima versione del progetto, di partecipare a più campi estivi.

Similmente al campo una persona è collegata a **room**, questa relazione è presente per memorizzare e rappresentare l'appartenenza di uno o più utenti ad una stanza.

Entità admin

L'amministratore è l'entità più semplice del DB: esso può -oltre alle funzioni basi di **person**- modificare i dati di un campo.

La relazione che lo permette è la stessa che collega la persona al campo.

Entità guest

L'ospite può sembrare un'entità complessa, ma in realtà, una volta capito il meccanismo, è abbastanza semplice.

Seguendo il modello del formulario di iscrizione, che è diviso a capitoletti in base a quello che viene chiesto, ho creato diverse tabelle che memorizzano tutti i dati degli utenti. Come si vede dall'identificatore esterno, tutte queste tabelle avranno come *foreign key* l'id del **guest**, che viene logicamente ereditato da **person**.

Ogni tabella memorizza quindi un set di dati che hanno un argomento comune. Le tabelle che usano questo meccanismo sono:

- *hygiene* → L'igiene personale dell'utente e le sue necessità nella pulizia personale e nelle funzioni corporali.
- *comunication* → Le sue abilità di comunicazione
- *psyc_state* → Lo stato psichico
- *alimentation* → Come l'ospite si alimenta e se ha necessità particolari
- *movement* → Le abilità motorie dell'ospite
- *last_recover* → Il suo ultimo eventuale ricovero in ospedale
- *healt_insurance* → Le informazioni sull'assicurazione sanitaria.

Entità autocertificate (gestione volontari)

Il volontario ha soltanto 2 campi in più rispetto all'entità **person**, tuttavia in maniera simile al **guest** ha anch'egli un'entità separata per memorizzare i dati medici.

Rispetto al primo progetto ho optato per un metodo di memorizzazione migliore. L'entità **autocertificate** ha ora pochi campi, quelli che sono legati per forza all'utente.

Le malattie e i problemi di cui un volontario può soffrire sono invece memorizzati in una tabella esterna. Il volontario, in fase di iscrizione, avrà la lista di problematiche fisiche, da cui potrà scegliere le opzioni che gli corrispondono. Una tabella ponte si occuperà di memorizzare tutte le relazioni

Entità camp

Il campo è l'entità più semplice del DB. Contiene semplicemente i dati di tutti i campi, e grazie alla relazione **belongs** permette l'associazione delle persone ad un campo.

Inoltre il campo è strettamente collegato alle stanze, ovvero all'entità **room**. Essa viene indentificata con un id, ma contiene pure il numero di stanza e il blocco, così come la capacità massima. Viene memorizzato anche l'id del campo per permettere più edizioni di anno in anno.

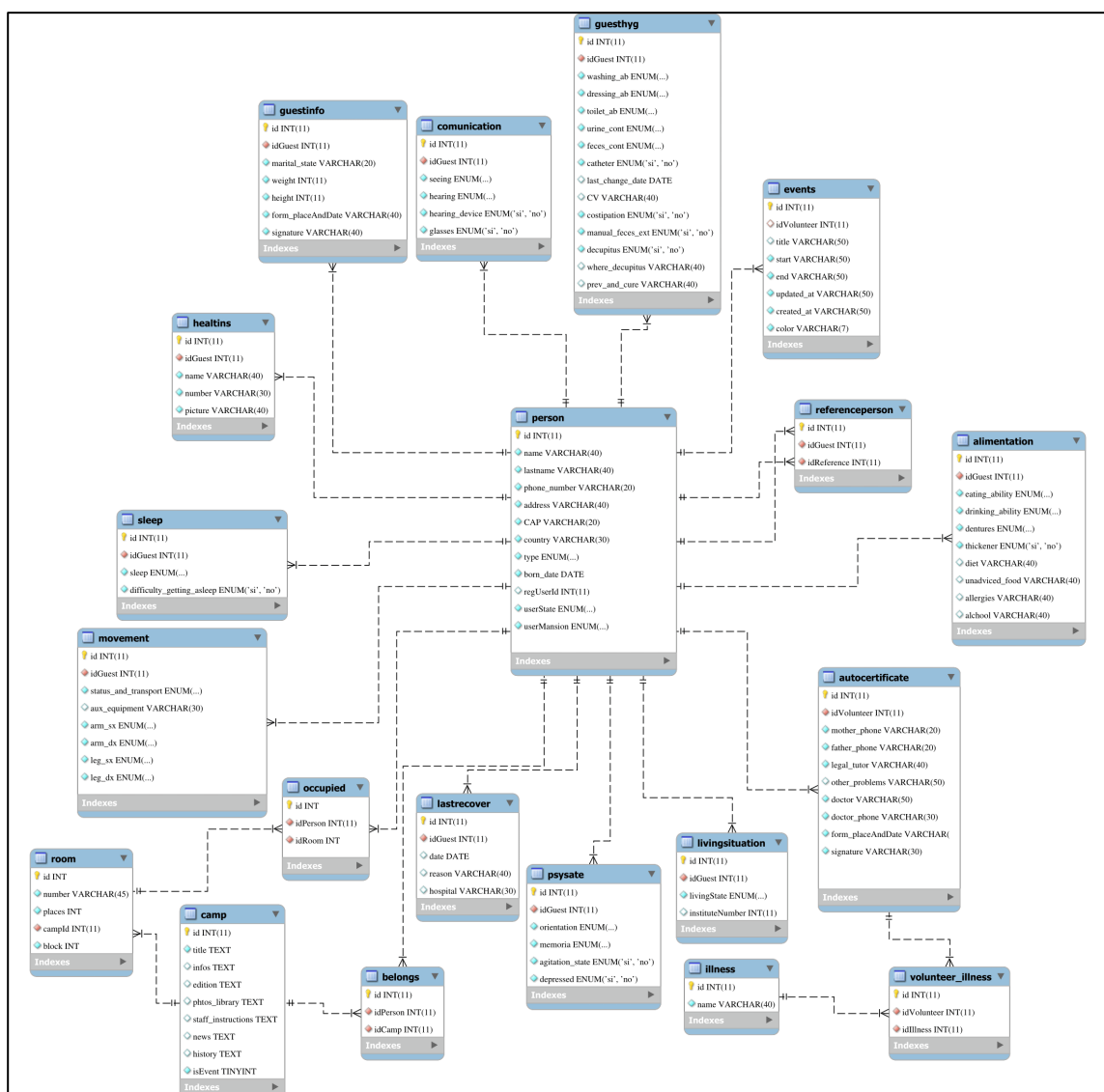


Figura 4 ER - Versione 2

Questo schema è molto più esplicativo per quanto riguarda la struttura “tecnica” del database.

Le entità sono diminuite, dato che la specializzazione non viene rappresentata. Tutti i campi presenti sull’entità **person** sono rimasti, e tutti gli altri sono stati divisi in tabelle.

Nelle tabelle di appoggio sono presenti degli id, come **idGuest** o **idVolunteer**. Questi non sono altro che *foreign key* che fanno riferimento all’*id* nella tabella **person**. Per il resto il DB rimane identico a prima.

Da notare che molti campi sono degli *enum*.

Ho deciso di adottare questo tipo di dato poiché nel *form* originale c’erano punti in cui bisognava scegliere tra una serie di opzioni: per esempio, nella sezione movimento bisognava selezionare per ogni arto il tipo di mobilità e le possibilità sono:

- Buona
- Ridotta
- Nessuna

Queste opzioni sono riportate nel campo *enum* da me creato.

Da notare che la tabella **guest** in questa versione non esiste. Infatti avendo risolto la specializzazione della prima versione ho dovuto rimuoverla. Al suo posto ho inserito **guestInfo**, che mi permette di immagazzinare le informazioni di base di tutti i *guest*; come il peso o l'altezza.

3.3 Design delle interfacce

3.3.1 Interfacce pubbliche

In questo capitolo sono raccolte le interfacce pubbliche o, in generale, accessibili e uguali per gli utenti nel campo. Ci sono casi in cui le interfacce cambiano leggermente in base all'utente autenticato.



Figura 5 Home page sito

Questa è la *home page* del sito internet. Come accennato nello scopo il progetto è per la fondazione “Vita Serena”, che gestisce anche un centro diurno. Per questo nella *home page* del sito ci devono essere due collegamenti, che portano al sito del Centro Diurno e al Campo estivo di Olivone.

Per il contenuto mi sono basato su quello che è attualmente il sito della fondazione, ma lo stile cambierà.

Questa è la *home page* del Campo Estivo.

Oltre ai collegamenti per le informazioni della fondazione, presenti anche nella home page generale, qui è possibile registrarsi o autenticarsi al sito tramite un *pop-up*.

Una volta registrati sarà possibile iscriversi al campo tramite un *form*.



Figura 6 Home page campo

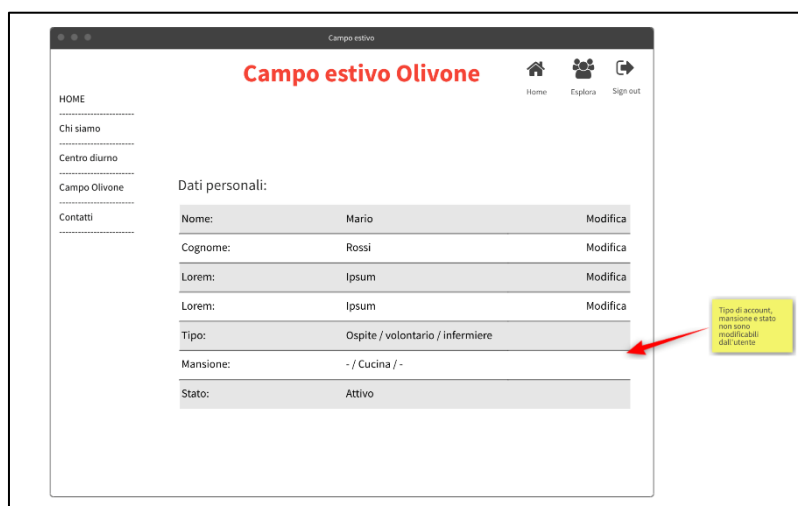


Figura 7 Pagina utente personale

Quando si è riempito il *form* di iscrizione si viene reindirizzati alla pagina persona. Qui l'utente può modificare i suoi dati personali. Gli unici campi non accessibili sono il tipo di account, la mansione e lo stato.

La mansione è un campo che viene riempito solo se l'utente è un volontario. Da questa pagina si può accedere alla sezione esplorativa del sito, ma solo se l'account è stato attivato.

Questa è la pagina esplorativa degli utenti del campo. È uguale per tutti.

In alto, con le icone, ci si può muovere nel sito. Solo l'amministratore avrà però accesso alla pagina dove sono elencate le disponibilità degli utenti.

Usando la barra di ricerca e il *pop-up* contenente i filtri si possono cercare uno o più utenti.

Cliccando sull'intestazione di una colonna i dati verranno ordinati in ordine crescente o per tipo. Cliccando nuovamente verranno ordinati al contrario.

Selezionando un utente si apre la sua pagina personale, che sarà praticamente identica a quella dell'account personale.

In base all'utente che la visualizza verranno mostrate più o meno informazioni.

- Un ospite vede solo i dati anagrafici.
- Un volontario, in aggiunta, vede le allergie.
- Un infermiere, in aggiunta, vede e può modificare i dati sanitari.
- Un amministratore vede e può modificare tutti i dati, **eccetto quelli sanitari, che sono nascosti**.

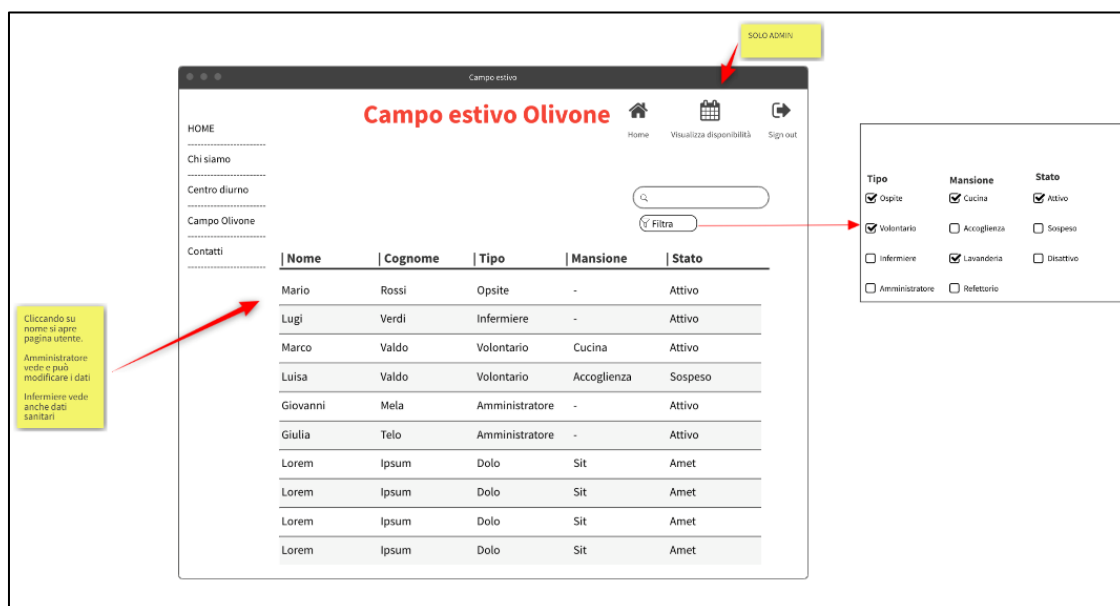


Figura 8 Pagina esplorativa

3.3.2 Interfacce gestione

In questo capitolo sono raccolte le interfacce visibili solo ad un infermiere.

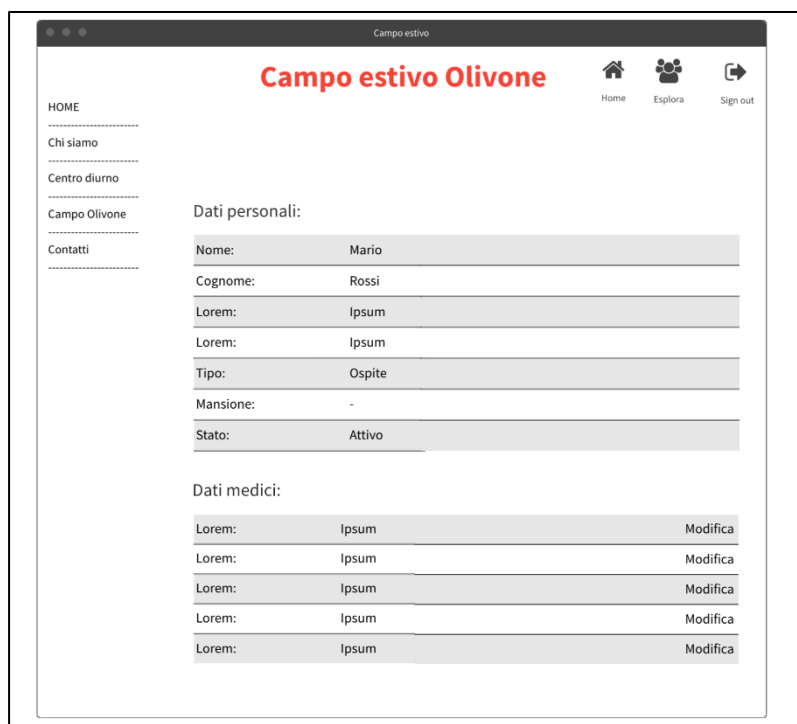


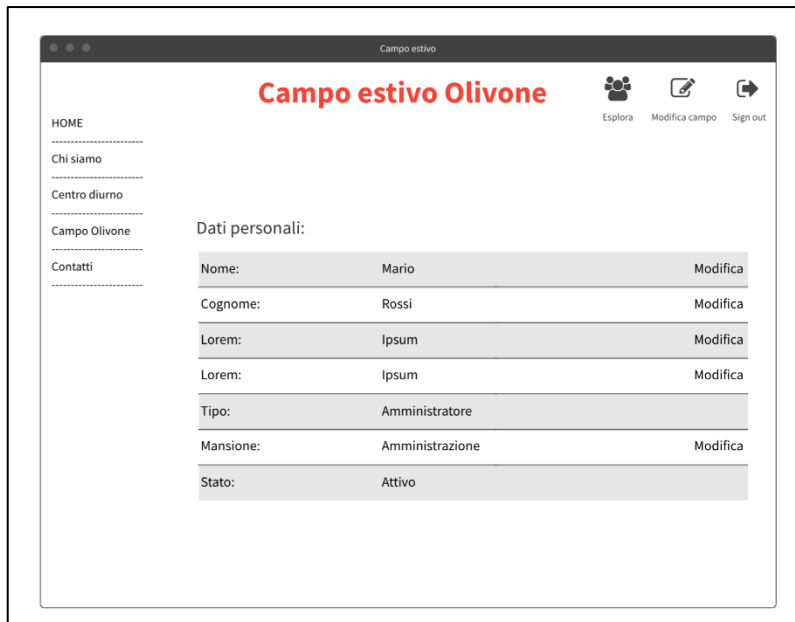
Figura 9 Pagina utente per infermiere

Come detto l'infermiere, oltre ai dati anagrafici, può vedere anche i dati sanitari. Questo è un esempio di come verranno mostrati i dati.

