

F1LiveryStyle 1.0

Titolo del progetto: F1 Livery Style
Alunno/a: Amel Cehic
Classe: I3BD
Anno scolastico: 2025/2026
Docente responsabile: Geo Petrini

Sommario

1	Introduzione.....	3
1.1	Informazioni sul progetto.....	3
1.2	Scopo	3
2	Analisi	4
2.1	Analisi del dominio	4
2.2	Analisi e specifica dei requisiti	4
2.2.1	Spiegazione elementi tabella dei requisiti:.....	8
2.3	Use Case.....	9
2.4	Pianificazione	10
2.5	Analisi dei mezzi.....	11
2.5.1	Software	11
2.5.2	Hardware.....	11
3	Progettazione	12
3.1	Design delle interfacce.....	12
3.1.1	Interfaccia Selezione modello	12
3.1.2	Interfaccia Personalizzazione modello	13
3.2	Design procedurale	14
4	Implementazione	15
4.1	Struttura cartella Prodotto	15
4.2	Pagina principale (Scelta modello).....	15
4.2.1	Interfaccia Grafica	15
4.2.2	Codice	16
4.3	Pagina di Personalizzazione	27
4.3.1	Interfaccia Grafica	27
4.3.2	Codice	28
5	Test.....	40
5.1	Protocollo di test.....	40
5.2	Risultati test.....	43
5.3	Tabella Riassuntiva dei Test Case.....	50
5.4	Mancanze/limitazioni conosciute.....	50
6	Consuntivo.....	51
7	Conclusioni	52
7.1	Sviluppi futuri.....	52
7.2	Considerazioni personali.....	52
8	Glossario	53
9	Bibliografia	54
9.1	Sitografia	54
10	Indice delle figure.....	55
11	Allegati	56

1 Introduzione

1.1 Informazioni sul progetto

- Nome del progetto: F1 Livery Style
- Nome e Cognome del progettista: Amel Cehic
- Scuola: CPT Trevano
- Sezione: SAMT Informatica
- Classe: I3BD
- Modulo: 306 - Realizzare un piccolo progetto IT
- Docente responsabile: Geo Petrini
- Data inizio progetto: 05.09.2025
- Data fine progetto: 19.12.2025

1.2 Scopo

Lo scopo principale di questo progetto è quello di implementare un applicativo a livello web, che permetterà agli utenti appassionati del mondo automobilistico, ed eventuali fanatici di modelli tridimensionali, di esprimere la propria creatività nella personalizzazione della vettura dei sogni. Oltre alla personalizzazione, lo scopo del progetto è quello di migliorare le mie competenze informatiche basate sull'uso e gestione dei modelli tridimensionali, e i vari linguaggi che andrò ad utilizzare. Migliorerà sicuramente le abilità di competenza nei linguaggi web di HTML, CSS, Javascript (con il framework babylon.js).

2 Analisi

2.1 Analisi del dominio

F1 Livery Style è un applicativo web creativo e user-friendly, semplice da utilizzare, che permette di personalizzare in modo libero le monoposto di Formula 1 (sport automobilistico), con modelli diversi, sia modelli storici che quelli moderni. L'obbiettivo principale del prodotto è quello di offrire alle persone la possibilità di esprimere la propria creatività nella personalizzazione tridimensionale delle vetture. Esso è destinato ad un pubblico vasto, come i fanatici della Formula 1, designer generali o semplici curiosi interessati nel mondo dello sport automobilistico. Per utilizzarlo, non si necessita di alcuna competenza o skill, è sufficiente la propria fantasia o creatività.

Al momento, sul mercato ci sono prodotti simili dedicati nel lavoro con modelli tridimensionali automobilistici, maggior parte a livello applicativo. Tuttavia, il mio applicativo sarà implementato a livello web utilizzando i 3 principali linguaggi informatici del mondo web (html, css e javascript). Uno aspetto che renderà il mio prodotto in modo unico si tratta della possibilità di avere una visualizzazione dell'oggetto in modo dettagliato e creativo (animazioni e background). Inoltre, si potrà esportare il modello personalizzato con estensioni destinati agli oggetti 3D (GLB), assieme ad immagini estetiche che rispecchieranno l'oggetto personalizzato.

2.2 Analisi e specifica dei requisiti

ID: REQ-01	
Nome	Creazione Interfaccia grafica (pagina principale)
Priorità	1
Versione	1.0
Note	L'utente deve poter visualizzare il titolo, descrizione del progetto, e selezionare il modello da personalizzare.
Sotto requisiti	
001	Si necessita il titolo del prodotto.
002	Si necessita di una piccola descrizione del prodotto.
003	Si necessita di un'area dove l'utente può scegliere il modello.
004	L'utente dovrà visualizzare il modello scelto.
005	Si necessita un bottone per confermare il modello da personalizzare.

ID: REQ-02	
Nome	Algoritmo di Visualizzazione del modello scelto nella pagina di Selezione Modello
Priorità	1
Versione	1.0
Note	L'utente deve poter visualizzare il modello scelto, per decidere se personalizzarlo o sceglierne un altro.
Sotto requisiti	
001	Si necessita di un canvas dove creare la scena, il motore e importare il file del modello scelto (con babylon.js).
002	Si necessita di applicare una rotazione automatica, per avere una visualizzazione del modello scelto più concreta e performante.

ID: REQ-03	
Nome	Creazione Interfaccia grafica (pagina personalizzazione)
Priorità	1
Versione	1.0
Note	L'utente deve poter visualizzare il modello scelto, per poi personalizzarlo, esportarlo ed applicando animazioni.
Sotto requisiti	
001	L'utente dovrà visualizzare il modello scelto, con le modifiche che verranno apportate.
002	Si necessita di un bottone dove si può tornare indietro alla pagina principale.
003	Si necessita di un bottone che possa esportare il modello modificato.
004	Si necessita di un'opzione per immettere delle animazioni alla vettura personalizzata.
005	Si necessita di una sezione dove impostare il background.
006	Si necessita di una sezione dove l'utente ha la possibilità di selezionare una zona specifica della vettura.
007	Si necessita di un modificare per colorare una zona scelta dall'utente.
008	Si necessita di una sezione dove impostare il materiale del modello.

ID: REQ-04	
Nome	Algoritmo di selezione zona, a cui inserisce un colore per un tempo breve
Priorità	1
Versione	1.0
Note	L'utente deve poter selezionare una zona, e visualizzare la zona scelta che sarà colorata per un breve periodo, in modo che si può capire che zona è stata scelta sul modello.
Sotto requisiti	
001	Si necessita di un select con le varie opzioni di seleziona zona.
002	Si necessita di una funzione che ricava la zona e la colora, visualizzando la zona selezionata colorata per un breve periodo.

ID: REQ-05	
Nome	Algoritmo di colorazione zona selezionata del REQ-04
Priorità	1
Versione	1.0
Note	L'utente deve poter colorare la zona scelta.
Sotto requisiti	
001	Si necessita di un input color per colorare la zona.

ID: REQ-06	
Nome	Algoritmo di inserimento immagine sulla zona selezionata nel REQ-04
Priorità	1
Versione	1.0
Note	L'utente deve poter inserire un'immagine (sponsor) nella zona selezionata.
Sotto requisiti	
001	Si necessita di un input file che accetta file immagini, a cui inserisce l'immagine nella zona selezionata.

ID: REQ-07	
Nome	Algoritmo dell'applicazione materiale sull'intero modello
Priorità	1
Versione	1.0
Note	L'utente deve poter selezionare un materiale predefinito e importarlo sul modello.
Sotto requisiti	
001	Si necessita di una sezione dove vengono inseriti l'immagine del materiale, a cui si possono selezionare.

ID: REQ-08	
Nome	Algoritmo di esportazione della vettura
Priorità	1
Versione	1.0
Note	L'utente deve poter esportare il modello personalizzato, ricevendo il modello e delle immagini della vettura.
Sotto requisiti	
001	Si necessita di esportare la cartella compromessa con il file GLB assieme alle immagini scattate.

ID: REQ-09	
Nome	Algoritmo di cambio background immagine/colore
Priorità	1
Versione	1.0
Note	L'utente deve poter modificare il background a propria scelta.
Sotto requisiti	
001	Si necessita di un picker, per scegliere un'immagine per il background.
002	Si necessita di un color picker per cambiare il colore del background.

ID: REQ-10	
Nome	Algoritmo di scelta tipo di gomma
Priorità	1
Versione	1.0
Note	L'utente deve poter selezionare un tipo di gomma da impostare al modello.
Sotto requisiti	
001	Si necessita di una zona che interpreta 3 immagini dei vari tipi di gomma.
002	L'utente deve poter selezionare uno dei 3 tipi e visualizzare la modifica importata al modello.

ID: REQ-11	
Nome	Algoritmo di abilitazione delle animazioni sul modello
Priorità	1
Versione	1.0
Note	L'utente deve poter testare animazioni sul modello.
Sotto requisiti	
001	Si necessitano dei button che avviano l'animazione.

2.2.1 Spiegazione elementi tabella dei requisiti:

ID: identificativo univoco del requisito.

Nome: breve descrizione del requisito.

Priorità: indica l'importanza di un requisito nell'insieme del progetto, definita assieme al committente. Ad esempio, poter disporre di report con colonne di colori diversi ha priorità minore rispetto al fatto di avere un database con gli elementi al suo interno. Solitamente si definiscono al massimo di 2-3 livelli di priorità.

Versione: indica la versione del requisito. Ogni modifica del requisito avrà una versione aggiornata.

Sulla documentazione apparirà solamente l'ultima versione, mentre le vecchie dovranno essere inserite nei diari.

Note: eventuali osservazioni importanti o riferimenti ad altri requisiti.

Sotto requisiti: elementi che compongono il requisito.

2.3 Use Case

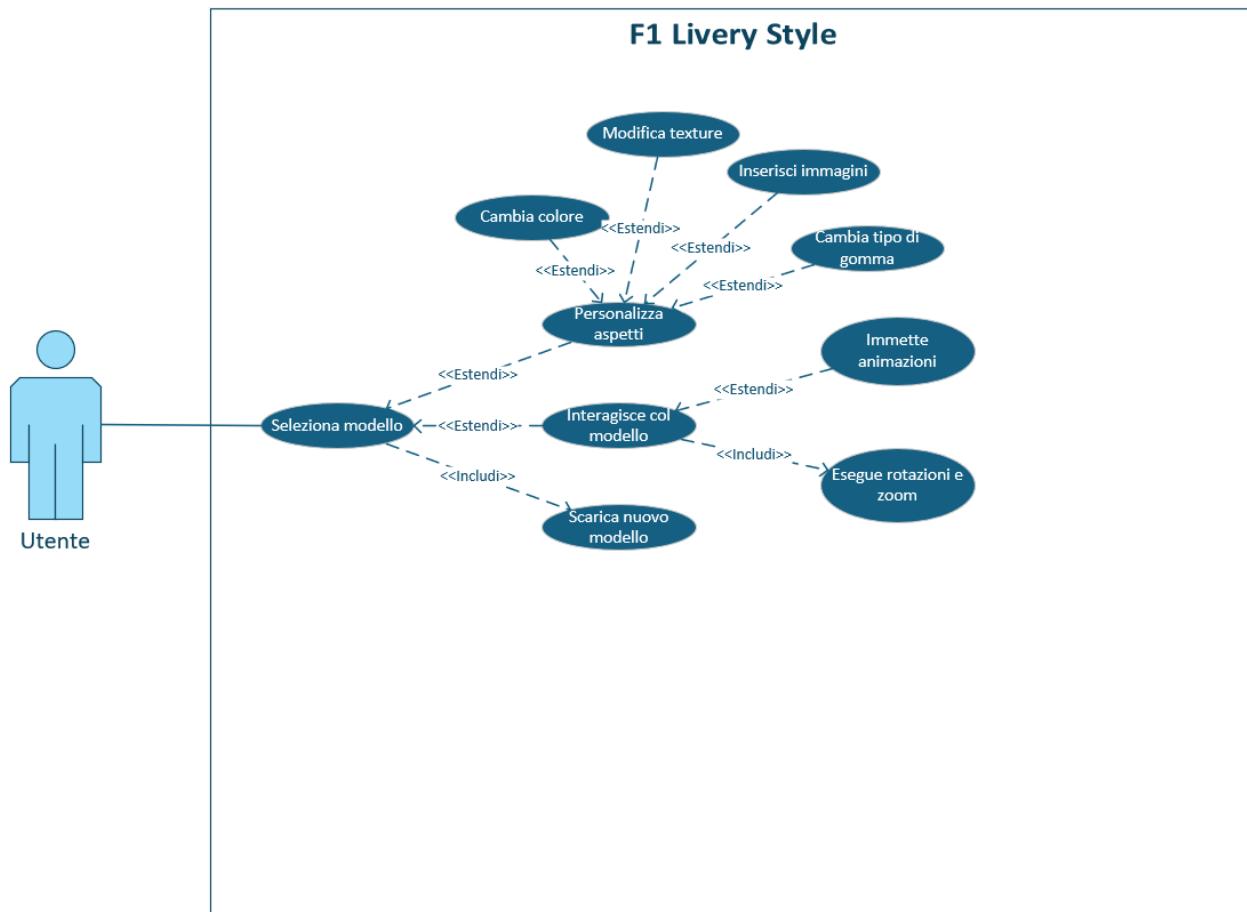


Figura 1: Use Case

Questo è lo Use Case dell'applicativo web F1LiveryStyle.

Il diagramma presenta un attore unico, **l'Utente**, quello che utilizzerà l'applicativo.

La funzione principale dell'Utente si tratta del selezionare il modello da personalizzare, altrimenti egli non potrà personalizzare la vettura che vuole. Dopodiché, si passerà alla personalizzazione, ove l'utente avrà **3** funzioni opzionali, che sono la **Personalizzazione dell'aspetto** del veicolo (colori, texture, inserimento immagini, tipo di gomma da usare), **Interazione con il modello** (ovvero, personalizzare anche la visualizzazione del modello modificato, con vari effetti, animazioni generali e di alcune zone della vettura, come le gomme), ed infine la funzione del **Scarica modello** (avendo il modello modificato, l'utente può scaricarlo con l'estensione dei formati tridimensionali, assieme ad una cartella avente immagini del veicolo con diverse angolazioni)

Documentazione F1LiveryStyle

2.4 Pianificazione

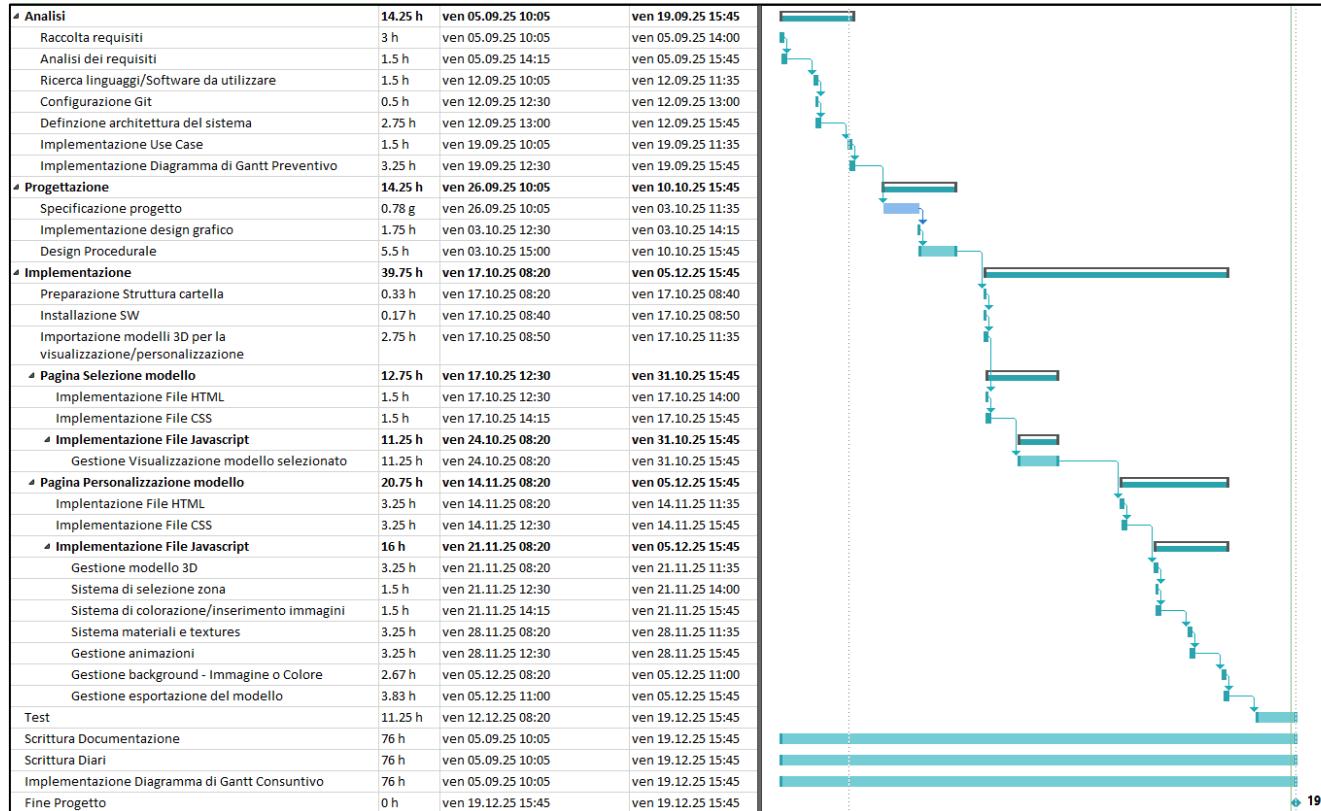


Figura 2: Diagramma di Gantt Preventivo

Il modello per implementare il diagramma di Gantt è Waterfall.

