Nivula SA

Web Booking Engine Smart

Manuale Tecnico

**Indice**

[1. Tecnologie Utilizzate 4](#_Toc21954381)

[1.1. Bootstrap 4](#_Toc21954382)

[1.2. jQuery 4](#_Toc21954383)

[1.3. GSAP 4](#_Toc21954384)

[1.4. Bootstrap-Datepicker 4](#_Toc21954385)

[1.5. Animate.css 5](#_Toc21954386)

[2. Software Utilizzati 6](#_Toc21954387)

[2.1. Visual Studio Code 6](#_Toc21954388)

[2.2. Laragon 6](#_Toc21954389)

[2.3. SourceTree 6](#_Toc21954390)

[3. Adobe XD 7](#_Toc21954391)

[4. Interfaccia 8](#_Toc21954392)

[4.1. Prima sezione 8](#_Toc21954393)

[4.1.1. Sfondo (1) 8](#_Toc21954394)

[4.1.2. Campi data (2) 9](#_Toc21954395)

[4.1.3. Campi con valore incrementale (3) 10](#_Toc21954396)

[4.1.4. Pulsante “Cerca camere” (4) 11](#_Toc21954397)

[4.2. Seconda Sezione 11](#_Toc21954398)

[4.2.1. Secondo form (1) 11](#_Toc21954399)

[4.2.2. Card info camera (2) 12](#_Toc21954400)

[4.2.3. Carrello camere scelte (3) 15](#_Toc21954401)

[4.2.4. Card camera selezionata (4) 15](#_Toc21954402)

[4.2.5. Offerte camera (5) 15](#_Toc21954403)

[4.3. Terza sezione 15](#_Toc21954404)

[5. Funzioni 16](#_Toc21954405)

[5.1. Datepicker 16](#_Toc21954406)

[5.1.1. Parametri base 16](#_Toc21954407)

[5.1.2. Valore minimo campo Data Ritorno 16](#_Toc21954408)

[5.1.3. Gestione errori 17](#_Toc21954409)

[5.2. Tooltip 19](#_Toc21954410)

[5.3. Contatori 20](#_Toc21954411)

[5.3.1. Età bambini 20](#_Toc21954412)

[5.4. Carosello 22](#_Toc21954413)

[5.4.1. Carosello modale 23](#_Toc21954414)

[6. Animazioni 24](#_Toc21954415)

[6.1. Animazione d’entrata 24](#_Toc21954416)

[6.1.1. Prima sezione 24](#_Toc21954417)

[6.1.2. Seconda sezione 24](#_Toc21954418)

[6.2. Transizioni 25](#_Toc21954419)

[6.2.1. Transizione da prima a seconda sezione 25](#_Toc21954420)

[6.2.2. Transizione da seconda a terza sezione 26](#_Toc21954421)

[6.3. Animazioni Elementi 26](#_Toc21954422)

[6.3.1. Offerte camera 26](#_Toc21954423)

[6.3.2. Contenitore Età bambini 28](#_Toc21954424)

[6.3.3. Carrello camere selezionate 28](#_Toc21954425)

[7. Differenze tra Desktop e Mobile 29](#_Toc21954426)

[8. Compatibilità Cross-Browser 30](#_Toc21954427)

[8.1. Chrome 30](#_Toc21954428)

[8.2. Firefox 30](#_Toc21954429)

[8.3. Edge 31](#_Toc21954430)

[8.4. Internet Explorer 31](#_Toc21954431)

[8.5. Safari 31](#_Toc21954432)

# Tecnologie Utilizzate

## Bootstrap

Bootstrap è il framework più utilizzato per lo sviluppo di applicazioni web tramite HTML, CSS e JavaScript, grazie soprattutto al fatto di essere compatibile con tutti i principali Browser, per il fatto di dare grande importanza al rendere le pagine multipiattaforma seguendo gli standard del web responsive design e la possibilità di modificare ogni singolo aspetto grafico tramite poche righe di CSS.

## jQuery

jQuery è una libreria JavaScript molto utile per la manipolazione di elementi HTML, nonostante JavaScript puro permetta già la possibilità di alterare gli elementi presenti all’interno del DOM, questa libreria ci da la possibilità di rendere questi processi molto più semplici da utilizzare, mettendo a nostra disposizione svariati metodi.

## Risultati immagini per gsapGSAP

GSAP è una libreria JavaScript interamente dedicata alle animazioni, che permettere di animare a nostro piacimento ogni elemento HTML.

## Bootstrap-Datepicker

Boostrap-Datepicker è una libreria JavaScript che si basa su jQuery e Bootstrap, per creare dei datepicker che seguano lo stile grafico tipico di quest’ultimo.

## Animate.css

Animate.CSS è una libreria CSS interamente dedicata alle animazioni, che dà la possibilità di implementare semplici animazioni in pochi secondi.

# Software Utilizzati

## Risultati immagini per visual studio codeVisual Studio Code

Visual Studio Code è un editor di codice, sviluppato da Microsoft, che si adatta particolarmente allo sviluppo di siti web, grazie alla sua compatibilità con la maggior parte dei linguaggi di programmazione utilizzati in questo ambiente e al vasto supporto offerto dalle compagnie di terze parti che mettono a disposizione un vastissimo numero di estensioni.

## Risultati immagini per laragonLaragon

Laragon è un software che permette di creare un server locale, in cui hostare i propri progetti web in maniera semplice e veloce.

## SourceTree

SourceTree è un software che agisce da client per Git, dando quindi la possibilità di interagire in maniera immediata con i repositories di GitHub e GitLab, attraverso funzioni come push, pull e commit.

# Risultati immagini per adobe xdAdobe XD

Adobe Experience Design o Adobe XD, è un software per dispositivi Windows e Mac che permette di progettare interfacce per differenti tipologie di dispostivi, come per esempio tablet, smartphone o desktop.

Questo programma è molto utile per la creazione di interfacce Utente, siccome ogni elemento inserito all’interno del nostro Artboard può essere animato, dandoci quindi la possibilità di creare prototipi molto vicini al prodotto finale. Vi è inoltre la possibilità di rendere i nostri progetti funzionali, tramite la creazione di semplici eventi che colleghino le varie schermate, questi possono essere il semplice click, il trascinare un elemento, tramite comando vocale o sfruttando la pressione di uno specifico pulsante nella tastiera o nel gamepad.

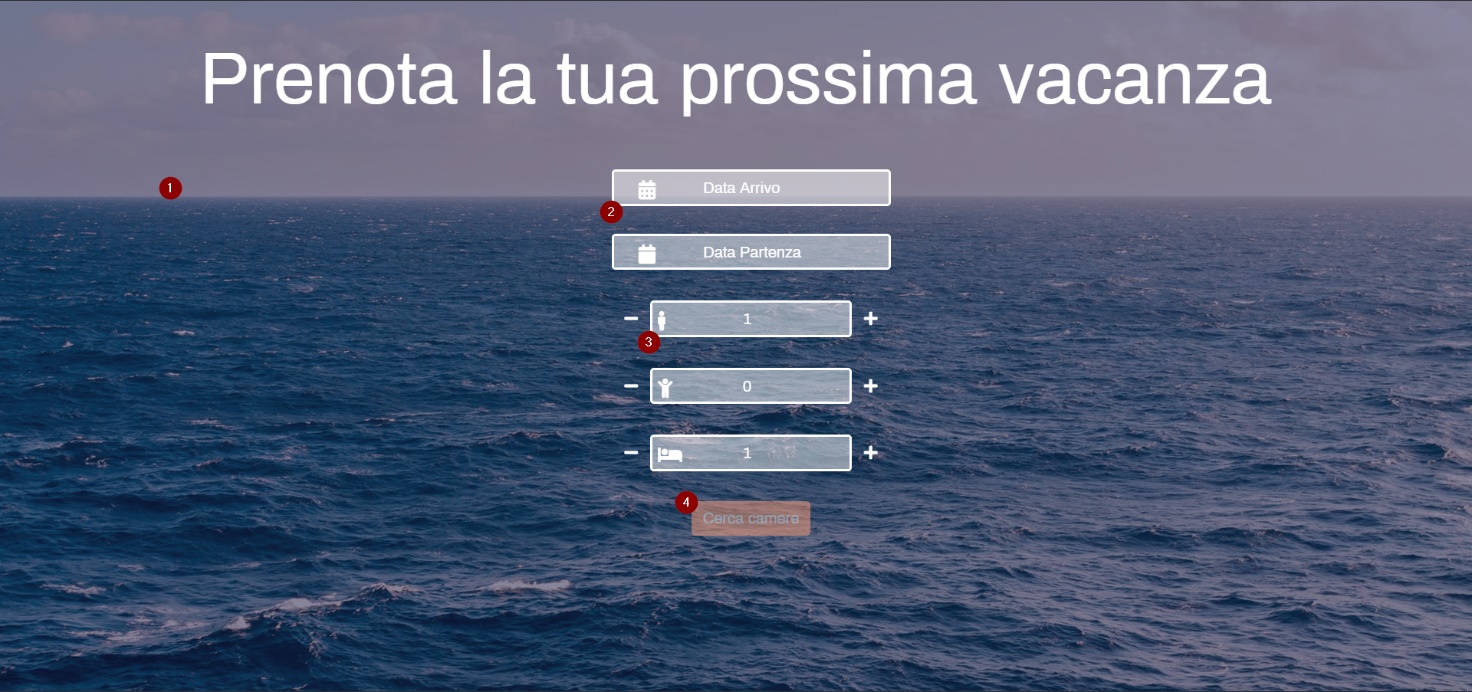
Adobe mette inoltre a disposizione una app per Android e IOS che permette di testare il funzionamento dei nostri mockup direttamente sui nostri smartphone.

Questo strumento dispone inoltre di un grande supporto da parte della community, che mette a disposizione svariati video e articoli istruttivi molto validi. Mentre le aziende di terze parti offrono una vasta gamma di plugins, che ci danno la possibilità per esempio di inserire immagini o illustrazioni royalty-free, nei nostri design.

Per questo progetto ho utilizzato la versione gratuita Adobe XD, sono disponibili due versioni a pagamento che permettono di avere a disposizione uno spazio di archiviazione cloud maggiore per salvare i vari prototipi, ma ho deciso di fare affidamento alla versione base, siccome ritengo che i 2 Gigabyte nel cloud offerti siano sufficienti per il mio utilizzo dell’applicazione.