L'utente può muoversi nel sito usando la barra superiore.

3.3.3 Interfacce amministrazione

In questo capitolo sono raccolte le interfacce visibili solo ad un amministratore.



Questa è la pagina personale di un amministratore. Rispetto ad un utente normale l'admin ha un accesso quasi completo ai suoi campi: lo stato è bloccato per motivi di sicurezza, così come il tipo.

Inoltre con la barra superiore l'amministratore può accedere alla modifica dei dati del campo.

Figura 10 Pagina utente personale admin

Questa è la pagina appena menzionata. Qui l'amministratore può modificare le informazioni del campo.

Cliccando il pulsante "modifica", potrà modificare il testo direttamente sulla pagina.

Potrà inoltre caricare una nuova *brochure*, che andrà a sovrascrivere quella già presente.



Figura 11 Pagina modifica campo

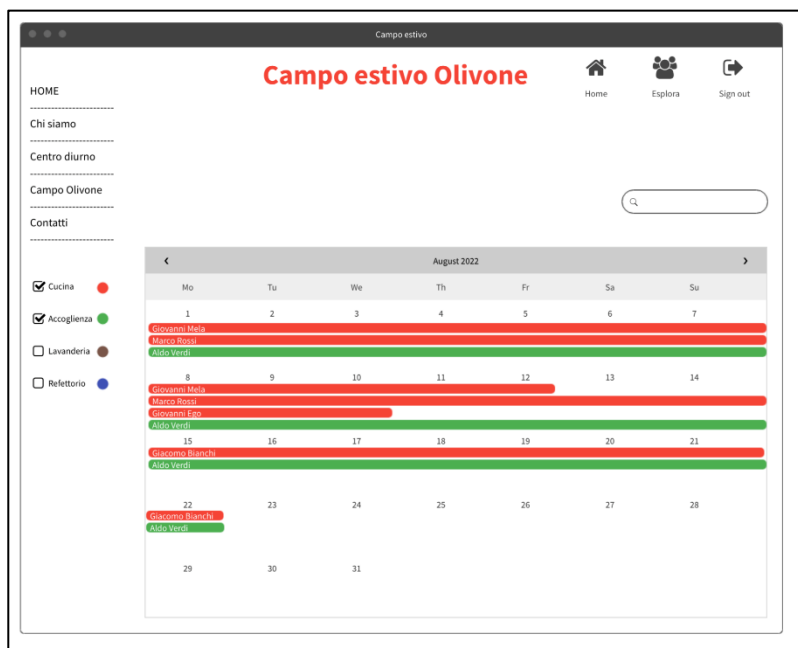
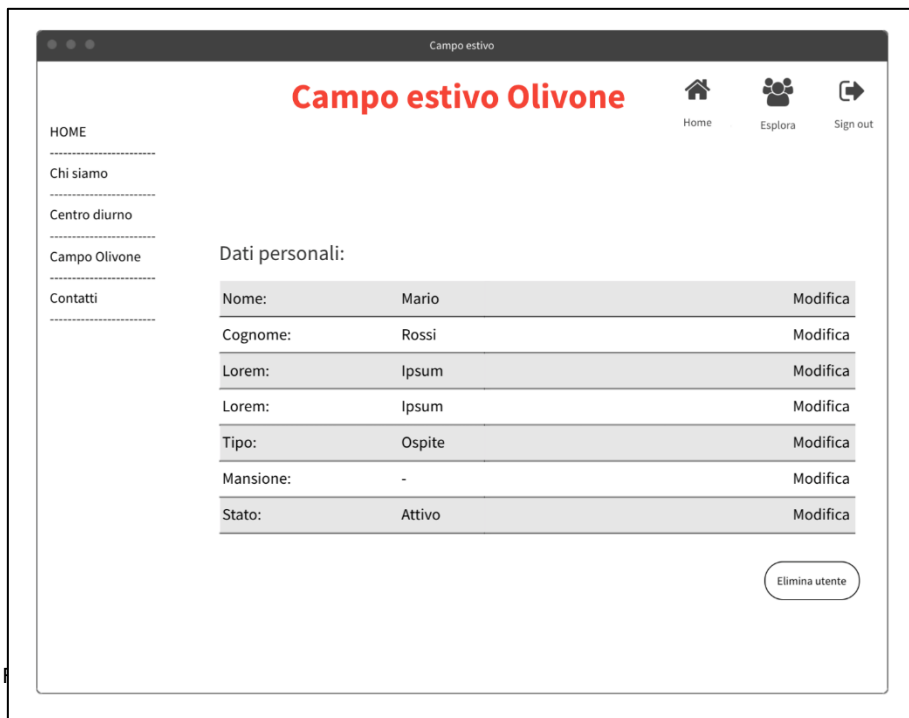


Figura 12 Pagina disponibilità

La pagina di disponibilità raggiungibile dalla pagina di esplorazione (Figura 8), permette all'admin di avere una visione di insieme delle disponibilità dei volontari.

Tramite i filtri a lato del calendario, è possibile visualizzare uno o più gruppi di volontari. È stato pensato per facilitare il compito all'amministratore che deve gestire quanti e quali persone sono assegnate alle varie mansioni.

Questo è un esempio di come un amministratore vede la pagina di un altro utente. Come si vede ha potere di modifica su tutti i campi, ma non vede i dati medici.



The screenshot shows the 'Campo estivo Olivone' user profile page. On the left, there's a sidebar with 'HOME' and a list of sections: 'Chi siamo', 'Centro diurno', 'Campo Olivone', and 'Contatti'. The main area displays 'Dati personali:' with a table of user information. Each row has a 'Modifica' button. At the bottom right, there is an 'Elimina utente' button.

Dati personali:		
Nome:	Mario	Modifica
Cognome:	Rossi	Modifica
Lorem:	Ipsium	Modifica
Lorem:	Ipsium	Modifica
Tipo:	Ospite	Modifica
Mansione:	-	Modifica
Stato:	Attivo	Modifica

Elimina utente

3.4 Design procedurale

Per questo capitolo ho voluto creare degli schemi di flusso. Uno solo non sarebbe bastato, infatti il campo prevede, come detto nei capitoli precedenti, l'accesso a più tipi di utenti: questo prevede diversi comportamenti da parte del sito.

Di seguito riporto gli schemi, dando una breve spiegazione.

3.4.1 Diagramma di flusso ospite

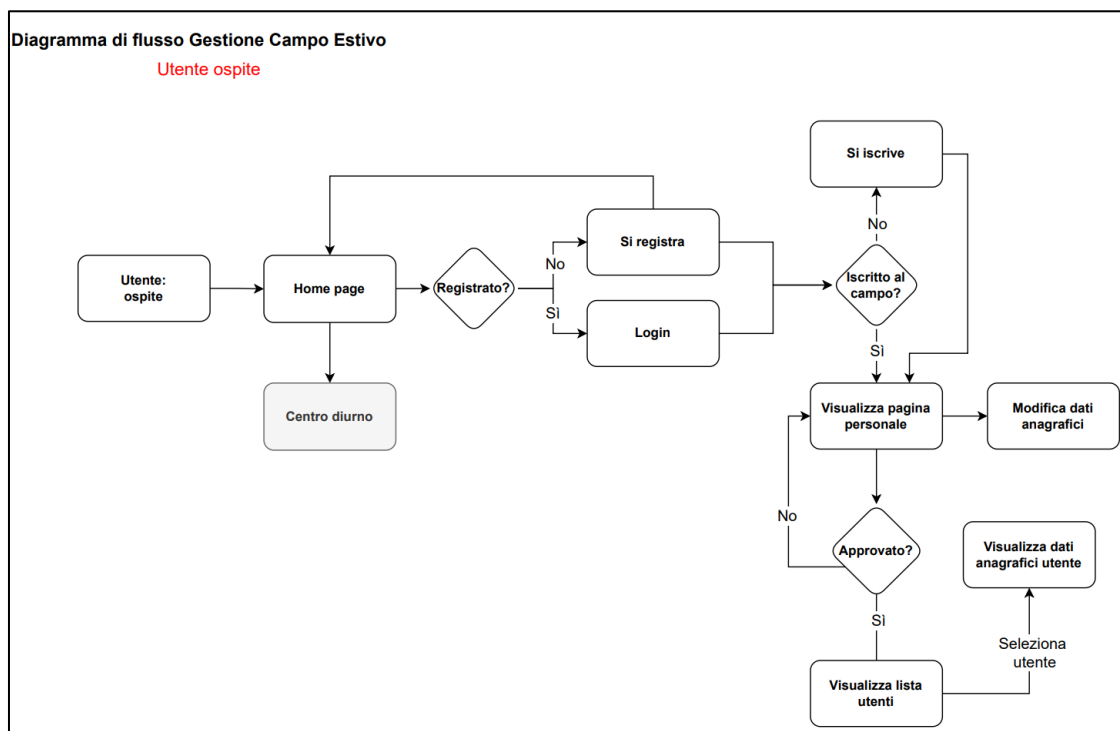


Figura 14 Diagramma di flusso ospiti

Il primo diagramma di flusso è quello che descrive l'uso del sito da parte di un ospite della fondazione.

Una volta raggiunta la *home page del campo estivo*, e non quella del sito in generale, l'ospite ha la possibilità di registrarsi o di effettuare il *login*.

Quando si sarà autenticato, se non lo ha già fatto, si può iscrivere al campo estivo presente sul sito. Poi viene reindirizzato alla sua pagina personale, dove può vedere e modificare i suoi dati.

Per ragioni di sicurezza il suo account rimane in sospeso, momentaneamente disattivato. Un amministratore, una volta eseguiti i controlli e gli accertamenti del caso, procederà alla sua attivazione. Ho pensato di implementare questo blocco poiché una volta approvato, l'account può visualizzare i dati anagrafici degli utenti del campo, perciò è opportuno che gli account vengano vidimati prima di avere questo accesso, per evitare utenti indesiderati con intenzioni maligne.

Quando l'account è stato approvato l'ospite può accedere alla lista degli utenti del campo, e visualizzarne i dati anagrafici.

3.4.2 Diagramma di flusso volontari ed infermieri

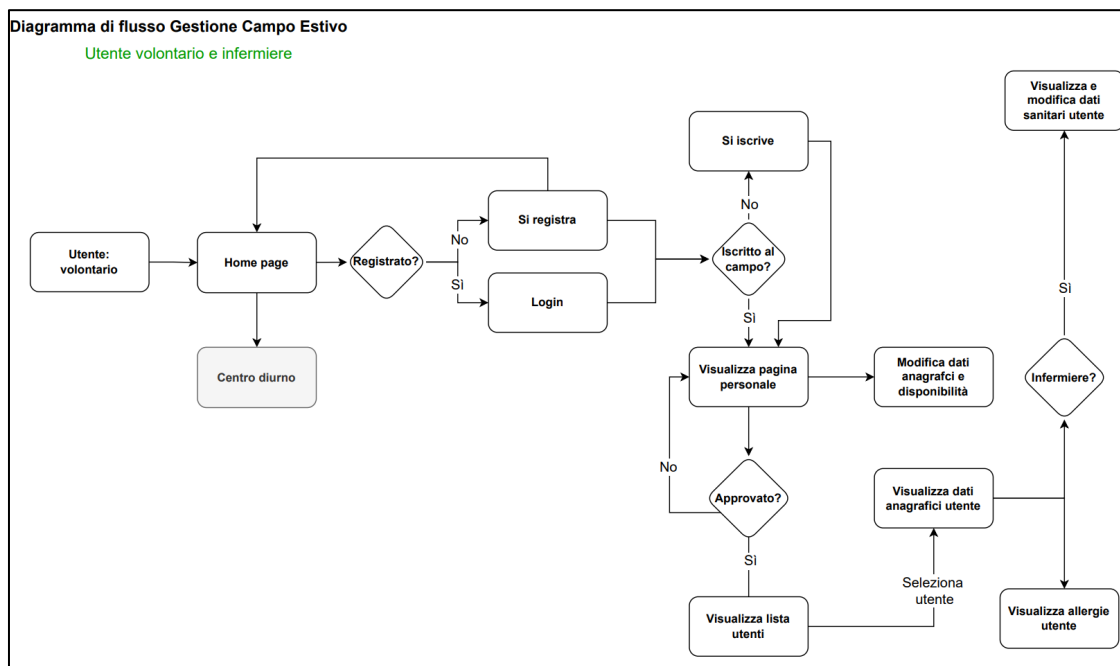


Figura 15 Diagramma di flusso volontari e infermieri

Benché differenti sotto alcuni aspetti, i volontari e gli infermieri sono la stessa cosa a livello organizzativo nel campo. Infatti un infermiere è un volontario approvato dall'amministratore e dal medico del campo. La procedura di iscrizione al sito e al campo è uguale a quella descritta per l'ospite, così come il blocco sull'account appena creato. In questo caso il blocco acquisisce importanza, poiché un account registrato come volontario può vedere solo le allergie, mentre un account infermiere può, in aggiunta, visualizzare e modificare i dati medici.

Infatti come illustrato anche dallo schema, una volta raggiunta la pagina di un utente del campo, in base al tipo di account, vengono mostrati anche i dati più sensibili degli utenti.

3.4.3 Diagramma di flusso amministratori

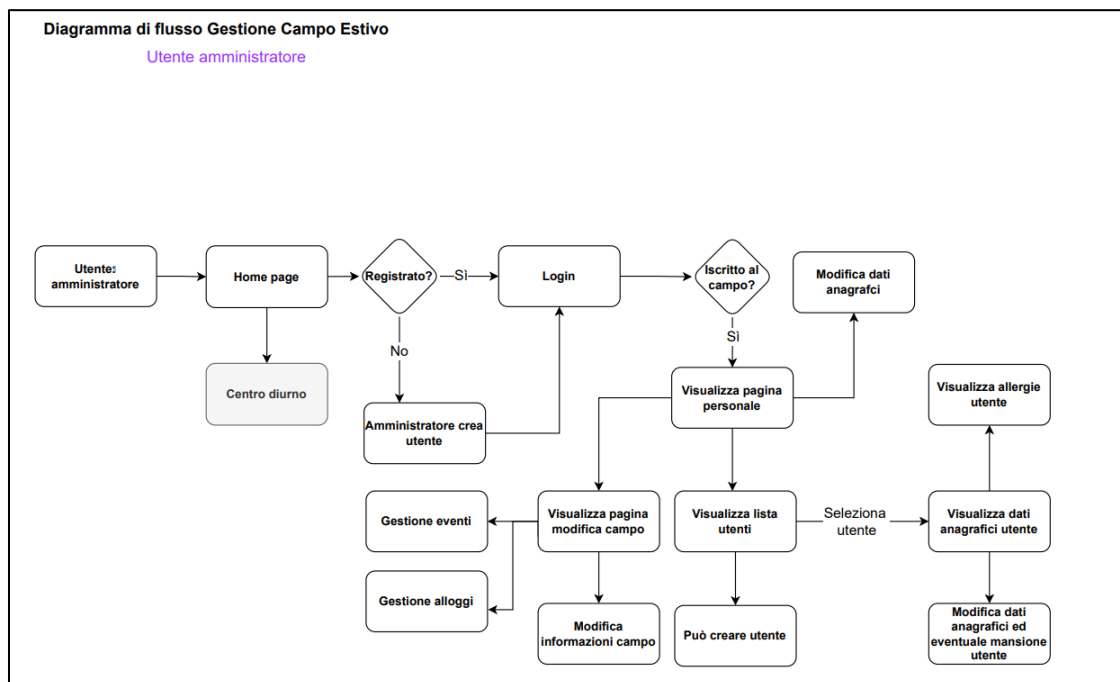


Figura 16 Diagramma di flusso amministratori

L'amministratore è l'account più potente del sito, può fare pressoché tutto, ad esclusione di visualizzare i dati medici.

La procedura di *login* è sempre la stessa, ma un utente non può, logicamente, crearsi un account amministratore in autonomia: solo un altro amministratore può farlo.

Per *default* tutti gli amministratori sono iscritti al campo, in questo modo la gestione dello stesso si semplifica.

Una volta raggiunta la pagina personale, l'amministratore può modificare le informazioni e la descrizione del campo attuale. Può, come tutti, visualizzare la lista di utenti e crearne uno nuovo.

Inoltre può, dopo aver selezionato un utente, visualizzarne e modificarne tutti i dati, eccetto quelli medici.

Inoltre può gestire gli alloggi e gli eventi del campo.

4 Implementazione

4.1 Il Database

Per il mio database ho optato per la scelta più classica ma a mio avviso più adatta al progetto, un DB relazionale con MySQL. Per la realizzazione sono ovviamente partito dallo schema ER, nello specifico dalla seconda versione, che era una rappresentazione quasi perfetta di quello che sarebbe stato poi implementato.

4.1.1 Tabella Person

La prima tabella che ho realizzato è la tabella *person*. In questa tabella verranno memorizzati tutti gli utenti del sito. Sono presenti attributi comuni a tutti quanti, come il nome, il cognome, la data di nascita, ecc.

