SEVEN CLICKERS

You Wish We Click

7clickersgroup@gmail.com

Specifica Tecnica

Versione	0.0.8
Stato	In Sviluppo
Uso	Esterno
Approvazione$_g$	-
Redazione	Mirko Stella
Verifica$_g$	-
Distribuzione	*Seven Clickers* Prof. Vardanega Tullio Prof. Cardin Riccardo

Descrizione

Specifica Tecnica dell'architettura di progetto

Registro delle modifiche

Vers.	Data	Autore	Ruolo	Descrizione
0.0.8	15-05-23	Mirko Stella Giacomo Mason Gabriele Mantoan	Responsabile Verificatori	Stesura sezione StoreDiagram
0.0.7	13-05-23	Mirko Stella Giacomo Mason Gabriele Mantoan	Responsabile Verificatori	Stesura sezione InterfaceFeaturesDiagram
0.0.6	12-05-23	Mirko Stella Giacomo Mason Gabriele Mantoan	Responsabile Verificatori	Stesura sezione SidebarFeaturesDiagram
0.0.5	11-05-23	Mirko Stella Giacomo Mason Gabriele Mantoan	Responsabile Verificatori	Stesura sezione PlayerFeaturesDiagram
0.0.4	08-05-23	Mirko Stella Giacomo Mason Gabriele Mantoan	Responsabile Verificatori	Stesura sezione CartFeaturesDiagram
0.0.3	07-05-23	Mirko Stella Giacomo Mason Gabriele Mantoan	Responsabile Verificatori	Stesura sezione Design pattern architetturale determinato dalle tecnologie adottate
0.0.2	05-05-23	Mirko Stella Giacomo Mason Gabriele Mantoan	Responsabile Verificatori	Stesura sezione Elenco dei componenti
0.0.1	04-05-23	Mirko Stella Giacomo Mason Gabriele Mantoan	Responsabile Verificatori	Stesura sezione Introduzione
0.0.0	03-05-23	Mirko Stella Giacomo Mason Gabriele Mantoan	Responsabile Verificatori	Creazione struttura documento

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

1.1 Glossario

In questo documento sono state segnate con il pedice "g" tutte le parole che, secondo noi, necessitano di una spiegazione ulteriore per evitare eventuali ambiguità o incomprensioni.

La spiegazione di questi termini la si può trovare nel documento di *Glossario*.

1.2 Scopo del documento

Il documento *Specifica Tecnica* ha lo scopo di descrivere i componenti utilizzati e le scelte progettuali fatte per la realizzazione del prodotto.

Dopo aver fornito un elenco descrittivo dei componenti verranno spiegati nel dettaglio, utilizzando l'ausilio degli schemi UML, i seguenti punti di interesse:

- Design pattern architetturale determinato dalle tecnologie adottate;
- Architettura logica (connessioni e interazioni tra componenti);
- Architettura di deployment (l'allocazione di componenti nel sistema in esecuzione);
- Idiomi, pattern di livello più basso che architetturale;
- Ogni altro aspetto progettuale che valorizza o caratterizza il design utilizzato.

1.3 Suggerimenti per la comprensione del documento

Per comprendere al meglio l'architettura utilizzata è importante comprendere in primo luogo il design pattern architetturale derterminato dalle tecnologie adottate in quanto gran parte delle scelte fatte si basano su di esso.

Si suggerisce quindi di leggere la sezione 2.2 riguardante il design pattern architetturale determinato dalle tecnologie adottate prima di proseguire con le successive.

2 Descrizione dell'architettura

2.1 Elenco dei componenti

- **Slices**:
  - **CartSlice**: componente che permette la gestione dello stato che contiene i dati riguardanti il carrello;
  - **ProductsSlice**: componente che permette la gestione dello stato che contiene i dati riguardanti i prodotti presenti all'interno dell'ambiente 3D.
  - **PlayerSlice**:
  - **RayCasterSlice**:
  - **DecorationSlice**:
  - **SidebarSlice**:
- **Initial states**:
  - **CartInitialState**:
  - **ProductsInitialState**:
  - **PlayerInitialState**:
  - **RayCasterInitialState**:

  - **DecorationInitialState**:
  - **SidebarInitialState**:
- **Actions**:
  - **sidebar.toggleSidebarIsOpen**:
  - **rayCaster.setLastProductPointed**:
  - **rayCaster.toggleRayCasterEnabled**:
  - **cart.addItems**:
  - **cart.removeItem**:
  - **cart.removeAll**:
- **Model components**:
  - **Coordinate**:
  - **Camera**:
  - **Octree**:
  - **Vector3**:
  - **Capsule**:
  - **Player**:
  - **CartItem**:
- **UI React components**:
  - **UI**:
  - **Crosshair**:
  - **Cart**:
  - **PlayerPosition**:
  - **CartItem**:
  - **ProductUI**:
  - **ProductInteractionPrompt**:
  - **Sidebar**:
  - **ColorSelector**:
  - **SelectColorItem**:
  - **ProductDetails**:
- **3D React components**:
  - **Canvas**:
  - **Scene**:
  - **PointerLock**:
  - **Environment**:
  - **Map**:
  - **Player**:
  - **Lights**:
  - **Models**:
  - **Decorations**:
  - **RayCaster**:

2.2 Design pattern architetturale determinato dalle tecnologie adottate

2.2.1 Redux-Toolkit

I componenti che costituiscono l'architettura utilizzata seguono il pattern offerto dalla libreria Redux-Toolkit.