# Interfaccia

## Prima sezione



### Sfondo (1)

Lo sfondo della prima sezione è coperto da un overlay, necessario per rendere i campi del form più visibili.



La base html di questo overlay è un semplice div, che viene espanso tramite il CSS per coprire l’intera schermata. Il colore e l’opacità di questo elemento è gestito attraverso la variabile CSS “overlay-color”, in cui viene specificato il colore e il grado di trasparenza tramite il metodo RGBA.

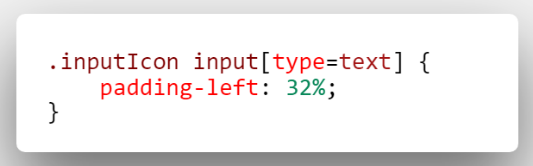
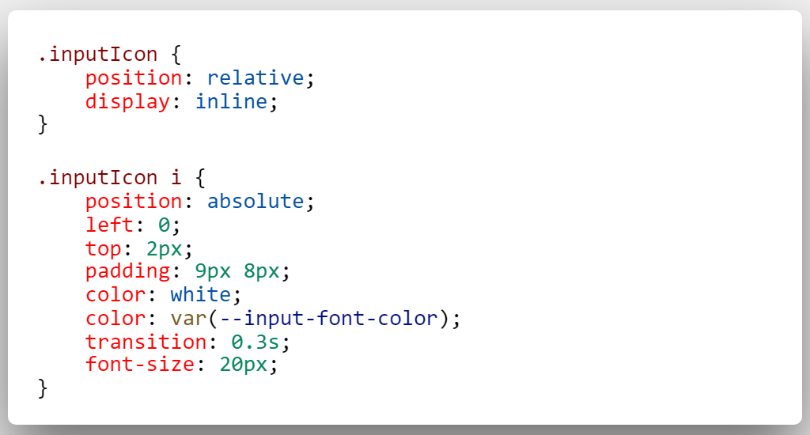
L’immagine utilizzata come sfondo, dovrà venir specificata dall’hotel, in modo che quest’ultimo possa scegliere una foto che rappresenti l’hotel, come per esempio uno scatto della struttura.

### Campi data (2)

Lo stile dei campi data presenti è gestito da più classi CSS, tra queste “form-control”, che viene fornita da Bootstrap, che in questo caso viene usata per gestire le caratteristiche base come le specifiche del testo.



La classe “form-style” gestisce lo sfondo, i bordi e il colore del testo, attributi delegati a delle variabili, che permettono ne permettono la facile modifica in futuro.

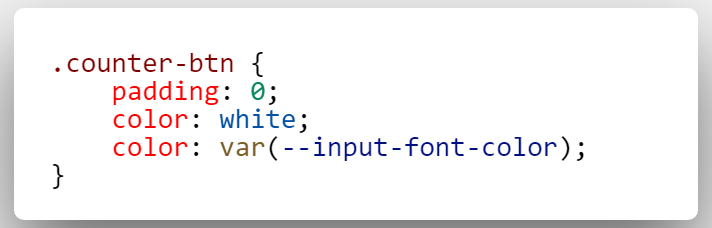


Le due icone utilizzate per questi due campi sono “calendar-alt” e “calendar” entrambe provenienti dal famoso set di icone Font Awesome[[1]](#footnote-1), queste vengono poi gestite tramite la classe “inputIcon” che si occupa di gestire la posizione, il colore e le dimensioni. Al contrario degli altri campi, i due campi data sono più larghi, perciò ho creato un selettore che discrimini esclusivamente questi due input, attraverso il valore specificato nell’attributo “type”, permettendomi di scrivere una regola che adatti il posizionamento delle icone diversamente.

Maggiori informazioni su come sono stati implementati i datepicker e le loro funzioni si possono trovare al capitolo **5.1 Datepicker**.

### Campi con valore incrementale (3)

Ciascuno dei campi con valore incrementale presente nella prima sezione, è accompagnato da due pulsanti: uno a sinistra che raffigura un meno ed uno a destra che invece raffigura un più, il primo aggiunto tramite l’icona “minus” mentre il secondo “plus” entrambe provenienti dal set messo a disposizione da Font Awesome.



Queste icone sono inserite all’interno di un semplice pulsante bootstrap, mentre le loro regole stilistiche vengono gestite dalla classe “counter-btn”.

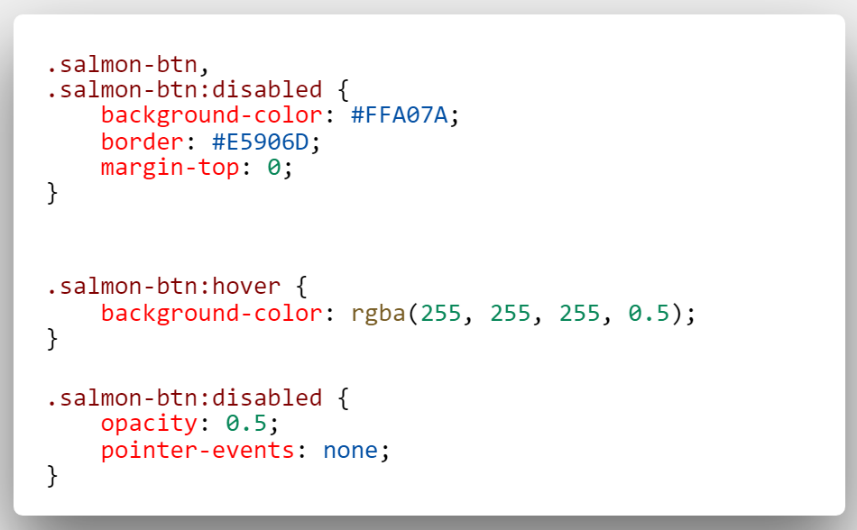


Il valore selezionato dall’utente viene visualizzato all’interno di un campo input, a cui è stato attributo lo stato “readonly”, questo elemento viene ancora gestito dalla classe “form-control” per gestire gli attributi di base, mentre il resto viene gestita dalla classe “form-counter”, nella quale mi sono visto costretto a fare più volte uso dell’attributo CSS “!important”, questo permette di sovrascrivere le regole scritte in altre classi che fanno riferimento allo stesso elemento.

Maggiori informazioni su come sono stati implementati i contatori si possono trovare al capitolo **5.3 Contatori**.

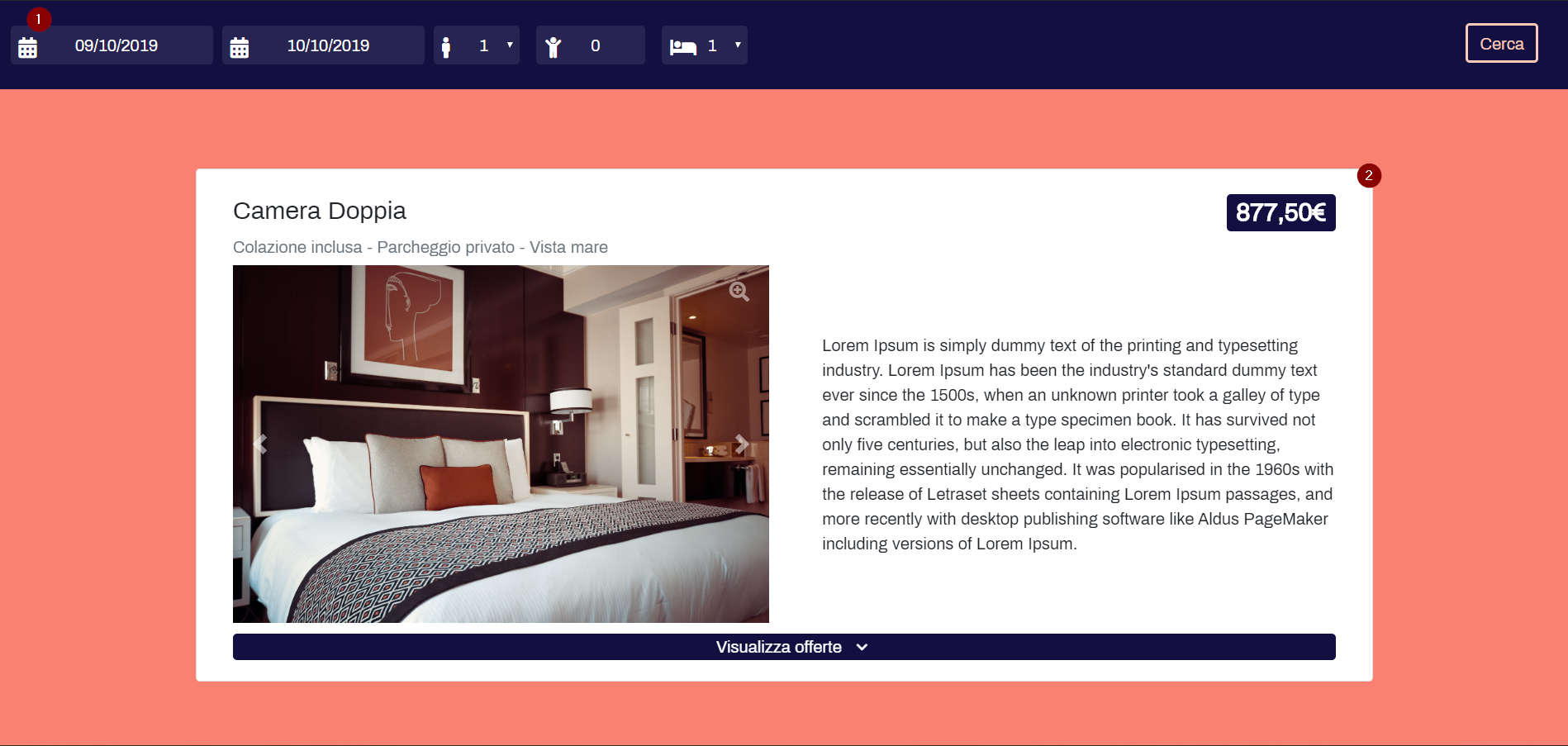
### Pulsante “Cerca camere” (4)

Il pulsante è stato creato sulla base della classe Bootstrap “btn-primary” a cui ho aggiunto modifiche tramite la classe “salmon-btn”.