Questa tabella contiene anche un *id* auto incrementale che è la chiave per tutti gli utenti.

Infine c'è un campo *enum* che specifica il tipo di utente: *guest*, *volontario*, *admin*, *reference*.

- Guest è l'ospite del campo
- Volontario è un volontario del campo
- Admin è l'amministratore
- Reference è la persona di riferimento di un'altra

4.1.2 Entità legate a ospiti

Tutti gli ospiti hanno poi delle entità legate. Infatti, come ho detto nella progettazione, c'erano molti campi da rappresentare, per questo ho diviso tutto in tabelle secondarie. Di seguito un esempio di tabella che sarà popolato da dati riferiti ad un *guest*. Si può vedere come questa tabella abbia un altro *id* che viene usato come chiave e una *foreign key* che permette di far riferimento all'ospite nella tabella *person*.

La maggior parte dei campi sono *NOT NULL*, vorrà dire che andranno per forza riempiti con qualcosa. Non è così per tutti i campi però, infatti *instituteNumber* accetta anche valori nulli. Questo perché suddetto campo è da riempire solo se viene selezionata l'opzione "in istituto" nel campo precedente. Questo non è tuttavia un controllo che può essere fatto a livello di SQL, ma andrà fatto con l'implementazione della pagina.

```
CREATE TABLE livingSituation(
  id INT(20) AUTO_INCREMENT NOT NULL,
  idGuest INT(20) NOT NULL,

  livingState ENUM('da solo', 'con il coniuge', 'in istituto', 'con i parenti') NOT NULL,
  instituteNumber INT(20),

  PRIMARY KEY(id),
  FOREIGN KEY (idGuest) REFERENCES person(id)
  ON UPDATE NO ACTION
  ON DELETE CASCADE
);
```

Le entità collegate all'ospite sono:

- guestInfo
- referencePerson
- livingSituation
- healthIns
- movement
- lastRecover
- alimentation
- guestHyg
- communication
- sleep
- psySate

C'è poi un'entità che connette un utente ad una stanza. È una tabella molto semplice, simile nella logica a quella creata per collegare un utente alla sua persona di riferimento.

```
DROP TABLE IF EXISTS occupied;
CREATE TABLE occupied(
    id INT AUTO_INCREMENT NOT NULL,
    idPerson INT NOT NULL,
    idRoom INT NOT NULL,
    PRIMARY KEY(id),
    FOREIGN KEY (idPerson) REFERENCES person(id),
    FOREIGN KEY (idRoom) REFERENCES room(id)
    ON UPDATE NO ACTION
    ON DELETE CASCADE
);
```

Infine un'altra entità importante è quella che specifica l'appartenenza di un utente ad un campo estivo, dato che tecnicamente potrebbero esserci più campi.

```
CREATE TABLE belongs (
    id INT AUTO_INCREMENT NOT NULL,
    idPerson INT NOT NULL,
    idCamp INT NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (idPerson) REFERENCES person (id),
    FOREIGN KEY (idCamp) REFERENCES camp (id)
);
```

Viene salvato l'id della persona, l'id della camera, e vengono collegati.

4.1.3 Entità legate a volontari

Il meccanismo di funzionamento per le entità dei volontari è leggermente differente rispetto a quello appena discusso. Alcune entità rispettano lo stesso meccanismo usato per i guest, tutte le entità hanno un *id* che utilizzano come chiave e una *FK* che si collega a *person*.

```
DROP TABLE IF EXISTS referencePerson;
CREATE TABLE referencePerson(
    id INT(20) AUTO_INCREMENT NOT NULL,
    idGuest INT(20) NOT NULL,
    idReference INT(20) NOT NULL,

    PRIMARY KEY(id),
    FOREIGN KEY (idGuest) REFERENCES person(id),
    FOREIGN KEY (idReference) REFERENCES person(id)
    ON UPDATE NO ACTION
    ON DELETE CASCADE
);
```

Le entità che usano questo meccanismo sono:

- referencePerson
- autocertificate

Il volontario è poi collegato alla tabella degli eventi e delle disponibilità, *events* appunto. Questa tabella ha una *foreign key* che permette di identificare a quale volontario si collega ogni disponibilità. Un evento invece non avrà questa *foreign key*.

```
CREATE TABLE events(
    id int NOT NULL AUTO_INCREMENT,
    idVolunteer int,
    title varchar(50),
    start varchar(50) NOT NULL,
    end varchar(50) NOT NULL,
    updated_at varchar(50) NOT NULL,
    created_at varchar(50) NOT NULL,
    color VARCHAR(7) NOT NULL DEFAULT "#FFCC99",
    isEvent BOOLEAN,
    PRIMARY KEY (id),
    FOREIGN KEY (idVolunteer) REFERENCES person(id)
);
```

L'attributo *color* verrà assegnato in base alla mansione dell'utente.

4.1.3.1 Autocertificazione

Rispetto al primo progetto l'autocertificazione è stata ripensata. Per permetter maggiore flessibilità ho deciso di creare una tabella che memorizzasse le malattie e i disturbi di cui un volontario può soffrire. Una tabella ponte si occuperà poi di collegare un volontario ad una malattia. In tutte queste tabelle viene usato un id auto incrementale per l'identificazione univoca.

```

/*****ILLNESS*****/
DROP TABLE IF EXISTS illness;
CREATE TABLE illness(
    id INT AUTO_INCREMENT NOT NULL,
    name TEXT NOT NULL,
    PRIMARY KEY(id)
);
INSERT INTO illness(name) VALUES
("Diabete"),
("Emicrania"),
...
("Problemi psichici (es: depressione, bipolarismo, ecc.)");

/*****VOLUNTEER_ILLNESS*****/
DROP TABLE IF EXISTS volunteer_illness;
CREATE TABLE volunteer_illness(
    id INT AUTO_INCREMENT NOT NULL,
    idVolunteer INT NOT NULL,
    idIllness INT NOT NULL,
    PRIMARY KEY(id),
    FOREIGN KEY (idVolunteer) REFERENCES autocertificate(id),
    FOREIGN KEY (idIllness) REFERENCES illness(id)
);

```

4.1.4 Entità legate al campo

Infine le ultime entità del DB, che non sono propriamente legata ad un utente, sono quelle che memorizzano campi estivi. Qui sono presenti dati come il titolo del campo, le novità e le informazioni, le direttive al personale ecc.

La prima entità legata al campo è quella atta a gestire le stanze. È un entità “classica” e relativamente semplice, viene poi collegata all'entità *occupied* in modo da sapere quali utenti occupano una determinata stanza.

```

CREATE TABLE room(
    id int NOT NULL AUTO_INCREMENT,
    number VARCHAR(10),
    block INT,
    places INT,
    campId INT,
    PRIMARY KEY (id),
    FOREIGN KEY (campId) REFERENCES camp(id)
);

```

Ho deciso di non salvare il numero di posti liberi, dato che può essere facilmente calcolato usando la funzione COUNT e il numero di posti totali.

Si noti che anche questa tabella ha come chiave primaria un id auto incrementale; avrei potuto utilizzare il blocco e il numero della stanza per fare una chiave, ma ho pensato che fosse più pulito in questo modo.

4.2 Sito web

Come primo passo per la realizzazione di questo progetto ho deciso di realizzare le pagine HTML, non partendo da uno sviluppo PHP bensì da uno puramente grafico. Questo mi ha permesso di avere da subito un'idea di come la pagina avrebbe funzionato alla fine; inoltre, in questo modo ho identificato alcune falle nel mio design, come le icone sbagliate o i collegamenti non ottimali, che ho potuto correggere subito e facilmente, visto che non avevo codice complesso da gestire. Per realizzare le pagine ho deciso di utilizzare il CMS, Nicepage. Un CMS, acronimo di **C**ontent **M**anagement **S**ystem, è un software che permette la creazione di pagine utilizzando l'approccio grafico. Trascinare i componenti, cambiando lo stile, e creare la pagina in questo modo, mi ha permesso di creare il sito in poco tempo, pur facendo un discreto lavoro.

4.2.1 Creazione delle views

Per la creazione delle *views*, come detto prima, ho utilizzato un CMS. Non mi dilungherò troppo nella spiegazione poiché non c'è molto da documentare; le *views* sono delle semplici pagine HTML con un CSS allegato.

Una cosa particolare di Nicepage e delle pagine da me create, è il fatto che è previsto l'utilizzo di un *header* sempre uguale. Per la pagina aperta al pubblico ho quindi creato un semplice *header* con il collegamento alle pagine:

- Home
- Contatti
- Chi siamo

Tuttavia questo non poteva funzionare per il resto delle mie pagine. Infatti la maggior parte delle pagine interne del sito, quelle che permettono di iscriversi o di gestire il campo estivo, hanno un *header* sempre dinamico. Ho quindi sfruttato un'opzione del CMS che permette di "nascondere l'*header*". Per la maggior parte delle pagine l'*header* non è quindi presente: ho creato a mano un'altra porzione di sito che fa le veci dell'*header*.

4.3 Laravel

4.3.1 Setup del progetto Laravel

Prima di creare il progetto Laravel, è necessario assicurarsi che sul proprio computer locale siano installati PHP e Composer. Se si sviluppa su MacOS, PHP e Composer possono essere installati tramite Homebrew.

Fortunatamente sul PC fornito dalla scuola entrambi i software erano presenti.
Per creare il progetto è sufficiente eseguire questo comando

```
composer create-project laravel/laravel example-app
```

Dopo aver creato il progetto è possibile eseguirlo con il comando:

```
php artisan serve
```

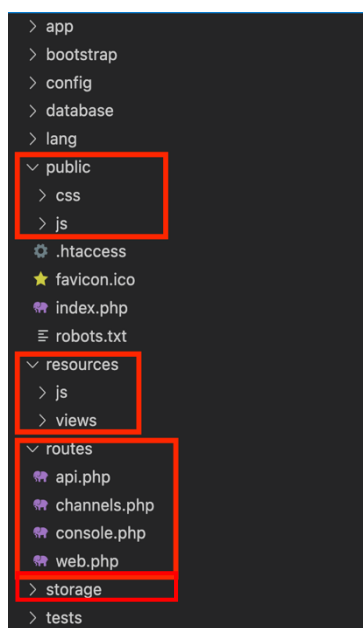


Figura 17 Struttura codice

Il progetto finito avrà una struttura simile, le sezioni più importanti sono evidenziate.

Nella cartella *public* verranno inseriti tutti i file CSS e JS.

In *resources* c'è una seconda cartella JS, che non viene utilizzata, e la cartella contenente tutte le view del progetto.

Infine nella cartella *routes* sono contenute tutte le informazioni per il buon funzionamento e la gestione del progetto; come il nome del sito, la versione di Laravel, ecc.

Il file *web.php* è molto importante poiché serve a “mappare” il sito. Tutte le pagine vengono indicate al suo interno, così che Laravel sappia dove cercare quando viene fatto un reindirizzamento.

Infine la cartella *storage* è volta al salvataggio delle immagini del sito.

4.3.2 Trasferimento delle views

Per trasferire le views non è bastato fare un semplice copia-incolla. Il primo passo è stato effettivamente copiare i file html nella cartella *views*, i file CSS in quella *CSS*, e così via.

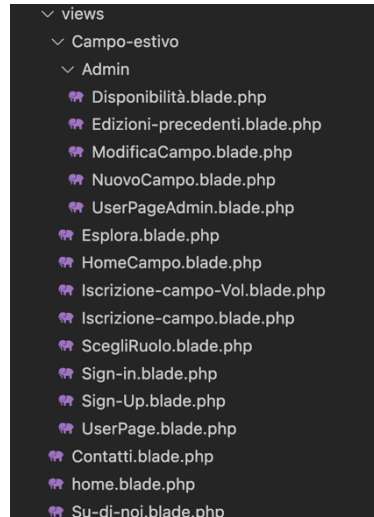


Figura 18 Struttura sito

Poi ho dovuto rinominare tutte le *views* seguendo una specifica nomenclatura. Infatti Laravel per identificare una *view* usa una specie di *template*. Per esempio: il file [home.html](#) è diventato [home.blade.php](#). Oltre a convertire il file da HTML a PHP ho quindi aggiunto la *keyword blade*. Nella foto si possono vedere tutte le *views* del progetto.

Un altro accorgimento che ho dovuto prendere è stato nell'adattare i collegamenti tra le pagine. Infatti il classico approccio che prevede l'uso di *href* in questo caso non funziona. Ho dovuto, nel file *web.php*, inserire tutte le pagine con la posizione ed un nome, di seguito un esempio.

```
Route::get('/Campo-estivo/Iscrizione-campo', function () {
    return view('Campo-estivo/Iscrizione-campo');
})->name('Iscrizione-campo')->middleware('auth');
```

Si utilizza quindi una funzione per ottenere il riferimento passando il percorso assoluto del file partendo dalla cartella *views*. In seguito si può specificare un nome tramite il quale viene identificata la pagina.

Il costrutto *middleware* viene utilizzato per specificare che l'accesso a questa pagina può avvenire solo se l'utente è autenticato al sito. Per gestire l'autenticazione ho usato un'estensione di Laravel chiamata Fortify. Ne parlo nel prossimo capitolo.

Per utilizzare questa *route* appena creata è sufficiente, in un attributo HTML che lo permette, includere la seguente istruzione:

```
href="{{route('Iscrizione-campo')}}"
```

Specificando il nome scelto nel file *web.php* è possibile raggiungere comodamente qualunque pagina.

Non sono solo i link delle pagine ad essere cambiati, ma anche quelli dei file CSS.

In questo caso però non è necessario indicarne la posizione nel file *web.php*, infatti Laravel prevede che essi si trovino tutti nella cartella apposita citata in precedenza. Per questo è sufficiente fare:

```
<link rel="stylesheet" href="{{URL::asset('css/Contatti.css')}}" />
```

Stesso meccanismo viene utilizzato per le immagini, ecco un esempio:

```

```

4.3.3 Login in Laravel

Per gestire il Login, come ho utilizzato Fortify.

Come accennato in precedenza Fortify è un'implementazione *backend* di autenticazione per Laravel. Fortify implementa le routes e i controller necessari per implementare tutte le funzioni di autenticazione di Laravel, tra cui il login, la registrazione, la reimpostazione della password, la verifica dell'e-mail, ecc.

Non è obbligatorio usare Fortify per utilizzare le funzioni di autenticazione di Laravel. Potevo infatti implementare manualmente tutte queste funzioni, tuttavia è consigliato utilizzare questo framework poiché è tra i più stabili.

Per installarlo è sufficiente eseguire i seguenti comandi:

```
composer require laravel/fortify
php artisan vendor:publish --provider="Laravel\Fortify\FortifyServiceProvider"
artisan migrate
```

L'ultimo comando serve a inserire nel DB le tabelle che Fortify utilizza per memorizzare gli utenti. Un vantaggio che si ha seguendo questo approccio è che la password è cifrata di default.

Poi dobbiamo includere manualmente *l'app service provider* nel progetto, un po' come fare l'import di una libreria in un normale codice sorgente. Per farlo bisogna aprire il file `/config/app.php`, qui bisogna inserire il collegamento a Fortify.

Nella sezione *providers* incollare quindi:

```
App\Providers\FortifyServiceProvider:: class,
```

La configurazione iniziale non è finita, nel file `/providers/AppServiceProvider.php` bisogna indicare che *view* devono venir utilizzate per il login e per la registrazione. Qui troveremo un metodo *boot* che è di default, al suo interno inserire le funzioni per specificare le *views*.

```
public function boot(){
    Fortify::loginView(function(){
        return view("Campo-estivo/Sign-In");
    });
    Fortify::registerView(function(){
        return view("Campo-estivo/Sign-Up");
    });
}
```


Ora si può finalmente utilizzare il login, assicurandosi che in entrambe le view specificate sia presente un *form* che abbia i corretti parametri dichiarati, ad esempio il metodo POST e la route corretta.

Inoltre è fondamentale che anche i campi per l'email e la password rispettino anch'essi una specifica sintassi. Ecco il mio form di registrazione, che oltre al nome e all'email, necessari per il buon funzionamento del form, presenta pure un campo per la conferma della password. L'ho semplificato togliendo lo stile ed altre cose superflue al contesto di Fortify.

```
<form method="POST" action="{{route('register')}}"> @csrf

    <!-- User name-->
    <label for="name">{{__('Name')}}</label>
    <input id="name" name="name" value="{{old('name')}}" required>
    @error('name')
        <span class="invalid-feedback" role="alert">
            <strong>{{ $message }}</strong>
        </span>
    @enderror

    <!-- User e-mail -->
    <label for="email">{{__('E-Mail Address')}}</label>
    <input id="email" type="email" name="email" value="{{old('email')}}" required>
    @error('email')
        <span class="invalid-feedback" role="alert">
            <strong>{{ $message }}</strong>
        </span>
    @enderror

    <!-- User password -->
    <label for="password">{{__('Password')}}</label>
    <input id="password" type="password" name="password" required>
    @error('password')
        <span class="invalid-feedback" role="alert">
            <strong>{{ $message }}</strong>
        </span>
    @enderror

    <!-- Password confirm-->
    <label for="password-confirm">{{__('Confirm Password')}}</label>
    <input id="password-confirm" type="password" name="password_confirmation" required>

    <!-- Button send-->
    <button type="submit">
        {{ __('Register') }}
    </button>
</form>
```

Il form di login è pressoché uguale, sia per struttura che sintassi dei campi. Tuttavia, come è logico che sia, ha solo 2 campi: uno per l'email e uno per la password. L'unica differenza degna di nota sta nella prima riga, dove viene specificato il tipo di form e la route. In questo caso la route è *login*.

```
<form method="POST" action="{{route('login')}}">
    ...
</form>
```

4.4 Realizzazione del progetto

4.4.1 Home principale

Nella home principale, seguendo i requisiti, ho incluso due semplici link che permettono di collegarsi rispettivamente al sito della fondazione e alla home page del campo. Di come funziona un link ho già parlato precedentemente, il meccanismo qui è lo stesso.

