10/01/2022

Software Cybersecurity - GRUPPO 1

UNIVPM, LM INGEGNERIA INFORMATICA E DELL’AUTOMAZIONE, AA 2020-2022



La biglietteria online sicura basata su blockchain | Front end del sito web |

Sommario

[Il Front end 2](#_Toc92148049)

[Struttura del codice 4](#_Toc92148050)

[HTML elements 5](#_Toc92148051)

[widget.js 5](#_Toc92148052)

[HTMLpage.js 5](#_Toc92148053)

[Elementi grafici 6](#_Toc92148054)

[Form personalizzati 9](#_Toc92148055)

[Scripts 12](#_Toc92148056)

[Web pages 14](#_Toc92148057)

[areaRiservata.js 14](#_Toc92148058)

# Il Front end

Durante la fase di progettazione sicura abbiamo individuato le diverse tipologie di attori che potrebbero interagire con la nostra applicazione web. Tra questi vi sono:

* il **cliente**, cioè l’attore che ha come obiettivo quello di acquistare biglietti per partecipare a uno degli eventi disponibili sul nostro sito web;
* l’**event manager**, ovvero l’attore che si occupa di organizzare eventi di intrattenimento, scegliendo e prenotando il luogo in cui si svolgerà l’evento, fissando una data, contattando gli artisti;
* lo **staff della biglietteria**, cioè gli attori che si occupano di mettere in vendita i biglietti per un dato evento;
* l’**annullatore dei biglietti**, ovvero l’attore che si occupa di scansione i codici QR dei biglietti prima che un cliente possa accedere all’evento, verificandone la validità;
* Il **sistema di pagamento**, cioè l’attore che si occupa di gestire le transazioni per l’acquisto dei biglietti e l’invio delle corrispondenti ricevute.

Ogni diversa tipologia di attore dovrebbe accedere solo a specifiche risorse ed eseguire solamente certe operazioni.

La necessità di garantire i requisiti di autenticazione e autorizzazione ha fatto emergere il bisogno di inserire un ulteriore attore: l’**utente ospite**, cioè quel particolare tipo di attore che non si è ancora autenticato con l’applicazione web. Le uniche operazioni che può svolgere sono:

* la **consultazione del catalogo** degli eventi;
* l’**autenticazione alla web app**, tramite una procedura di login;
* l’**iscrizione alla web app**, nel caso sia la prima volta che l’utente visita il sito web.

Attraverso l’impiego dei diagrammi i\* abbiamo deciso di scomporre la nostra applicazione in più moduli software. Ognuno di questi si occupa di interagire con una diversa tipologia di attore umano. Abbiamo individuato quattro diversi sottomoduli:

* **Modulo amministrazione evento**, che si occupa dell’interazione con gli organizzatori degli eventi;
* **Modulo gestione biglietti**, per l’interazione con lo staff della biglietteria;
* **Modulo vendite biglietti**, che si occupa dell’interazione con il cliente;
* **Modulo annullatore biglietti**, per l’interazione con l’invalidatore dei biglietti;
* **Modulo utente ospite**, che si occupa dell’interazione con gli utenti non autenticati.

Il sistema centrale (server) delega, quindi, la comunicazione con gli attori esterni a questi sottomoduli, i quali costituiscono il **front end** dell’applicazione.

Ciascun attore interagisce con l’applicazione navigando all’interno di una serie di pagine web. Ogni pagina, ad eccezione della pagina di login e di quella di iscrizione al sito, include una **barra di navigazione**. La barra di navigazione è disponibile per tutte le diverse tipologie di attori, tranne che per l’invalidatore dei biglietti, poiché l’unica operazione a lui permessa è la scansione dei codici QR associati ai biglietti.

Nella **barra di navigazione**, oltre al logo dell’applicazione web, sono presenti un insieme di pulsanti che permettono di consultare le diverse sezioni del catalogo degli eventi. Ancora più a destra, con un click sul pulsante Account, si apre un menu a tendina che mostra una serie di azioni che l’utente può eseguire.

Tutti gli utenti che si sono autenticati hanno disponibili tre operazioni, ovvero:

* la **consultazione del proprio profilo utente**;
* la **modifica della password** del proprio account, nel caso non fosse più sicura;
* il **logout** dall’area riservata.