Per la fase dell'Analisi ho previsto 3 lezioni, tempo necessario per comprendere il progetto, raccogliendo e specificando i requisiti, fare ricerche per prendere qualche spunto e creare i diagrammi Use Case e Gantt Preventivo. Per la fase di Progettazione ho definito nuovamente 3 lezioni, per definire l'architettura del sistema e design delle interfacce assieme al design procedurale. Per la fase più lunga, Implementazione, ho stabilito 4 lezioni per lo sviluppo del codice, con 2 pagine web dove una servirà a selezionare il modello, mentre l'altra pagina per la personalizzazione del modello scelto. Infine ho preso 2 lezioni, le finali, per eseguire i test necessari, e per risolvere problemi in caso di test fallito.

2.5 Analisi dei mezzi

Per l'implementazione del prodotto e la scrittura della sua documentazione utilizzerò il computer fornito dalla scuola SAMT assieme ai software installati.

2.5.1 Software

- Visual Studio Code
Versione 1.105.1
Per scrittura/esecuzione codici dei linguaggi informatici web del progetto
- Blender 3D
Versione 4.5.3
Per gestire/implementare i modelli tridimensionali
- Microsoft Visio 2016
Versione 2509 (Build 16.0.19231.20138) a 64 bit
Per l'implementazione Use Case, Design, Diagramma Architetturale, Swimlane
- Microsoft Project 2016 Professional
Versione 1711 (Build 8730.2122) a 64 bit
Per realizzare il diagramma di Gantt (Preventivo/Consuntivo)
- Microsoft Word per Microsoft 365
Versione 2311 (Build 16.0.17029.20028) a 64 bit
Per fare la documentazione del prodotto richiesto

2.5.2 Hardware

- SSD SanDisk Extreme 55AE
SCSI Disk Device 1TB
Dove sarà contenuto il prodotto

3 Progettazione

3.1 Design delle interfacce

Per la realizzazione delle interfacce, sono state progettate con il software Microsoft Visio 2016.

3.1.1 Interfaccia Selezione modello

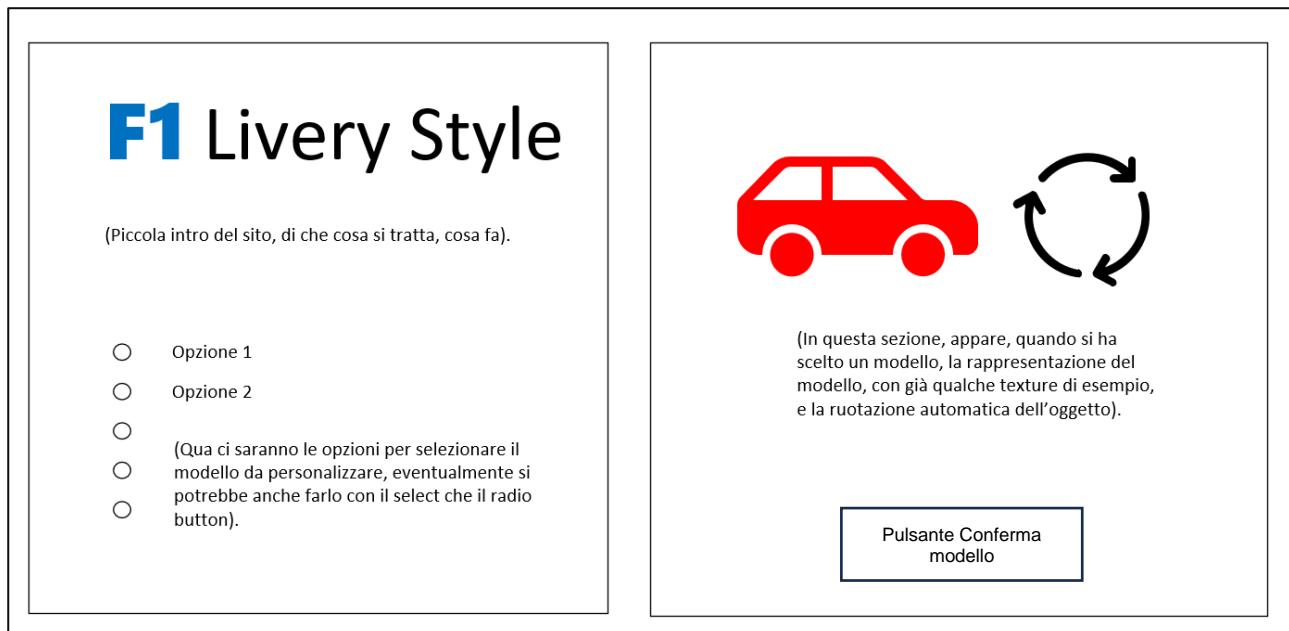


Figura 3: Design Pagina Principale

Questa è la pagina principale, ovvero la pagina del Selezione modello. Essa conterrà 2 sezioni:

- **Sezione scelta:** si tratta della parte input, ove l'utente legge il titolo del prodotto, la sua descrizione e seleziona il modello da personalizzare (con il nome e anno della vettura).
- **Sezione visualizza e conferma:** si tratta della parte output, dove l'utente potrà visualizzare il modello (con qualche texture e dei colori sopra inseriti, come esempio) selezionato tridimensionalmente ed esteticamente, con una rotazione automatica dell'oggetto per visualizzarlo meglio. Quando avrà deciso il modello da personalizzare, selezionerà Conferma modello, per passare alla personalizzazione.

3.1.2 Interfaccia Personalizzazione modello

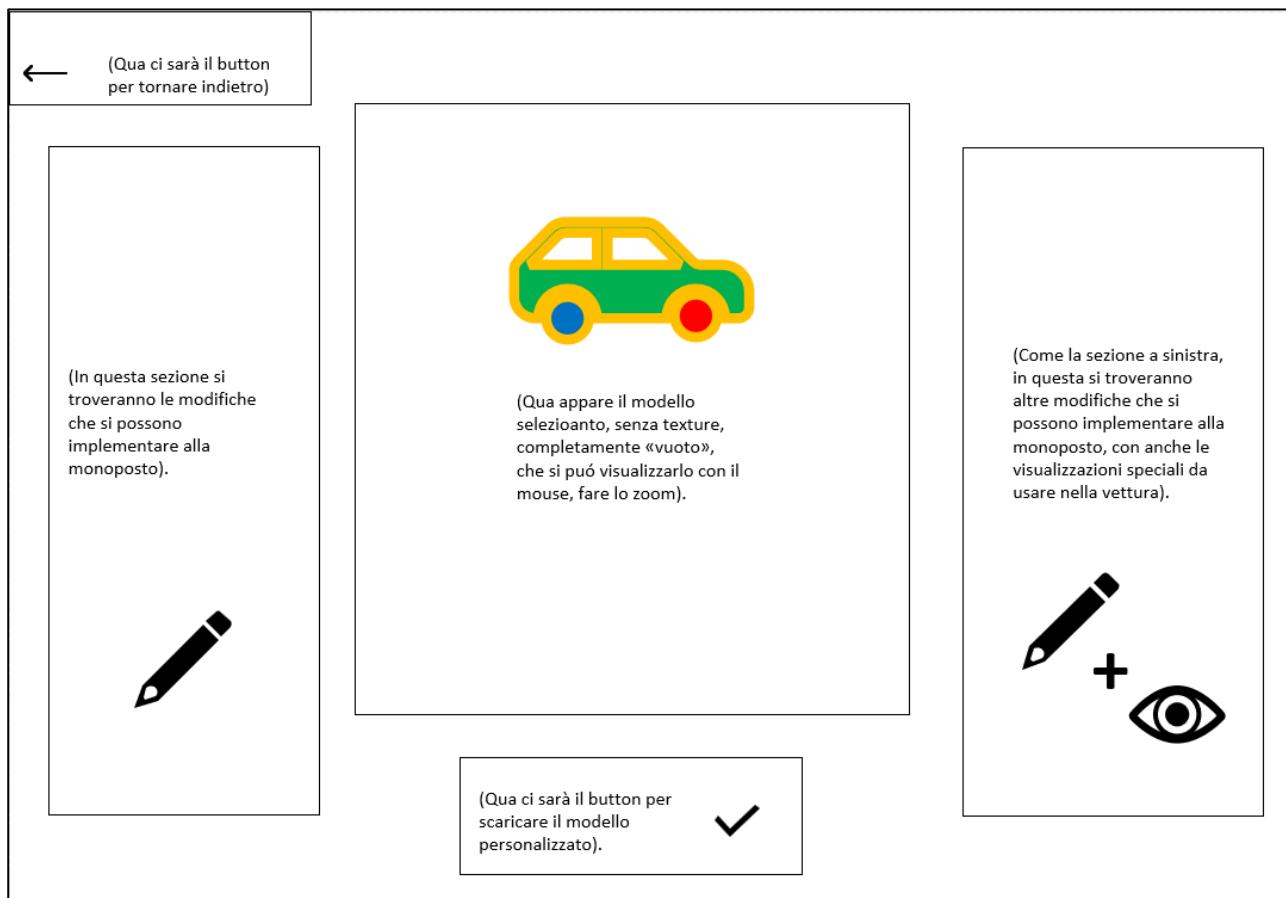


Figura 4: Design Pagina Personalizzazione

Dopo la selezione del modello, si passerà alla pagina di Personalizzazione modello. Al centro della pagina, ci sarà il modello presentato, con le modifiche a cui saranno apportate dall'utente. Nelle parti laterali della pagina, verranno presentati i modificatori che si possono immettere all'oggetto, con anche la parte della visualizzazione dettagliata con animazioni (inserita assieme ai modificatori della parte laterale a destra). Dopodiché, sottostante a tutto ciò, viene presentato il pulsante per esportare l'oggetto personalizzato, assieme alle immagini dell'oggetto. Infine, nella parte sopra a sinistra, si trova il pulsante per tornare indietro alla pagina principale del Selezione modello.

3.2 Design procedurale

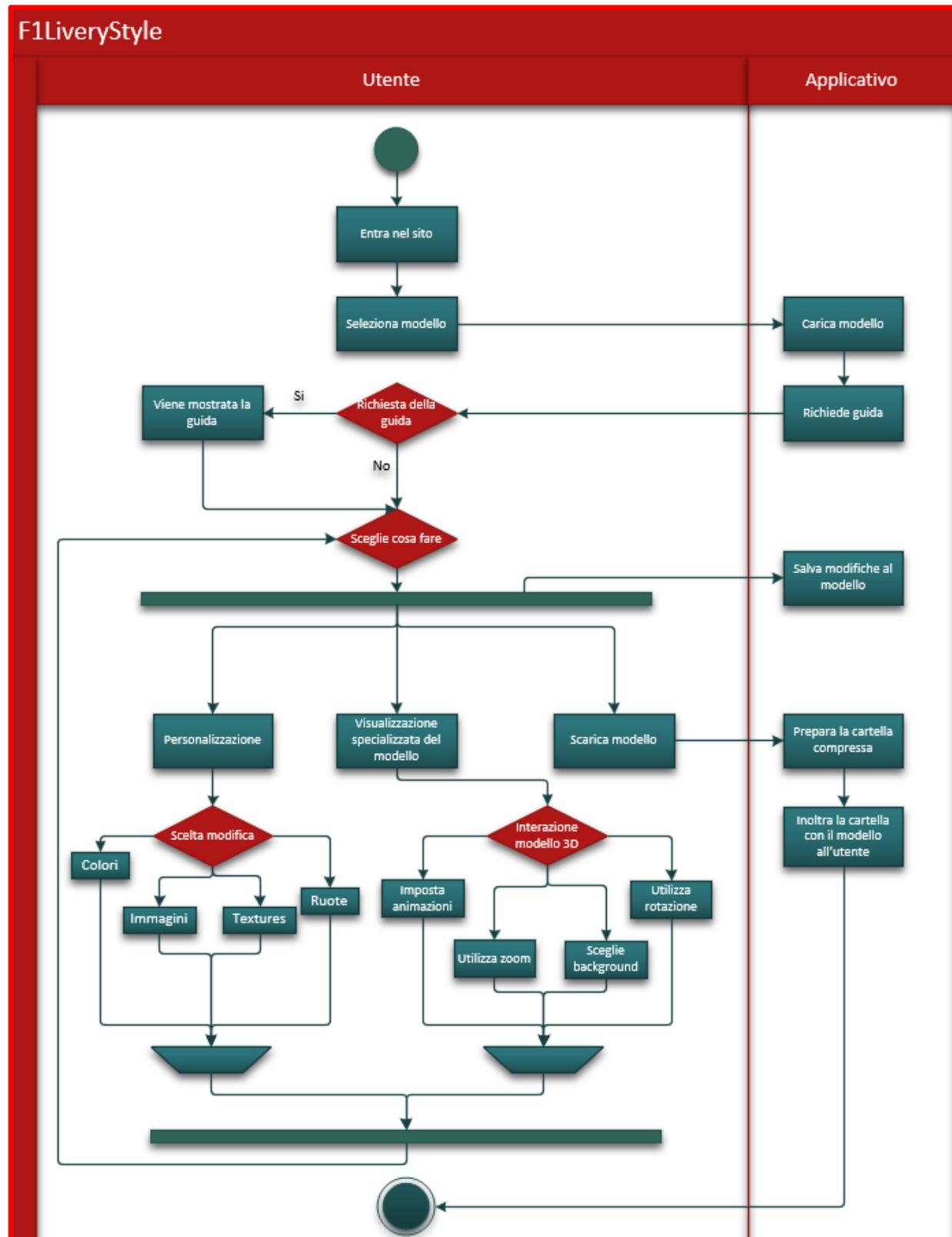
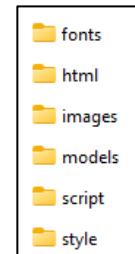


Figura 5: Diagramma di flusso

4 Implementazione

4.1 Struttura cartella Prodotto

Innanzitutto, creo le cartelle essenziali per organizzare i vari file, con le cartelle **fonts** (dove conterrà i fonts che si utilizzeranno da importare nel progetto), **html** (per i file html), **images** (per le immagini che si metteranno nel sito), **models** (modelli di visualizzazione e quelli da modificare), **script** (vari script per far funzionare il prodotto) e **style** (per i stili che si applicheranno nel html).



4.2 Pagina principale (Scelta modello)

Figura 6: Struttura cartelle

Questa sarà la pagina principale del prodotto, dove l'utente ha la possibilità di visualizzare la lista di modelli che si possono scegliere da personalizzare, visualizzando quando è stato scelto, il modello anticipatamente. Dopo aver deciso il modello, apparirà il pulsante “conferma”, che permetterà di passare alla fase di personalizzazione.

4.2.1 Interfaccia Grafica

L'interfaccia grafica della pagina di selezione modello l'ho implementata dando estetica e facilità nell'utilizzarla. Infatti, viene presentato la zona Scelta come “input”, con titolo, descrizione del sito e la scelta di modello, mentre dall'altra parte c'è la sezione “output” del seleziona modello, ovvero la visualizzazione del modello scelto.