4.4.2 Home campo

4.4.2.1 Visualizzazione

Questa è la pagina principale del campo: qui sono presenti i collegamenti alle pagine di *Sign up* e di *login*. Tramite i link in cima alla pagina si può inoltre accedere alla propria pagina utente e iscriversi al campo.

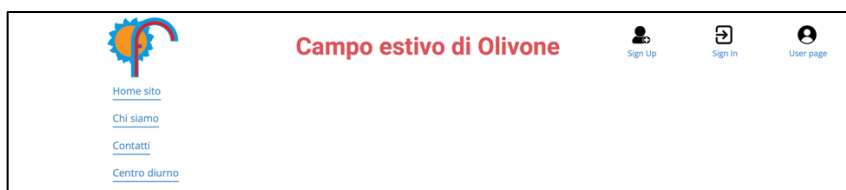


Figura 19 Home - 1

In questa pagina ho poi inserito tutte le sezioni presenti nel sito originale, come le informazioni sul campo.



Figura 20 Home - 2

Oppure le direttive per il personale.

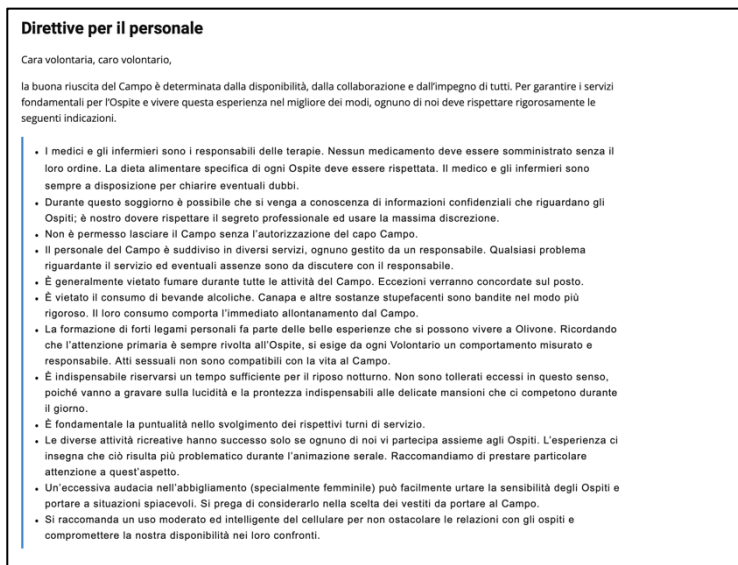


Figura 21 Home - 3

Tutti i dati di questa pagina, a differenza della prima versione del progetto, vengono presi dal database. Non tutti sono però modificabili dall'amministratore. Infatti per motivi logici e di formattazione HTML la storia del campo e le direttive del personale non sono modificabili.

Se ci sono novità di qualunque genere è possibile inserirle nel campo *news*.

4.4.2.2 Modifica home campo

A differenza del primo progetto, in questa seconda versione ho implementato la modifica di un campo. Se si è amministratori, una volta raggiunta la home page, vengono caricati dei bottoni che permettono la modifica. Per gestirlo ho creato un controller chiamato **HomeController**.

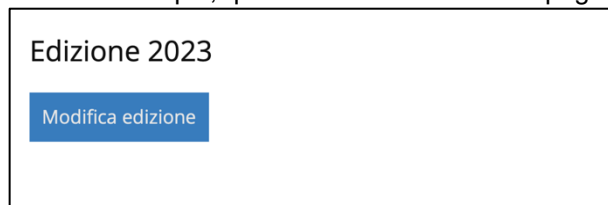
Viene invocato dalla view in questo modo, in questo esempio è possibile gestire la sezione per l'edizione del campo.

```
<form action="{route('editCamp', 'edition')}" method="GET"> @csrf
<?php
    $edition = (new App\Http\Controllers\HomeController)->getEdition();
    print($edition);

    if(Session::get("role")=="admin"){
        if(Session::get("editEdition")=="edit"){
            print('<br><input type="submit" value="Salva edizione">');
        }else{
            print('<br><input type="submit" value="Modifica edizione">');
        }
    }
?>
</form>
```

Viene controllato se l'utente è un amministratore, poi in base allo stato della sessione **editEdition** viene stampato un pulsante adibito alla modifica o alla stampa dell'edizione.

Ecco un esempio, quando si arriva alla home page:

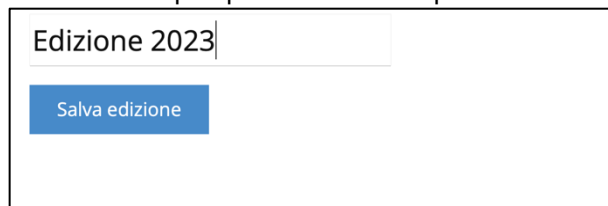


Edizione 2023

Modifica edizione

Figura 22 Home - 4

Ecco un esempio quando si clicca il pulsante:



Edizione 2023

Salva edizione

Figura 23 Home - 5

In pratica viene fatto un *toggle* di quello che viene stampato. Il tutto viene gestito dalla funzione `editCamp`:

```
...
elseif ($field == "edition") {
    if (!empty(Session::get("editEdition"))) {
        Session::forget("editEdition");

        if (is_null($request->edition)) {
            $this->getEdition();
        } else {
            $edition = $this->filterField($request->edition);
            $this->queryData("edition", $edition);
        }

        $request->request->remove("edition");
        return redirect()->route("HomeCampo");
    } else {
        Session::put("editEdition", "edit");
        return redirect()->route("HomeCampo");
    }
}
```

Essa prende come *input* la richiesta *http* del form, chiamata `request`, e il parametro fornito dal codice, in questo caso è `edition`. Grazie a questo parametro e con una sequenza di *if-elseif-else* viene poi identificato cosa si sta modificando.

Alla prima esecuzione la sessione `editEdition` sarà vuota, quindi verrà ritornata la view che caricherà il pulsante di modifica utilizzando il getter apposito, di cui parlerò in seguito.

Alla seconda esecuzione invece, quando l'utente avrà inserito un nuovo valore nel campo, grazie al parametro `request`, viene preso il valore attuale dell'edizione. Viene inoltre dimenticata la variabile di sessione, in modo da far funzionare il meccanismo di *toggle*.

Se il valore inserito è vuoto viene preso il valore attuale dell'edizione con un getter. Altrimenti il valore viene filtrato (con `filterField`) e poi viene salvato.

Per ottenere il valore viene usato questo getter:

```
public function getEdition(){
    $edition = DB::table("camp")
        ->where("id", Session::get("campId"))
        ->first(["edition"]);

    if (Session::get("editEdition") == "edit") {
        $str = "'" . ((array) $edition)["edition"] . "'";
        return '<input type="text" style="font-size: 1.875rem" value=' .
            $str . 'name="edition"></input>';
    } else {
        return '<span style="font-size: 1.875rem">' .
            $edition->edition . '</span>';
    }
}
```

Viene usato l'id del campo, ottenuto in precedenza, e viene preso il primo (e unico) valore con l'edizione.

Se la sessione `editEdition` è vuota viene ritornato il contenuto attuale del campo nel DB, ma questa volta circondato da un input di testo. In questo modo nella view apparirà un campo di testo con scritto dentro il valore attuale del campo `edition` sul DB.

Alla seconda volta che verrà schiacciato, dato che la sessione è già stata eliminata in precedenza, verrà solo visualizzato il testo.

Per salvare il valore viene usato questo *setter*:

```
public function queryData($field, $data){
    DB::table("camp")
        ->where("id", "=", Session::get("campId"))
        ->update([$field => $data]);
}
```

Non è altro che un metodo che prende come parametri il campo da modificare (*\$field*) e il valore da salvare (*\$data*). Usando le strutture di Laravel viene poi salvato tutto nel DB, sempre avvalendosi dell'id del campo.

Per ottenere l'id del campo viene eseguita una funzione al caricamento che prende l'ultimo campo nel DB, che sarà sicuramente quello che si vuole modificare (il campo attuale essenzialmente).

```
public function setCampId(){
    $idCamp = DB::table("camp")
        ->latest("id")
        ->first();
    $idCamp = $idCamp->id;
    Session::put("campId", $idCamp);
}
```

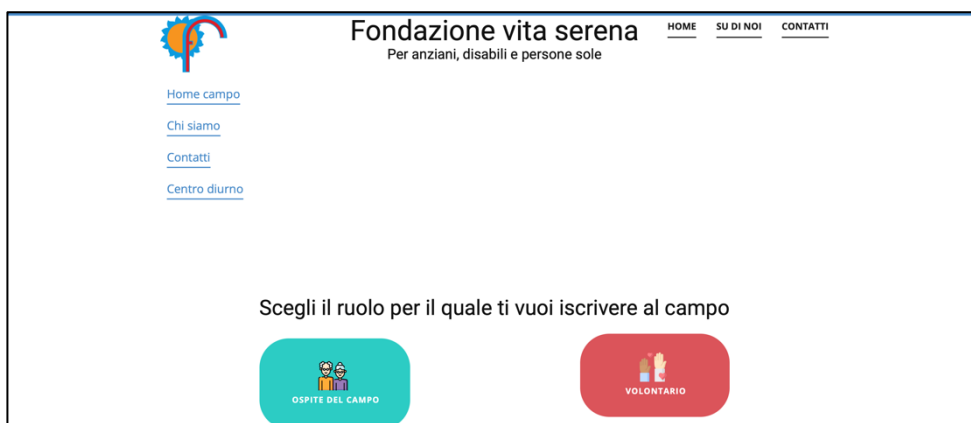
La stessa cosa documentata qui l'ho ripetuta per tutti i dati modificabili del campo.

4.4.3 Registrazione e login

Di questo aspetto ho già parlato in modo esaustivo nel capitolo dedicato all'autenticazione in Fortify. Anche in questo caso quindi non mi dilungherò dato che non ho implementato nulla di più rispetto a quanto già documentato.

Unico aspetto degno di nota è che inizialmente avevo previsto che il login e la registrazione avvenissero tramite un pop-up. Tuttavia alla fine ho utilizzato una pagina poiché Fortify lo prevedeva, mi sono quindi dovuto adattare.

4.4.4 Registrazione al campo



Prima di registrarsi ad un campo è necessario scegliere il proprio ruolo, che può essere volontario o ospite. Questo viene fatto tramite una pagina intermedia una volta che si clicca su "iscriviti al campo".

Figura 24 Scelta ruolo

Una volta scelto il ruolo si viene portati alla pagina che permette di completare l'iscrizione al campo. Per creare il form mi sono basato su quello che ho già fatto per il *login*.

Di seguito la dichiarazione del form.

```
<form action="{route('QueryAllData')}}" method="POST" name="form"> @csrf
```

Si può vedere che nell'action viene specificato **QueryAllData**.

Questo non è altro che un metodo da me creato nel controller della pagina. Con il seguente comando ho creato un controller:

```
php artisan make:controller NomeController --invokable
```

Poi nel file *web.php* ho definito la route che collegasse il form nella pagina di iscrizione alla funzione **QueryAllData**.

```
Route::post("/Campo-estivo/Iscrizione-campo",
[\App\Http\Controllers\IscrizioneOspController::class, 'QueryAllData'])->name('QueryAllData');
```

Questo mi ha permesso di passare allo sviluppo della funzione vera e propria, chiamata **QueryAllData**.

Il primo passo è stato prelevare i dati dal form, per farlo ho usato la seguente sintassi:

```
public function QueryAllData(Request $request){
    $nameTxt = $request->nameTxt;

    ...
}
```

Dichiarando **\$request** posso prendere i dati semplicemente indicando il nome del campo al quale voglio accedere.

Ho ripetuto questo processo per tutti i campi del form, poi sono passato al salvataggio nel DB.

Per farlo ho usato molto la guida ufficiale di Laravel e ho trovato un metodo semplice per farlo, ecco un esempio:

```
/*
 * Inserimento di un set di dati in una tabella.
 */
DB::table("person")->insert([
    "name" => $nameTxt,
    "lastname" => $lastNameTxt,
    "phone_number" => $phoneNumber,
    "address" => $viaTxt,
    "CAP" => $CAP,
    "country" => $nazione,
    "type" => "guest",
    "born_date" => $bornDate,
]);
```

Specificando la tabella e usando il costrutto *insert* posso inserire i dati presi dai campi direttamente nel DB. Questo esempio rappresenta il primo inserimento che ho fatto. Poi, con il codice seguente, ho salvato l'id dell'utente appena inserito in modo da avere la FK da utilizzare per tutti gli altri inserimenti:

```
$idGuest = DB::table("person")
    ->latest("id")
    ->first();
$idGuest = $idGuest->id;
```

Per il resto delle tabelle ho usato lo stesso procedimento usato per la tabella *person*.

Dopo aver inserito tutti i dati ho ritornato la *view* della pagina utente.

```
return View::make('Campo-estivo/UserPage');
```

L'utente, dopo la registrazione, viene così reindirizzato alla sua pagina personale.

4.4.5 User page

4.4.5.1 Visualizzazione

Per la pagina utente ho utilizzato un approccio leggermente diverso rispetto a prima. Ho infatti realizzato tutto il codice PHP direttamente nella pagina, senza utilizzare un controller. Il mio obiettivo era ottenere una lista degli attributi dell'utente corrente. A questo scopo ho usato la classe *Auth*, implementata nativamente in Laravel/Fortify.

```
$user=Auth::user()->name;
```

Ho poi selezionato l'utente dal DB e l'ho salvato in una variabile

```
$results = DB::select("select * from person where name = '".$user.'");
```

Poi ho definito un array che contenesse le etichette di tutti gli attributi dell'utente:

```
$data = array("Id", "Nome", "Cognome", "Numero di telefono",
"Via", "CAP", "Nazione", "Ruolo", "Data di nascita", "");
```

Infine ho scritto il ciclo che stampasse tutti i dati:

Ho dovuto fare un doppio ciclo poiché dal database viene estratta una matrice. Perciò il primo ciclo itera tutte le "righe" della matrice, ogni riga corrisponde ad un dato dell'utente, quindi: nome, cognome, ecc.

Il secondo ciclo itera attraverso l'array temporaneo appena creato e mi permette di concatenare così l'etichetta e il dato dell'utente.

Ho dovuto usare la funzione `array_values` poiché i dati estratti non usavano un indice numerico ma una stringa.

```
foreach ($results as $el){
    $realArray = (array)$el;
    $realArray=array_values($realArray);
    for ($i=0;$i<count($realArray);$i++){
        print("<tr><td>".$data[$i]."</td><td>".$realArray[$i]."</td></tr>");
    }
}
```

Ecco come si presenta la pagina.

Campo estivo di Olivone

Dati personali:

Id	3
Nome	Admin
Cognome	Super
Numero di telefono	0041-12-345-67-89
Via	Admin
CAP	1234
Nazione	Admin
Ruolo	admin
Data di nascita	1979-07-09
Userid	2
Stato profilo	approved
Mansione	

Modifica dati

Figura 25 User Page - 1

Ora è doverosa una puntualizzazione, in ordine cronologico ho sviluppato prima la pagina di esplorazione, poi le pagine per visualizzare i dati di un altro utente del campo, e infine la modifica e la visualizzazione dei dati di un utente.

Dico questo poiché spesso negli esempi di codice si vedranno delle chiamate a delle funzioni in altri controller, questo perché spesso ho riutilizzato il codice già scritto per evitare ripetizioni ed inconsistenze.

Detto questo, passiamo alla modifica dei dati.

4.4.5.2 Modifica

Per la modifica ho usato un meccanismo simile a quello usato per la modifica dei dati del campo. Ho un pulsante che, con un sistema di sessione e di *toggle* viene caricata prima la *view* che permette di consultare i dati. Cliccando sul pulsante viene invece trasformata in una *view* per modificare questi dati.

Dati personali:

Id	3
Nome	<input type="text" value="Admin"/>
Cognome	<input type="text" value="Super"/>
Numero di telefono	<input type="text" value="0041-12-345-67-89"/>
Via	<input type="text" value="Admin"/>
CAP	<input type="text" value="1234"/>
Nazione	<input type="text" value="Admin"/>
Ruolo	<input type="text" value="admin"/>
Data di nascita	<input type="text" value="05/04/2023"/> Attuale: 1979-07-09
Userid	2
Stato profilo	approved
Mansione	

Salva dati

Figura 26 Modifica dati

Quando clicco di nuovo i dati vengono salvati.