In aggiunta a queste, a seconda dei privilegi che possiedono, possono effettuare anche altre operazioni.

# Struttura del codice

Il codice relativo al front end è contenuto nel percorso **ticketTwo/front end** nella directory principale del progetto. La directory è organizzata in sottocartelle secondo la seguente struttura:

* Sottocartella **CSS**, contiene tutti i file di stile CSS allegati alle pagine web;
* Sottocartella **HTML elements**, contiene una serie di elementi grafici HTML utilizzati all’interno delle pagine web del sito (sono codificati in linguaggio javascript lato server);
* Sottocartella **images**, contiene tutte le immagini utilizzate all’interno delle pagine web del sito (loghi, sfondi, …);
* Sottocartella **scripts**, contiene tutto il codice che viene eseguito lato client dal browser dell’utente (script allegati alle pagine html);
* Sottocartella **web pages**, contiene i sottomoduli software che si interfacciano con i client che visitano il sito web (sono codificati in linguaggio javascript lato server).

# HTML elements

## widget.js

Il file **widget.js** è il cuore di tutto il front end. In esso viene dichiarata la classe **Widget**,che permette di costruire in maniera dinamica un elemento HTML da aggiungere ad una pagina web.

Il costruttore della classe richiede come parametri il tipo di tag da generare e il testo da includere nel tag (parametro opzionale). Il compito del costruttore è quello di allocare in memoria lo spazio per quattro strutture dati:

* **attributi**, un oggetto le cui chiavi sono i nomi degli attributi del tag HTML e i valori sono stringhe che rappresentano il valore dell’attributo associato;
* **children**, un array in cui verranno inseriti i figli del tag che si vuole generare;
* **scripts**, un set in cui sono inseriti i nomi dei moduli javascript necessari al funzionamento dell’elemento HTML (sono i moduli contenuti nella cartella **scripts**);
* **stile**, un oggetto le cui chiavi sono i nomi delle proprietà CSS da associare al tag HTML e i valori sono stringhe che rappresentano il valore della proprietà associata.

La classe fornisce inoltre una serie di metodi, tutti pubblici, che possono essere utilizzati per personalizzare il tag:

* **addChild()**, permette di aggiungere un figlio al tag HTML;
* **addScript()**, permette di importare un modulo javascript nella pagina web in cui viene inserito il tag;
* **getScripts()**, restituisce un set contenente tutti i moduli javascript utilizzati dall’elemento HTML e dai figli;
* **get()**, genera una stringa contenente il tag HTML richiesto a partire dai parametri dell’oggetto;
* **setAttribute()**, permette di impostare il valore di uno degli attributi del tag;
* **setProperty()**, permette di impostare il valore di una delle proprietà CSS da associare al tag.

La funzione **get()** è la funzione più importante della classe. Inizializza una stringa a cui aggiunge come prefisso il nome del tag, racchiuso tra parentesi angolari, e come suffisso il nome del tag, preceduto da slash tra parentesi angolari. All’interno del prefisso vengono aggiunti tutti gli attributi del tag, seguendo la sintassi del linguaggio HTML. Tra questi vi è anche l’attributo **style**, che va a contenere come valore una stringa con tutte le proprietà CSS del tag.

Nella parte centrale della stringa vengono aggiunti tutti i figli del tag ed eventualmente il testo del tag (opzionale). I figli sono aggiunti invocando la funzione **get()** in maniera ricorsiva per ogni figlio.

## HTMLpage.js

All’interno di **HTMLpage.js** è definita la classe **HTMLpage**, che permette di generare una pagina web in maniera dinamica, prima che questa venga inviata al client.

Nel costruttore della classe viene istanziato un oggetto della classe **Widget**. Questo oggetto rappresenta la radice dell’albero DOM associato alla pagina HTML (tag HTML).

A questo tag, utilizzando la funzione **addChild(),** vengono aggiunti due figli:

* Un tag di tipo **head**, che rappresenta l’intestazione della pagina web;
* Un tag di tipo **body**, che rappresenta il corpo della pagina web.