Questa classe gestisce il colore del bordo e dello sfondo, inoltre attraverso l’uso delle pseudo classi “:hover” e “:disabled“ ho aggiunto regole che vengono utilizzate rispettivamente quando l’utente passa con il puntatore sopra l’elemento e quando quest’ultimo è disabilitato, in modo da evidenziare il fatto che non sia ancora utilizzabile.

## Seconda Sezione



### Secondo form (1)

Il secondo form, che ha la funzione di ricordare all’utente i parametri selezionati in precedenza e di permettere di eseguire una nuova ricerca, è contenuto all’interno di un div a cui è stata aggiunta la classe “row” che ci dà la possibilità di utilizzare il grid system di Bootstrap[[2]](#footnote-2), per gestire il posizionamento dei vari campi input che compongono il nostro form.

I campi data all’interno di questo form sono rimasti invariati rispetto a quelli visti nella prima sezione, da un punto di vista prettamente strutturale, gli altri campi invece sono stati modificati, i campi adulti e camere sono diventati degli elenchi a discesa, mentre il campo relativo al numero dei bambini apre un contenitore separato in cui vengono gestite le varie età (vedi capitolo **5.3.1 Età bambini**).



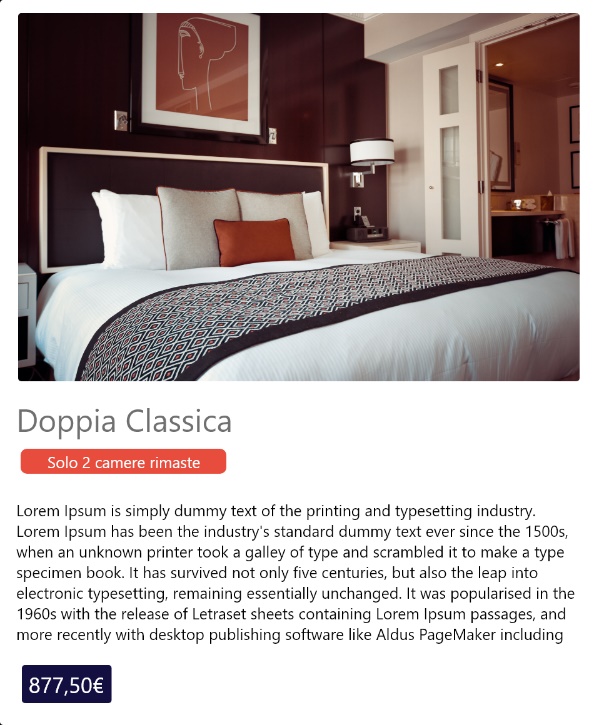
Tutti i campi sono gestiti dalla classe “form-style” alla quale sono state aggiunte alcune regole create appositamente per questi elementi, che permettono di rimuove il bordo e cambiare lo sfondo, per adattarlo a quello del contenitore.



La classe “nav-btn” gestisce lo stile del pulsante “Cerca”, la grandezza e la posizione.

### Card info camera (2)

Le card che contengono le informazioni riguardanti le varie tipologie di camere che si possono vedere ora, sono frutto di un processo di studio, durante il quale ho valutato diverse tipologie di layout:

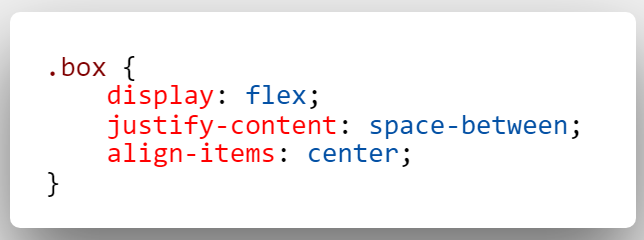


La prima che sfruttava un approccio quasi esclusivamente verticale, viene dato molto spazio alla descrizione che insieme all’immagine occupa gran parte dello spazio.

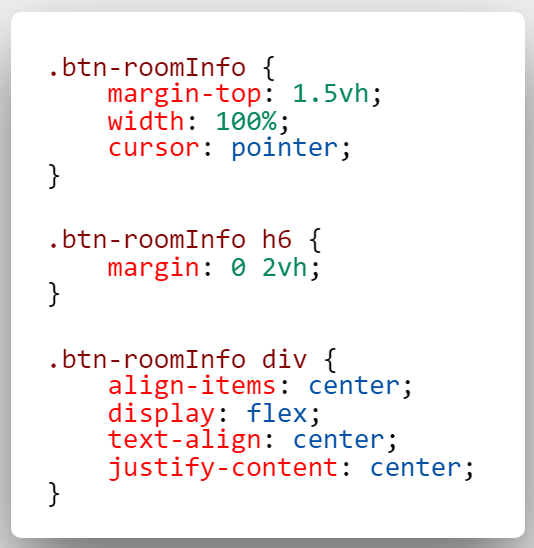


Si è poi deciso di passare ad un design che sfruttasse maggiormente la larghezza, allineando alcuni elementi solo stesso piano orizzontale, come le foto e la descrizione, in modo da diminuire considerevolmente l’altezza, rendendo il tutto più compatto. È stato inoltre introdotto il concetto di parole chiave, che da la possibilità all’utente di farsi un’idea dei servizi offerti, senza doversi per forza leggere l’intera descrizione.

Per poi giungere infine al design attuale, che raggruppa il titolo e le parole chiave in alto sinistra, il prezzo è stato spostato in alto a destra per separarlo dagli altri elementi e renderlo più visibile, è stato poi aggiunto un pulsante che da la possibilità di visualizzare le varie offerte disponibili per la camera.

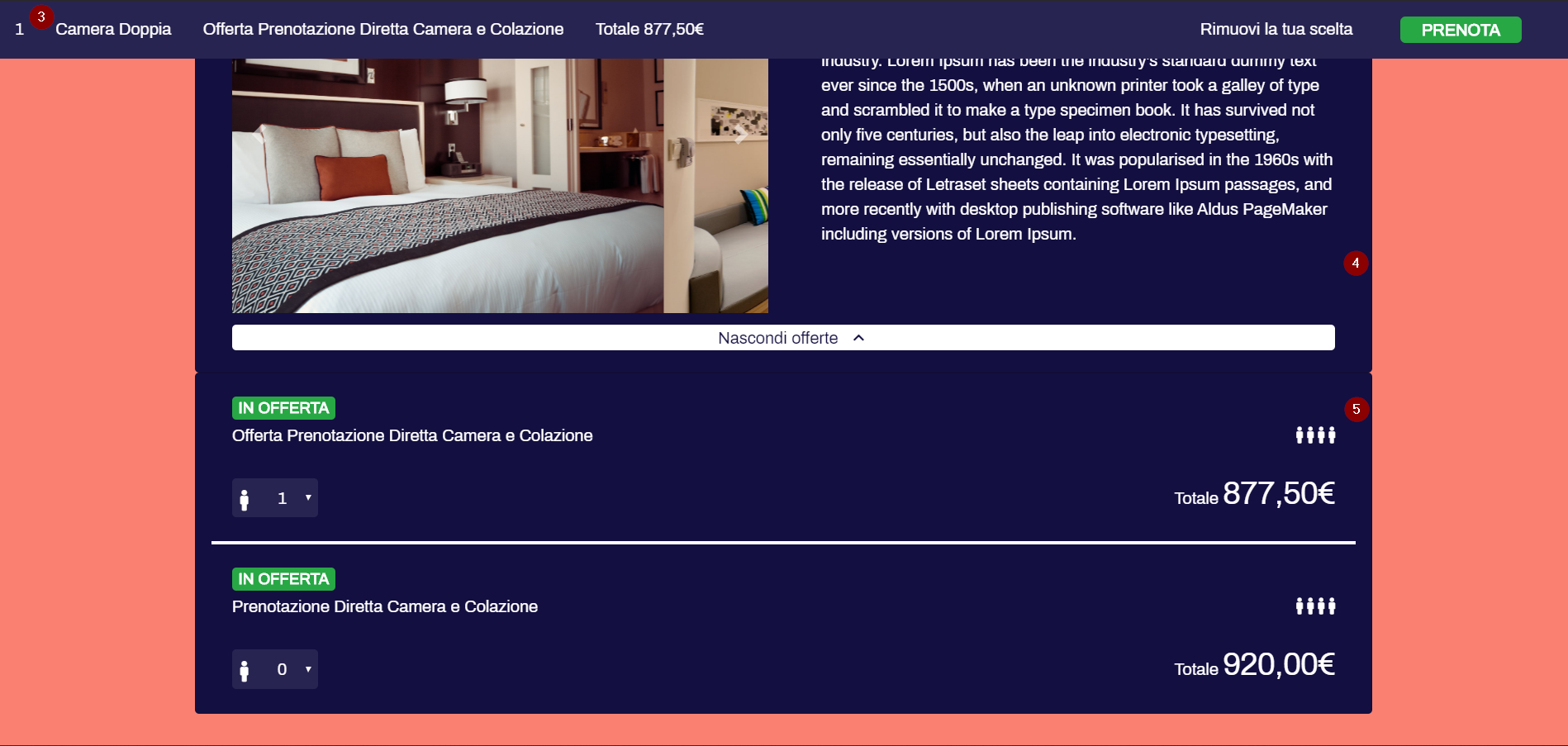


L’allineamento dei vari elementi è stato eseguito grazie alle regole contenute nella classe “box”, ho sfruttato inoltre il grid system di Bootstrap per impostare in che modo gli elementi si sono suddivisi lo spazio disponibile.



Il pulsante è regolato dalla classe “btn-roomInfo”, questa gestisce gli spazi e allinea il testo e l’icona, “chevron-down” di Font Awesome.

Il titolo della camera, le parole chiave, la descrizione, il prezzo e le foto da mostrare nel carosello vengono fornite dal back-end.