Per farlo nel controller di questa pagina è presente il metodo `editData`.

Esso ha come parametro un oggetto di tipo `Request` che rappresenta i dati inviati dal client attraverso una richiesta HTTP.

Nel codice, viene prima controllato se una variabile di sessione chiamata "editing" esiste già. Se esiste, viene cancellata. In questo modo si evita che l'utente possa eseguire più operazioni di modifica contemporaneamente.

Successivamente, i dati inviati dal client vengono raccolti nell'array `$data` e viene eseguito uno "shift" per rimuovere il primo elemento, che non è necessario ai fini della modifica. Viene inoltre estratto l'id dell'utente corrente attraverso il metodo `Auth::user()` e salvato nella variabile `$userId`.

Un'altra operazione importante è la conversione della data di nascita in un formato corretto e la rimozione di eventuali caratteri indesiderati.

```
if(!empty(Session::get("editing"))){
    Session::forget("editing");
    $data=$request->all();
    array_shift($data);
    $userId=Auth::user()->id;
    $data=$this->fillData($data);
    $time = strtotime($data[6]);
    if ($time) {
        $data[6] = date('Y-m-d', $time);
    } else {
        return;
    }
}
```

Il metodo `fillData` viene chiamato per riempire i dati mancanti o che l'utente non ha modificato in fase di modifica, sarà documentato in dettaglio dopo.

Infine, i dati vengono salvati nella tabella "person" del database attraverso una *query* SQL di tipo UPDATE. Viene selezionata la riga della tabella corrispondente all'utente corrente (identificato dall'id) e vengono aggiornati i campi con i nuovi valori.

```
...
DB::table("person")
    ->where('regUserId', '=', $userId)
    ->update([
        "name" => $data[0],
        "lastname" => $data[1],
        "phone_number" =>$data[2],
        "address" =>$data[3],
        "CAP" =>$data[4],
        "country" =>$data[5],
        "born_date" => $data[6],
    ]);
return redirect()->route('UserPage');return;
...
```

Se la variabile di sessione "editing" non esiste, viene impostata e viene effettuato un *redirect* alla stessa pagina. In caso contrario, viene eseguita la modifica e viene effettuato un *redirect* alla pagina dell'utente. Questo è essenziale per far funzionare il meccanismo di *toggle*.

Come detto `fillData` prende come argomento un array `$data` contenente i dati di un utente e restituisce lo stesso array con i campi vuoti riempiti con i dati precedenti dell'utente.

La funzione utilizza l'ID dell'utente attualmente autenticato per ottenere i dati dell'utente dalla tabella `person` del database. La query SQL seleziona solo i campi `name`, `lastname`, `phone_number`, `address`, `CAP`, `country` e `born_date` per l'utente con lo stesso ID dell'utente autenticato.

La funzione confronta poi ogni campo di `$data` con il valore corrispondente nel vecchio array `$oldData`, e se il campo in `$data` è vuoto, lo sostituisce con il valore corrispondente in `$oldData`. Infine, la funzione restituisce l'array `$data` con i campi vuoti riempiti con i valori precedenti.

```
public function fillData($data){
    $data=array_values($data);
    $userId=\Auth::user()->id;

    $keys = array(
        "name", "lastname", "phone_number",
        "address", "CAP", "country", "born_date"
    );

    $oldData="select name,lastname,phone_number,address,CAP,country,born_date from person where
regUserId =
    ".$userId;

    $oldData = DB::select($oldData);
    $oldData=array_values(((array)$oldData[0]));

    for($i=0;$i<count($data);$i++){
        if(empty($data[$i])){
            $data[$i]=$oldData[$i];
        }
    }
    return $data;
}
```

4.4.5.3 Dati medici

Per visualizzare i dati medici uso un a funzione sviluppata in origine per le pagine degli altri utenti. Dato che il concetto è quello ho solo incluso nella `view` del codice una chiamata a quella funzione:

```
<?php

print("<br><br><h2>Dati generali</h2><br>");

$id = (new App\Http\Controllers\UserPageController)->toId(Session::get("regUserId"));
$genData = (new App\Http\Controllers\UserPageExploreController)->generalData($id);
print($genData."<br>");
print("<br><br><h2>Dati medici</h2><br>");

$medData = (new App\Http\Controllers\UserPageExploreController)->medicalData($id);
print($medData);
?>
```

Ho diviso i dati medici in 2 parti, dato che anche i volontari devono vedere almeno le allergie dell'utente. Di questa funzione parlerò in seguito.

4.4.5.4 La view

Nella view sono presenti alcune particolarità per far funzionare il tutto.

Una volta che la variabile con tutti dati viene restituita alla view è necessario, ovviamente, stamparla.

Ho quindi creato un ciclo che si occupa di scorrere tutti i dati ricevuti dal controller, salvati in `$realArray`, e stamparli.

Finche si parla di una semplice stampa di dati basta un ciclo come questo:

```
for ($i = 0; $i < count($realArray); $i++) {
    $toPrint = $toPrint . "<tr>" . "<td><b>" . $data[$i] . "</b></td>"
    . "<td>" . $realArray[$i] . "</td></tr>";
}
```

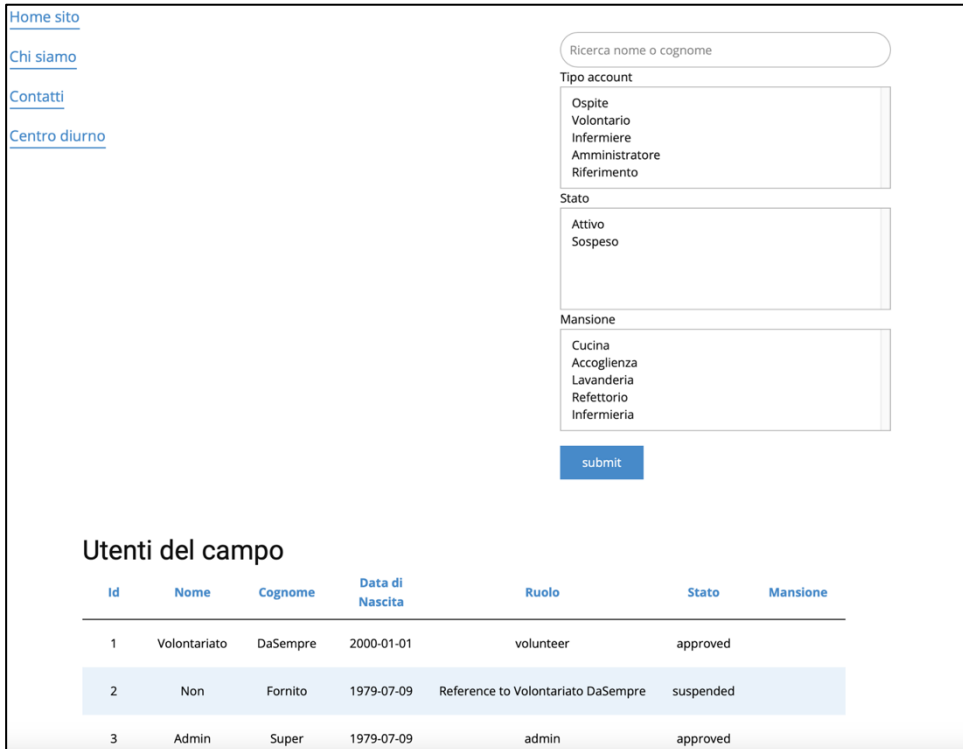
Tuttavia in fase di modifica la situazione si complica. Nel seguente codice viene controllato l'indice dell'array, questo è fatto per adattarsi alle differenze di tipo e accesso nei campi.

Soprattutto in fase di modifica. Infatti, per esempio il campo `bornDate`, non può avere un *placeholder*, così ho deciso di stampare la data precedente a fianco. I campi come l'id o lo stato del profilo invece non possono essere modificati da un utente comune, quindi anche essi sono bloccati.

Tutto questo viene fatto, come al solito, concatenando in una variabile i dati formattati come una tabella HTML, che alla fine viene stampata.

```
for ($i = 0; $i < count($realArray); $i++) {
    if ($i == 0 || $i == 7 || $i > 8) {
        $toPrint = $toPrint . "<tr>" . "<td><b>" . $data[$i] . "</b></td>"
        . "<td>" . $realArray[$i] . "</td></tr>";
    } else if ($i == 8) {
        $toPrint = $toPrint . "<tr>" . "<td><b>" . $data[$i] . "</b></td>"
        . "<td><input type='date' name=" . $data[$i] . "> Attuale: " . $realArray[$i] . "</input></td></tr>";
    } else {
        $toPrint = $toPrint . "<tr>" . "<td><b>" . $data[$i] . "</b></td>"
        . "<td><input type='text' name=" . $data[$i] . " placeholder=" . $realArray[$i] . "></div></td></tr>";
    }
}
```

4.4.6 Pagina esplorativa



The screenshot shows a web interface for user management. On the left is a sidebar with links: Home sito, Chi siamo, Contatti, and Centro diurno. The main area contains search filters: a text input for 'Ricerca nome o cognome', a dropdown for 'Tipo account' (Ospite, Volontario, Infermiere, Amministratore, Riferimento), a dropdown for 'Stato' (Attivo, Sospeso), and a dropdown for 'Mansione' (Cucina, Accoglienza, Lavanderia, Refettorio, Infermeria). A blue 'submit' button is below the filters. Below the filters is a table titled 'Utenti del campo' with columns: Id, Nome, Cognome, Data di Nascita, Ruolo, Stato, and Mansione. The table contains three rows of data.

Id	Nome	Cognome	Data di Nascita	Ruolo	Stato	Mansione
1	Volontariato	DaSempere	2000-01-01	volunteer	approved	
2	Non	Fornito	1979-07-09	Reference to Volontariato DaSempere	suspended	
3	Admin	Super	1979-07-09	admin	approved	

La pagina esplorativa è una pagina molto importante per tutti gli utenti del campo. Contiene un meccanismo di ricerca e filtraggio combinato con una lista completa di tutti gli utenti:

Per questa funzione dividerò l'implementazione in parti, dato che il codice è tanto.

Figura 27 Esplora

4.4.6.1 searchUser

La funzione accetta un oggetto di tipo Request come argomento che rappresenta la richiesta HTTP inviata dal client. Questa richiesta contiene tutti i filtri che l'utente ha deciso di applicare alla pagina. In base a che cosa arriva nell'array vengono preparate le variabili per contenere i dati. Successivamente, i filtri per `userType`, `userState` e `userMansion` vengono estratti dalla richiesta e se non sono presenti nella richiesta, vengono assegnati dei valori di default.

```
if (array_key_exists("userType", $userData)) {
    $userType = $userData["userType"];
} else {
    $userType = ["guest", "volunteer", "nurse", "admin", "reference"];
}
```

La funzione `filterField` viene utilizzata per filtrare il valore della variabile `$userName`, rimuovendo eventuali caratteri indesiderati.

La funzione `toQuery` viene utilizzata per trasformare gli array di filtri in stringhe che possono essere utilizzate nella `query` SQL.

Viene quindi costruita una stringa di `query` SQL e passata alla funzione `DB::select`, che esegue la `query` e restituisce un array di oggetti rappresentanti i risultati.

La variabile `$query` viene poi salvata nella sessione.

```
$query= "select id,name,lastname,born_date,type,userState,userMansion from person
where (name like " . "'" . $userName . "'"
OR lastname like " . "'" . $userName . "'"
AND ( " . $userType . " )
AND ( " . $userState . " )
AND ( " . $userMansion . " )";
$users = DB::select($query);
```

L'array di risultati viene convertito in un array associativo e passato alla funzione `returnData`, che lo formatta e lo prepara per la visualizzazione.
Infine, viene restituita una view chiamata `Campo-estivo.Esplora` che contiene i dati cercati e formattati, nella variabile `$searchedUser`.

```
$realArray = (array) $users;
$searchedUser = array_values($realArray);
$searchedUser=$this->returnData($searchedUser);
return view("Campo-estivo.Esplora", ["searchedUser" => $searchedUser]);
```

4.4.6.2 returnData

Questo metodo consente di cercare le persone di riferimento degli ospiti all'interno dell'elenco degli utenti e di scrivere i dati corretti nel campo giusto. Questo metodo viene usato da molte funzioni. In pratica prende una lista di utenti e ne trova le referenze degli ospiti.

Il metodo inizia creando un array vuoto `$references` che conterrà gli ID degli utenti che sono referenze. Viene eseguito un ciclo for per iterare attraverso l'array `$searchedUser`, passato dalla e controllare il valore della chiave `$type` di ogni elemento dell'array. Se il valore di `$type` è uguale a `$reference`, l'ID dell'utente viene *pushato* nell'array `$references`. In questo modo o un array con tutti gli id degli utenti referenze.

```
for($i=0;$i<count($searchedUser);$i++){
    $state=((array)$searchedUser[$i])["type"];
    if($state=="reference"){
        array_push($references, ((array)$searchedUser[$i])["id"]);
    }
}
sort($references);
```

Successivamente, gli ID degli utenti contenuti in `$references`, una volta passati nella funzione `toQuery` vengono usati per effettuare una *query* per ottenere l'ID degli ospiti che sono collegati loro. L'array degli ID degli ospiti viene poi creato e iterato attraverso il ciclo `for` per ottenere i nomi e i cognomi degli ospiti. Il risultato viene poi salvato nell'array `$referredUserInfo`.

```
$queryReferredUser=$this->toQuery($useresIds,"id");
$queryReferredUser=str_replace(' ','',$queryReferredUser);
$queryReferredUser=DB::select("select id,name,lastname from person where ".$queryReferredUser);

$referredUserInfo=array();
foreach ($queryReferredUser as $el) {
    $realArray = (array) $el;
    $realArray = array_values($realArray);
    array_push($referredUserInfo,$realArray);
}

for($i=0;$i<count($referredUserInfo);$i++){
    array_push($referredUserInfo[$i],$references[$i]);
}
```

Nella sezione successiva, il metodo cerca la presenza di `$references` nell'intero array degli utenti. Quando trova `$references`, cerca l'ID corretto nell'array contenente i dati di riferimento e sostituisce quello che era l'id della persona di riferimento con `Reference to [nomeGuest] [cognomeGues]`

Infine, il metodo restituisce l'array `$searchedUser` con i dati corretti.

```
foreach($searchedUser as $el){
    $realArray = (array) $el;
    $realArray = array_values($realArray);
    for($i=0;$i<count($realArray);$i++){
        if($realArray[$i]=="reference"){
            for($j=0;$j<count($referredUserInfo);$j++){
                //3 element is referenceId
                if($realArray[0]==$referredUserInfo[$j][3]){
                    //access to name and lastname
                    $realArray[$i]= "Reference to ".$referredUserInfo[$j][1]."."
                        ".$referredUserInfo[$j][2];
                }
            }
        }
    }
    array_push($finalArray,$realArray);
    $searchedUser=$finalArray;
}
```

4.4.6.3 toQuery

La funzione `toQuery` permette di creare una stringa di *query* SQL utilizzando un array di dati e il tipo di ricerca da effettuare. In particolare, questa funzione accetta due parametri:

- `$data`: l'array contenente i dati da cercare.
- `$type`: il tipo di dato da cercare.

All'interno della funzione, viene inizializzata una variabile `$res` che conterrà la stringa di *query* SQL. Successivamente, viene effettuato un ciclo `for` su tutti gli elementi dell'array passato come parametro.

Ad ogni iterazione viene aggiunto alla stringa `$res` un pezzo di *query* SQL, tipo `"OR = "`.

```
public function toQuery($data, $type){
    $res = "";
    for ($i = 0; $i < count($data); $i++) {
        $res = $res . $type . " = " . "'" . $data[$i] . "'" . " OR ";
    }
    ...
}
```

Alla fine si otterrà qualcosa del tipo:

```
$type = "valore" OR = "valore1" OR = "valore2" OR...
```

Alla fine del ciclo `for`, viene utilizzata la funzione `substr` per rimuovere gli ultimi tre caratteri (`OR`) dalla stringa di *query*. Infine, la funzione restituisce la stringa di *query* SQL ottenuta.

4.4.6.4 orderBy

Questa funzione ordina il risultato di una *query* in base ad un elemento specificato e memorizza l'ordine e il colore dell'elemento selezionato nella sessione dell'utente.

Cliccando sull'intestazione della lista viene invocato questa funzione.

Inizialmente viene controllato l'elemento scelto e viene impostato il colore rosso corrispondente nella sessione, mentre gli altri colori vengono rimossi.