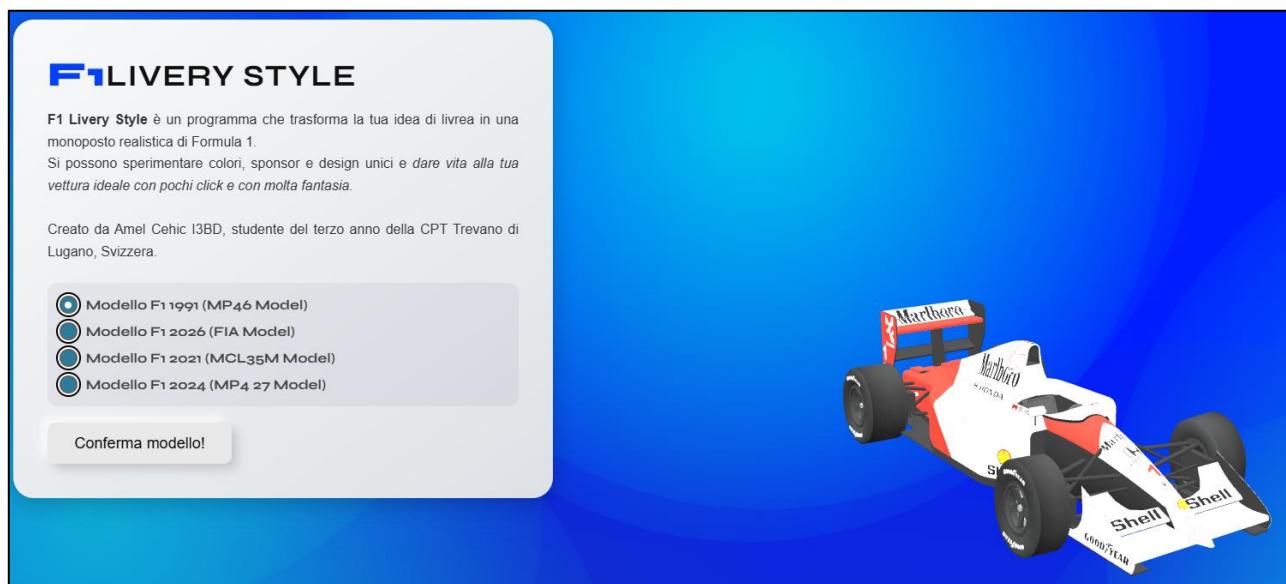


Figura 7: Interfaccia Grafica - Pagina Selezione modello

4.2.2 Codice

Per implementare la pagina del Seleziona modello, ho creato i file selectModel.html (nella cartella html), style.css (nella cartella style) e script.js (nella cartella script). Come aggiuntivo ho scaricato un font per generare un buon titolo, e ho scaricato un'immagine come icona di una vettura F1 per metterlo nel favicon.

4.2.2.1 File HTML

4.2.2.1.1 Head

Per il file html, inizio dichiarando <!DOCTYPE html> (che indica al browser che il documento utilizza lo standard di html5) e aproendo il tag <html> che racchiude l'intero contenuto della pagina. Poi apro il tag head dove vengono inserite informazioni per il browser come 2 commenti che indicano le date inizio e fine dell'implementazione e il nome del file e il title per indicare il titolo della pagina che apparirà nel browser.

Dopodiché inserisco i metatag che forniscono informazioni aggiuntive, come codifica dei caratteri (UTF-8, per mostrare correttamente i caratteri speciali), una breve descrizione della pagina (description, utile per il motore di ricerca), il nome dell'autore (author) e la viewport (per la gestione di scaling e addattamento grafico).

Infine, inserisco nell'head 2 collegamenti, uno per l'icona del sito che verrà mostrata nella pagina (icon), mentre l'altra per lo stile e aspetto grafico della pagina (stylesheet), preso dal file style.css.

```
<!DOCTYPE html>
<html>

<head>
    <!-- data creazione: 03.10.2025 data ultima modifica: 31.10.2025 -->
    <!-- selectModel.html -->
    <title>Personalizza la tua vettura!</title>
    <meta charset="UTF-8">
    <meta name="description" content="Scelta di modello F1 da personalizzare">
    <meta name="author" content="Amel Cehic">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="icon" href="../images/f1.png">
    <link rel="stylesheet" href="../style/style.css">
</head>
```

Figura 8: File HTML - Head

4.2.2.1.2 Body

Per la parte body, apro il tag <body>, dove tutti gli elementi al suo interno verranno presentati nella pagina. Innanzitutto, inserisco un div con classe header finisher, per immettere il background in movimento, che conterrà tutti gli elementi restanti. Implemento il canvas con un id “canvasOggetto”, che servirà per far visualizzare il modello selezionato, con delle texture sopra anticipatamente.

Inserisco la parte Introduttiva, che avrà il titolo, una descrizione, una selezione e una conferma, facendo un div della classe “intro” con:

- **Titolo** → **h1** (font personalizzato) e **span** (colorato di blu, in grassetto e grande di dimensione 40px).
- **Descrizione** (**p**) → descrive lo scopo del prodotto, assieme al tag **p** per paragraph, classe “paragrafo”, con al suo interno i tag **strong** (testo in grassetto), **br** (a capo) ed **em** (corsivo).
- **Selezione** (div class=“select”) → permette di scegliere il modello, implementata assieme al label classe “radioButton” con l’**input** (type radio) e il **testo** (span classe “label-next”).

Quando viene scelto un modello, il button “conferma” sarà visibile per passare alla fase della personalizzazione. Infine, includo quattro script che serviranno a far funzionare la pagina:

- <https://cdn.babylonjs.com/babylon.js> → motore Babylon 3D per la visualizzazione di modelli.
- <https://cdn.babylonjs.com/loaders/babylonjs.loaders.min.js> → per caricare formati 3D.
- <..//script/finisher-header.es5.min.js> → script preso da <https://www.finisher.co/lab/header/>, per l’animazione grafica del header (sfondo animato).
- <..//script/script.js> → script del prodotto per gestire il modello, per caricarlo, e per varie configurazioni della pagina web.

```

<body>
    <div class="header finisher-header">
        <canvas id="canvasOggetto"></canvas>
    </div>
    <div class="intro">
        <h1 style="font-family: font1;text-align: left;"><span
            style="color:#0040ff;font-weight: bolder;font-size: 40px;">F1</span>Livery Style</h1>
        <p class="paragrafo">
            <strong>F1 Livery Style</strong>
            è un programma che trasforma la tua idea di
            livrea in una monoposto realistica di Formula 1. <br> Si possono sperimentare colori, sponsor e design unici e <em>dare vita
            alla tua vettura ideale con pochi click e con molta fantasia.</em> <br><br>
        </p>
        <div class="select">
            <label class="radioButton">
                <input type="radio" name="modello" value="1991" id="1991" />
                <span class="label-text">Modello F1 1991 (MP46 Model)</span>
            </label>
            <label class="radioButton">
                <input type="radio" name="modello" value="2026" id="2026" />
                <span class="label-text">Modello F1 2026 (FIA Model)</span>
            </label>
            <label class="radioButton">
                <input type="radio" name="modello" value="McLaren" id="McLaren" />
                <span class="label-text">Modello F1 2021 (MCL35M Model)</span>
            </label>
            <label class="radioButton">
                <input type="radio" name="modello" value="2024" id="2024" />
                <span class="label-text">Modello F1 2024 (MP4 27 Model)</span>
            </label>
        </div><br>
        <button class="conferma">Conferma modello!</button>
    </div>
</div>

<script src="https://cdn.babylonjs.com/babylon.js"></script>
<script src="https://cdn.babylonjs.com/loaders/babylonjs.loaders.min.js"></script>
<script src="..//script/finisher-header.es5.min.js" type="text/javascript"></script>
<script src="..//script/script.js"></script>
</body>

</html>

```

Figura 9: File HTML – Body

4.2.2.2 File Style CSS

Per l'implementazione del file style.css, inizio inserendo un'importazione del font Syne da Google Fonts, con i pesi 400,700 e 800, così da poterlo utilizzare nella pagina di selezione. Poi definisco un font personalizzato che viene ricavato dal file sotto la cartella fonts, nominandolo "font1", che verrà utilizzato nell'intero documento.

Dopodiché, imposto lo stile di base dell'intera pagina, inserendo il font da utilizzare (quello personalizzato, font1) ed eliminando spazi predefiniti dei browser. Per il header.finisher-header (lo sfondo animato) ho preso il codice generato dal sito, dove definisce la posizione di elementi rispetto al contenitore (position: relative), impostando la larghezza fissa al 100%, quindi tutta la pagina, per un'altezza minima di 100vh (800px) ed un overflow hidden, cioè che tutto ciò che esce dai bordi del contenitore viene nascosto.

Per il canvas, nominato l'id "canvasOggetto", definisco la posizione assoluta (libera), con top 0 (messo alla parte superiore), un right di 10% (spostato verso sinistra di 10% dalla destra dello schermo), lo width di 50% (larghezza che prende metà della pagina), altezza di 100% (occupa tutta l'altezza disponibile), z-index 1 (ovvero, che sta sopra lo sfondo, per sicurezza) e infine disattivando l'interazione del mouse con il canvas (pointer-events: none).

```
@import url('https://fonts.googleapis.com/css2?family=Syne:wght@400;700;800&display=swap');

@font-face {
    font-family: "font1";
    src: url(../fonts/Syne-VariableFont_wght.ttf);
}

body {
    font-family: "font1", sans-serif;
    margin: 0;
    padding: 0;
}

.header.finisher-header {
    position: relative;
    width: 100%;
    min-height: 100vh;
    overflow: hidden;
}

#canvasOggetto {
    position: absolute;
    top: 0;
    right: 10%;
    width: 50%;
    height: 100%;
    z-index: 1;
    pointer-events: none;
}
```

Figura 10: File CSS - Parte 1

Per la classe intro della parte scelta modello, ho inserito uno sfondo gradient orientato a 145 gradi che passa da un colore chiaro a un grigio tenue, assieme ad un border-radius arrotondato a 1.5rem (23px, 1rem=16px), spazio interno a 2.5rem, un massimo di larghezza di 550px in modo da evitare che l'intro diventi troppo larga sugli schermi grandi, un'ombra esterna che crea quel effetto di profondità (neo morfismo), il tutto allineato a sinistra (sia con content che con text per sicurezza), una transizione fluida di 0.3 secondi (avviata a qualsiasi cambiamento, come hover), margine 50px e un z-index 0 per metterlo alla base.

Con l'intro, applico un effetto quando viene fatto l'evento hover (quando il cursore passa sopra la sezione intro) con una sollevazione verso l'alto di 5px (translateY) e aumentando l'ombra, rendendo quell'effetto sollevato più comprendibile. Mentre per i titoli (h1) rimuovo margini predefiniti (margin 0), applico la dimensione del font di 2.2rem, coloro il testo di nero, trasformo in maiuscolo il testo ed applico uno spazio tra le lettere di 1px. Invece per nuovamente i titoli (h1) e il tag span applico il colore nero, aumentando lo spessore del carattere di 800 e applicando la dimensione di 2.5rem.

Per la classe paragrafo, utilizzata nella descrizione del prodotto, utilizzo della famiglia Arial il font sans-serif, colorando con un grigio scuro, impostando una dimensione fissa di 1rem, aumentando l'interlinea di 1.6 per renderlo più leggibile, aggiungendo spazio sopra il paragrafo per separarlo dal titolo (margin-top) e giustificando il testo per averlo più ordinato.

```
.intro {  
background: linear-gradient(145deg, #f8f9fa, #dee2e6);  
border-radius: 1.5rem;  
padding: 2.5rem;  
max-width: 550px;  
box-shadow: 0 8px 24px rgba(0, 0, 0, 0.2);  
justify-content: left;  
text-align: left;  
transition: all 0.3s ease;  
margin: 50px;  
z-index: 0;  
  
}  
  
.intro:hover {  
transform: translateY(-5px);  
box-shadow: 0 12px 32px rgba(0, 0, 0, 0.25);  
}  
  
.intro h1 {  
margin: 0;  
font-size: 2.2rem;  
color: #111;  
text-transform: uppercase;  
letter-spacing: 1px;  
}  
  
.intro h1 span {  
color: #0B1D51;  
font-weight: 800;  
font-size: 2.5rem;  
}  
  
.paragrafo {  
font-family: Arial, sans-serif;  
color: #333;  
font-size: 1rem;  
line-height: 1.6;  
margin-top: 1rem;  
text-align: justify;  
}
```

Figura 11: File CSS - Parte 2

Con la classe del select, utilizzata per la selezione modello, imposto il layout come flexbox e lo organizzo in colonna, dando uno spazio tra gli elementi di 0.75rem, assieme ad uno spazio nella parte superiore di 1.5rem, con lo spazio interno di 1rem, arrotondando gli angoli 0.75rem, applicando il colore del background di un leggero blu scuro e dando un effetto sfocato allo sfondo sottostante (blur).

Per il radioButton in se, uso anche in quel caso un flexbox centrati verticalmente, avente uno spazio tra gli elementi di 0.75rem, rendendo “cliccabile” il contenuto cambiando tipo di cursore, impostando la dimensione di 1rem e il spessore ben evidenziato, colorato con un grigio scuro e una transizione di 0.2 secondi del colore. Invece, quando il cursore è sopra all’elemento (avvio della transizione) si applica un nuovo colore più “evidente” per dare un effetto.

Nel radioButton, nel specifico nell’input di tipo radio, rimuovo lo stile predefinito del browser (appearance, per il radio), imposto la dimensione come un cerchio normale (width e height), colorandolo di blu, rendendolo più circolare con gli angoli arrotondati (border-radius), impostando nessun bordo, applicando un triplo livello di ombra per dare quell’effetto tridimensionale (box-shadow, con un’ombra più vicina e scura per dare profondità, un’ombra più chiara bianca per simulare luce e un’ombra più ampia e scura per evidenziare il “rialzo”), gestendo la posizione e mantenendo la cliccabilità assieme ad una transizione di tutto in 0.2 secondi, ad esempio per le ombre tridimensionali.

```
.select {  
    display: flex;  
    flex-direction: column;  
    gap: 0.75rem;  
    margin-top: 1.5rem;  
    padding: 1rem;  
    border-radius: 0.75rem;  
    background: rgba(11, 29, 81, 0.05);  
    backdrop-filter: blur(4px);  
}  
  
.radioButton {  
    display: flex;  
    align-items: center;  
    gap: 0.75em;  
    cursor: pointer;  
    font-size: 1rem;  
    font-weight: bold;  
    color: #444;  
    transition: color 0.2s ease;  
}  
  
.radioButton:hover {  
    color: #0B1D51;  
}  
  
.radioButton input[type="radio"] {  
    appearance: none;  
    width: 1.25em;  
    height: 1.25em;  
    background: #2b7d9e;  
    border: none;  
    border-radius: 50%;  
    box-shadow:  
        0 0 0.15em #636363,  
        0 0 0.3em #fff,  
        0 0 0.45em #000;  
    margin: 0;  
    transition: all 0.2s ease;  
    position: relative;  
    cursor: pointer;  
}
```

Figura 12: File CSS - Parte 3

Per gestire il pallino quando si seleziona la scelta del radio button, utilizzo il ::before, con il content vuoto che “fungerà” per il pallino interno, impostando le dimensioni di un cerchio di 0.6rem, colorandolo di bianco, applicando ai suoi angoli l’effetto circolare (border-radius), posizionandolo più al centro possibilmente rispetto al radio button esterno (position, top, left), facendo un transform(-50%, -50%) per centralarlo esattamente con lo scale(0) rendendolo invisibile (grandezza 0) e applicando una transizione ad ogni cambiamento di 0.2s sul transform.

Quando è stato selezionato (:checked::before), il pallino passa da scale(0) a scale(1) espandendosi rendendo visibile, creando quindi un buon effetto di selezione animata.

Mentre quando si ha il cursore sopra, colora il cerchio esterno di azzurro, per far capire che è cliccabile.

Per il button Conferma, imposto un colore nero, con spazi interni “larghi” (0.7rem 1.7rem), con la dimensione di 18px, arrotondando gli angoli di 0.5rem, con uno sfondo molto chiaro, impostando il cursore puntatore, mettendo un bordo solido 1px di colore bianco, dando una transizione 0.3 secondi di tutto, applicando un’ombra tridimensionale (ombra esterna verso il basso a destra e un’ombra luce dall’alto a sinistra) e impostando la visibilità dell’elemento a invisibile perché verrà visibile quando verrà selezionato uno dei radio button.