### Carrello camere scelte (3)

### Card camera selezionata (4)

### Offerte camera (5)

## Terza sezione



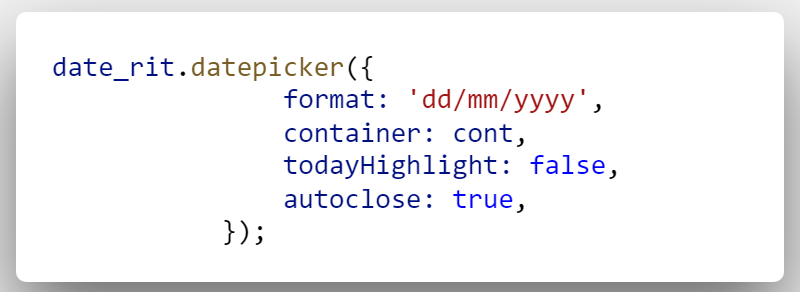
# Funzioni

## Datepicker

Tutti i campi data presenti nell’applicazione sono stati implementati con l’aiuto della libreria Bootstrap-Datepicker, ho inoltre aggiunto alcune modifiche per adattare maggiormente i datepicker alle necessità del progetto.

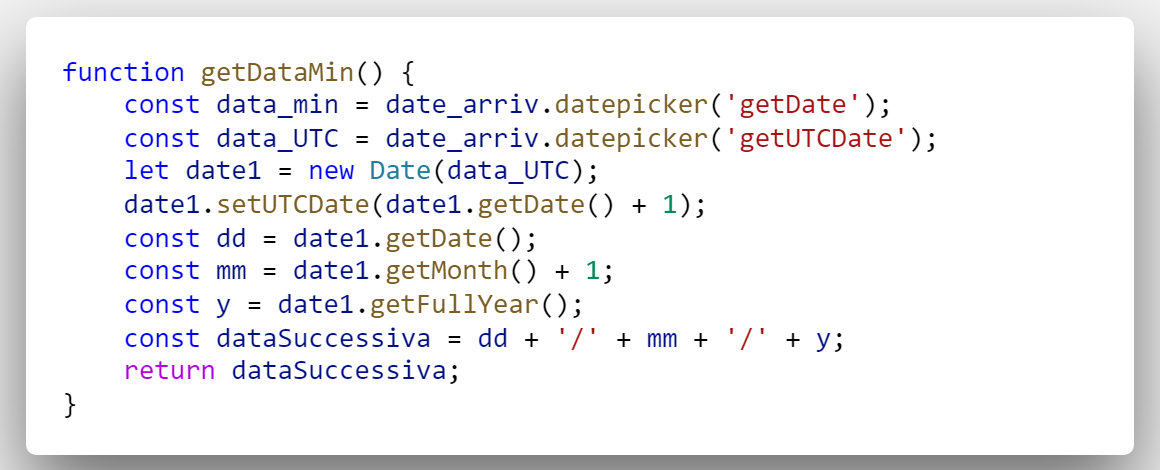
### Parametri base

Queste linee di codice permettono di modificare i parametri base del datepicker, **format** si riferisce ovviamente al formato in cui verrà visualizzata la data, **container** invece serve per selezionare in quale elemento verrà appeso il campo data che nel nostro caso lasceremo come da default, **todayHighlight** invece si riferisce alla funzione che evidenzia la data odierna, che nel caso della data di arrivo sarà “true” invece per la data di ritorno sarà “false”, in fine **autoclose** la funzione che chiuderà automaticamente il pop-up del datepicker una volta che l’utente avrà selezionato la data, che nel nostro è proprio quello che vogliamo.



### Valore minimo campo Data Ritorno

Per evitare che l’utente possa inserire una data di ritorno inferiore a quella di arrivo, è importante fare in modo che la data minima selezionabile sia superiore a quella inserita nel campo data d’arrivo.



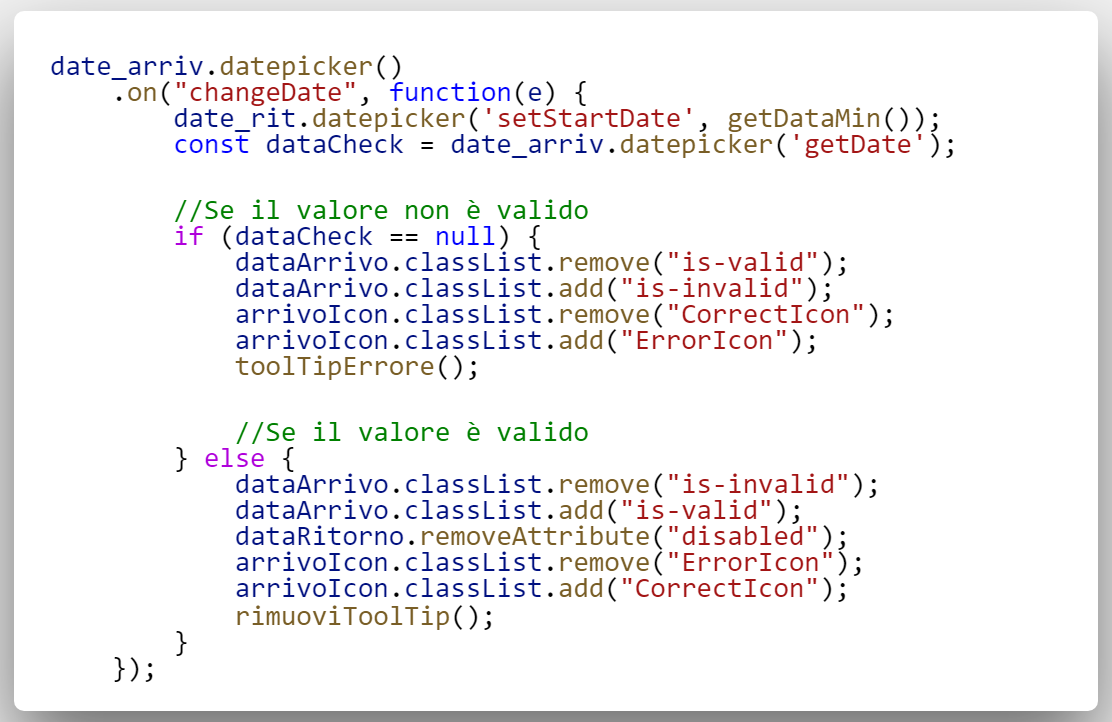
Ho quindi deciso di creare una funzione che si occupi di calcolare la data minima selezionabile, sulla base del valore selezionato nel campo data precedente, sulla base di quest’ultimo viene creata una variabile che conterrà la data in formato UTC, a cui viene aggiunto un uno, equivalente ad un giorno, il valore ottenuto da questa addizione si suddivide poi in tre variabili che vengono poi unite per creare il valore data nel formato normale.

Il valore ottenuto da questa funzione viene poi impostato come data d’inizio del datepicker per la data di ritorno, con questa linea di codice.

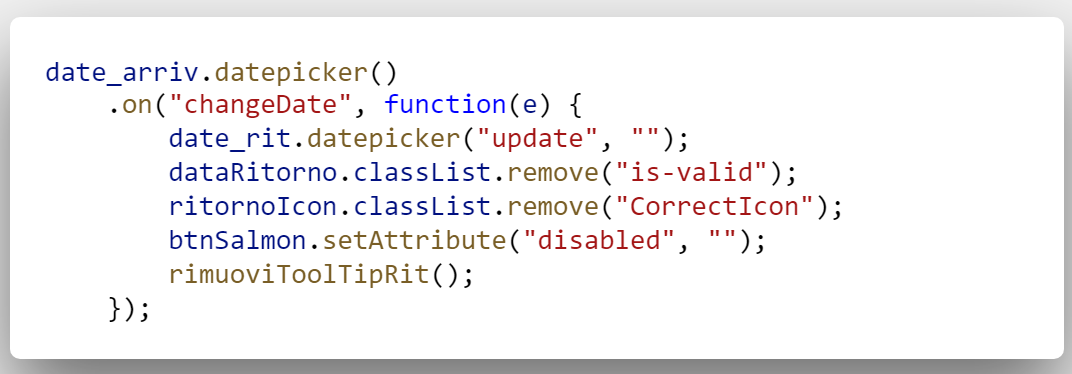


### Gestione errori

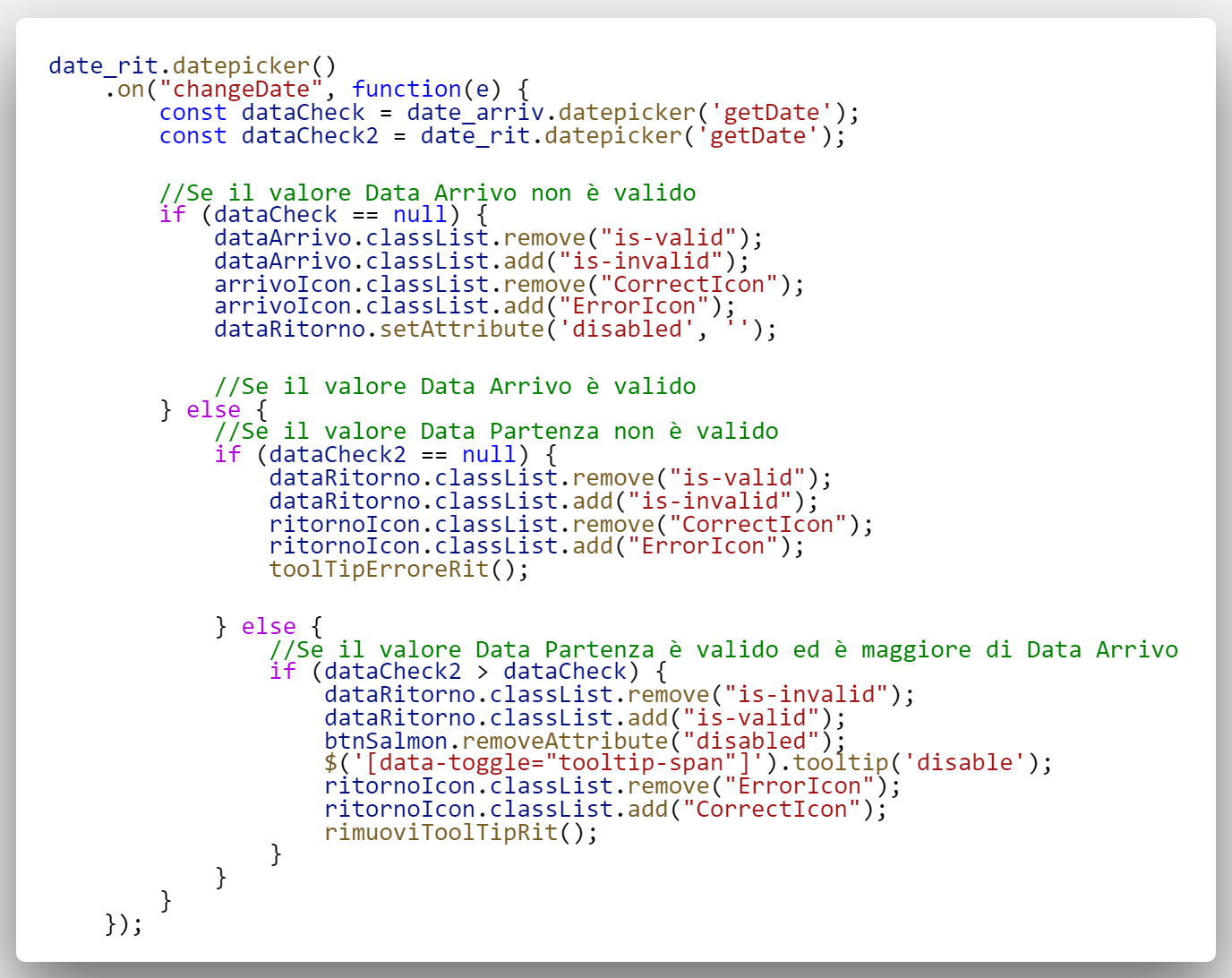
Per evitare che l’utente possa inserire dati che potrebbero causare problemi al sistema, è necessario creare delle funzioni che si occupino di gestire gli errori, per nostra fortuna Bootstrap-Datepicker si occupa già in parte di fare ciò. Infatti se l’utente inserisce un dato non valido, questo verrà sostituito dalla odierna.