All’interno dell’intestazione vengono impostati alcuni parametri della pagina web, come il titolo, la descrizione, la codifica di caratteri da impiegare (in questo caso UTF-8) e i file di stili CSS da utilizzare.

La classe **HTMLpage** fornisce una serie di metodi, tutti pubblici, che possono essere utilizzati per aggiungere elementi alla pagina web e inviarla come risposta ad una richiesta del client:

* **addChild()**, permette di aggiungere elementi HTML al corpo della pagina web (oggetti della classe Widget);
* **send()**, invia la pagina web al client come risposta ad una richiesta HTTP di tipo GET.

La funzione **send()** ottiene una lista di tutti i moduli necessari al funzionamento degli elementi in essa contenuti, invocando la funzione **getScripts()** per ogni figlio del body in maniera ricorsiva.

Per ogni modulo individuato viene aggiunto un tag di tipo **script** nell’intestazione, in cui si specifica di importare le funzioni definite in esso.

La funzione **send()** invoca poi il metodo **get()** sulla radice dell’albero DOM, che in maniera ricorsiva va a generare il documento HTML, aggiungendo i vari figli dei tag fino alle foglie.

Il documento HTML viene poi allegato alla risposta da inviare al client.

## Elementi grafici

Gli elementi grafici di base utilizzati per riempire il corpo delle pagine web del sito sono tutti contenuti nella cartella **widgets**. Ogni elemento grafico è associato ad una classe che eredita dalla classe **Widget**.

* All’interno del file **bar.js** viene definita la classe **Bar**, che si occupa di generare la barra di navigazione contenuta nella maggior parte delle pagine disponibili sul sito web. Quando viene invocato il costruttore viene generato un tag di tipo “div”. A questo vengono aggiungi come figli tutti gli elementi che saranno mostrati nella barra di navigazione:
  + il logo del sito web;
  + i pulsanti che permettono di navigare all’interno delle sezioni del catalogo degli eventi;
  + il pulsante Account per aprire il menu a tendina dell’utente;
  + il menu a tendina stesso (che non viene visualizzato fino a quando non si fa click sull’opportuno pulsante).



Figura 1 Barra di navigazione del sito web

* All’interno del file **card.js** è definita la classe **Card**, la quale costruisce un elemento HTML utilizzato per visualizzare i dati relativi ad un evento o ad un biglietto. L’elemento è organizzato in tre colonne: nella colonna di sinistra viene visualizzata l’immagine dell’evento, in quella centrale il titolo e i dati dell’evento, in quella di destra è presente un contenitore che può essere personalizzato inserendo altri elementi HTML, come ad esempio dei pulsanti.



Figura 2 Elemento HTML che visualizza i dati dell’evento

* All’interno del file **form.js** è definita la classe **Form**, che crea un form HTML che permette all’utente di inserire dati sulla pagina web ed inviarli al server. I dati possono essere di qualsiasi tipo: dati di iscrizione, dati di autenticazione, o dati relativi ad un evento.



Figura 3 Form HTML per l’inserimento dei dati

* All’interno del file **info.js** è definita la classe **Info**. Questa permette di visualizzare un messaggio formattato all’interno di una pagina web (tag di tipo “H5”).
* All’interno del file **logo.js** è definita la classe **Logo**, che permette di visualizzare il logo del sito web all’interno di una pagina HTML (viene usato nella pagina di login).

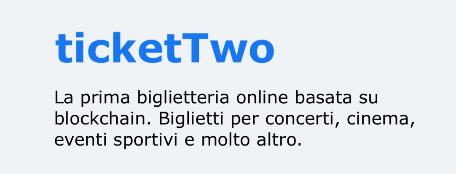


Figura 4 Classe Logo nella pagina di login

* All’interno del file **QRscanner.js** è definita la classe **QRscanner**. La classe implementa uno scanner di codici QR che permette di scansionare i codici dei biglietti al momento dell’ingresso all’evento.



Figura 5 Scanner di codici QR utilizzato dall’invalidatore

* All’interno del file **scrollView.js** è definita la classe **ScrollView**. Questa non fa altro che costruire una lista di oggetti di tipo **Card**, mostrati all’interno di una pagina web, tramite una scrollview. Permette di scorrere il catalogo di eventi che si trova nella home page del sito e la lista dei biglietti acquistati da un cliente.
* All’interno del file **table.js** è definita la classe **Table**, che permette di visualizzare in una tabella i dati contenuti nel database della web app.