Questo viene fatto perché nella view, nell'*header* della tabella, è presente del codice che selettivamente colora di rosso il titolo e li mette una freccia che indica l'ordine dell'ordinamento. Cliccando di nuovo si inverte l'ordine.

```
<th><a style="color:<?php echo(Session::get("colorId"));?>" href="{ route('orderBy', 'id')
}>">Id<?php
if(!empty(Session::get("colorId"))){
    if(Session::get("order")== "ASC"){
        echo("<b>". "↑" . "</b>");
    }else{
        echo("<b>". "↓" . "</b>");
    }
}
?></a></th>
```

Per cambiare il colore vengono appunto usate le variabili di sessione: viene settata solo la variabile di sessione corrispondente all'ordinamento della lista. Per evitare colori multipli le altre vengono eliminate:

```
if($el=="id"){
    Session::put("colorId","red");
    Session::forget("colorName");
    Session::forget("colorLastName");
    Session::forget("colorBornDate");
    Session::forget("colorType");
    Session::forget("colorUserState");
    Session::forget("colorMansion");
    ...
}
```

Viene quindi verificato se l'elemento di ordinamento scelto è lo stesso dell'ordine precedente. Se ciò è vero viene invertito l'ordine, e in caso contrario viene impostato un nuovo parametro di ordinamento. Per farlo la query viene ripetuta ma con un ordinamento di SQL diverso (ASC o DESC).

```
$prev = Session::get("previous");
if ($el != $prev) {
    if(empty(Session::get("order"))){
        Session::put("order", "ASC");
    }else if (Session::get("order") == "ASC") {
        Session::put("order", "DESC");
    } else {
        Session::put("order", "ASC");
    }
} else {
    Session::put("previous", $el);
}
```

Infine, viene eseguita la *query* completa con la nuova impostazione di ordinamento e i risultati della ricerca vengono visualizzati nella vista *Esplora*. per formattare i dati in una tabella HTML viene usata la funzione `returnData`, documentata precedentemente.

```
$query = $query . " ORDER BY " . $el . Session::get("order");
$users = DB::select($query);
...
$searchedUser=$this->returnData($searchedUser);
return view("Campo-estivo.Esplora", ["searchedUser" => $searchedUser]);
```

4.4.6.5 openPage

Questa funzione ha lo scopo di consentire l'apertura e il caricamento della pagina utente. Il metodo richiede due parametri, `$index` e `$tok`. Il primo rappresenta l'indice dell'utente da aprire e il secondo è un *flag* che mi permette di distinguere se sto modificando il mio utente o un utente terzo.

La prima riga del metodo imposta il valore di `$editId` nella classe Session a `$index`, rendendolo disponibile per un uso successivo e memorizzando l'utente che viene modificato. La riga successiva esegue una *query* SQL per recuperare i dettagli dell'utente con il `$index` specificato. I risultati di questa *query* sono memorizzati nella variabile `$results`.

Se l'array `$results` è vuoto, il metodo stampa un messaggio che indica che l'utente non è stato trovato. Altrimenti, il metodo procede a definire un array chiamato `$data` che contiene le etichette per ciascun dato che verrà visualizzato sulla pagina utente.

```
Session::put("editId", $index);
$results = DB::select("select * from person where id = " . "'" . $index . "'");
if(empty($results)){
    print("<tr><h6>Utente non trovato</h6></tr>");
}
$data = array("Id", "Nome", "Cognome",
    "Numero di telefono", "Via", "CAP",
    "Nazione", "Ruolo", "Data di nascita",
    "UserId", "Stato profilo", "Mansione");
```


Il blocco di codice successivo recupera il ruolo dell'utente attualmente loggato eseguendo una *query* SQL per selezionare i campi di tipo utente e stato utente. Questi valori vengono memorizzati nella variabile `$userRoleArray`. Se `$userRoleArray` non è nullo, i valori vengono estratti e memorizzati nella variabile globale `$_SESSION`. Il valore del ruolo viene memorizzato nella variabile `$_SESSION["role"]`, mentre il valore di stato viene memorizzato nella variabile `$_SESSION["state"]`.

```
$userId = \Auth::user()->id;
$userRoleArray = DB::select(
    "select type,userState from person where regUserId =" .
    "'" . $userId . "'"
);
if ($userRoleArray != null) {
    $realArray = (array) $userRoleArray[0];
    $realArray = array_values($realArray);
    $_SESSION["role"] = $realArray[0];
    $_SESSION["state"] = $realArray[1];
}
```

La riga successiva chiama il metodo `returnData` con la variabile `$results` come parametro. Questo metodo elabora l'array `$results` e lo restituisce come un array di array.

La variabile `$toPrint` viene quindi inizializzata come una stringa vuota e il metodo procede a eseguire un ciclo sui risultati del metodo `returnData`. Per ogni elemento nell'array, il codice estrae i valori e li memorizza nella variabile `$realArray`.

Se `Session::get("editing")` è vuoto, il metodo esegue un ciclo sui valori di `$realArray` e stampa una riga di tabella per ogni elemento nell'array. La riga della tabella consiste di un'etichetta dall'array `$data` e il valore corrispondente dall'array `$realArray`.

Se `Session::get("editing")` non è vuoto invece, il metodo esegue nuovamente un ciclo sui valori di `$realArray`. Tuttavia, questa volta, il metodo stampa un campo di input di un form per ogni valore che è modificabile. Il campo di input è una casella di testo o un *dropdown*, a seconda del tipo di dato che viene modificato. Il codice include anche alcune istruzioni condizionali che determinano quali campi possono essere modificati dall'utente in base al suo ruolo e se sta modificando il proprio profilo o quello di qualcun altro.

```
//userState and mansion are editable only by the Amdin.
if($i==10 && $tok=="otherUser"){
    $toPrint = $toPrint . "<tr>" . "<td><b>" . $data[$i] . "</b></td>"
    <td><select name='Stato profilo' class='...' required='required'>;
    if($realArray[$i]=="approved"){
        $toPrint=$toPrint.'<option selected="selected" "value="approved">approved</option>'
        <option value="suspended">suspended</option>';
    }else{
        $toPrint=$toPrint.'<option "value="approved">approved</option>'
        <option selected="selected" value="suspended">suspended</option>';
    }
}
$toPrint=$toPrint."</select></div></td></tr>";
...

```

4.4.6.6 La view

Infine mi sembra opportuno mostrare qualcosa anche della view.
Ho incluso solo un allegato

Infine, il metodo restituisce la variabile `$toPrint`, che contiene il codice HTML per la pagina utente.
Questo è il codice che genera la lista.

Viene stampata tutta la variabile costruita dal controller, includendo le chiamate alla funzione `openPage`.
È infatti proprio lei che “collega” la pagina di esplorazione e la pagina di un utente. Fornendo l'id dell'utente corrente posso aprirne la pagina dettagliata.

```
foreach ($searchedUser as $el) {
    $realArray = (array) $el;
    $realArray = array_values($realArray);
    print '<tr style="height: 65px;">';
    for ($i = 0; $i < count($realArray); $i++) {

        print("<td class=u-table-cell><a style='color: black';
            href=\".route(\"openPage\",['index' => $realArray[0], 'tok' => 'tok']\".\">\".
            $realArray[$i] .\"</a></td>\"");
    }

    print "</tr>";
}
```

4.4.7 Pagine esplorativa utenti

Queste pagine hanno lo stesso principio della pagina personale. Alcuni metodi sono uguali o molto simili nelle dinamiche, quindi non mi soffermerò. Commenterò però i metodi per l'ottenimento dei dati medici, che sono molto importanti anche per le pagine personali degli utenti.

4.4.7.1 generalData

Questo metodo consente di stampare i dati generali di un ospite. Accetta un parametro `$id`, che è l'id dell'ospite che deve essere visualizzato. Il metodo prima verifica se ci sono dati per l'ospite selezionato attraverso il metodo `hasData`.

Se ci sono dati, viene creato un array multidimensionale `$fields` che contiene i campi dei dati dell'ospite, come nome, cognome, dieta, allergie, ecc. Questi campi contengono le etichette dei campi che verranno visualizzate nella pagina

```
$hasData=$this->hasData($id);
if($hasData){
    $fields=array(
        array("Nome ", "Cognome"),
        array("Mangia", "Beve", "Dentiera",
            "Addensante", "Dieta", "Alimenti sconsigliati",
            "Allergie", "Alcolici")
    );
    ...
}
```

Se invece non ci sono dati per l'ospite selezionato nella variabile di stampa viene inserita una stringa di errore.

Successivamente, viene creato l'`$tables` che contiene le tabelle del database che devono essere interrogate per ottenere i dati dell'ospite. L'array `$titles` contiene invece titoli delle tabelle corrispondenti, che verranno visualizzati nella pagina web.

Il metodo quindi richiama il metodo `getDatas` per ottenere i dati dell'ospite. Questi dati vengono quindi restituiti e salvati nell' array `printData` che può essere utilizzato per visualizzare i dati dell'ospite nella pagina web.

```
...
    $tables=array("referencePerson","alimentation");
    $titles=array("Persona di riferimento","Alimentazione");
    $printData=$this->getDatas($id,$fields,$tables,$titles);
} else {
    $printData="Nessun dato per questo utente.";
}
return $printData;
```

In sintesi, questo metodo accetta un parametro id, verifica se ci sono dati per l'ospite corrispondente, organizza i campi dei dati e le tabelle del database, quindi chiama il metodo `getDatas` per recuperare i dati e restituisce i dati dell'ospite in un array.

```
...
$tables=array(
    "guestInfo","livingSituation","lastRecover",
    "healtIns","movement","guestHyg",
    "comunication","sleep","psySate"
);
$titles=array(
    "Informazioni ospite","Situazione di vita","Ultimo ricovero",
    "Cassa malati","Movimenti","Igene personale",
    "Comunicazione","Sonno","Stato psichico"
);
```

4.4.7.2 medicalData

Questa funzione ha lo stesso principio della precedente. Qui vengono interrogate però le tabelle con i dati medici degli utenti.

```
...
$tables=array(
    "guestInfo","livingSituation","lastRecover",
    "healtIns","movement","guestHyg",
    "comunication","sleep","psySate"
);
$titles=array(
    "Informazioni ospite","Situazione di vita","Ultimo ricovero",
    "Cassa malati","Movimenti","Igene personale",
    "Comunicazione","Sonno","Stato psichico"
);
```

4.4.7.3 getData

Questa funzione consente di stampare tutti i dati relativi ad un utente identificato dall'ID fornito. La funzione accetta quattro parametri:

- **id**: che rappresenta l'ID dell'utente
- **fields**: che rappresenta un array con tutti i titoli dei campi
- **tables**: che rappresenta un array con le tabelle da interrogare
- **titles**: che rappresenta un array con tutti i titoli delle tabelle

All'interno della funzione, viene inizializzato un valore vuoto **\$printData** che conterrà tutti i dati da stampare. Successivamente, viene effettuato un ciclo for per ogni tabella presente nell'array **\$tables**. Per ogni tabella, viene costruita una tabella HTML che include il titolo della tabella e le colonne relative ai campi specificati nell'array **\$fields**.

Successivamente, viene effettuata una query sul database per recuperare i dati relativi all'utente specificato tramite l'ID. Se la tabella corrente è **\$referencePerson**, viene effettuata una traduzione dell'ID in nome e cognome della persona di riferimento.

```
...
for($i=0;$i<count($tables);$i++){
    $printData=$printData."<h4>".$titles[$i]."</h4><table>";
    $data = DB::table($tables[$i])->where('idGuest', $id)->get();
    $currFields=$fields[$i];

    //translate id into Name and lastname
    if($tables[$i]=="referencePerson"){
        $data=(array)$data[0];
        $data=array_values($data);
        $data=$data[2];
        $data = DB::table("person")->where('id', $data)->get(['name','lastname']);
    }
}
...
```

Se non ci sono dati corrispondenti, viene inserito un messaggio di "Nessun dato". In caso contrario, viene utilizzato il metodo **toTable** per formattare i dati recuperati in una tabella HTML. Infine, viene restituito il valore di **printData** contenente tutte le tabelle HTML create in precedenza.

4.4.7.4 toTable

Il metodo **toTable** serve a formattare i risultati di una query in una tabella HTML. Prende due parametri:

- **fields** è un array contenente i nomi dei campi dati della query
- **data** è l'array contenente i dati restituiti dalla query

Il metodo restituisce una stringa HTML che rappresenta la tabella con i dati restituiti.

La funzione inizia convertendo l'oggetto restituito dalla query in un array e restituendo solo i valori effettivi, scartando gli eventuali valori nulli. In seguito, viene creato un ciclo for per concatenare la stringa HTML con tutti i valori della query, in modo che venga formattato come una tabella HTML. Infine, la stringa HTML formattata viene restituita.

```
$final="";
for($i=0;$i<count($data);$i++){
    $final=$final."<tr><td>".$fields[$i]."</td><td>".$data[$i]."</td></tr>";
}
```

4.4.7.5 hasData

La funzione **hasData** controlla se l'utente con l'id specificato ha dei dati registrati nella tabella *person*.

La funzione prende come *input* un parametro **\$id** che rappresenta l'id. Viene poi eseguita una *query* sul *database*, cercando l'utente nella tabella *person* e selezionando il campo **type**. Se la *query* restituisce almeno una riga di risultati, l'utente esiste.

Se esiste, viene estratto il valore del campo **type** e se l'utente è un guest viene restituito *true*, altrimenti *false*. La funzione non restituisce nulla se non viene trovato alcun utente con l'id specificato.

```
$data = DB::table("person")->where('id', $id)->get(['type']);
if(count($data)>0){
    $data=(array)$data[0];
    $data=array_values($data);
    return $data[0]=="guest";
}
```

4.4.8 Gestione disponibilità

Per la gestione di questa parte del progetto ho usato una libreria JS chiamata Fullcalendar. FullCalendar è una libreria open source per la creazione di calendari interattivi, scritta in JavaScript e basata sulla libreria jQuery. Essa offre una vasta gamma di funzionalità, come la possibilità di creare eventi, visualizzare e modificare eventi esistenti, trascinare e rilasciare gli eventi per spostarli, zoomare e scorrere il calendario per visualizzare diverse date, e altro ancora. FullCalendar può essere utilizzato in diversi contesti, come ad esempio la gestione di appuntamenti medici, eventi aziendali, prenotazioni di viaggi e vacanze, e molto altro.

Per farlo ho creato un controller dedicato, che contiene tre funzioni essenziali alla gestione delle disponibilità. Quando la pagina home viene caricata, grazie ad ajax, viene caricato il metodo `index()`.

4.4.8.1 checkCalendar

Questa funzione è quella che gestisce il form di filtraggio, di cui parleremo in seguito. Riceve come parametro una richiesta HTTP e restituisce la *view* del calendario.

Il primo passo della funzione consiste nell'eliminare eventuali dati di sessione esistenti relativi al nome utente e all'ID degli utenti. Infatti questi dati saranno quelli che la prossima funzione usa per restringere e applicare i parametri della ricerca sul DB; siccome vengono riutilizzati è importante svuotarli.

Successivamente, i dati della richiesta vengono ottenuti e salvati in una variabile separata.

La variabile ad essere estratta è *\$cerca*, che contiene la stringa inserita nel campo di testo del form. viene estratta dalla richiesta e viene pulita attraverso la funzione *filterField*.

Se *cerca* è nulla, viene impostata come una stringa vuota. In questo modo non avrà effetto sulla ricerca, dato che la *query* di sql una sintassi del tipo:

```
like %.$cerca.%
```

I campi del form vengono estratti dalla richiesta, rimuovendo il primo e l'ultimo elemento, che contengono dati di Laravel.

Viene quindi ottenuto l'elenco dei nomi dei campi del form e salvato nell'array *\$mansioni*.

Infatti, selezionando un *checkbox*, esso viene inserito nella richiesta HTTP. In questo modo quindi otterrò le mansioni selezionate nella ricerca.

```
Session::forget("userName");
Session::forget("usersId");

//get data from the form, save them in separate variable.
$request = $request->all();

$cerca = $request["cerca"];
$cerca = $this->filterField($cerca);

if (is_null($cerca)) {
    $cerca = "";
}

//remove first and last element, useless. Save then array key, same as the fields names in the DB
array_shift($request);
array_pop($request);
$mansioni = array_keys($request);
```

Ora sfruttiamo la funzione *toQuery* di *EsploraController* per ottenere una *query* che abbia tutte le mansioni citate nella ricerca. La *query* viene poi usata a sua volta selezionare gli ID degli utenti dalle tabelle di database che hanno come mansione una di quelle scelte.

Gli ID degli utenti risultanti vengono quindi salvati nell'array `$ids`.

```
if (count($mansions) > 0) {
    $queryVol = (new EsploraController())->toQuery($mansions,"userMansion");
    $query = "select regUserId from person where " . $queryVol;

    //then get the actual id of the person searched.
    $idStd = DB::select($query);
    $realArray = array_values(((array) $idStd));
    foreach ($realArray as $el) {
        $el = (array) $el;
        array_push($ids, $el["regUserId"]);
    }
    ...
}
```

Infine, i valori del nome utente e degli ID degli utenti vengono salvati nelle sessioni e viene restituita la *view* del campo.

Ritornando la *view* saremo così sicuri che il calendario si ricaricherà, eseguendo la prossima funzione.

4.4.8.2 Index

Questa funzione viene eseguita ad ogni caricamento o modifica della pagina. Proprio perché il metodo POST ricarica la pagina ho deciso di usare l'approccio con le variabili di sessione appena citato.