Siccome il campo data partenza è dipende dal valore inserito nel campo precedente, è fondamentale bloccare l’utente dall’utilizzo del campo data partenza nel caso in cui non abbia ancora inserito la data di arrivo. Per questo ho aggiunto al codice html corrispondente al campo data partenza l’attributo **disabled**, che impedisce all’utente di utilizzare quell’elemento, ho inoltre creato una funzione che controlla se il campo data arrivo sia vuoto o meno, nel primo caso la funzione aggiunge due classi CSS che rendono il campo rosso e un tooltip con il messaggio “La data inserita non è corretta.”, in modo da evidenziare il fatto che il campo deve essere ancora riempito, nel secondo caso invece viene rimosso l’attributo che disabilità il campo successivo e si aggiungono due classi che colorano il campo di verde e cambiano il tooltip per evidenziare che il valore inserito è corretto.



Un’altra possibilità della quale bisogna tenere conto è il cambio della data del campo data arrivo dopo che è stata inserita una data nel campo data partenza, siccome questa situazione potrebbe presentare la possibilità che la nuova data inserita nel primo campo sia superiore a quella inserita nel secondo campo. Per evitare questa eventualità ho creato una funzione che rimuove la data inserita nel campo data partenza, nel caso in cui il valore del primo campo venisse modificato.

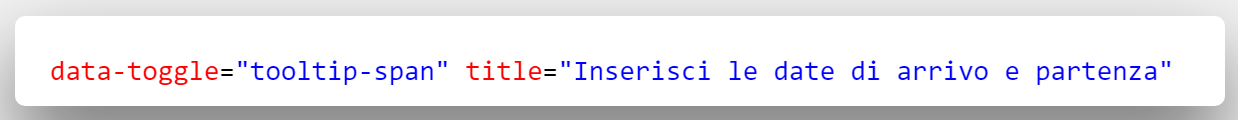


Ho poi sviluppato una funzione che si occupa di verificare la correttezza del valore inserito nel campo data partenza, prima controllando se è il campo non è vuoto e poi controllando se la data inserita sia effettivamente superiore a quella indicata nel campo precedente.

Infine ho aggiunto l’attributo **disabled** anche al pulsante che permette di inviare i dati inseriti all’utente tramite il metodo get, per evitare che l’utente possa cliccarlo senza aver prima inserito entrambe le date, ho quindi deciso di sfruttare la stessa funzione descritta poco fa, che si occupa dunque anche di rendere utilizzabile il bottone all’utente, nel caso in cui tutti i controlli abbiano un responso positivo.

## Tooltip

Tutti i tooltip presenti all’interno del nostro Web Booking Engine sono stati implementati attraverso Bootstrap.



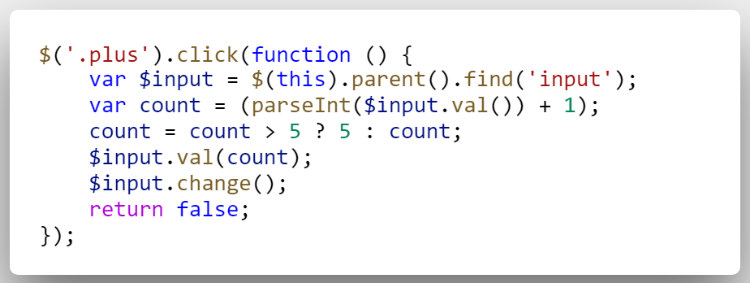
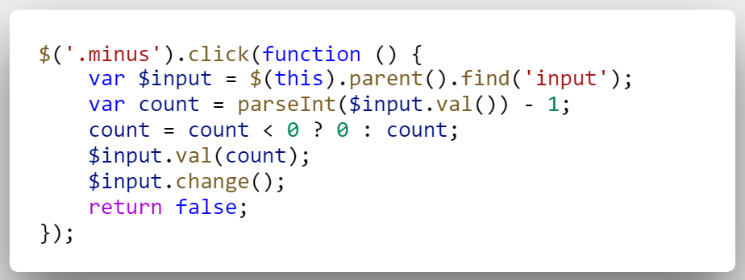
Per inserire un tooltip in un elemento, il primo passo è quello di indicare il **data-toggle**, che verrà utilizzato in seguito per inizializzare il tooltip, bisogna poi indicare il **title**, ossia il testo che verrà inserito all’interno del nostro tooltip.



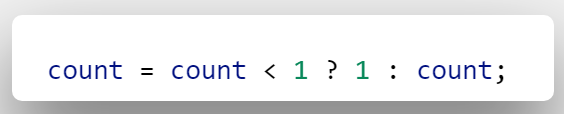
Una volta specificati i due attributi, non ci resterà che inizializzare il tooltip tramite jQuery, per fare ciò ci basterà richiamare il **data-toggle** che abbiamo inserito in precedenza, come indicato dallo screenshot qua sopra.

## Contatori

I contatori che si trovano nella prima parte del sito e che permettono di selezionare il numero di adulti, bambini e camere sono stati implementati utilizzando un codice proveniente da un thread sul sito Stack Overflow.



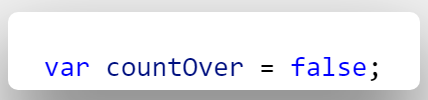
Questo è il codice di base, che ogni volta in cui l’utente clicca sull’elemento con la classe “minus” estrare il valore del padre e se il valore è superiore a 0 ne sottrae un’unità, altrimenti lo lascia invariato. Mentre nel caso in cui l’altro evento venisse chiamato, viene controllato se il valore è inferiore a 5 o meno, se questo controllo ritorna un responso positivo, viene sommata al valore un’unità.



Per alcuni campi ho dovuto modificare leggermente il codice, come nel caso del numero di adulti e quello delle camere, siccome non possono contenere un valore inferiore ad 1, ho modificato il controllo per l’evento che si occupa di gestire le sottrazioni con la linea di codice rappresentata qua sopra.

### Età bambini

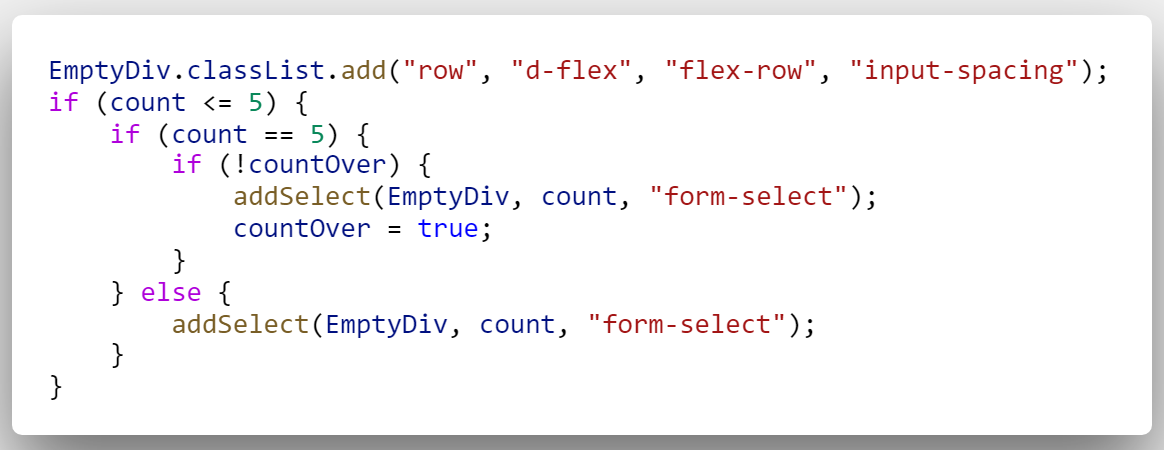
All’interno Web Booking Engine vi sono due casi particolari che richiedono l’implementazione di ulteriori modifiche, ovvero i campi relativi al numero dei bambini, dato che questi gestiscono anche la creazione degli elementi dropdown per l’età di ogni bambino selezionato.



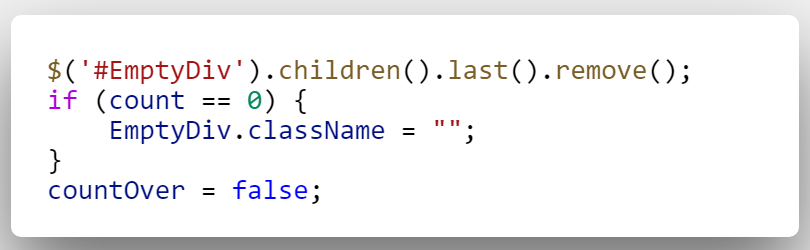
Innanzitutto ho creato un flag chiamato “coutOver” che servirà per capire quando si è raggiunto il numero massimo di dropdown necessari, per evitare che il numero totale di quest’ultimi ecceda il limite stabilito.