Quando esso viene premuto, cambia colore più scuro e simulando un’ombra verso l’interno.

```
.radioButton input[type="radio"]::before {  
    content: "";  
    width: 0.6em;  
    height: 0.6em;  
    background: #fff;  
    border-radius: 50%;  
    position: absolute;  
    top: 50%;  
    left: 50%;  
    transform: translate(-50%, -50%) scale(0);  
    transition: transform 0.2s ease;  
}  
  
.radioButton input[type="radio"]:checked::before {  
    transform: translate(-50%, -50%) scale(1);  
}  
  
.radioButton input[type="radio"]:hover {  
    background: #6dc6e9;  
}  
  
.conferma {  
    color: #090909;  
    padding: 0.7em 1.7em;  
    font-size: 18px;  
    border-radius: 0.5em;  
    background: #e8e8e8;  
    cursor: pointer;  
    border: 1px solid #e8e8e8;  
    transition: all 0.3s;  
    box-shadow: 6px 6px 12px #c5c5c5, -6px -6px 12px #ffffff;  
    visibility: hidden;  
}  
  
.conferma:active {  
    color: #666;  
    box-shadow: inset 4px 4px 12px #c5c5c5, inset -4px -4px 12px #ffffff;  
}
```

Figura 13: File CSS - Parte 4

4.2.2.3 File Script

Per il file script nominato script.js, ho inserito codici che gestiscono l'animazione del modello quando verrà scelto da uno dei radio button.

Per farlo, ho iniziato istanziando delle variabili globali che verranno utilizzate in ogni funzione, ricavando il canvas, creando il “motore”, ovvero l’engine della scena, il pulsante conferma a cui servirà a renderlo visibile quando è stato scelto qualcosa, e le variabili vuote scena, currentModel (value del radio per il modello scelto) e la camera che verrà utilizzata in uno switch che la configura per ogni modello:

```
const canvasOggetto = document.getElementById("canvasOggetto");
const engine = new BABYLON.Engine(canvasOggetto, true);
//true --> abilita anti-aliasing (per avere un immagine piu "morbida")
const confermaButton = document.querySelector(".conferma");
let scena;
let currentModel = null;
let camera;
```

Figura 14: File JS - Variabili Globali

```
function creaScena() {
    scena = new BABYLON.Scene(engine);
    scena.clearColor = new BABYLON.Color4(0, 0, 0, 0);

    camera = new BABYLON.ArcRotateCamera(
        "camera",
        -Math.PI / 4, //alpha
        Math.PI / 3, //beta
        5000, //radius
        new BABYLON.Vector3(0, 0, 0), //target
        scena
    );
    camera.attachControl(canvasOggetto, false);
    camera.lowerRadiusLimit = 50;
    camera.upperRadiusLimit = 4000;
    //ROTAZIONE AUTOMATICA LENTA (+= 0.009)
    scena.registerBeforeRender(function () {
        camera.alpha += 0.009;
    });

    //luce che illumina dall'alto
    const luce1 = new BABYLON.HemisphericLight(
        "luce",
        new BABYLON.Vector3(0, 1, 0), //direzione
        scena
    );
    luce1.intensity = 4;

    //luce direzionale
    const luce2 = new BABYLONDirectionalLight(
        "luce2",
        new BABYLON.Vector3(-1, -2, -1),
        scena
    );
    luce2.intensity = 4;

    return scena;
}
```

Figura 15: File JS - Funzione creaScena()

Poi inizio a creare la scena con la funzione creaScena() che verrà chiamata in generale nel file, ovvero che si avvia subito. Con questa creo una nuova scena con il motore inserito nelle variabili globali e togliendo il colore cosicché lo sfondo della scena sia trasparente.

Con la variabile camera creata precedentemente, la configuro inserendogli il nome “camera”, con l’alpha (angolazione orizzontale, girare attorno) impostata a “-Pi greco / 4” (quindi l’angolo orizzontale ruotata di circa 45 gradi verso sinistra attorno all’oggetto), assieme al beta (angolazione verticale, come un’inquadratura dall’alto) “Pi greco / 3” (ovvero 60 gradi, con vista più frontale), radius lontano dal modello di 5000, con il target creato con Vector3 (per impostare in punto o vettore nello spazio 3D) messo al centro e chiamando la scena dove impostare questa camera, ovvero scena.

Con la camera implementata, la collego al canvas dove verrà mostrata (attachControl), imposta quanto si possa avvicinarsi o allontanarsi (per sicurezza, 50 e 4000), applicando una rotazione automatica dove per ogni frame aumenta l’alpha di 0.009 (registerBeforeRender).

Per le luci, ne creo 2 dove una illumina dall’alto, quella emisferica (Hemispheric) e quella direzionale (Directional), impostando ad entrambe un’intensità elevata di 4. Infine la funzione ritorna la scena configurata.

Avendo la camera della scena configurata, implemento una funzione che ricava il nome del modello (valore del radio button scelto), e modifica dei parametri della camera, poiché ogni modello ha la camera preimpostata diversamente. Per farlo inserisco uno switch del nomeFile, mettendo i case dei value dei radio button, configurando e modificando i valori del radius, assieme al limit. Inserisco un console.log per assicurarmi che si sta utilizzando la camera giusta per il modello giusto e metto il case default, che si mette sempre nello switch, con valori preimpostati del modello nullo.

```
function impostaCamera(nomeFile) {
    switch (nomeFile) {
        case "1991":
            camera.radius = 15;
            camera.lowerRadiusLimit = 10;
            camera.upperRadiusLimit = 15;
            console.log("camera modello 1991");
            break;

        case "2026":
            camera.radius = 20;
            camera.lowerRadiusLimit = 0;
            camera.upperRadiusLimit = 20;
            console.log("camera modello 2026");
            break;

        case "McLaren":
            camera.radius = 9;
            camera.lowerRadiusLimit = 0;
            camera.upperRadiusLimit = 9;
            console.log("camera modello mclaren");
            break;

        case "2024":
            camera.radius = 10;
            camera.lowerRadiusLimit = 0;
            camera.upperRadiusLimit = 10;
            console.log("camera modello mercedes W11");
            break;

        default:
            camera.radius = 1000;
            camera.lowerRadiusLimit = 50;
            camera.upperRadiusLimit = 1000;
            console.log("camera default");
            break;
    }
}
```

Figura 16: File JS - Funzione impostaCamera()

Avendo gestito la camera della scena per ogni modello, ora implemento la funzione che va a caricare il modello sul canvas. Nella funzione, che verrà chiamata quando uno dei radio button verrà selezionato, passo la variabile nomeFile (valore del radio button), controllando subito se effettivamente è già presente un modello, quindi rimuovendolo (tutti meshes, per evitare le sovrapposizioni).

Richiamo la funzione impostaCamera(), creo la variabile costante dove si trovano i models da farli visualizzare sotto “./models/views”, e per ultimo faccio il SceneLoader.ImportMesh, che carica il modello inserendo il nome meshes vuoto, la path root, il nome del file con estension GLB che si deve caricare, dove deve lavorare, quindi nella scena ed una funzione per il parametro onSuccess (quando ha successo il caricamento, esegue la funzione) che passa i meshes.

Nella funzione ricava i mesh mettendoli in una variabile currentModel, trova automaticamente il punto centrale del modello utilizzando i suoi bounding box (variabili dimensione e centro), per ogni mesh sposta la loro posizione al centro con il SubtractInPlace(centro), calcolo la grandezza reale del modello, sposta automaticamente la camera alla distanza giusta avendo la dimensione massima ed infine fa un print nella console, un log di conferma che è stato caricato con successo. Mentre se non viene caricato esegue una funzione nel onError, ove stampa in console il messaggio di errore.

Infine, richiamo la funzione creaScena() per “configurare” i parametri della scena e poi caricarla.

```
function caricaModello3D(nomeFile) {
    if (currentModel) {
        currentModel.forEach(mesh => mesh.dispose());
        currentModel = null;
    }
    impostaCamera(nomeFile);

    const percorsoModelli = "../models/views/";
    BABYLON.SceneLoader.ImportMesh(
        "", //nome meshes
        percorsoModelli, //path root
        nomeFile + ".glb",
        scena,
        function (meshes) { //parametro onsuccess, callback quando il modello è stato caricato correttamente
            currentModel = meshes;
            const dimensione = meshes[0].getHierarchyBoundingVectors(); //meshes[0] - nodo principale

            //la funzione getHierarchyBoundingVectors, serve per determinare punto estremo alto a sinistra e quello in basso a destra
            const centro = BABYLON.Vector3.Center(dimensione.min, dimensione.max);
            meshes.forEach(mesh => {
                mesh.position.subtractInPlace(centro); //sposta la posizione al centro
            });

            const grandezzaModello = dimensione.max.subtract(dimensione.min);
            //calcola dimensioni reali del modello | max - min
            const dimensioneMax = Math.max(grandezzaModello.x, grandezzaModello.y, grandezzaModello.z);
            /*Trova la dimensione più grande del modello,
            per capire quanto lontano deve stare la camera*/
            camera.radius = dimensioneMax * 2;
            //il * 2, viene eseguito, perché più è grande, più sta lontano

            console.log("modello: ", nomeFile);
        },
        null, //parametro onprogress
        function (scena, message) { //parametro onerror
            console.error("errore: ", message);
        }
    );
}

creaScena();
```

Figura 17: File JS - Funzione caricaModello()

Per chiamare le funzioni implementate precedentemente, devo gestire gli eventi che l'utente possa fare e gli aggiornamenti ad ogni momento nel browser. Gestisco l'evento del quando uno dei radio button viene selezionato, utilizzando un for per ognuno dei radio button, aggiungo l'evento "change" con la funzione che ricava il valore (quindi nomeFile), il button di conferma diventa visibile, invoco la funzione per caricare il modello passando il nome del modello da caricare e faccio una stampa alla console per il target camera, che mi mostra tutte le informazioni della camera.

Mentre per il pulsante conferma, quando viene premuto ricava il radio button selezionato e controlla se non è vuoto, per salvare l'informazione nel LocalStorage (informazioni dentro il browser), impostando la chiave e valore, che lo userò per ricavarlo nella prossima pagina, di personalizzazione, e sicuramente reindirizzo l'utente alla pagina di personalizzazione con il modello selezionato. Chiaramente è fondamentale anche aggiornare la scena ad ogni momento (come la grandezza) assieme alla pagina del browser.

Infine, inserisco il codice che mi permette di applicare il background animato preso dal sito (FinisherHeader).

```
//Dopo aver scelto, mostra il modello
const radioB = document.querySelectorAll('input[name="modello"]');
for (let radio of radioB) {
  radio.addEventListener("change", function () {
    let modello = this.value;
    confermaButton.style.visibility = "visible";
    caricaModello3D(modello);
    console.log("Target camera:", camera.target);
  });
}

//Confermi e vai a personalizzare
confermaButton.addEventListener("click", function () {
  const selezionato = document.querySelector('input[name="modello"]:checked');

  if (selezionato) {
    /*Salva il riferimento del modelli selezionato nel browser,
    a cui servirà per la pagina di personalizzazione*/
    localStorage.setItem("modelloSelezionato", selezionato.value);
    //Apre la pagina di personalizzazione
    window.location.href = "styleModel.html";
  } else {
    alert("seleziona un modello");
  }
});

//Aggiorna ogni momento la scena
engine.runRenderLoop(function () {
  scena.render();
});

//Aggiorna la dimensione della finestra ogni volta che si cambia, quindi il canvas
window.addEventListener("resize", () => {
  engine.resize();
});

new FinisherHeader({
  "count": 10,"size": {"min": 974, "max": 1261,"pulse": 0},
  "speed": {"x": {"min": 0.1,"max": 0.8}, "y": {"min": 0.1,"max": 0.8}},
  "colors": {"background": "#001dff", "particles": ["#00ffdd", "#284292", "#23dbb6"]},
  "blending": "overlay",
  "opacity": {"center": 0.5,"edge": 0.05},
  "skew": 0,
  "shapes": ["c"]
});
```

Figura 18: File JS - Gestione eventi dall'utente, aggiornamenti pagina e aggiunta script Background

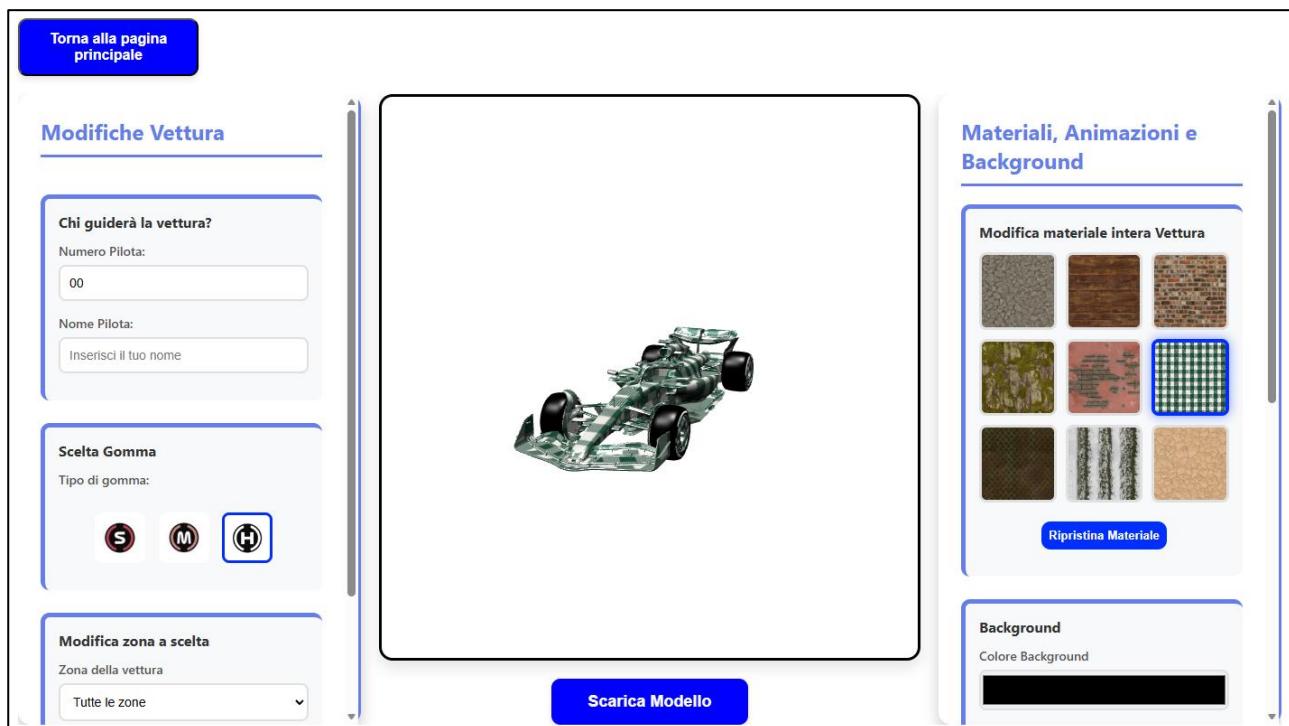
4.3 Pagina di Personalizzazione

Per la pagina di personalizzazione ho implementato i file styleModel.html, style2.css e i due script script2.js e personalizzazioni.js. Il file script2.js conterrà la gestione della visualizzazione del modello, mentre lo script personalizzazioni.js conterrà tutta la gestione di personalizzazioni sul modello.

4.3.1 Interfaccia Grafica

Per la pagina di personalizzazione, ho voluto fare qualcosa di estetico, inserendo:

- Pulsante “Torna Indietro”.
- Pannello Sinistro con:
 - Modificatori di base sulla vettura (Colorazione, Immagini, Gomma).
- Pannello Centrale con:
 - Rappresentazione tramite canvas del modello personalizzato.
 - Pulsante “Scarica Modello”.
- Pannello Destro con:
 - Materiali da applicare su tutto il modello.
 - Background da impostare
 - Animazioni da abilitare e disattivare



4.3.2 Codice

Per il codice mostrerò i codici più importanti da implementare, basandoci sul file html e i 2 script script2.js e personalizzazioni.js.