Figura 6 Tabella generica per la visualizzazione dei dati del database

## Form personalizzati

All’interno del sito web sono presenti svariati form che permettono di aggiungere o modificare record del database. Questi form sono implementati come classi che ereditano dalla classe più generica **Form** e sono contenuti all’interno della cartella **forms**.

* All’interno del file **formMFA.js** è definita la classe **FormMFA**. Qui è definito un particolare tipo di form, che viene mostrato all’utente quando deve inserire un codice OTP per completare la procedura di autenticazione a due fattori.



Figura 7 Form per l’inserimento dell’OTP

* All’interno del file **gestioneVendite.js** è definita la classe **FormGestioneVendite**. Nella classe è definito un particolare tipo di form, che viene visualizzato quando un membro dello staff della biglietteria richiede di aprire o di chiudere le vendite dei biglietti per un certo evento. All’interno del form è presente una casella di testo in cui l’utente deve inserire la password del proprio wallet in modo da autorizzare l’operazione.



Figura 8 Form per l’inserimento della password del wallet

* All’interno del file **formModificaPassword** è definita la classe **FormModificaPassword**, nella quale è definito un particolare tipo di form, che viene mostrato all’utente quando richiede la modifica della password associata al suo account.



Figura 9 Form per la modifica della password

* All’interno del file **profiloUtente.js** è definita la classe **ProfiloUtente**. Nella classe è definito un particolare tipo di form, che permette ad un utente di visualizzare i dati associati al proprio profilo.



Figura 10 Visualizzazione dei dati del proprio account

* All’interno del file **modificaProfiloUtente.js** è definita la classe **ModificaProfiloUtente**, dove viene definito un particolare tipo di form, che permette ad un utente di modificare alcuni dei dati associati al proprio profilo. Non è possibile modificare né la mail né i privilegi associati all’utente.