Ho fatto questa classe che si occuperà di creare i vari dropdown, con il nome, l’id e un tooltip apposito e iniettarli direttamente nell’HTML.



Questo è il codice che è stato aggiunto per gestire l’incremento del valore. Vengono prima di tutto aggiunte le classi necessarie al div che conterrà i dropdown, viene eseguito un controllo sul valore del campo, se è uguale al limite, che in questo caso è 5, ed il flag creato in precedenza è impostato su “false” viene creato un dropdown sfruttato la funzione apposita ed il valore del flag diventa “true”. Mentre nel caso in cui il valore sia inferiore al limite, si aggiunge il dropdown ma il flag rimane invariato.



Infine la diminuzione del valore è gestita tramite questo codice, che rimuove per prima cosa l’ultimo dropdown che è stato aggiunto, se il valore equivale a 0 si occupa anche di eliminare le classi dal contenitore. Viene infine impostato il flag su “false” per permettere poi all’evento per l’addizione di capire che il limite non è ancora stato raggiunto.

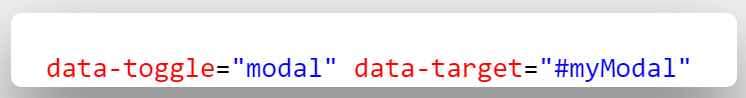
## Carosello

Anche i caroselli, che si occupano di mostrare all’utente le foto delle camere, sono stati inseriti con l’aiuto di Bootstrap[[3]](#footnote-3), ai quali non ho aggiunto alcuna modifica, siccome il layout di default, si adatta perfettamente alle nostre esigenze.

### Carosello modale

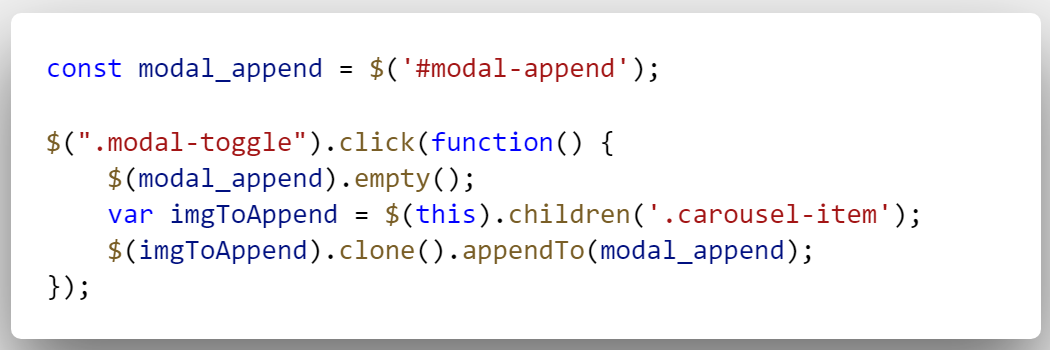
Questo speciale carosello ha richiesto alcune modifiche strutturali, al contrario di quelli “normali”.

La prima di queste modifiche è forse quella più evidente, ovvero il fatto che questo carosello è contenuto in una finestra modale, anche in questo caso implementato grazie a Bootstrap.



I due attributi html, rappresentati nello screenshot qua sopra, sono viene aggiunto a ciascun carosello “normale”, queste due proprietà chiamano la finestra modale con l’id identico a quanto inserito in **data-target**, quando l’utente clicca sull’elemento che contiene questi attributi.

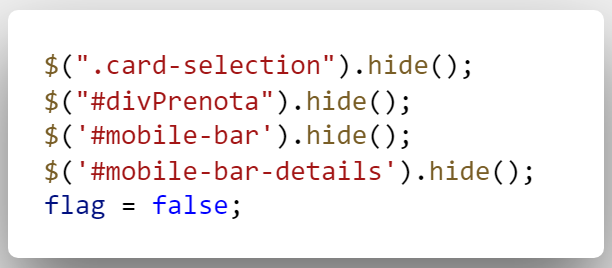
Ora però noi abbiamo più caroselli “normali” ma solo uno modale, quindi se vogliamo fare in modo che il nostro carosello “speciale” mostri ogni volta le immagini della camera giusta dobbiamo renderlo dinamico.



Ho quindi creato una funzione, che si attiva ogni volta in cui la finestra modale viene chiamata, svuota il contenuto del div contenente le immagini, trova il medesimo contenitore nel carosello che ha chiamato la funzione e ne coppia il contenuto all’interno del div che è stato in precedenza.  
Questo procedimento permette al carosello nella finestra modale di mostrare ogni volta le immagini della camera giusta, evita inoltre la possibilità che rimangano le foto della camera che l’utente ha cliccato in precedenza, siccome ogni volta che la funzione viene chiamata elimina tutte le fotografie che il carosello modale conteneva in precedenza.

## Offerte camera

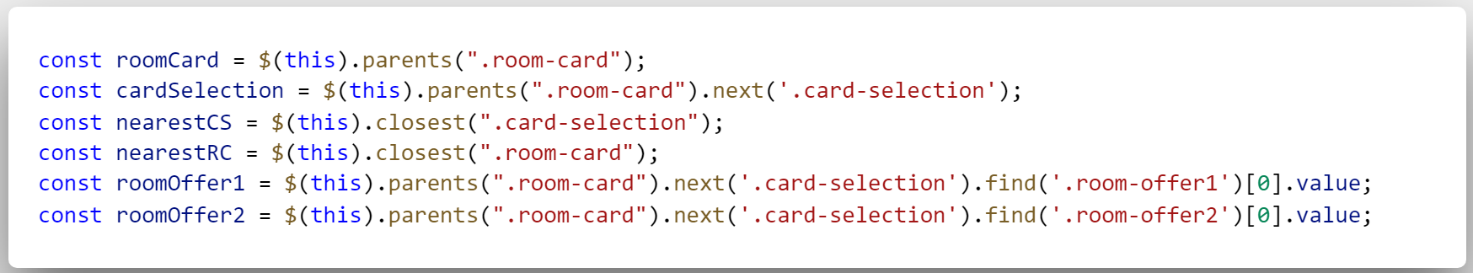
Il codice che gestisce i contenitori che ospitano i dettagli delle varie offerte delle stanze è stato creato interamente da me, l’unica libreria che è stata usata in questo caso è **jQuery**, che mi ha permesso creare delle funzionalità che probabilmente utilizzando unicamente JavaScript non si sarebbero potute fare.



Prima ancora di creare la funzione, dobbiamo assicurarci che tutti gli elementi verranno resi visibili in seguito, siano nascosti, usiamo quindi il metodo “hide” per fare ciò. Inizializziamo inoltre una variabile booleana che verrà utilizzata in seguito come flag.



L’intera funzione è contenuta all’interno di un evento, che viene chiamato quando l’utente clicca sul pulsante con la classe “btn-roomInfo”, ossia il pulsante che accoglie la scritta “Visualizza offerte”.



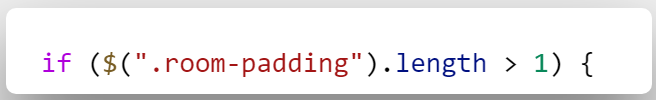
Inizializziamo poi, tramite jQuery le varie variabili a cui saranno collegati i molteplici elementi HTML, che andremo a modificare nella funzione.  
La variabile “roomCard” si riferisce a tutte le card contenenti le informazioni sulle camere presenti sul sito, “cardSelection” a tutte le card che contengono le offerte delle varie camere, “nearestCS” solo alla Room Card che contiene il pulsante che ha cliccato l’utente, “nearestRC” solo alla Selection Card più vicina al pulsante cliccato, “roomOffer1” al valore del numero di stanze che l’utente desidera per la prima offerta della camera selezionata e “roomOffer2” al valore della seconda offerta.



Viene poi fatto un controllato se ci sono già delle Card Offerte visibili, se la condizione è vera: viene aggiunta la classe “selection-hide” (per maggiori info vedi capitolo **6.3.1 Offerte camera**), si resetta la posizione dell’icona del pulsante (vedi **6.3.4 Icone “Chevron”**), il testo del pulsante viene resettato, si aggiunge infine la classe “room-card”, che gestisce lo spazio tra le Card Room quando le Card Offerte sono nascoste.



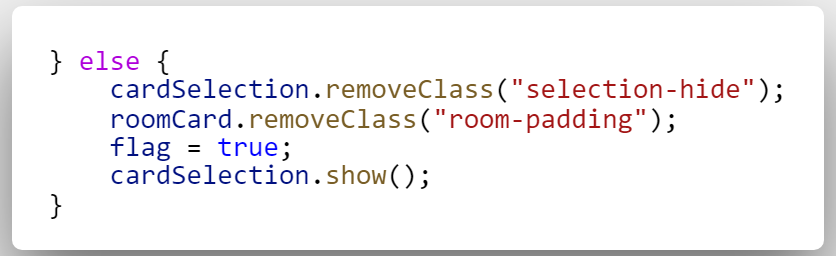
In seguito si esegue un controllo sul pulsante cliccato, se la posizione dell’icona non è stata alterata, quindi se la Card Offerte relativa a questo elemento è ancora nascosta: ruotiamo l’icona, cambiamo il testo, nascondiamo tutte le Card Offerte, ad esclusione di quella più vicina alla camera selezionata dall’utente e aggiungiamo a tutte le Room Card, tranne quella relativa a questo elemento, la classe “room-padding”.  
Se invece l’icona è già stata ruotata e di conseguenza la Card Offerte è già visibile: riportiamo il pulsante al suo stato originario.



Viene poi eseguito un ulteriore controllo, per ragioni di spazio ho deciso di suddividere in più parti, la condizione in questo caso sarà il numero di elementi con la classe “room-padding” che deve essere superiore a 1, che corrisponde ad un metodo più semplice di accertarsi che ci siano più camere senza la Card Offerte visibile.