4.3.2.1 File HTML

4.3.2.1.1 Gestione Tipo Gomma

Per la gestione del Tipo di gomma che si può applicare ho utilizzato un button che all'evento click esegue la funzione cambiaTipoGomma() passando il tipo di gomma. Dentro al button ho inserito l'immagine della gomma rappresentativa.

```
<!--Selezione Gomma da impostare-->
<div class="sezione">
    <h3>Scelta Gomma</h3>
    <div class="campo">
        <label>Tipo di gomma:</label><br>
        <div class="immaginiGomme">
            <button class="logoGomma" onclick="cambiaTipoGomma('soft')">
                
            </button>
            <button class="logoGomma" onclick="cambiaTipoGomma('medium')">
                
            </button>
            <button class="logoGomma" onclick="cambiaTipoGomma('hard')">
                
            </button>
        </div>
    </div>
</div>
```

4.3.2.1.2 Pannello Centrale (canvas + download)

Per la gestione del modello, applico un contenitore con l'id contenitoreCanvas, inserendo al suo interno il canvas utilizzato dalla scena del modello personalizzato. Mentre per il download del modello implemento un button che all'evento click esegue la funzione downloadModello.

```
<div class="centro">
    <div id="contenitoreCanvas">
        <canvas id="canvasOggetto"></canvas>
    </div>
    <button class="download" onclick="downloadModello()">
        Scarica Modello
    </button>
</div>
```

4.3.2.1.3 Applicazione Materiale sul modello

Per la gestione di visualizzazione e funzionamento dell'applicazione dei materiali sul modello, implemento la stessa struttura del tipo gomma, dove inserisco i button con l'evento che richiama la funzione dagli script, e al suo interno immetto l'immagine rappresentativa del materiale da selezionare. Infine applico il button che va a richiamare la funzione ripristinaMateriale.

```
<div class="sezione">
    <h3>Modifica materiale intera Vettura</h3>
    <div class="campo">
        <div class="grigliaSfondi">
            <button class="buttonSfondo" onclick="cambiaMaterialeModello('../images/textures/materials/rocks.jpg')">
                
            </button>
            <button class="buttonSfondo" onclick="cambiaMaterialeModello('../images/textures/materials/wood.jpg')">
                
            </button>
            <button class="buttonSfondo" onclick="cambiaMaterialeModello('../images/textures/materials/brick.jpg')">
                
            </button>
            <button class="buttonSfondo" onclick="cambiaMaterialeModello('../images/textures/materials/aerial.jpg')">
                
            </button>
            <button class="buttonSfondo" onclick="cambiaMaterialeModello('../images/textures/materials/rebar_reinforced.jpg')">
                
            </button>
            <button class="buttonSfondo" onclick="cambiaMaterialeModello('../images/textures/materials/gingham.jpg')">
                
            </button>
            <button class="buttonSfondo" onclick="cambiaMaterialeModello('../images/textures/materials/metal_plate.jpg')">
                
            </button>
            <button class="buttonSfondo" onclick="cambiaMaterialeModello('../images/textures/materials/snow.jpg')">
                
            </button>
            <button class="buttonSfondo" onclick="cambiaMaterialeModello('../images/textures/materials/mud.jpg')">
                
            </button>
        </div><br>
        <div id="buttonAnimazioni" style="text-align: center;">
            <button onclick="ripristicaMateriale()>Ripristina Materiale</button>
        </div>
    </div>
</div>
```

4.3.2.1.4 Implementazion Script Esterini sul file HTML

In fondo alla pagina carico gli script che mi permettono di caricare ed eseguire i codici Javascript. I seguenti script hanno le diverse funzionalità:

- **cdnjs.cloudflare.com** → Libreria per creare, leggere e modificare file ZIP (per il download).
- **cdn.babylonjs.com/babylon.js** → Motore 3D WebGL, utilizzato per creare e gestire le scene 3D.
- **cdn.babylonjs.com/loaders/babylonjs.loaders.min.js** → Loader di Babylon.js che permette di caricare modelli 3D già definiti.
- **cdn.babylonjs.com/serializers/babylonjs.serializers.min.js** → Serializer Babylon.js, che serve a esportare le scene o i mesh in formati come GLB.
- **script2 e personalizzazioni** → sono gli script implementati da me che serviranno a gestire il modello e permettere di personalizzarlo.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jszip/3.10.1/jszip.min.js"></script>
<script src="https://cdn.babylonjs.com/babylon.js"></script>
<script src="https://cdn.babylonjs.com/loaders/babylonjs.loaders.min.js"></script>
<script src="https://cdn.babylonjs.com/serializers/babylonjs.serializers.min.js"></script>
<script src="..//script/script2.js"></script>
<script src="..//script/personalizzazioni.js"></script>
```

4.3.2.2 File script2

Nel file script2 implemento nuovamente le funzioni di gestione di visualizzazione modello, implementato e documentato nel capitolo del file script.js della pagina di selezione modello.

4.3.2.2.1 Gestione iniziale caricamento modello

L'utente può entrare liberamente nel sito di personalizzazione grazie all'URL, senza selezionare il modello. Per gestire questa problematica, all'inizio ho implementato un controllo che se il modello, ricavato dal localStorage, sia nullo, vuoto, allora esegue un alert e reindirizza l'utente alla pagina di selezione modello.

```
const canvasOggetto = document.getElementById("canvasOggetto");
const engine = new BABYLON.Engine(canvasOggetto, true);
const modello = localStorage.getItem("modelloSelezionato");

let scena;
let camera;
let currentModel = null;

if (!modello) {
    alert("Non hai selezionato nessun modello, cosa vuoi personalizzare?? Ritorna alla pagina principale...");
    window.location.href = "selectModel.html";
}
```

4.3.2.2.2 Colorazione Gomma al caricamento del modello

Nella funzione del caricaModello, nel parametro del callback onSuccess, ho inserito alla fine il codice che va colorare i mesh, le zone della gomma, in modo che l'utente appena apre la pagina di personalizzazione, visualizzerà le gomme del modello nel colore nero, così che è già impostato (non personalizzabile). Questo non influisce nel tipo di gomma, poiché quella è utilizzata per applicare nella zona “laterale” della gomma.

```
//Colora ruote
const ruoteDaColorare = ["Object_235", "Object_219", "Object_227", "Object_243"];
const materiale = new BABYLON.StandardMaterial("nero", scena);
materiale.diffuseColor = new BABYLON.Color3(0, 0, 0);

ruoteDaColorare.forEach(nomeRuota => {
    let meshRuota = scena.getMeshByName(nomeRuota);

    if (meshRuota) {
        meshRuota.material = materiale;
    }
});
```

4.3.2.2.3 Gestione pulsante torna alla pagina principale

Per gestire l'evento del button “torna indietro”, ho inserito un controllo del confirm, dove se l’utente seleziona “Ok”, allora nel localstorage viene eliminato l’informazione del nome del modello, reindirizzando l’utente alla pagina principale della selezione modello, mentre se seleziona “Annulla”, allora prosegue con la personalizzazione del modello già scelto precedentemente.

```
function tornaIndietro() {
    if(confirm("Stai per perdere il tuo lavoro, sei sicuro di eliminarlo??")){
        localStorage.removeItem("modelloSelezionato");
        window.location.href = "selectModel.html";
    }
}
```

4.3.2.3 File personalizzazioni

Nel file personalizzazioni.js, ho inserito tutte le funzioni che vanno a modificare e usare il modello in sé, includendo le funzioni come per la visualizzazione della zona selezionata, colorazione della zona selezionata, materiale, sfondo con il colore/immagine e animazioni varie.

4.3.2.3.1 Variabili iniziali

Implemento le variabili iniziali per lo sfondo, zona selezionata, animazioni e i valori degli input presi dal html.

```
//Gestione Background
let sfondoAttivo = null;
let tipoSfondo = 'colore';
let valoreSfondo = '#000000';

//Colora zona
let zonaSelezionata = "";
const coloraZona = document.querySelector('input[name="coloraZona"]');

//Selezione zone
const selectZonaVettura = document.getElementById("selectZonaVettura");

//Rotazione Vettura
let modelloGira = false;
let velocita = 0.009;

//Mostra wireframe
let wireframeAttivo = false;

//Rotazione ruote
let rotazioneRuote = false;
let funzioneRotazioneRuote = null;
let velocitaRuote = 0.05;

//Animazione Luce rossa posteriore
let luceAccesa = false;
let timerLampaggio = null;
let materialeRosso = null;
let lucePosteriore = null;

//Gomme
const gomme = ["Object_235", "Object_219", "Object_227", "Object_243", "Object_217", "Object_225",
    "Object_241", "Object_233"];
const tipoGomma = ["Object_217", "Object_225", "Object_241", "Object_233"];
let tipoDiGommaSelezionato = false;

//Informazioni pilota
const numeroPilota = document.querySelector('input[name="numeroPilota"]');
const nomePilota = document.querySelector('input[name="nomePilota"]');
```

4.3.2.3.2 Cambio colore sfondo

Per gestire lo sfondo, applico la funzione del cambiamento quando l'input color è stato modificato, ricavando il colore e il canvas. Poi per tutti gli elementi nella classe buttonSfondo, rimuove lo stato attivo, in modo che dal sito si vedrà che il background immagine verrà automaticamente deselezionata (serve per l'interfaccia grafica). Controlla con la variabile booleana che ci sia già uno sfondo attivo, così da rimuoverlo e colorando la scena, impostando le variabili che serviranno per altre funzioni di controlli/gestione.

```
function cambiaColoreSfondo() {
    const colore = document.getElementById("backgroundColorPicker").value;
    const canvas = document.getElementById("contenitoreCanvas");
    canvas.style.backgroundImage = "none";
    canvas.style.background = colore;
    document.querySelectorAll('.buttonSfondo').forEach(btn => btn.classList.remove('active'));

    if (sfondoAttivo) {
        sfondoAttivo.dispose();
        sfondoAttivo = null;
    }
    scena.clearColor = new BABYLON.Color4(rgb.r / 255, rgb.g / 255, rgb.b / 255, 1);
    tipoSfondo = 'colore';
    valoreSfondo = colore;
}
```

4.3.2.3.3 Cambio immagine sfondo

Per cambiare lo sfondo con un'immagine inizio applicando le nuove modifiche al canvas, per background image, size e position, rimuovo a tutti i button della classe buttonSfondo, lo stato attivo (per interfaccia grafica) ed eseguo un for che vado a ricavare l'immagine selezionata per cambiare lo stato, in attivo. Verifico se la variabile è true (presenza di un altro sfondo), eliminandolo e modificando la variabile in false. Implemento un nuovo layer che servirà ad applicarlarlo alla scena, in modo che si applica l'immagine come sfondo.

```
function cambiaSfondoImmagine(percorso) {
    const canvas = document.getElementById("contenitoreCanvas");

    canvas.style.backgroundImage = "url(" + percorso + ")";
    canvas.style.backgroundSize = "cover";
    canvas.style.backgroundPosition = "center";

    let bottoni = document.querySelectorAll(".buttonSfondo");

    for (var i = 0; i < bottoni.length; i++) {
        bottoni[i].classList.remove("active");
    }

    for (var i = 0; i < bottoni.length; i++) {
        let img = bottoni[i].querySelector("img");

        if (img && img.src.indexOf(percorso) !== -1) {
            bottoni[i].classList.add("active");
        }
    }

    if (sfondoAttivo !== null) {
        sfondoAttivo.dispose();
        sfondoAttivo = null;
    }

    let nuovoLayer = new BABYLON.Layer(
        "backgroundLayer",
        percorso,
        scena
    );
    nuovoLayer.isBackground = true;
    sfondoAttivo = nuovoLayer;
    tipoSfondo = "immagine";
    valoreSfondo = percorso;
}
```

4.3.2.3.4 Gestione selezione Gomma

Per gestire la selezione del tipo di gomma, ricavo i buttons, e per ognuno rimuovo lo stato active per l'interfaccia grafica, e attivo quella selezionata. Poi implemento un nuovo materiale e una texture di un'immagine esistente, dove alla texture scalo orizzontalmente di -1, la sposto orizzontalmente di 1, ruota la texture di 180 gradi e cambio la luminosità di 2.0, in modo da impostare la texture perfettamente al mesh. Faccio una serie di controlli, dove coloro il materiale in base al nome del tipo di gomma. Diffondo la texture sul materiale, e per ogni mesh dell'array gomma, applico il materiale.

```
function cambiaTipoGomma(tipo) {
    const tuttiBottoniGomma = document.querySelectorAll('.logoGomma');
    for (let i = 0; i < tuttiBottoniGomma.length; i++) {
        tuttiBottoniGomma[i].classList.remove('active');
    }
    event.currentTarget.classList.add('active');
    tipoDiGommaSelezionato = true;

    const mat = new BABYLON.StandardMaterial("gommaMat", scena);
    const tex = new BABYLON.Texture("../images/textures/wheel/wheel.png", scena);

    tex.uScale = -1;
    tex.uOffset = 1;
    tex.wAng = Math.PI;
    tex.level = 2.0;

    if (tipo === "soft") {
        mat.diffuseColor = new BABYLON.Color3(1, 0, 0);
    } else if (tipo === "medium") {
        mat.diffuseColor = new BABYLON.Color3(1, 1, 0);
    } else if (tipo === "hard") {
        mat.diffuseColor = new BABYLON.Color3(1, 1, 1);
    }

    mat.diffuseTexture = tex;

    for (let i = 0; i < tipoGomma.length; i++) {
        const mesh = scena.getMeshByName(tipoGomma[i]);
        if (mesh) {
            mesh.material = mat;
        }
    }
}
```

4.3.2.3.5 Animazione rotazione vettura + modifica velocità di rotazione

Per eseguire la funzione di rotazione, inverto lo stato dell'animazione (accendi/spegni), se è true crea una funzione dove aumenta nella camera, l'alpha (angolazione orizzontale), prendendo la funzione la eseguo per ogni frame. Mentre se è false la variabile dello stato dell'animazione, elimino l'evento e metto la funzione null. Per l'impostazione della velocità, ricavo la velcoita in float dal value del range, e se la rotazione e lo stato della funzione sono true, allora riesegue l'evento per ogni frame (in modo da continuarlo poi all'infinito).

```
function eseguiRotazioneVettura() {
    rotazione = !rotazione;
    if (rotazione) {
        funzioneRotazione = function () {
            camera.alpha += velocita;
        };
        scena.registerBeforeRender(funzioneRotazione);
    } else {
        if (funzioneRotazione) {
            scena.unregisterBeforeRender(funzioneRotazione);
            funzioneRotazione = null;
        }
    }
}

function velocitaRotazione() {
    velocita = parseFloat(target.value);

    if (rotazione && funzioneRotazione) {
        scena.unregisterBeforeRender(funzioneRotazione);
        funzioneRotazione = function () {
            camera.alpha += velocita;
        };
        scena.registerBeforeRender(funzioneRotazione);
    }
}
```

4.3.2.3.6 Evento quando si seleziona una zona del modello

Per gestire l'evento dell'input del select, ricavo la zona con value, e se non è vuoto richiamo la funzione coloraZonaScelta(zona), che entra in uno switch per ogni case (zona), viene dato il nome del mesh ed esegue la funzione visualizzaZona() con il nome passato.

```
selectZonaVettura.addEventListener("input", function () {
    const zona = this.value;
    if (zona !== "") {
        coloraZonaScelta(zona);
    }
});
```

Nella funzione visualizzaZona(), ricavo il nome del mesh, controllo che siano validi e che non hanno un materiale già impostato. Creo un nuovo materiale, che prendere il materiale del mesh (in modo da averlo come backup), creo un ulteriore materiale che lo colora di rosso. La funzione aspetta 1 secondo e rimette il materiale originale, aggiornando la variabile della zona selezionata, ritornando il mesh.

```
function visualizzaZona(nomeMesh) {
    const mesh = scena.getMeshByName(nomeMesh);
    if (!mesh || !mesh.material){
        return;
    }

    const materialeOriginale = mesh.material.clone(mesh.material.name + "_backup");

    const materialeRosso = new BABYLON.StandardMaterial("matRosso", scena);
    materialeRosso.diffuseColor = new BABYLON.Color3(1, 0, 0);

    mesh.material = materialeRosso;

    setTimeout(() => {
        mesh.material = materialeOriginale;
    }, 1000);

    zonaSelezionata = nomeMesh;
    return nomeMesh;
}
```

4.3.2.3.7 Visualizza tutte le zone

Per questa funzione l'ho applicata al case di tutte le zone, eseguendo il codice simile alla funzione visualizzaZona(), dove pero di diverso, metto tutte le mesh in un array che servirà ad applicare il materiale a tutte le mesh, per un secondo, e rimettendo il materiale originale. Infine ritorna il nome del mesh, in questo caso "All", tutti.