Figura 11 Form per la modifica dei dati del proprio account

# Scripts

* All’interno del file **bar.js** sono definite le due funzioni che permettono di aprire e chiudere il menu a tendina presente nella navigation bar dell’applicazione web:
  + La funzione **openMenu(),** quando viene invocata, apre il menu a tendina. La funzione non fa altro che impostare a **block** il valore della proprietà CSS **display** del tag associato al menu a tendina;
  + La funzione **closeMenu(),** quando invocata, chiude il menu a tendina. La funzione non fa altro che impostare a **none** il valore della proprietà CSS **display** del tag associato al menu a tendina.
* All’interno del file **form.js** è definita una serie di funzioni che implementano la logica dei form dell’applicazione web:
  + La funzione **showInfo()**, quando viene invocata, mostra una messaggio di errore all’interno del form ad essa associato. Il testo del messaggio va specificato come parametro della funzione;
  + La funzione **sendDataToServer()** raccoglie il valore di tutti i campi contenuti nel form e li inserisce all’interno di un oggetto **JSON**. L’oggetto contenente i dati del form viene inviato al server tramite una richiesta HTTP. Se il server risponde che la richiesta è stata servita con successo, il client viene reindirizzato ad una nuova pagina web, altrimenti nel form viene visualizzato un messaggio che spiega il motivo per cui la richiesta è fallita. L’**url dell’api** a cui sono inviati i dati del form, il **tipo di richiesta HTTP** e l’**url di reindirizzamento** vengono specificati come parametri della funzione. Se la risposta del server contiene dei token di autenticazione, la funzione li memorizza tra i **cookies** del client;
  + La funzione **confermaPassword()** verifica se la stringa contenuta nel campo **password** del form coincide con quella contenuta nel campo **conferma password**. In caso negativo visualizza un messaggio di errore e impedisce al client di inviare il contenuto del modulo al server;
  + La funzione **read()** legge un file dal disco locale del client e ne carica il contenuto all’interno del modulo, in modo che possa essere inviato al server insieme agli altri dati. Viene utilizzato per fare l’**upload** delle immagini degli eventi;
  + La funzione **showField()** mostra una casella di testo all’interno del form quando si seleziona un determinato **radio button**. Questa viene utilizzata nella pagina per la concessione dei privilegi: quando si seleziona come **privilegio** da concedere l’opzione “**Organizzatore eventi**”, essa visualizza la casella di testo in cui inserire il nome dell’organizzatore.
* All’interno del file **logout.js** è definita la funzione **logout()** che permette di eseguire il logout dall’applicazione web. La funzione semplicemente elimina i **cookies** relativi al sito web (i token di autenticazione sono contenuti nei cookies) e lo reindirizza alla pagina di **login**. Per eliminare i cookies si imposta la loro scadenza ad una data già trascorsa (1° gennaio 1970 - Unix epoch).
* All’interno del file **QRscanner.js** sono definite le funzioni necessarie per scansionare i QR code associati ai biglietti al momento dell’ingresso all’evento:
  + La funzione **scan()** accende la fotocamera del dispositivo del client e abilita la funzione per decodificare i QR code. Questa viene eseguita ogni 2 secondi, in maniera periodica fino a quando non viene disabilitata;
  + La funzione **stop()** spegne la fotocamera del dispositivo del client e disabilita la funzione per decodificare i QR code;
  + La funzione **decode()** esegue la decodifica dei QR code. Per prima cosa scatta un’**istantanea** del flusso video proveniente dalla fotocamera, poi effettua una chiamata alla libreria **jsqr** per tentare di decodificare il codice QR presente nell’immagine, se presente. Quando viene rilevato un codice QR i dati in esso contenuti vengono inviati al server, che provvederà ad eseguire l’invalidazione del biglietto.
* All’interno del file **table.js** sono definite le due funzioni che permettono di evidenziare le righe di una tabella quando si passa sopra col mouse:
  + La funzione **enableHover()** modifica il colore di una riga della tabella, quando si passa sopra col mouse, rendendola un pò più scura;
  + La funzione **disableHover()** ripristina il normale colore della riga, una volta che si sposta il mouse.
* La cartella **jsqr** contiene la libreria esterna che viene utilizzata lato client per scansionare i codici QR associati ai biglietti.

# Web pages

La cartella Web pages contiene i cinque moduli software che hanno il compito di interfacciarsi con gli utenti. In ogni modulo viene definita la struttura delle pagine web visualizzate dagli utenti che si interfacciano con quel modulo.

## areaRiservata.js

I moduli Amministrazione evento, Gestione biglietti e Vendite biglietti contengono un file chiamato **areaRiservata.js**. All’interno del file viene definita la classe **AreaRiservata**, implementata in maniera diversa in ciascuno dei tre moduli.

La classe eredita da **HTMLpage**. Il costruttore non fa altro che invocare il costruttore della superclasse per generare una nuova pagina web. A questa viene poi aggiunta come figlio la barra di navigazione che consente di navigare il catalogo ed aprire il menu a tendina.

Il menu a tendina della barra viene quindi personalizzato aggregando una serie di operazioni che l’utente può eseguire dalla propria area riservata. Le voci del menu a tendina sono aggiunte invocando il metodo **addItem()** della classe **Bar**.

Il menu a tendina viene personalizzato in modo diverso in ciascuno dei tre moduli, poiché gli utenti che interagiscono con esse hanno privilegi diversi. Tutti i menu a tendina presentano però delle operazioni comuni:

* **Visualizza profilo utente**, permette all’utente di visualizzare i dati del suo profilo ed eventualmente di modificarli;
* **Cambia password**, permette all’utente di modificare la password del proprio account;
* **Logout**, permette all’utente di eseguire il logout dall’applicazione web.

Le pagine web del sito, definite nei cinque moduli, sono rappresentate come classi javascript che ereditano dalla classe **HTMLpage** (annullatore e utente ospite) o dalla classe **AreaRiservata** (staff biglietteria, cliente, organizzatore eventi). Il corpo delle pagine web viene poi riempito aggiungendo oggetti della classe **Widget** o di sue sottoclassi.

In questa sezione non viene approfondita la struttura delle pagine web, che sono descritte in maniera più dettagliata all’interno della Guida Utente di ticketTwo.