In seguito viene verificato lo stato del flag. Nel caso in cui questo abbia un valore positivo, viene eseguito un ulteriore controllo sul pulsante per capire se la Card offerte sia ancora nascosta o meno, nel primo caso aggiungiamo la classe “selection-hide” per attivare l’animazione, nascondiamo poi l’elemento con un delay di 500 millisecondi, utilizzando il metodo “setTimeout” di JavaScript.  
Nell’altro caso invece vengono rimosse le classi “room-padding” e “selection-hide”, il flag viene attivato e la Card offerte viene resa visibile.



Se il flag non è attivo, vengono utilizzate nuovamente le linee di codice viste poco fa’ per rendere il contenitore con le offerte visibile.



In conclusione ci occupiamo della parte di codice che viene attivata quando il controllo sulla classe “room-padding”, visto in precedenza, da un risultato negativo.  
Viene aggiunta la classe “selection-hide” e il flag viene disattivato. A seguito di un delay di 600 millisecondi viene: nascosta la Card offerte, aggiunta la classe “room-padding”, fatto un controllo sul numero indicato dall’utente nei menu a tendina, nel caso in cui in almeno in uno dei due input sia stato selezionato un numero superiore a 0 viene aggiunta alla Room Card la classe “room-selected” per evidenziare che è stata selezionata un’offerta di questa camera, nel caso in cui il valore non sia superiore a 0, la classe “room-selected” viene invece rimossa. Infine viene attivata la funzione “checkRoomSelection”, spiegata nel capitolo **5.6 Carrello camere selezionate**.

### Chiusura automatica

Mi è stato richiesto da Francesco Luconi di creare una funzione che chiuda in automatico la Card Offerte quando l’utente seleziona un valore nuovo in uno dei due input a tendina relativi al numero di stanze per offerta.



La funzione è contenuta all’interno di un evento, questo viene chiamato quando il valore di uno dei campi a tendina presenti sul sito, di cui si parlava in precedenza, viene modificato.



Vengono aggiunte le classi “room-padding” e “selection-hide”, a seguito di un delay di 600 millisecondi: viene creato un evento che si occupa di resettare il testo e l’icona di tutti i pulsanti che attivano la Card Offerte, viene richiamata la funzione “checkRoomSelection”, nascosta la Card Offerte e il flag viene disattivato.

## Carrello camere selezionate

Al contrario delle altre funzioni presenti sul sito, il processo che gestisce il funzionamento del carrello è suddiviso in più parti, vista la sua complessità.

### Attivazione



L’attivazione dal carrello viene gestita da una funzione apposita, nominata “checkRoomSelection”, questa viene chiamata in altri processi per valutare se sia o meno il caso di rendere visibile il carrello.



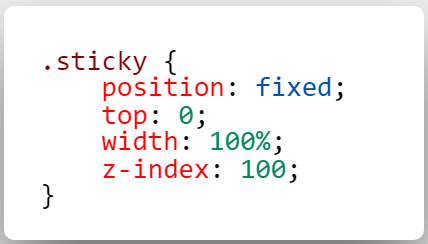
Il sistema controlla se vi sono elementi con classe “room-selected” presenti sul sito, nel caso in cui il responso sia positivo, il recipiente che contiene il carrello viene reso visibile, viene poi eseguita la gestione delle classi che si occupano delle animazioni, vedi capitolo **6.3.3 Carrello camere selezionate** per maggiori informazioni a riguardo.



Nel caso invece, in cui la verifica fatta in precedenza dia un risultato negativo, innanzitutto vengono gestite le classi per attivare l’animazione d’uscita, a seguito di un delay di 500 vengono nascosti tutti gli elementi che compongono il carrello.

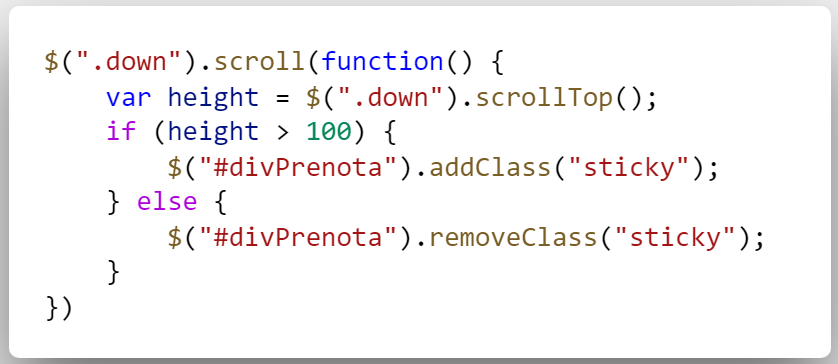
### Funzione sticky

Mi è stato richiesto da Francesco Luconi, di fare in modo che il carrello seguisse l’utente, quindi di rendere l’elemento sticky.



Ho creato una classe CSS che si occupa di fissare in cima alla pagina l’elemento, sfruttando gli attributi “position” e “top”, ho inoltre inserito un valore molto alto al parametro “z-index” per evitare che qualche altro elemento all’interno della pagina possa venir sovrapposto al carrello.

Il nostro carrello però è posizionato in basso al form che contiene i dati di ricerca, perciò non possiamo fissare subito l’elemento perché andrebbe a sovrapporsi con quello che lo precede, perciò è necessario fare in modo che la classe “sticky” venga aggiunta solo quando il form non è più presente nella schermata.



Per questo motivo ho creato un evento che si attiva quando l’utente scorre verticalmente la seconda sezione, rappresentata dalla classe “down”. All’interno di questo evento, viene inizializzata una variabile che contiene la posizione verticale corrente. Tramite questa viene eseguito un controllo, se la posizione attuale è superiore a 100 viene aggiunta la classe che rende il carrello sticky, nel caso contrario invece, la classe viene rimossa.

### Inserimento dati

# Animazioni

## Animazione d’entrata

### Prima sezione

Le animazioni d’entrata degli elementi presenti nella prima sezione sono state implementate grazie ad Animate.CSS.

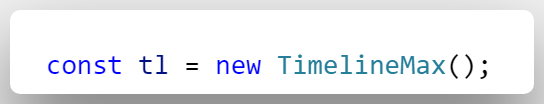


Basta aggiungere la classe “animated” nell’elemento che si vuole animare e indicare il nome dell’animazione che si vuole ottenere. L’elenco completo delle animazioni disponibili è accessibile attraverso la documentazione ufficiale di Animate.CSS[[4]](#footnote-4).

### Seconda sezione

Le animazioni d’entrata usate nella seconda sezione sfruttano invece la libreria JavaScript GSAP, sfruttando inoltre TweenMax e TimeLineMax, quest’ultimi plugin aggiuntivi di GSAP.

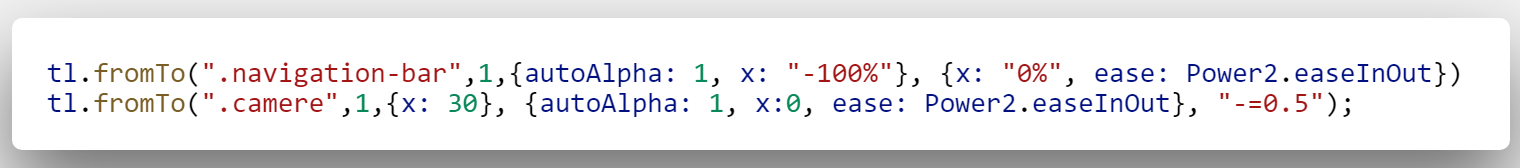
Ho creato una funzione chiamata “AnimationSlide” che conterrà tutte il codice necessario, permettendoci quindi accedere a tutte le animazioni chiamando semplicemente la funzione.



Innanzitutto dobbiamo creare la nostra timeline tramite TimeLineMax, questa ci darà la possibilità di gestire i tempi d’inizio delle varie animazioni.



Passiamo poi a settare i valori iniziali degli elementi che in questo caso vogliamo inizialmente invisibili, selezioniamo per prima cosa la durata dell’animazione, in questo caso inseriamo il valore minimo possibile ovvero 0.1, indichiamo poi l’attributo che vogliamo modificare, nel nostro caso “autoAlpha” che si occupa di gestire sia la visibilità dell’elemento che l’opacità, nel nostro caso dev’essere invisibile, perciò indichiamo 0 come valore.



Passiamo infine all’implementazione vera è propria dell’animazioni. Utilizzando il metodo “fromTo” di TimeLineMax possiamo indicare quali siano gli attributi che vogliamo animare. Indichiamo per prima cosa la durata dell’animazione, per passare poi agli attributi di partenza dell’animazione, ed in conclusione indicare gli attributi alla fine dell’animazione, con la possibilità di indicare un metodo di “easing” ovvero un metodo che rallenterà la conclusione dell’animazione, per evitare che appaia tagliata a metà.

## Transizioni

### Transizione da prima a seconda sezione

La transizione tra la prima sezione e la seconda, che si attiva quando l’utente clicca sul pulsante “Cerca camere”, viene gestita con GSAP.  
Eviterò di spiegare ancora l’intero processo, siccome ne abbiamo già discusso in precedenza.



In questo caso ho sfruttato il metodo “to” di TimeLineMax, che differentemente da quanto visto in precedenza con il metodo “fromTo”, anima gli elementi dai loro attributi attuali a quelli che andremo ad impostare.  
Le animazioni seguono l’ordine impostato sulla timeline, se invece vogliamo che due animazioni vengano eseguite in contemporanea, dovremo indicare un valore negativo nella proprietà delay, come possiamo vedere nell’ultima linea di codice dello screenshot qua sopra.

### Transizione da seconda a terza sezione

La transizione tra la seconda sezione e la terza invece è gestita tramite una funzione presa da un video tutorial su YouTube di Kevin Powell[[5]](#footnote-5).

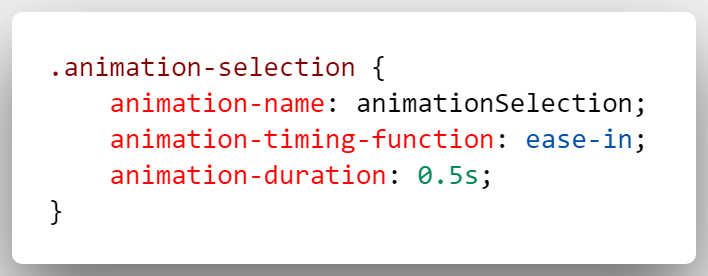


Innanzitutto viene creato un evento che si attiva ogni volta che l’elemento con la classe “scroll” viene cliccato. La prima riga contenuta nella funzione evita la possibilità che il link porti ad una nuova URL.  
Attraverso il metodo jQuery “animate” animiamo la transizione tramite la proprietà “scrolltop” che si occuperà di passare dalla posizione sull’asse verticale attuale a quella indicata nel campo “href” dell’elemento che ha chiamato l’evento, la durata dell’animazione viene dichiarata in millisecondi, in questo caso noi vogliamo che duri 1 secondo, perciò è stato inserito il valore 1000.