In una prima fase questa funzione prende le variabili di sessione settate dalla funzione `checkCalendar` e le traduce in modo da poterle usare nelle *query*. Soprattutto quando parliamo degli id. se essi non sono presenti è necessario eseguire un'altra *query* rispetto a quella di default.

A tale scopo viene usata una variabile booleana apposita.

```
//get the name
$name = "";
$userMansion = [""];
if (Session::get("userName") != null) {
    $name = Session::get("userName");
}

//verify is there are users with specific mansions to search
$canProcede = false;
if (Session::get("usersId") != null) {
    $usId = Session::get("usersId");
    $usId = $usId[0];
    if (!is_null($usId)) {
        $canProcede = true;
    }
}

}
```

In seguito, se ci sono id presenti, viene fatta la seguente *query*: vengono presi solo gli utenti che hanno una disponibilità nel *range* specificato dalla richiesta AJAX e che sono contraddistinti da uno degli id specificati nell'array appena ottenuto.

Inoltre il loro nome deve adattarsi al *pattern* ricercato.

```
$ids = Session::get("userId");
$data = Event::whereDate("start", ">=", $request->start)
    ->whereDate("end", "<=", $request->end)
    ->whereIn("idVolunteer", $ids)
    ->where("title", "like", "%" . $name . "%")
    ->get([
        "id",
        "idVolunteer",
        "title",
        "start",
        "end",
        "color",
    ]);
```

Se invece non è stata specificata alcuna mansione, e quindi non ci sono utenti specifici da cercare, la linea che specifica gli id dei volontari viene omessa.

Infine i dati vengono ritornati.

```
//return data
$response = response()->json($data);
return $response;
return view("Campo-estivo.Admin.Disponibilita");
```

4.4.8.3 action

Il metodo `action()` viene chiamato quando l'utente aggiunge o modifica un evento o una disponibilità nella vista del calendario completo. Il metodo controlla se la richiesta è stata effettuata tramite AJAX e procede solo se lo è. Se la richiesta è per aggiungere un nuovo evento, il metodo crea un nuovo evento con il titolo, l'ora di inizio e l'ora di fine specificati e imposta l'attributo `isEvent` su `true`. Quindi restituisce il nuovo evento come risposta JSON. Se la richiesta è per aggiornare un evento esistente, il metodo trova l'evento con l'id specificato e aggiorna il titolo, l'ora di inizio e l'ora di fine. Quindi restituisce l'evento aggiornato come risposta JSON.

```
public function action(Request $request){
    if ($request->ajax()) {
        if ($request->type == "add") {
            $event = Event::create([
                "title" => $request->title,
                "start" => $request->start,
                "end" => $request->end,
                "isEvent" => true,
            ]);
            return response()->json($event);
        }
        ...
        if ($request->type == "delete") {
            $event = Event::find($request->id)->delete();
            return response()->json($event);
        }
    }
}
```


4.4.8.4 La view

Nella view la prima cosa utilizzata è uno script JS, che interagisce con il calendario.

Esso è diviso in due parti principali, la prima parte è l'inizializzazione del calendario, mentre la seconda parte consiste in una serie di *callback* che vengono richiamati quando l'utente interagisce con il calendario.

La prima cosa che viene fatta è il set up dell'header del calendario, con l'aggiunta dei bottoni per spostarsi avanti e indietro tra i mesi, la visualizzazione della data corrente, e la scelta tra le diverse visualizzazioni (mese, settimana, giorno). In seguito viene caricata la lista degli eventi dal server, utilizzando la funzione `$.ajax()`, che carica la funzione *index*.

Il calendario è configurato per consentire la selezione di un intervallo di tempo, attraverso la proprietà `selectable`. Quando l'utente seleziona un intervallo di tempo, viene chiamata la funzione `select()`, che apre una finestra di dialogo in cui l'utente può inserire il titolo dell'evento. Se l'utente inserisce un titolo, viene effettuata una chiamata AJAX per aggiungere l'evento al database.

```
var calendar = $('#calendar').fullCalendar({
  editable: true,
  header: {
    left: 'prev,next today',
    center: 'title',
    right: 'month,agendaWeek,agendaDay'
  },
  events: '/Campo-estivo/Admin/Disponibilita',
  selectable: true,
  selectHelper: true,
  select: function(start, end, allDay) {
    var title = prompt('Titolo evento:');
    if (title) {
      var start = $.fullCalendar.formatDate(start, 'YYYY-MM-DD HH:mm:ss');
      var end = $.fullCalendar.formatDate(end, 'YYYY-MM-DD HH:mm:ss');
      //add event
      $.ajax({
        url: "/Campo-estivo/Admin/Disponibilita/action",
        type: "POST",
        data: {
          title: title,
          start: start,
          end: end,
          type: 'add',
          isEvent: true,
        },
        success: function(data) {
          calendar.fullCalendar('refetchEvents');
        }
      })
    }
  },
  ...
});
```

Successivamente viene definito un comportamento per la modifica degli eventi. Quando un evento viene ridimensionato o spostato, viene chiamata una funzione per aggiornare il database. In entrambi i casi viene effettuata una chiamata AJAX con i nuovi dati dell'evento.

Infine, viene definita una funzione per la cancellazione degli eventi. Quando un evento viene cliccato, l'utente viene avvertito con una finestra di dialogo che gli chiede se desidera eliminare l'evento. Se l'utente conferma, viene effettuata una chiamata AJAX per cancellare l'evento dal database.

Per gestire le chiamate AJAX, viene utilizzata la funzione `$.ajaxSetup()`, che viene richiamata all'inizio del codice. Questa funzione viene utilizzata per impostare l'header X-CSRF-TOKEN per proteggere l'applicazione dalle richieste Cross-Site Request Forgery (CSRF).

Ecco un esempio di evento

```

editable: true,
//update event -> resize
eventResize: function(event, delta) {
  var start = $.fullCalendar.formatDate(event.start, 'Y-MM-DD HH:mm:ss');
  var end = $.fullCalendar.formatDate(event.end, 'Y-MM-DD HH:mm:ss');
  var title = event.title;
  var id = event.id;
  $.ajax({
    url: "/Campo-estivo/Admin/Disponibilita/action",
    type: "POST",
    data: {
      title: title,
      start: start,
      end: end,
      id: id,
      type: 'update'
    },
    success: function(response) {
      calendar.fullCalendar('refetchEvents');
    }
  })
},
...

```

Un'altra cosa degna di nota in questa view è il form, che non si distacca dagli altri form creati per il sito. In questo form sono filtrabili gli utenti per mansione e per nome. Usando dei *checkbox* si possono selezionare le mansioni desiderate ed inserire un nome da cercare. Il form poi agirà sulla funzione `checkCalendar`, documentata prima.

4.4.9 Gestione alloggi

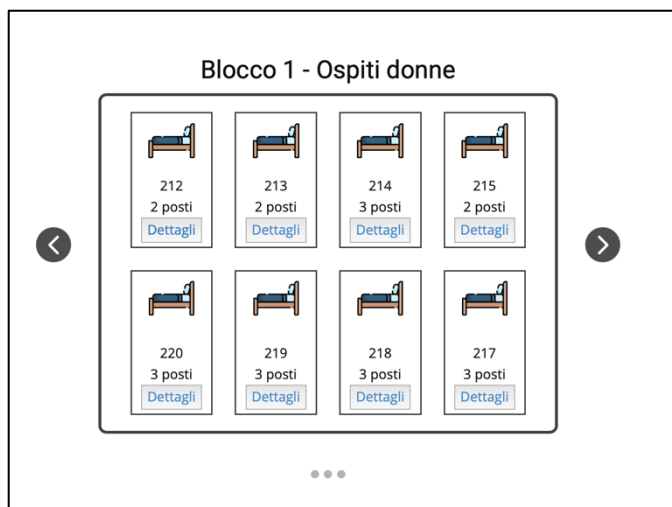


Figura 28 Alloggi

Per la gestione degli alloggi sono partito, come per il resto dal progetto, sono partito dalla realizzazione della GUI.

Ho avuto subito in mente il *design* e, sempre con l'aiuto di Nicepage, l'ho implementato. Ho pensato ad un carosello che scorresse e che mostrasse uno dopo l'altro i 3 blocchi di stanze. Dopo una discussione con il committente, e basandomi sulla planimetria fornitami dal Sig. Mombelli, ho ottenuto da lui le informazioni necessarie per sapere quante stanze fossero adibite a ospiti e quante stanze fossero adibite a volontari.

Tecnicamente cliccando su un pulsante viene visualizzata la lista degli occupanti. Il codice c'è ma non è completo e non si presta ad essere documentato più di tanto. Per ora ho solo fatto un recupero degli utenti che hanno una relazione con

la stanza.

Ogni stanza passa al controller il numero della stanza assieme al blocco, con questo formato: [blocco.stanza].

In questo modo nel codice PHP posso dividere e ottenere i dati necessari per risalire a che stanza è stata selezionata

```
// $roomData contains block.number
$roomData=explode(".", $roomData);
$block=$roomData[0];
$number=$roomData[1];
$campId=Session::get("campId");

//taking the id and the original places available in the room
$roomData = DB::table("room")
    ->where("block", $block)
    ->where("number", $number)
    ->where("campId", $campId)
    ->get(["id", "places"]);

//get the actual data
$roomData=json_decode($roomData, true)[0];
$roomId=$roomData["id"];
$places=$roomData["places"];
```

Poi prendo tutti gli utenti che sono in quella stanza

```
//search all the times that the roomId appears in the occupied table.
$occupied=DB::table("occupied")
    ->where("idRoom", $roomId)
    ->get(["idPerson"]);
$occupied=json_decode($occupied, true);
```

E infine ottengo e salvo l'id di tutte le persone in quella stanza:

```
$available=0;
if(!empty($occupied) || !is_null($occupied)){
    //get the id of all the person in the specified room.
    $personsIn=array();
    foreach($occupied as $oc){
        array_push($personsIn,$oc["idPerson"]);
    }

    //get the number of available places.
    $occupation=count($personsIn);
    $available=$places-$occupation;
}
```

Poi entra in gioco un'altra funzione che mi permette di ottenere i dati base dell'utente (nome, cognome, data di nascita) grazie all'id che viene passato

```
public function getUsersData($ids){
    //matrix with the data
    $allData=array();
    for($i=0;$i<count($ids);$i++){
        //getting data, an array is returned
        $data=DB::table("person")
            ->where("id",$ids[$i])
            ->get(["name","lastname","born_date"]);
        //decoding the array to simplify it and putting in the matrix
        array_push($allData,json_decode($data,true)[0]);
    }
    return $allData;
}
```

Infine, seguendo il principio della pagina utente, prendo questi dati e li inserisco in una tabella HTML, per poi stamparli.

```
foreach($data as $row){
    $toPrint=$toPrint."<tr>";
    foreach($row as $col){
        $toPrint=$toPrint."<td>".$col."</td>";
    }
    $toPrint=$toPrint."<tr>";
}
return view("Campo-estivo.Admin.Alloggi")
```

4.4.10 Iscrizione volontari

Anche in questo caso l'iscrizione dei volontari è una copia carbone dell'iscrizione degli ospiti.

Esempio di campo nella view.

```
<div class="u-form-group u-form-name u-label-top">
  <label for="name-6797" class="u-label">Nome</label>
  <input type="text" id="name-6797" name="nameRef" class="u-input u-input-rectangle">
</div>
```

Poi i dati vengono salvati. Siccome un utente maggiorenne non è obbligato a fornire i suoi dati medici e i dati della persona di riferimento ho dovuto mettere dei riempimenti di default nel codice PHP. Questo perché le tabelle hanno campi NOT NULL.

```
$nameRef = $request->nameRef;
$nameRef = $nameRef ?? 'Non';

$lastNameRef = $request->lastNameRef;
$lastNameRef = $lastNameRef ?? 'Fornito';
```

Poi i dati vengono inviati al DB.

```
DB::table("person")->insert([
  "name" => $nameTxt,
  "lastname" => $lastNameTxt,
  "phone_number" => $phoneNumber,
  "address" => $viaTxt,
  "CAP" => $CAP,
  "country" => $nazione,
  "type" => "volunteer",
  "born_date" => $bornDate,
  "regUserId" => $userId,
  "userState" => "suspended",
  "userMansion"=> "",
]);
```

5 Test

5.1 Protocollo di test

Test Case:	TC-01	Nome: Collegamento al sito della fondazione
Riferimento:	REQ-01	
Descrizione:	La pagina base ha il collegamento al sito della fondazione	
Prerequisiti:	-	
Procedura:	<ol style="list-style-type: none"> 1. Collegarsi al sito 2. Cliccare su “Centro diurno” 	
Risultati attesi:	<ol style="list-style-type: none"> 1. Si viene reindirizzati al sito della fondazione 	

Test Case:	TC-02	Nome: Registrazione al sito
Riferimento:	REQ-02	
Descrizione:	È possibile registrarsi al sito	
Prerequisiti:	-	
Procedura:	<ol style="list-style-type: none"> 1. Collegarsi al sito 2. Cliccare su “Campo estivo” 3. Cliccare su “Sign-up” e registrarsi 	
Risultati attesi:	<ol style="list-style-type: none"> 1. Si viene correttamente registrati 	

Test Case:	TC-03	Nome: Login al sito
Riferimento:	REQ-03	
Descrizione:	È possibile autenticati al sito	
Prerequisiti:	-	
Procedura:	<ol style="list-style-type: none"> 1. Collegarsi al sito 2. Cliccare su “Campo estivo” 3. Cliccare su “Sign-in” e registrarsi 	
Risultati attesi:	<ol style="list-style-type: none"> 1. Si viene correttamente autenticati 	

Test Case:	TC-04	Nome: Iscrizione al campo
Riferimento:	REQ-04	
Descrizione:	È possibile iscriversi al campo	
Prerequisiti:	-	
Procedura:	<ol style="list-style-type: none"> 1. Collegarsi al sito 2. Cliccare su “Campo estivo” 3. Autenticarsi 4. Cliccare su “iscrizione al campo” 5. Inserire tutti i dati 	
Risultati attesi:	1. Si viene correttamente registrati al campo	

Test Case:	TC-05	Nome: User page
Riferimento:	REQ-05	
Descrizione:	Visualizzare i dati del campo	
Prerequisiti:	Un utente registrato al campo	
Procedura:	<ol style="list-style-type: none"> 1. Collegarsi al sito 2. Cliccare su “Campo estivo” 3. Autenticarsi 4. Cliccare su “User page” 5. Controllare la presenza di dati e modificarne uno 	
Risultati attesi:	1. I dati sono consultabili e modificabili	

Test Case:	TC-06	Nome: Pagina esplorativa
Riferimento:	REQ-06	
Descrizione:	Un utente loggato può visualizzare i dati di altri utenti	
Prerequisiti:	Un utente registrato al campo	
Procedura:	<ol style="list-style-type: none"> 1. Collegarsi al sito 2. Cliccare su “Campo estivo” 3. Autenticarsi 4. Cliccare su “User page” 5. Cliccare su “Esplora” 	
Risultati attesi:	1. Si visualizza un’anteprima dei dati anagrafici di altri utenti	

Test Case:	TC-07	Nome: Pagine esplorative filtrabili
Riferimento:	REQ-06	
Descrizione:	Un utente loggato può visualizzare i dati di altri utenti e ordinarli	
Prerequisiti:	Un utente registrato al campo	
Procedura:	<ol style="list-style-type: none"> 1. Collegarsi al sito 2. Cliccare su “Campo estivo” 3. Autenticarsi 4. Cliccare su “User page” 5. Cliccare su “Esplora” 6. Provare a filtrare gli utenti cliccando sul titolo della lista 	
Risultati attesi:	<ol style="list-style-type: none"> 1. Si visualizza un’anteprima dei dati anagrafici di altri utenti 	

Test Case:	TC-08	Nome: Pagine personali altri utenti
Riferimento:	REQ-07	
Descrizione:	Un utente loggato può visualizzare i dati di un utente aprendolo dalla pagina esplorativa	
Prerequisiti:	Un utente registrato al campo	
Procedura:	<ol style="list-style-type: none"> 1. Collegarsi al sito 2. Cliccare su “Campo estivo” 3. Autenticarsi 4. Cliccare su “User page” 5. Cliccare su “Esplora” 6. Provare a cliccare su un utente 	
Risultati attesi:	<ol style="list-style-type: none"> 1. Si visualizzano tutti i dati anagrafici di un utente 2. Con utente “admin” è possibile modificare i dati 	

Test Case:	TC-09	Nome: Creazione e modifica di un campo
Riferimento:	REQ-08	
Descrizione:	Un amministratore, può modificare i dati sul campo corrente o su quelli passati. Inoltre può creare una nuova edizione del campo, inserendo tutte le informazioni necessarie.	
Prerequisiti:	Un admin autenticato al sito	
Procedura:	<ol style="list-style-type: none"> 1. Collegarsi al sito 2. Cliccare su “Campo estivo” <p>I dati sono modificabili</p> <ol style="list-style-type: none"> 1. Cliccare su “User page” 2. Cliccare su “Esplora” 3. Cliccare su “Gestione Alloggi” 4. Cliccare su “Nuovo campo” 	

Test Case:	TC-10	Nome: Creazione account ospite
Riferimento:	REQ-09	
Descrizione:	Un amministratore può autonomamente creare un account per un ospite.	
Prerequisiti:	Un admin autenticato al sito	
Procedura:	1. Meccanica non prevista	
Risultati attesi:		

Test Case:	TC-11	Nome: Attivazione ed eliminazione account
Riferimento:	REQ-10	
Descrizione:	Un amministratore può attivare o eliminare l'account di un volontario o di un ospite.	
Prerequisiti:	Un admin autenticato al sito	
Procedura:	<ol style="list-style-type: none"> 1. Collegarsi al sito 2. Cliccare su "Campo estivo" 3. Autenticarsi 4. Cliccare su "User page" 5. Cliccare su "Esplora" 6. Modificare il campo "Stato" 	
Risultati attesi:	Viene modificato lo stato dell'utente.	