```
function visualizzaTutteLeZone() {
    const materialiOriginali = [];

    const materialeRosso = new BABYLON.StandardMaterial("matRossoAll", scena);
    materialeRosso.diffuseColor = new BABYLON.Color3(1, 0, 0);

    scena.meshes.forEach(mesh => {
        if (mesh.material) {
            if(!gomme.includes(mesh.name)){
                materialiOriginali.push({
                    mesh: mesh,
                    materiale: mesh.material
                });
            }

            mesh.material = materialeRosso;
        }
    });
}

setTimeout(() => {
    materialiOriginali.forEach(item => {
        item.mesh.material = item.materiale;
    });
}, 1000);

return zonaSelezionata = "All";
}
```

4.3.2.3.8 Evento cambio colore zona selezionata

Con la zona selezionata l'utente la puo colorare con l'input color. Per gestirlo implemento l'evento input, che richiama la funzione coloraZonaSceltaInput().

```
coloraZona.addEventListener('input', (e) => {
    coloraZonaSceltaDelInput(e.target.value);
});
```

Nella funzione coloraZonaSceltaInput(), controllo il modello non è valido, implemento un nuovo colore, preso dall'input, e faccio un controllo con all'interno dei controlli per il caso se fossero tutte le zone (controllo se è "All" il nome, per ogni mesh controllo ha il materiale, se non è incluso tra i mesh delle gomme e controllo il tipo di materiale). Infine se ha il nome, che non è "All", faccio una serie di controlli applicando il materiale e il colore, in caso non hanno il materiale.

```
function coloraZonaSceltaInput(color) {
    if (!modelloSelezionato || zonaSelezionata === "") {
        alert("Seleziona prima una zona dalla lista!");
        return;
    }

    const colore = BABYLON.Color3.FromHexString(color);

    if (zonaSelezionata === "All") {
        modelloSelezionato.forEach(mesh => {
            if (mesh.material) {
                if(!gomme.includes(mesh.name)){
                    if (mesh.material instanceof BABYLON.PBRMaterial) {
                        mesh.material.albedoColor = colore;
                    } else if (mesh.material instanceof BABYLON.StandardMaterial) {
                        mesh.material.diffuseColor = colore;
                    }
                }
            });
        });
        return;
    }

    const mesh = scena.getMeshByName(zonaSelezionata);
    if (!mesh) {
        console.error("Mesh non trovato:", zonaSelezionata);
        return;
    }

    if (!mesh.material) {
        mesh.material = new BABYLON.StandardMaterial("mat_" + zonaSelezionata, scena);
    }
    if (mesh.material.name === "matRosso" || mesh.material.name === "matRossoAll") {
        mesh.material = new BABYLON.StandardMaterial("mat_" + zonaSelezionata, scena);
    }

    if (mesh.material instanceof BABYLON.PBRMaterial) {
        mesh.material.albedoColor = colore;
    }
    else if (mesh.material instanceof BABYLON.StandardMaterial) {
        mesh.material.diffuseColor = colore;
    }
}
```

4.3.2.3.9 Animazione: Visualizza Wireframe

Per gestire la visualizzazione del WireFrame, controllo ogni mesh che il suo wireframe non sia definito, per poi attivarlo mettendogli true (variabile wireFrameAttivo).