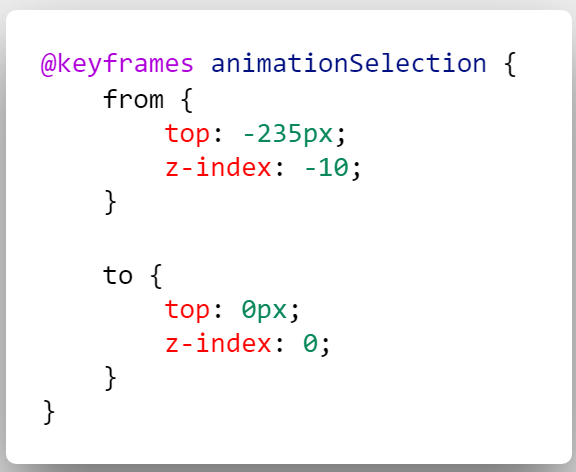
## Animazioni Elementi

### Offerte camera

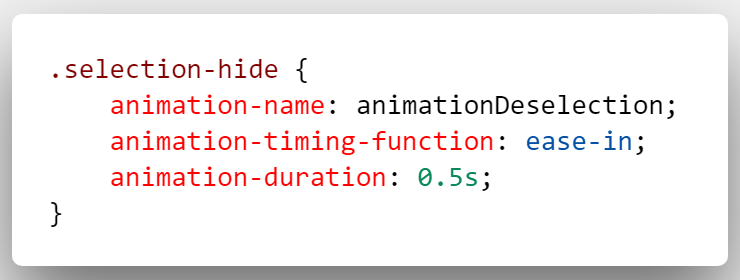
Le animazioni di entrate e uscita della card che contiene le offerte della camera sono gestite tramite CSS.



La classe “animation-selection” gestisce i parametri base dell’animazione d’entrata, come il keyframe che verrà utilizzato, lo stile di easing utilizzato e la durata dell’animazione.

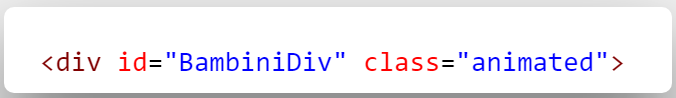


Attraverso il keyframe “animationSection” selezioniamo gli attributi dell’elemento che vogliamo animare, la loro posizione di partenza “from” e quella finale “to”.  
In questo caso vogliamo animare gli attributi top e z-index, siccome l’effetto desiderato è quello di far sembrare che le offerte siano nascoste dietro ai dettagli della camera e poi farle scendere, alziamo il nostro elemento assegnando a “top” un valore negativo e lo andiamo a nascondere attraverso un “z-index” negativo.  
Le posizioni finali saranno invece quelle nomarli, perciò assegneremo uno 0 ad entrambi gli attributi.



Nel keyframe “animationDeselection”, che viene gestito dalla classe “selection-hide”, vengono invece inseriti i dettagli dei parametri che verranno animati per nascondere le offerte, al contrario di quanto visto nel keyframe precedente, qua si partirà dai valori normali per poi passare a quelli che occulteranno il nostro elemento dietro ai dettagli della camera.

### Contenitore Età bambini

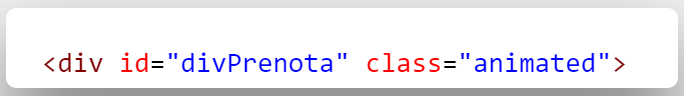


Per il contenitore delle età dei bambini, che possiamo trovare nella seconda sezione del sito, è stata utilizzata **Animate.ccs** per gestire le animazioni.



Per le animazioni iniziale viene utilizzata la classe “fadeIn”, invece l’animazione d’uscita utilizza “fadeOut”. Quando il contenitore è visibile vengono chiamate le linee di codice nello screen a sinistra, che rimuovono la classe dell’animazione d’uscita e aggiungono quella d’uscita, mentre quando dev’essere nascosto accade il contrario.

### Carrello camere selezionate



Anche il sistema a mo’ di carello sfrutta **Animate.ccs** per le animazioni.



L’entrata viene coordinata dalla classe “fadeInDown”, mentre l’uscita da “fadeOutUp”. Le linee di codice a sinistra vengono chiamate prima che l’elemento viene reso visibile, mentre quelle a destra vengono chiamate appena un po’ prima che l’elemento sia nascosto.



Il contenitore che viene creato quando vengono selezionate più camere differenti, utilizza invece utilizza “fadeInDown” e “fadeOut”, rispettivamente per l’animazione iniziale e quella finale. Come visto in precedenza a sinistra il codice per l’entrata e a destra il codice per l’uscita.

### Icone “Chevron”

# Differenze tra Desktop e Mobile

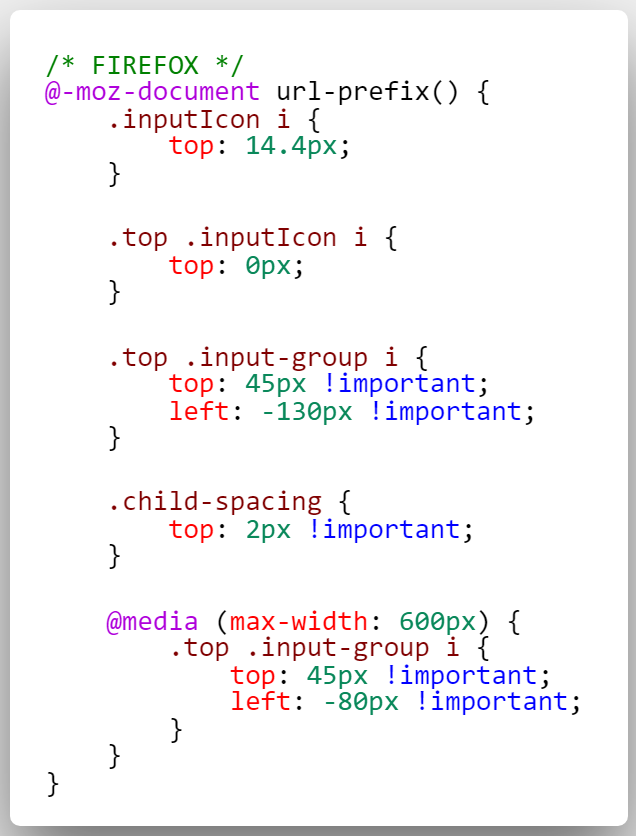
# Compatibilità Cross-Browser

## Risultati immagini per chromeChrome

Essendo Chrome il Browser di riferimento per lo sviluppo di questo applicativo web, non è stato necessario creare funzioni o regole esclusive per questo programma.

## Firefox

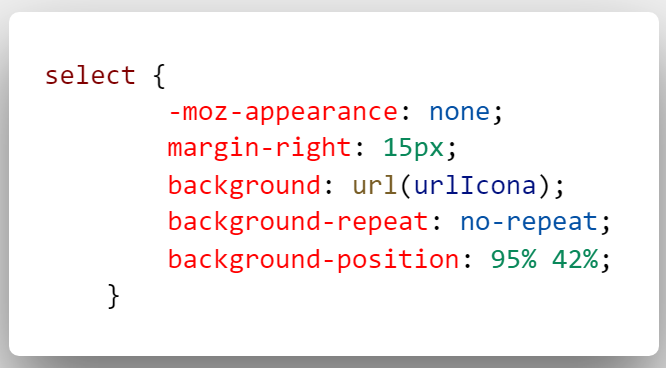
 Le prove fatte su Firefox non hanno evidenziato alcun problema per quanto riguarda le funzioni implementate tramite JavaScript e jQuery, che funzionano esattamente come su Chrome, un problema riscontrato è quello relativo al posizionamento delle icone dei vari campi input presenti nel sito, che su questo Browser venivano spostate verso l’alto.



Ho utilizzato una media query CSS che mi permettesse di targettizzare unicamente Firefox, all’interno ho inserito alcune regole che permettessero di eliminare le differenze di posizionamento, agendo sulle proprietà “top” e “left” degli elementi che presentavano questo problema.



Un’altra differenza rispetto riscontrata, rispetto al browser di riferimento, è lo styling degli elementi “select”, ovvero i dropdown (screenshot qua sopra). Firefox forza sugli elementi delle regole, che a mio modo di vedere non si adattano bene al sito.



Perciò ho inserito delle regole, trovate su un post in Stack Overflow[[6]](#footnote-6), all’interno della media query che fanno riferimento unicamente agli elementi “select” all’interno della pagina.  
Per prima cosa rimuoviamo lo styling di Firefox attraverso la proprietà “-moz-appearance”, inseriamo un margine per staccare brevemente l’icona dal valore del campo con “margin-right”, inseriamo inseguito il “background” ovvero l’icona che andremo ad utilizzare, infine ci assicuriamo che il posizionamento venga fatto correttamente sfruttando gli attributi “background-repeat” e “background-position”.

## Edge

## Internet Explorer

## Safari

1. Sito ufficiale di Font Awesome <https://fontawesome.com/> [↑](#footnote-ref-1)
2. Pagina nella documentazione di Bootrap dedicata al grid system <https://getbootstrap.com/docs/4.3/layout/grid/> [↑](#footnote-ref-2)
3. Pagina nella documentazione dedicata ai caroselli <https://getbootstrap.com/docs/4.0/components/carousel/> [↑](#footnote-ref-3)
4. Reperibile tramite questo sito <https://daneden.github.io/animate.css/> [↑](#footnote-ref-4)
5. “How to add Smooth Scrolling to your one page website with jQuery” <https://www.youtube.com/watch?v=x0YnVwAuNQI&t=653s> [↑](#footnote-ref-5)
6. Why does Firefox set a background color on the select arrow button when I set a custom border or background color? <https://stackoverflow.com/a/29513693> [↑](#footnote-ref-6)