Test Case:	TC-12	Nome: Disponibilità volontari
Riferimento:	REQ-11	
Descrizione:	Un amministratore può visualizzare e modificare la disponibilità dei suoi volontari	
Prerequisiti:	Un admin autenticato al sito	
Procedura:	<ol style="list-style-type: none"> 1. Collegarsi al sito 2. Cliccare su "Campo estivo" 3. Autenticarsi 4. Cliccare su "User page" 5. Cliccare su "Esplora" 6. Cliccare su "Gestione disponibilità" 7. Visualizzare e modificare la disponibilità dal calendario 	
Risultati attesi:	Vengono visualizzate le disponibilità, filtrabili.	


Test Case:	TC-13	Nome: Gestione eventi
Riferimento:	REQ-12	
Descrizione:	Un amministratore può visualizzare e modificare gli eventi del campo	
Prerequisiti:	Un admin autenticato al sito	
Procedura:	<ol style="list-style-type: none"> 1. Collegarsi al sito 2. Cliccare su “Campo estivo” 3. Autenticarsi 4. Cliccare su “User page” 5. Cliccare su “Esplora” 6. Cliccare su “Gestione disponibilità” 7. Visualizzare e modificare gli eventi 	
Risultati attesi:	Vengono creati/spostati/eliminati degli eventi	

Test Case:	TC-14	Nome: Gestione alloggi
Riferimento:	REQ-13	
Descrizione:	Un amministratore può visualizzare e modificare gli alloggi del campo	
Prerequisiti:	Un admin autenticato al sito	
Procedura:	<ol style="list-style-type: none"> 1. Collegarsi al sito 2. Cliccare su “Campo estivo” 3. Autenticarsi 4. Cliccare su “User page” 5. Cliccare su “Esplora” 6. Cliccare su “Gestione alloggi” 7. Visualizzare e modificare gli occupanti 	
Risultati attesi:	Vengono assegnati/tolti utenti alle stanze	

Test Case:	TC-15	Nome: Sito responsive
Riferimento:	REQ-14	
Descrizione:	Il sito è <i>responsive</i>	
Prerequisiti:	-	
Procedura:	<ol style="list-style-type: none"> 1. Collegarsi al sito 2. Provare a ridimensionare la pagina 	
Risultati attesi:	<ol style="list-style-type: none"> 1. Le informazioni vengono sempre presentate bene 	

5.2 Risultati test


Test Case	Esito	Data ultimo test	Risultato	Commenti
TC-01	Passato	05.04.2023	Pagina della fondazione aperta	
TC-02	Passato	05.04.2023	Utente aggiunto e autenticato al sito	
TC-03	Passato	05.04.2023	Utente autenticato al sito	
TC-04	Passato	05.04.2023	Si viene correttamente registrati al campo	
TC-05	Passato	05.04.2023	I dati sono consultabili e modificabili	
TC-06	Passato	05.04.2023	Si visualizza un'anteprima dei dati anagrafici di altri utenti	
TC-07	Passato	05.04.2023	Si visualizza un'anteprima dei dati anagrafici di altri utenti	
TC-08	Passato	05.04.2023	<ul style="list-style-type: none"> Si visualizzano tutti i dati anagrafici di un utente. Con utente "admin" è possibile modificare i dati 	
TC-09	Parzialmente passato	05.04.2023	<ul style="list-style-type: none"> I dati vengono modificati Impossibile creare il campo 	Solo modifica
TC-10	Fallito	05.04.2023	Fallito	Funzione non implementata
TC-11	Passato	05.04.2023	Gli account vengono attivati o disattivati	
TC-12	Passato	05.04.2023	La disponibilità è presente e modificabile	
TC-13	Parzialmente passato	05.04.2023	L'evento viene creato	Funzione un po' legnosa
TC-14	Fallito	05.04.2023	Fallito	Funzione non finita
TC-15	Passato	05.04.2023	Il sito è responsive	

	SAMT – Sezione Informatica	Pagina 68 di 73
	Gestione Campo Estivo – Versione 2.0	

5.3 Mancanze/limitazioni conosciute

Sono abbastanza felice del punto raggiunto dal mio progetto e di aver soddisfatto quasi tutti i requisiti richiesti. Tuttavia, non posso fare a meno di sentire un po' di delusione per le mancanze ancora presenti. In particolare, la gestione degli alloggi non è completa e so che ci sono delle aree in cui posso ancora migliorare la grafica e la sicurezza. Inoltre, avrei voluto implementare una gestione dei certificati medici per i volontari, ma non ho avuto il tempo di farlo.

Nonostante queste mancanze, sono comunque orgoglioso del lavoro svolto e credo di aver creato un prodotto comunque funzionante e usabile dagli utenti del campo estivo. Sono grato per le sfide che ho incontrato lungo il percorso, poiché mi hanno aiutato a crescere come sviluppatore e ad affinare le mie abilità.

	SAMT – Sezione Informatica	Pagina 69 di 73
	Gestione Campo Estivo – Versione 2.0	

6 Consuntivo

Rispetto al primo progetto le tempistiche sono funzionate meglio. Ho previsto le attività in maniera abbastanza corretta, anche se ho sovrastimato il tempo per la gestione del calendario, il che mi ha permesso di guadagnare tempo.

Allo stesso tempo per creare alcune parti del progetto, come l'ordinamento della lista nella pagina di esplorazione che comprendesse un indicazione della direzione e il colore alla colonna selezionata, mi hanno portato via più tempo del previsto. Anche indicare in questa lista le persone di riferimento e a chi appartenessero non è stato evidente, dato che ho dovuto lavorarci a lungo per ottenere il risultato che desideravo. Infine un'operazione chirurgica non prevista mi ha portato via circa una settimana di lavoro, che mi ha probabilmente rallentato nello sviluppo. Sono comunque riuscito a completare una parte dei requisiti e sono più soddisfatto rispetto al primo semestre.

7 Conclusioni

Riscrivo e correggo il testo:

Dopo quasi un anno di lavoro su questo progetto, mi ritengo mediamente soddisfatto del risultato finale. Se nel progetto precedente mi ero concentrato principalmente sulla progettazione delle interfacce, il funzionamento del backend e la struttura del database, in questo progetto ho potuto sviluppare molto di più la programmazione. Grazie al fatto di avere già le interfacce pronte, ho potuto dedicare molto tempo allo sviluppo del codice e concentrarmi sull'implementazione delle soluzioni desiderate. Ammetto di non essere mai stato un grande fan della programmazione di backend web, specialmente del PHP. Per me è sempre stato un linguaggio inutilmente complesso e con troppe differenze rispetto ad altri linguaggi. Tuttavia, ho cambiato un po' la mia opinione, poiché mi sono abituato a vederlo e ho acquisito sempre più familiarità con le sue particolarità. Sono convinto che questo sia dovuto in gran parte a Laravel, un framework molto utilizzato e affidabile nel mondo del lavoro e della produzione, con il quale si possono realizzare progetti molto grandi e importanti. Come ho scoperto nel primo semestre, la curva di apprendimento di Laravel è considerevole e non è facile iniziare a comprendere tutte le sue particolarità e regole. Tuttavia, una volta apprese le basi, si può fare sul serio. Infatti, questa volta è andata molto meglio, ho potuto utilizzare le tecniche apprese nel primo semestre e metterle in pratica nel secondo.

Non è stata la prima volta che mi sono dedicato al refactoring di un progetto. L'anno scorso, nel primo semestre, abbiamo sviluppato in gruppo Drone2.0, una versione evoluta di un progetto dello stesso gruppo dell'anno precedente. Nel secondo semestre, abbiamo sviluppato Dino Run and Jump, un altro progetto evoluto di un progetto precedente sviluppato da un altro gruppo. Trovo questa attività di refactoring molto interessante, soprattutto se viene effettuata su del codice che si conosce e non su quello di qualcun altro. Pertanto, il processo di refactoring e analisi del codice precedente è stato molto efficace ed è una delle parti di cui sono più soddisfatto.

Rispetto al primo semestre, mi ritengo abbastanza soddisfatto del progetto. Tuttavia, ci sono alcune mancanze che avrei voluto colmare. Sono comunque contento di aver soddisfatto buona parte dei requisiti e di aver risolto dei problemi che inizialmente pensavo di dover lasciare nella consegna finale. È stato stimolante programmare fin dall'inizio, poiché ho potuto concentrarmi sui dettagli tecnici e logici del codice. Questo secondo progetto mi ha permesso di migliorare le mie competenze in un sito che ormai conosco bene. In conclusione, mi dispiace non aver terminato il progetto, ma forse senza ritardi nel lavoro e assenze personali, avrei potuto fare di più.

7.1 Sviluppi futuri

Rispetto al primo semestre vedo meno sviluppi futuri, anche se rimangono simili nelle idee.

In questa versione per il futuro della gestione del Campo Estivo di Olivone, ci sono molte ancora per migliorare e sviluppare il progetto.

In primo luogo, credo che ci sia un'opportunità per migliorare il sito web del campo estivo. Potrebbe beneficiare di un *restyling* grafico per renderlo più accattivante e facile da navigare. Inoltre, siccome ci sono dati medici coinvolti, si potrebbe pensare di implementare una crittografia per garantire la sicurezza delle informazioni.

La gestione degli alloggi è un'area che richiede ulteriore lavoro. Attualmente, il codice scritto è minimo e non soddisfacente, quindi ci sono molte opportunità per migliorarlo e renderlo più efficiente.

Per quanto riguarda la gestione degli eventi si può sicuramente. Al momento, è troppo basilare e presenta alcune imperfezioni. Ci sono molte opportunità per rendere questa parte del progetto più efficiente e *user-friendly*.

Infine, la gestione dei certificati dei volontari è assente e questo rappresenta un problema in termini di usabilità complessiva, anche se questa parte può essere gestita ancora su carta senza pregiudicare troppo il funzionamento del progetto. Questa rimane comunque un'area che richiederà uno sviluppo futuro.

Complessivamente, sebbene ci siano ancora molte aree che richiedono attenzione e sviluppo, sono soddisfatto di questa evoluzione.

8 Bibliografia

8.1 Sitografia

- <https://fullcalendar.io/docs/getting-started>
- <https://laravel.com/docs/9.x/authentication>
- <https://laravel.com/docs/9.x/blade>
- <https://laravel.com/docs/9.x/configuration>
- <https://laravel.com/docs/9.x/controllers>
- <https://laravel.com/docs/9.x/database>
- <https://laravel.com/docs/9.x/database>
- <https://laravel.com/docs/9.x/frontend>
- <https://laravel.com/docs/9.x/installation>
- <https://laravel.com/docs/9.x/logging>
- <https://laravel.com/docs/9.x/middleware>
- <https://laravel.com/docs/9.x/migrations>
- <https://laravel.com/docs/9.x/pagination>
- <https://laravel.com/docs/9.x/queries>
- <https://laravel.com/docs/9.x/requests>
- <https://laravel.com/docs/9.x/responses>
- <https://laravel.com/docs/9.x/routing>
- <https://laravel.com/docs/9.x/seeding>
- <https://laravel.com/docs/9.x/session>
- <https://laravel.com/docs/9.x/starter-kits>
- <https://laravel.com/docs/9.x/structure>
- <https://laravel.com/docs/9.x/views>
- https://www.mrw.it/mysql/come-eliminare-violazioni-integrita-database-mysql_12933.html
- https://www.mrw.it/mysql/delete-cumulativi-tabelle-mysql_12947.html
- https://www.mrw.it/mysql/insert-into-select_12954.html
- <https://www.w3schools.com/js/default.asp>
- https://www.w3schools.com/js/js_ajax_http_response.asp
- https://www.w3schools.com/js/js_ajax_http_send.asp
- https://www.w3schools.com/js/js_ajax_http.asp
- https://www.w3schools.com/js/js_ajax_intro.asp
- https://www.w3schools.com/js/js_ajax_php.asp
- <https://www.w3schools.com/php/>
- https://www.w3schools.com/php/php_date.asp
- https://www.w3schools.com/php/php_forms.asp
- https://www.w3schools.com/php/php_functions.asp
- https://www.w3schools.com/php/php_if_else.asp
- https://www.w3schools.com/php/php_json.asp
- https://www.w3schools.com/php/php_switch.asp
- https://www.w3schools.com/sql/sql_and_or.asp
- https://www.w3schools.com/sql/sql_delete.asp
- https://www.w3schools.com/sql/sql_drop_table.asp
- https://www.w3schools.com/sql/sql_select.asp
- https://www.w3schools.com/sql/sql_top.asp
- https://www.w3schools.com/sql/sql_update.asp
- <https://www.geekandjob.com/wiki/ajax>
- <https://bloginnovazione.it/composer-php/35920/>
- <https://vitolavecchia.altervista.org/definizione-caratteristiche-e-tipologie-di-controller-in-informatica/>
- <https://nordvpn.com/it/blog/csrf-xsrf/>
- <https://aulab.it/notizia/272/cose-laravel-e-come-impararlo>
- https://www.ilsoftware.it/articoli.asp?tag=Header-in-informatica-a-cosa-servono-nelle-email-nei-file-e-in-ambito-networking_25715

9 Glossario

AJAX; 54; 55; 56

Abbreviazione di Asynchronous JavaScript and XML, indica una combinazione di tecnologie di sviluppo usate per creare pagine web dinamiche

callback; 55

In programmazione, una callback o richiamo è generalmente una funzione o un "blocco di codice" che viene passato come parametro a un'altra funzione.

Composer; 29

Composer è uno strumento di gestione delle dipendenze, open source per PHP, creato principalmente per facilitare la distribuzione e la manutenzione dei pacchetti PHP come singoli componenti dell'applicazione.

controller; 31; 34; 37; 38; 39; 40; 42; 49; 52; 57

un dispositivo hardware o un programma software che gestisce o dirige il flusso di dati tra due entità.

CSRF; 56

CSRF sta per Cross-Site Request Forgery, ovvero falsificazione di richieste tra siti: si tratta fondamentalmente di una richiesta di azione che viene inviata da un sito web a un altro sfruttando una sessione autenticata di un utente

FK; 26; 38

vedi foreign key

foreign key; 15; 16; 25; 27

Una chiave esterna (in inglese foreign key), nel contesto dei database relazionali, è un vincolo di integrità referenziale tra due o più tabelle.

Fortify; 30; 31; 32; 36; 38

Libreria open source di Laravel per il controllo di accesso ai siti

GUI; 57

Graphical user interface: l'interfaccia grafica.

header; 28; 46; 55; 56

Nel contesto dell'HTTP, gli header sono utilizzati per trasmettere informazioni aggiuntive tra client e server.

Laravel; 2; 4; 10; 13; 14; 29; 30; 31; 36; 37; 38; 53; 68

Laravel è un framework scritto in PHP e usato per sviluppare applicazioni web

MVC; 14

Model-View-Controller (MVC) è un modello di architettura del software

primary key; 14

Insieme di attributi che permette di individuare univocamente un record o tupla o ennupla in una tabella o relazione.

route; 30; 32; 34; 35; 37; 40; 46; 49

In Laravel, una "route" è un meccanismo per definire come l'applicazione risponde a una determinata richiesta HTTP.

sessione; 34; 35; 39; 40; 44; 46; 53; 54

Sessione, in informatica e telecomunicazioni, è un termine che può essere utilizzato per indicare: il colloquio tra utente ed elaboratore su un determinato argomento, tipicamente un'applicazione software;

template; 30

Modello. Per esempio di un sito

toggle; 35; 39; 40

Meccanismo per cambiare da uno stato all'altro TOKEN; 56

Un indicatore univoco registrato, con funzione di rappresentare un oggetto digitale

view; 29; 30; 31; 32; 34; 35; 38; 39; 41; 42; 44; 46; 47; 49; 52; 54; 55; 57; 59

Vista. In questo caso la gui che l'utente vede.

10 Allegati

- QdC
- Abstract
- Diari di lavoro
- Sito
- Script database
- Gantt Preventivo
- Gantt Consuntivo
- Allegati Campo Estivo
 - Formulari originali
 - Corredo
 - Regole
 - Ecc.
- Progettazione
 - Diagrammi di flusso
 - Design interfacce
 - Schema DB
 - Use Case