```
function mostraWireframe() {  
    wireframeAttivo = !wireframeAttivo;  
    scena.materials.forEach(materiale => {  
        if (materiale.wireframe !== undefined) {  
            materiale.wireframe = wireframeAttivo;  
        }  
    });  
}
```

4.3.2.3.10 Dichiarazioni

La funzione del download, ho utilizzato l'intelligenza artificiale per gestirmi il download e la creazione dei poster. Questo perché non sono riuscito ad implementarlo correttamente, e ho riscontrato molte problematiche. Infatti questo download era richiesto di implementarlo, scritto nei requisiti, ma purtroppo non sono riuscito a gestirlo correttamente, e per questo ho utilizzato AI.

5 Test

5.1 Protocollo di test

Test Case:	TC-001	Nome:	Interfaccia Grafica di Selezione modello
Riferimento:	REQ-01		
Descrizione:	Verificare che l'utente possa visualizzare correttamente il titolo, la descrizione, la sezione di scelta modello, visualizzazione del modello scelto e il pulsante "Conferma" dopo che si ha scelto un modello.		
Prerequisiti:	Connessione internet per caricare il framework Babylon.js.		
Procedura:	<ol style="list-style-type: none"> 1. Aprire il sito web. 2. Verificare la presenza del titolo F1 Livery Style. 3. Verificare la presenza della lista dei modelli selezionabili tramite radio button. 		
Risultati attesi:	La visualizzazione degli elementi titolo, descrizione e sezione scelta modello è presente.		

Test Case:	TC-002	Nome:	Visualizzazione del modello scelto nella pagina di selezione
Riferimento:	REQ-02		
Descrizione:	Verificare che il modello selezionato dalla lista, venga visualizzato assieme alla rotazione automatica.		
Prerequisiti:	Pagina principale (selectModel.html) caricata e connessione internet attiva.		
Procedura:	<ol style="list-style-type: none"> 1. Selezionare un modello dalla lista tramite i radio button. 2. Osservare il canvas a destra e verificare se c'è la presenza del modello scelto. 3. Verificare che il modello esegue una rotazione automatica. 		
Risultati attesi:	Il modello è visibile ed esegue la rotazione automatica.		

Test Case:	TC-003	Nome:	Interfaccia Grafica di Personalizzazione
Riferimento:	REQ-03		
Descrizione:	Verificare che l'utente possa entrare nella pagina di personalizzazione dopo che ha scelto un modello dalla pagina principale, e visualizzare tutti gli elementi di personalizzazione.		
Prerequisiti:	Aver selezionato un modello dalla pagina principale Seleziona modello ed aver cliccato il pulsante apparso "Conferma".		
Procedura:	<ol style="list-style-type: none"> 1. Dalla pagina principale, scegliere un modello e cliccare il pulsante "Conferma". 2. Verificare che venga caricata la pagina di personalizzazione. 3. Verificare la presenza del modello selezionato nella pagina di selezione, senza texture, al centro della pagina. 4. Verificare la presenza del pulsante "Torna Indietro". 5. Verificare la presenza del pulsante "Esporta". 6. Verificare la presenza dei due pannelli laterali contenenti i strumenti di modifica. 7. Ripetere con i modelli restanti. 		
Risultati attesi:	La pagina viene caricata correttamente con tutti gli elementi definiti, ma a dipendenza dal modello selezionato, solo il modello 2024 funziona, mentre gli altri non carica il modello stabilito.		

Test Case:	TC-004	Nome:	Selezione zona specifica del modello
Riferimento:	REQ-04		
Descrizione:	Verificare che l'utente possa selezionare una zona specifica tramite select, dov'essa cambia colore per un tempo breve.		
Prerequisiti:	Avere la pagina di personalizzazione aperta assieme al modello scelto visualizzabile.		
Procedura:	<ol style="list-style-type: none"> Nel pannello laterale sinistro, nella sezione “Modifica zona a scelta” e selezionarne uno. Verificare che la zona viene colorata. Ripetere con le altre zone. 		
Risultati attesi:	Con la selezione di una zona, questa si colora per tempo breve correttamente.		

Test Case:	TC-005	Nome:	Colorazione zona selezionata
Riferimento:	REQ-04 REQ-05		
Descrizione:	Verificare che l'utente possa applicare materiali già pronti all'intera vettura.		
Prerequisiti:	Avere la pagina di personalizzazione aperta assieme al modello scelto visualizzabile e avere scelto una zona nel select che non sia “Seleziona una zona”.		
Procedura:	<ol style="list-style-type: none"> Con l'input colore, cambiare il colore. Verificare il nuovo colore della zona 		
Risultati attesi:	La zona viene colorata correttamente.		

Test Case:	TC-006	Nome:	Inserimento immagine zona selezionata
Riferimento:	REQ-04 REQ-06		
Descrizione:	Verificare che l'utente possa applicare materiali già pronti all'intera vettura.		
Prerequisiti:	Avere la pagina di personalizzazione aperta assieme al modello scelto visualizzabile e avere scelto una zona nel select che non sia “Seleziona una zona”.		
Procedura:	<ol style="list-style-type: none"> Con l'input file, inserire un'immagine. Verificare il nuovo materiale della zona 		
Risultati attesi:	La zona viene colorata correttamente.		

Test Case:	TC-007	Nome:	Applicazione materiali predefiniti all'intero modello
Riferimento:	REQ-07		
Descrizione:	Verificare che l'utente possa applicare materiali già pronti all'intera vettura.		
Prerequisiti:	Avere la pagina di personalizzazione aperta assieme al modello scelto visualizzabile.		
Procedura:	<ol style="list-style-type: none"> Nel pannello laterale destro, recarsi sulla Sezione “Materiali”, e selezionarne uno. Verificare che il materiale venga applicato all'intera vettura. Ripetere con i materiali restanti. 		
Risultati attesi:	L'applicazione del materiale sul modello viene applicato senza problemi.		

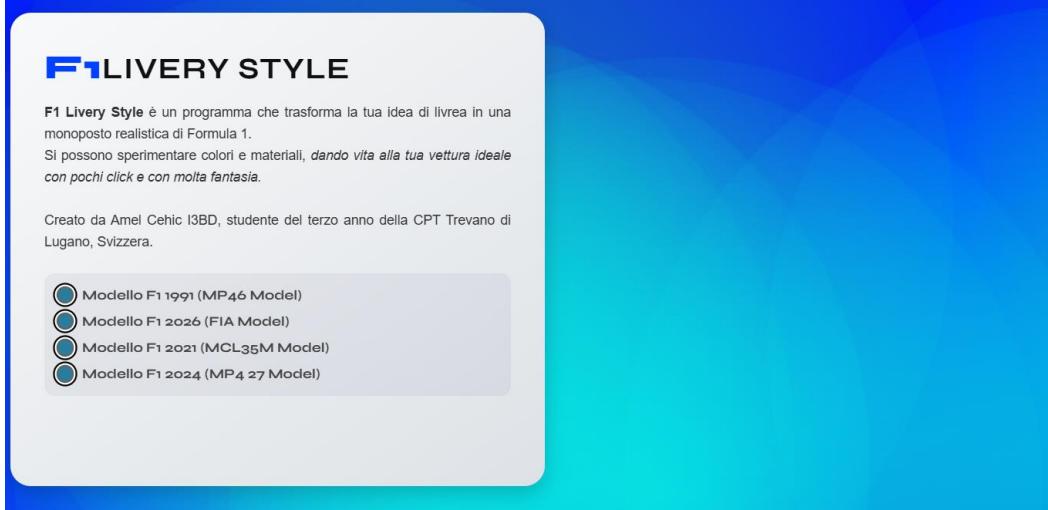
Test Case:	TC-008	Nome:	Esportazione del modello personalizzato
Riferimento:	REQ-08		
Descrizione:	Verificare che l'utente possa esportare il modello in formato GLB con immagini di anteprima.		
Prerequisiti:	Avere la pagina Personalizzazione con il modello funzionante aperta, aver inserito nome e numero pilota, scegliendo un tipo di gomma (se si vuole, selezionare un background in modo che nelle 3 immagini verrà perlustrato il modello con il background scelto)..		
Procedura:	<ol style="list-style-type: none"> Premere il pulsante al centro in basso, nominato “Scarica Modello”. Verificare che venga scaricata dal browser una cartella che contiene il modello GLB assieme a tre immagini del modello. Ripetere con un background scelto. 		
Risultati attesi:	Il download avviene correttamente.		

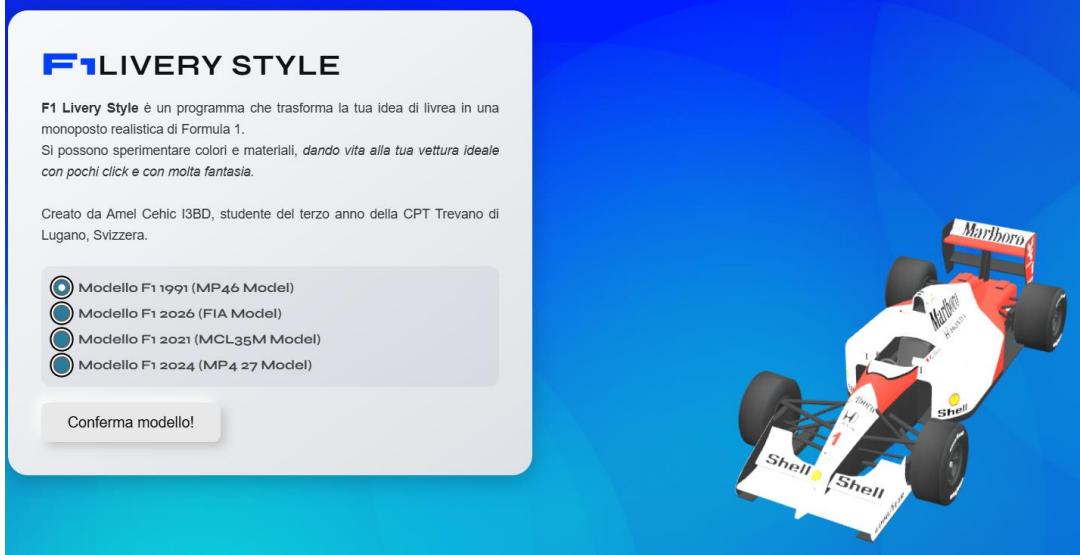
Test Case:	TC-009	Nome:	Modifica del background della scena
Riferimento:	REQ-09		
Descrizione:	Verificare che l'utente possa modificare il background cambiando il colore o selezionando un'immagine predefinita.		
Prerequisiti:	Pagina di personalizzazione aperta.		
Procedura:	<ol style="list-style-type: none"> Nel pannello laterale destro, nella sezione “Background”, usare l’input del colore Verificare che il background cambi colore Nella sezione delle immagini background, selezionare un’immagine Verificare che l’immagine venga applicata come sfondo. 		
Risultati attesi:	Il background cambia correttamente in base alla selezione.		

Test Case:	TC-010	Nome:	Selezione tipo di gomma
Riferimento:	REQ-10		
Descrizione:	Verificare che l'utente possa scegliere uno dei tre tipi di gomma e visualizzare la modifica sul modello.		
Prerequisiti:	Avere la pagina di personalizzazione aperta assieme al modello scelto visualizzabile.		
Procedura:	<ol style="list-style-type: none"> Nel pannello laterale sinistro, nella sezione “Scelta Gomme”, selezionare un tipo di gomma. Verificare che la zona laterale delle gomme del modello, cambi aspetto. Ripetere con gli altri tipi di gomme. 		
Risultati attesi:	Il tipo di gomma del modello viene modificato correttamente.		

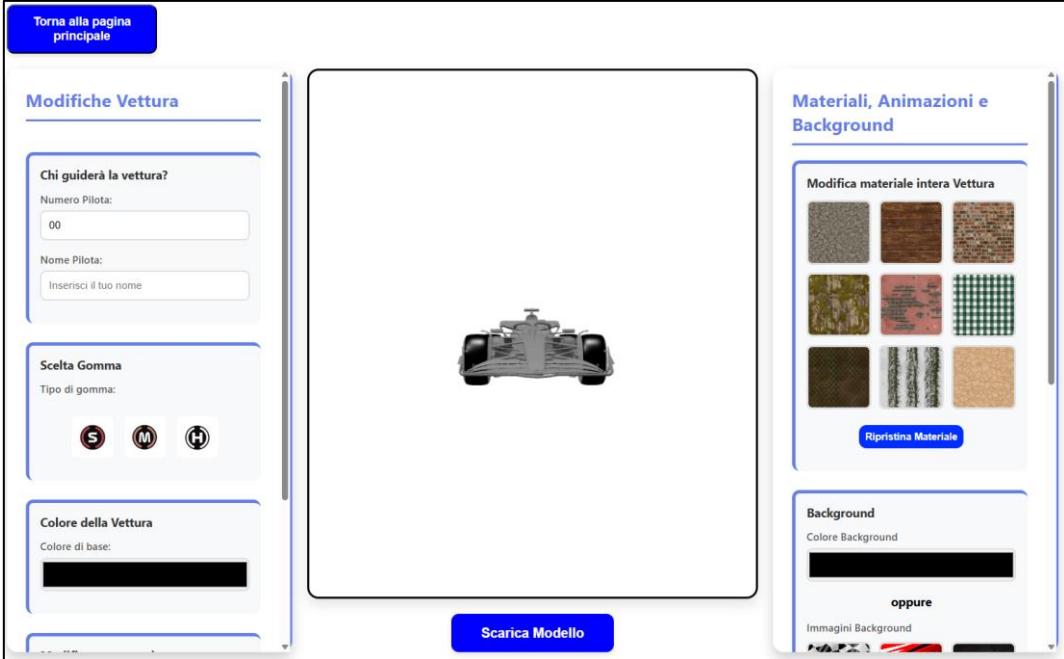
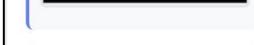
Test Case:	TC-011	Nome:	Attivazione animazioni sul modello
Riferimento:	REQ-11		
Descrizione:	Verificare che l'utente possa applicare animazioni predefinite sul modello		
Prerequisiti:	Avere la pagina di personalizzazione aperta assieme al modello scelto visualizzabile.		
Procedura:	<ol style="list-style-type: none"> Nel pannello laterale destro, nella sezione “Animazioni e Movimenti”, cliccare una delle tre animazioni (se è presente, con l'input range testare la velocità dell'animazione). Verificare che l'animazione venga eseguita. Cliccare nuovamente il pulsante dell'animazione. Verificare che l'animazione venga rimossa. Ripetere con le altre animazioni. 		
Risultati attesi:	Le animazioni vengono eseguite correttamente, e quando si preme nuovamente, vengono rimosse con successo.		

5.2 Risultati test

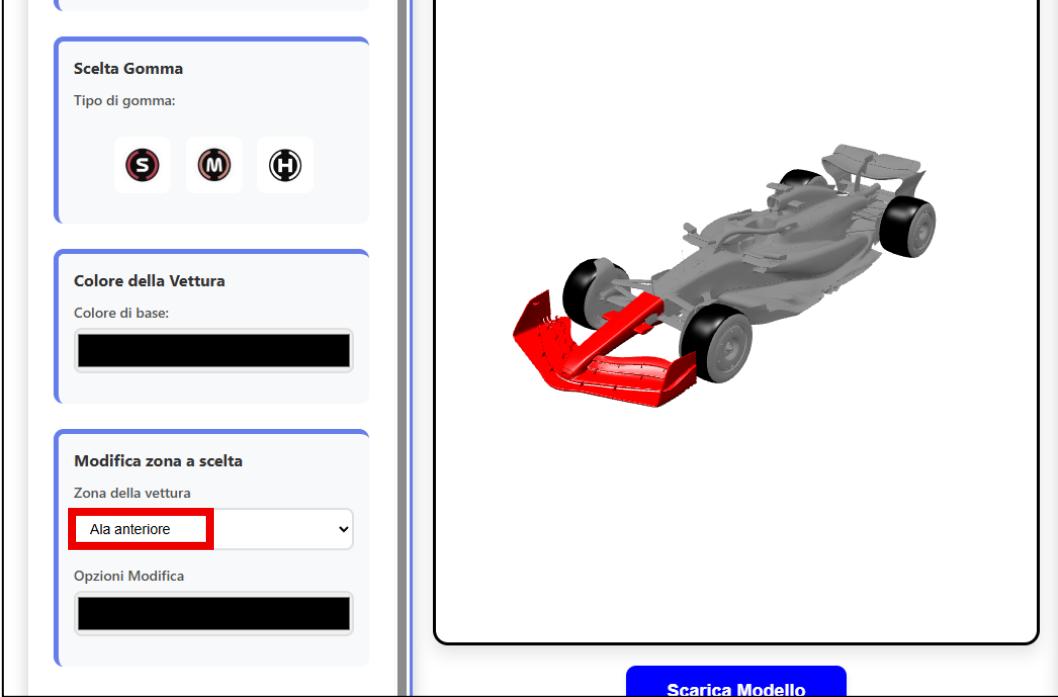
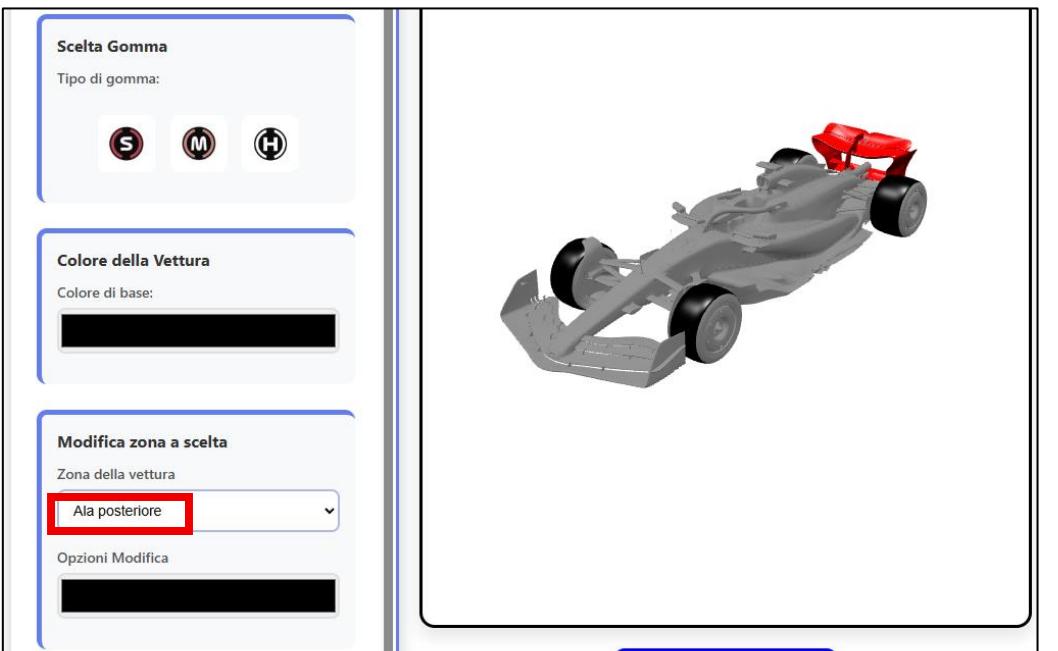
Test Case:	TC-001	Data:	19.12.2025
Esito:	PASSATO		
Risultato:	 <p>The screenshot shows the F1 Livery Style application window. At the top, it displays the title "F1 LIVERY STYLE". Below the title, there is a descriptive text: "F1 Livery Style è un programma che trasforma la tua idea di livrea in una monoposto realistica di Formula 1. Si possono sperimentare colori e materiali, dando vita alla tua vettura ideale con pochi click e con molta fantasia." Underneath this text, it says "Creato da Amel Cehic I3BD, studente del terzo anno della CPT Trevano di Lugano, Svizzera." A list of four model options is shown in a box: "Modello F1 1991 (MP46 Model)", "Modello F1 2026 (FIA Model)", "Modello F1 2021 (MCL35M Model)", and "Modello F1 2024 (MP4 27 Model)".</p> <p>La pagina mostra tutti gli elementi correttamente.</p>		

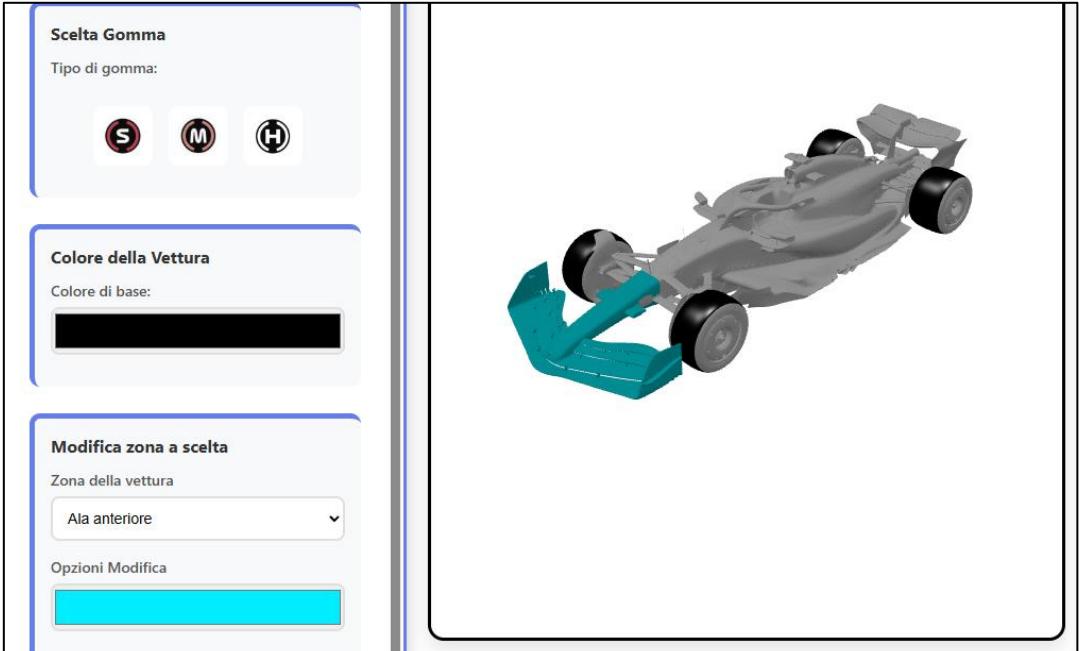
Test Case:	TC-002	Data:	19.12.2025
Esito:	PASSATO		
Risultato:	 <p>F1 LIVERY STYLE</p> <p>F1 Livery Style è un programma che trasforma la tua idea di livrea in una monoposto realistica di Formula 1. Si possono sperimentare colori e materiali, dando vita alla tua vettura ideale con pochi click e con molta fantasia.</p> <p>Creato da Amel Cehic I3BD, studente del terzo anno della CPT Trevano di Lugano, Svizzera.</p> <ul style="list-style-type: none"> <input checked="" type="radio"/> Modello F1 1991 (MP46 Model) <input checked="" type="radio"/> Modello F1 2026 (FIA Model) <input checked="" type="radio"/> Modello F1 2021 (MCL35M Model) <input checked="" type="radio"/> Modello F1 2024 (MP4 27 Model) <p>Conferma modello!</p>		

Il funzionamento del caricamento modello selezionato dall'utente funziona correttamente e il button "Conferma" viene mostrato dopo la selezione. Stessa cosa vale per gli altri modelli.

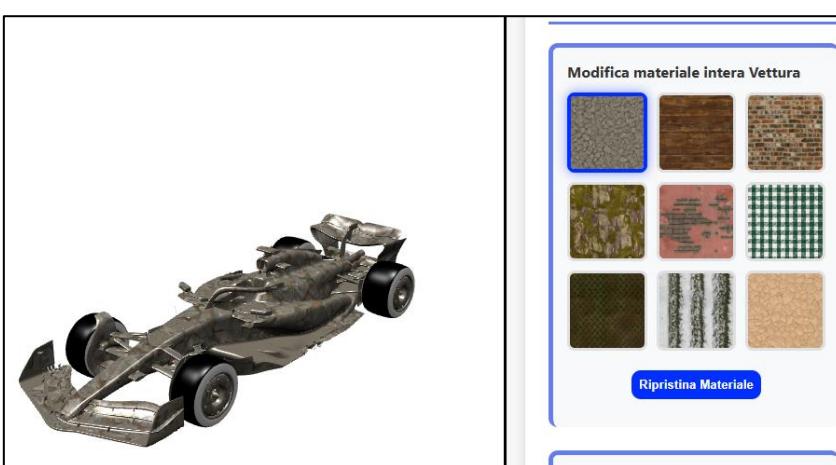
Test Case:	TC-003	Data:	19.12.2025
Esito:	PARZIALMENTE PASSATO		
Risultato:	 <p>Torna alla pagina principale</p> <p>Modifiche Vettura</p> <p>Chi guiderà la vettura?</p> <p>Numero Pilota: 00</p> <p>Nome Pilota: Inserisci il tuo nome</p> <p>Scelta Gomma</p> <p>Tipo di gomma: <input checked="" type="radio"/> S <input type="radio"/> M <input type="radio"/> H</p> <p>Colore della Vettura</p> <p>Colore di base: </p> <p>Scarica Modello</p> <p>Materiali, Animazioni e Background</p> <p>Modifica materiale intera Vettura</p> <p>Colore Background  oppure Immagini Background</p>		

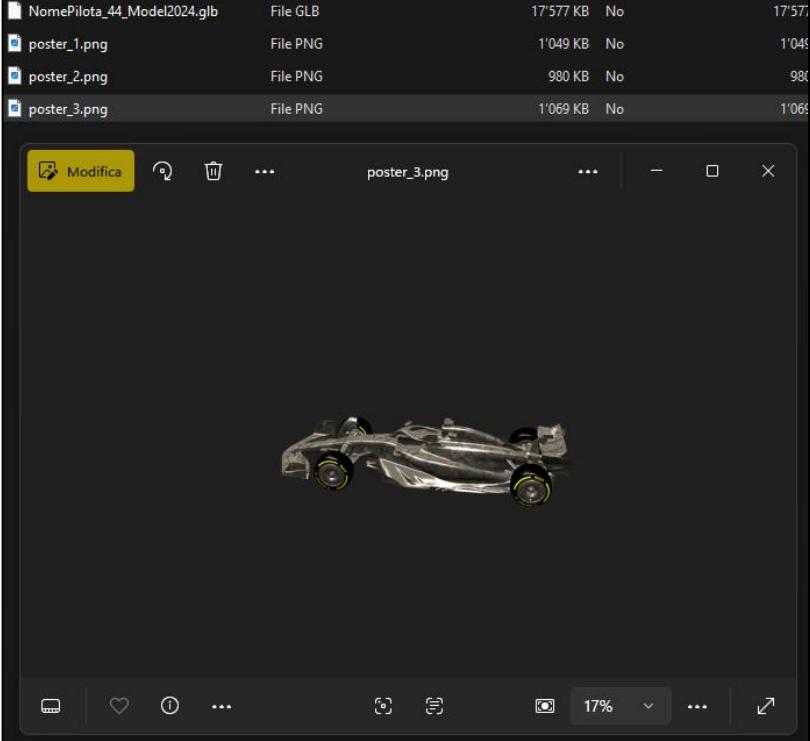
Questo test funziona parzialmente, poiché non presenta la visualizzazione del modello nel centro per diversi modelli. Non sono riuscito a fare la personalizzazione per i modelli 1991, 2026 e 2021, mentre il modello 2024 l'ho gestito e viene interpretato correttamente. Per il resto, tutti gli elementi vengono visualizzati correttamente.

Test Case:	TC-004	Data:	19.12.2025
Esito:	PASSATO		
			
Risultato:	<p>Quando si seleziona una zona, la zona nel modello viene colorata per un breve tempo, a cui serve ad indicare all'utente su che zona del modello sta lavorando.</p> 		

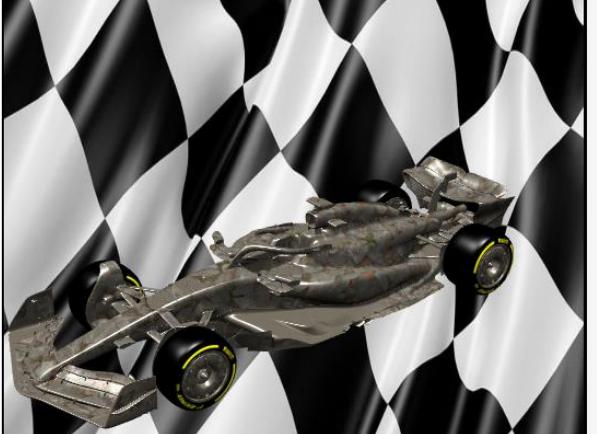
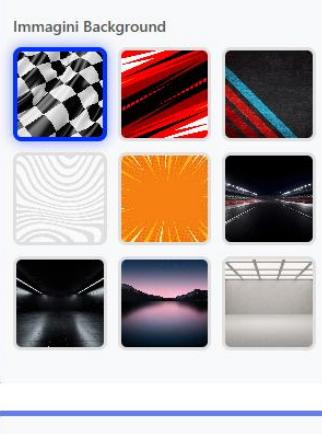
Test Case:	TC-005	Data:	19.12.2025
Esito:	PASSATO		
Risultato:	 <p>Selezionando una zona del modello, e impostando un colore all'input, la zona specifica si colora senza problemi.</p>		

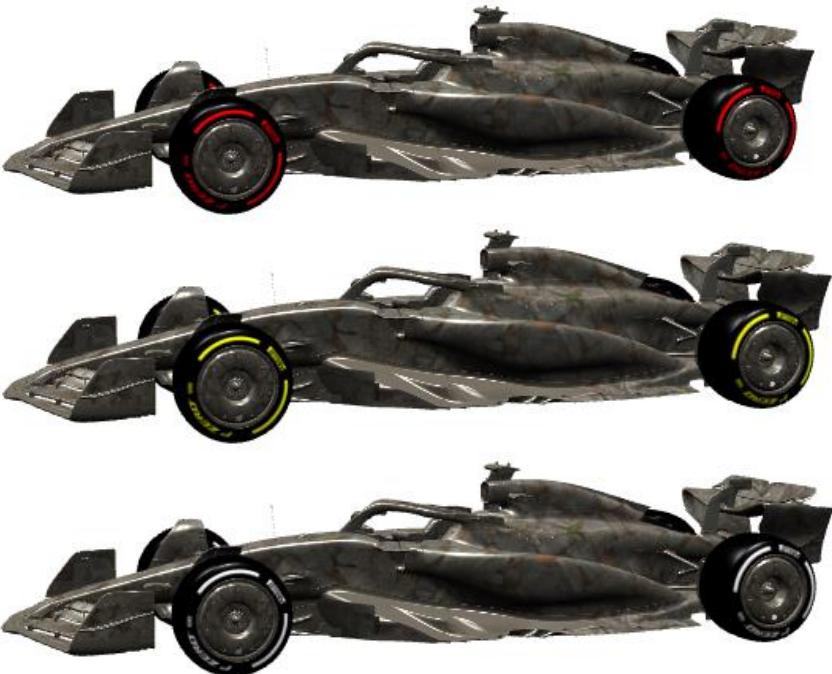
Test Case:	TC-006	Data:	19.12.2025
Esito:	NON PASSATO		
Risultato:	Purtroppo, il TC-006 non passa, poiché non ho gestito gli inserimenti di immagini sul modello.		

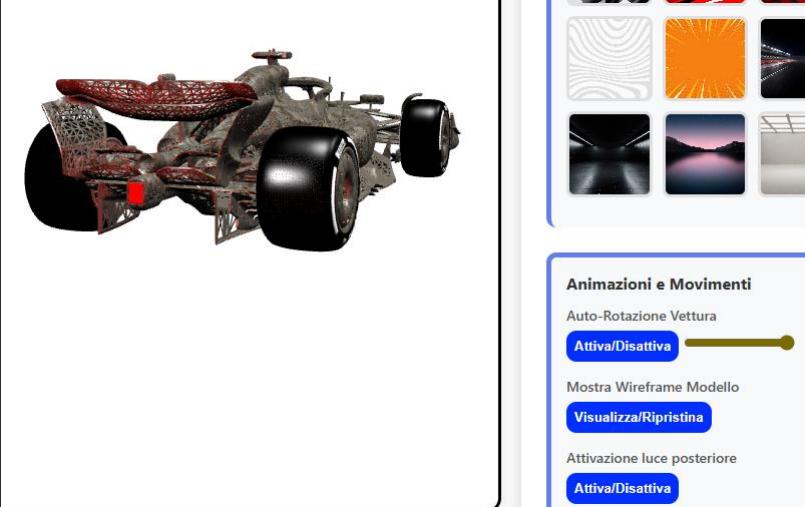
Test Case:	TC-007	Data:	19.12.2025
Esito:	PASSATO		
Risultato:	 <p>Scegliendo un materiale, al modello viene applicato correttamente il materiale.</p>		

Test Case:	TC-008	Data:	19.12.2025
Esito:	PASSATO		
Risultato:	 <p>NomePilota_44_Model2024.glb File GLB 17'577 KB No 17'577 poster_1.png File PNG 1'049 KB No 1'049 poster_2.png File PNG 980 KB No 980 poster_3.png File PNG 1'069 KB No 1'069</p> <p>Modifica ⌂ poster_3.png</p> <p>Risultato:</p> 		

Esportando il modello personalizzato, mi viene scaricato il modello GLB con 3 poster, e nell'immagine viene presentato il modello con diverse angolazioni per ogni immagine.

Test Case:	TC-009	Data:	19.12.2025
Esito:	PASSATO		
Risultato:	 		
	<p>Cambiando l'input color del colore background, il background imposta correttamente il colore. Stessa cosa vale per l'immagine scelta, viene impostata correttamente.</p>  		

Test Case:	TC-010	Data:	19.12.2025
Esito:	PASSATO		
Risultato:	 <p>Selezionando uno dei tre tipi di gomma, le gomme del modello avranno il tipo scelto dall'utente, impostati correttamente.</p>		

Test Case:	TC-011	Data:	19.12.2025
Esito:	PARZIALMENTE PASSATO		
Risultato:	 <p>Abilitando le funzioni, esse funzionano correttamente, sia per l'abilitazione che per lo spegnimento dell'animazione. Ma purtroppo non lo ritengo passato, poiché avevo stabilito di implementare l'animazione della rotazione delle gomme, che non sono riuscito a farlo.</p>		

5.3 Tabella Riassuntiva dei Test Case

Test Case	Esito
TC-001	PASSATO
TC-002	PASSATO
TC-003	PARZIALMENTE PASSATO
TC-004	PASSATO
TC-005	PASSATO
TC-006	NON PASSATO
TC-007	PASSATO
TC-008	PASSATO
TC-009	PASSATO
TC-010	PASSATO
TC-011	PARZIALMENTE PASSATO

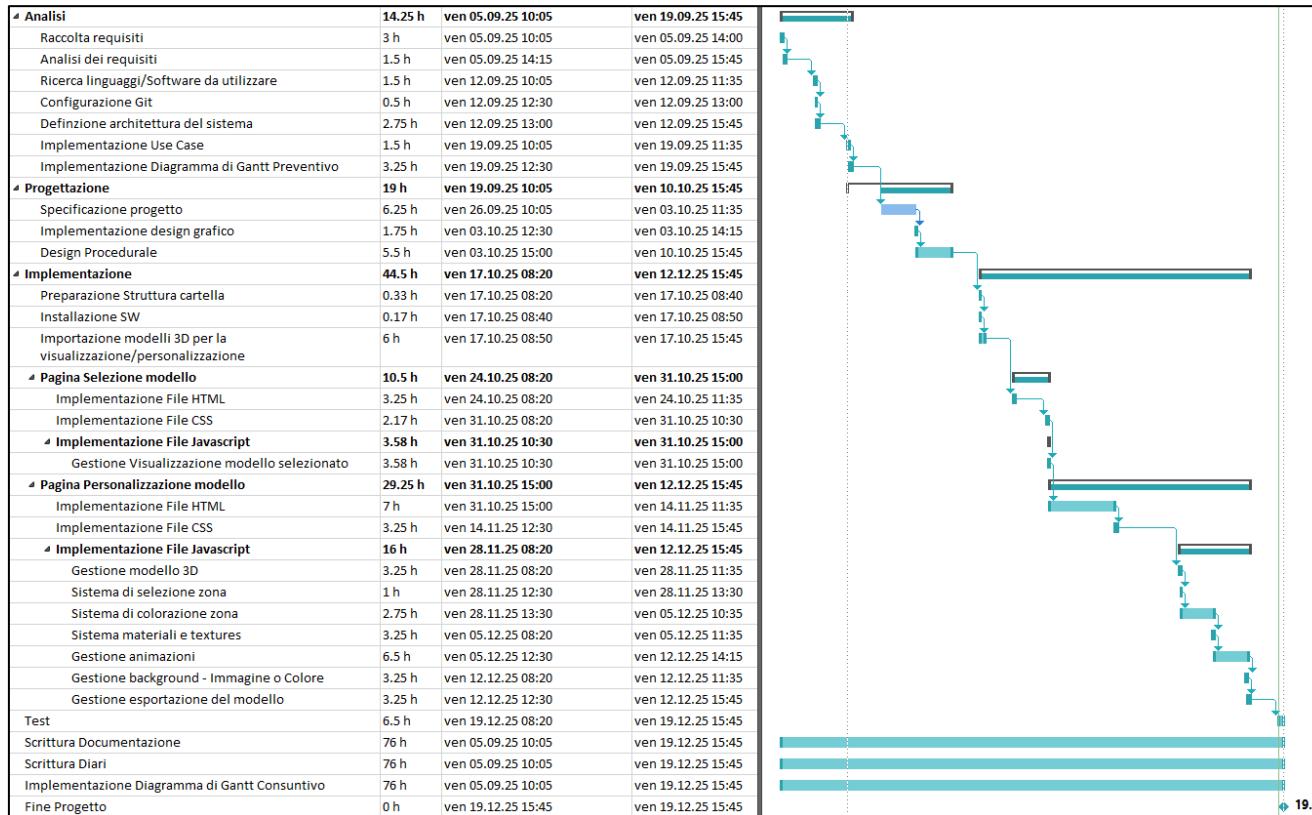
5.4 Mancanze/limitazioni conosciute

Una limitazione del prodotto è la lentezza di caricamento dei modelli nella pagina di selezione modello. Se l'utente rapidamente tutti i radio button, il caricamento del modello può corrompersi, arrivando a visualizzare un modello sopra a un altro.

Per quanto riguarda le mancanze, una delle principali è quella di gestire tutti i modelli, che permette all'utente di scegliere liberamente quale modello personalizzare. Questa mancanza è grave, perché limita l'esperienza all'utente e riduce il numero di scelte di funzionalità del progetto.

Un ulteriore mancanza, è quella della gestione dell'inserimento di immagini sulle zone specifiche, selezionate dall'utente. Per ora, quello che si può personalizzare su una zona specifica è solo la modifica del colore, limitando le possibilità di personalizzazione e creativa del prodotto.

6 Consuntivo



Il diagramma di Gantt Consuntivo rispetto a quello Preventivo, viene presentato diversamente, per le date, poiché ho avuto molte difficoltà nel comprendere come sviluppare dei concetti e non ho gestito i tempi necessari. Le fasi Analisi e Progettazione sono state rispettate, non ho avuto difficoltà e ritardi. Nella fase dell'Implementazione, i tempi sono diventati molto più elevati. I problemi che ho riscontrato erano la gestione di visualizzazione dei modelli tridimensionali, ho perso molto tempo nel capire cosa fosse il problema. Inoltre sono stato assente il 21.11.2025, non ho potuto continuare il progetto. Ho rinunciato all'inserimento delle immagini, poiché il tempo stava finendo. Questo perché perso tempo anche nello sviluppo delle pagine, concentrandomi nell'estetica che del funzionamento del prodotto. Con questo difetto, viene causato anche per i tempi non rispettati dei Test, che sono stati eseguiti l'ultima lezione.

7 Conclusioni

7.1 Sviluppi futuri

Tra le migliori che ritengo fondamentali, applicherei l'aggiunta delle immagini/texture personalizzate sul modello, per aumentare le possibilità di personalizzazione, rendendo il prodotto più "ricco" di funzionalità. Infine, sarebbe opportuno applicare l'animazione delle ruote stabilità all'inizio, per rendere il prodotto più realistico, insieme ad un sistema che salva il modello nel browser, così da evitare perdite delle modifiche elaborate durante la personalizzazione.

Mentre per le migliori ed estensioni opzionali, applicherei un ulteriore animazione nominata "Aerodinamica", nella quale viene simulato una passata d'aria leggermente bianca contro il modello, permettendo una visualizzazione del modello più realistica, analizzando il comportamento aerodinamico del modello.

Un altro possibile sviluppo futuro, legato agli sfondi e i materiali, è quello di permettere all'utente di aggiungere al prodotto la propria immagine da utilizzare come sfondo, o materiale, offrendo un numero maggiore di scelte per la personalizzazione.

Infine, un ulteriore sviluppo futuro, che ritengo complesso ma interessante, una visualizzazione del modello tramite dispositivo mobile. L'idea è quella di aggiungere un pulsante (ideale nella pagina di personalizzazione, ma sarebbe anche interessante inserirlo nella pagina di selezione per visualizzare il modello con le texture), che genera un codice QR, dove lo scannerizzi con il cellulare, così che potrai visualizzare il modello sul tuo dispositivo mobile e nello spazio reale con la fotocamera.

7.2 Considerazioni personali

Attraverso questo progetto, ho imparato ad utilizzare un nuovo framework di Javascript nominato Babylon.js, che mi ha dato la possibilità di acquisire nuove conoscenze nell'ambito tridimensionale. Principalmente, ho imparato come vengono gestiti e visualizzati i modelli tridimensionali nelle pagine web, approfondendo le mie competenze dei linguaggi web HTML, CSS e Javascript.

Prima d'ora, non avevo mai implementato un prodotto web idoneo all'interazione con modelli 3D assieme ai framework esterni. Durante le implementazioni, ho riscontrato problematiche, principalmente nella comprensione del come personalizzare zone specifiche di un modello e del caricamento di un modello in una pagina web. Sfortunatamente, questa problematica mi ha fatto prendere molto tempo, e non sono riuscito a gestire tutti i modelli stabiliti, solamente uno.

Nonostante tutto, sono soddisfatto del risultato finale e del mio impegno applicato. Sono riuscito a gestire un modello tridimensionale, raggiungendo un buon risultato.

8 Glossario

Termino	Descrizione
HTML	HyperTest Markup Language: linguaggio utilizzato per strutturare il contenuto di una pagina web
CSS	Cascading Style Sheets: linguaggio che permette di definire il layout e la grafica di una pagina web.
Javascript	Linguaggio di programmazione utilizzato per rendere pagine web dinamiche e interattive, gestendo eventi ed animazioni.
Mesh	Oggetto tridimensionale composto da vertici, facce e spigoli, utilizzato per rappresentare modelli 3D.
GLB	Formato di file per modelli 3D basato su glTF, che contiene geometria, materiali, texture e animazioni.
F1	Formula 1: sport automobilistico su pista, caratterizzato da monoposto ad alte prestazioni.
Babylon.js	Framework di Javascript che consente la creazione e la gestione di scene tridimensionali sul browser.
localStorage	Sistema di memorizzazione del browser di salvare dati localmente.
Wireframe	Schema grafico preliminare che rappresenta la struttura e la disposizione degli elementi di un modello.
Canvas	Tag html che consente rendering grafico dinamico, utilizzato per visualizzare scene 2D/3D.
Framework	Insieme di librerie e strumenti che facilitano lo sviluppo di applicazioni software fornendo funzionalità predefinite.
Texture	Immagine applicata alla superficie di un modello 3D per definire l'aspetto visivo
Scena	Ambiente tridimensionale che contiene tutti gli elementi 3D come modelli, luci e camere.

9 Bibliografia

9.1 Sitografia

- <https://xboxdesignlab.xbox.com/fr-ch/configure/xbox-elite-wireless-controller-series-2>,
Nuove idee di progetto, 19.09.2025
- <https://modelviewer.dev/editor/>,
Nuovi parametri da inserire nel progetto, 31.10.2025
- <https://chatgpt.com>,
Funzione creaPoster implementata con uso di AI, 28.11.2025
- https://doc.babylonjs.com/features/introductionToFeatures/chap1/first_app/,
Come si crea una scena Babylon, 31.10.2025
- <https://blender.stackexchange.com/questions/7160/can-i-delete-all-materials-of-all-objects-in-a-scene-quickly>,
Script per eliminare tutti i materiali di tutti mesh, 14.11.2025
- <https://www.finisher.co/lab/header/>
Generatore codice background, 31.10.2025
- <https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage>,
Funzionamento localStorage, 14.11.2025
- https://doc.babylonjs.com/features/featuresDeepDive/materials/using/materials_introduction,
Gestione Materiali, 14.11.2025
- <https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener>,
Gestione Eventi input, 03.09.2025
- <https://developer.mozilla.org/en-US/docs/Web/API/Window/setInterval>,
Gestione setInterval, 12.12.2025
- <https://playground.babylonjs.com/>,
Comprensione codice della scena, 10.09.2025
- https://developer.mozilla.org/en-US/docs/Web/API/File_API,
Gestione input file, 14.10.2025

10 Indice delle figure

Figura 1: Use Case	9
Figura 2: Diagramma di Gantt Preventivo	10
Figura 3: Design Pagina Principale	12
Figura 4: Design Pagina Personalizzazione	13
Figura 5: Diagramma di flusso	14
Figura 6: Struttura cartelle	15
Figura 7: Interfaccia Grafica - Pagina Selezione modello	15
Figura 8: File HTML - Head	16
Figura 9: File HTML – Body	17
Figura 10: File CSS - Parte 1	18
Figura 11: File CSS - Parte 2	19
Figura 12: File CSS - Parte 3	20
Figura 13: File CSS - Parte 4	21
Figura 14: File JS - Variabili Globali	22
Figura 15: File JS - Funzione creaScena()	23
Figura 16: File JS - Funzione impostaCamera()	24
Figura 17: File JS - Funzione caricaModello()	25
Figura 18: File JS - Gestione eventi dall'utente, aggiornamenti pagina e aggiunta script Background	26

11 Allegati

[Design](#)

[Diagrammi UseCase e Swimlane](#)

[Gantt Consuntivo](#)

[Gantt Preventivo](#)