Redux-Toolkit è pensato per integrarsi con React e il principale vantaggio che offre è quello di poter gestire i dati condivisi tra i componenti React in modo centralizzato semplificando la gestione dello stato globale dell'applicazione (in alternativa ogni componente React dovrebbe passare il proprio stato tramite props ai suoi diretti discendenti).

I componenti che formano l'architettura di Redux-Toolkit sono:

- **Store**: componente che contiene lo stato globale dell'applicazione.

  All'avvio dell'applicazione viene configurato utilizzando RootReducer e i componenti che utilizzano lo stato globale fanno il subscribe allo *store* in modo da venire renderizzati ogni volta che un dato di interesse cambia valore. Questo modo di operare può essere visto come un pattern *Observer* in cui lo *store* è il *Subject* e gli *Observers* sono i componenti React che hanno fatto il subscribe allo *store*;

- **RootReducer**: componente utilizzato per configurare lo store combinando più slice;

- **Slice**: componente che contiene un proprio stato che rappresenta una porzione dello stato globale dell'applicazione, i *reducer* che operano sullo stato e i *selector* per consentire ai suoi client il reperimento dei dati.

  Per definire una *slice* è buona norma raggruppare i dati in modo che siano legati da un sottoinsieme di funzionalità offerte dal sistema che lavorano su dati comuni;

- **Reducer**: componente che riceve come parametri uno stato iniziale (*InitialState*) e una *action* (composta da un type e un payload) e restituisce lo stato dopo aver operato sui dati.

  React-Toolkit gestisce le chiamate ai *reducer* quindi i dispatch delle *action* avvengono specificando solamente l'oggetto che rappresenta il payload;

- **Actions**: oggetto composto da un type e da un payload di cui viene effettuato il dispatch quando opportuno.

  Il payload è un oggetto che contiene i dati da passare al *reducer* che catturerà l'*action*;

- **InitialState**: componente che contiene i dati di una *slice* su cui essa opera.

  Importante precisare che Redux-Toolkit utilizzando la libreria immer gestisce anche l'immutabilità dei dati in modo che i reducer restituiscano delle copie dello stato in modo che esso non possa venire modificato dall'esterno e utilizzato in modo improrio.

  L'unico modo per modificare i dati dello stato globale è quindi con il dispatch di un'*action*;

- **Selector**: funzione che prende lo stato corrente di una *slice* come argomento e ritorna un sottoinsieme specifico del suo stato. In altre parole, un *selector* consente di 'selezionare' una parte specifica dello stato in modo da poterla utilizzare in modo isolato all'interno di un componente React.

2.2.2 React-three-fiber

Questa libreria fornisce un 'punto d'incontro' tra React (libreria javascript per la creazione di interfacce utente) e Three.js (libreria usata per la modellazione dell'ambiente 3D) semplificando la creazione dei componenti da inserire all'interno dell'ambiente 3D.

React-three-fiber rende la scrittura del codice dichiarativa creando dei componenti React 'preconfezionati' che rappresentano i componenti 3D. Questi componenti sono personalizzabili modificandone le caratteristiche tramite le props di React.

Un esempio è il componente *Canvas* che fornisce con un'unica dichiarazione la *scene* e la *camera* con una configurazione standard adatta alla maggior parte dei casi di utilizzo. Per inserire i componenti all'interno dell'ambiente è sufficiente dichiararli come figli del componente *scene*.

2.3 Architettura logica

Per facilitare la lettura dei diagrammi delle classi è stato scelto di organizzarli per feature in modo che ogni diagramma rappresenti i componenti che permettono l'implementazione di funzionalita' specifiche. Sono presenti dei diagrammi che non seguono questa convenzione che sono utili per avere una visione generale sulle dipendenze di alcuni componenti.

I diagrammi prodotti che rappresentano funzionalità specifiche sono:

- **CartFeaturesDiagram:** include i componenti che svolgono le funzioni riguardanti il carrello;
- **PlayerFeaturesDiagram:** include i componenti che svolgono le funzioni riguardanti le interazioni dell'utente con l'ambiente 3D;
- **ProductSidebarFeaturesDiagram:** include i componenti che svolgono le funzioni riguardanti la side-bar;
- **InterfaceFeaturesDiagram:** include i componenti necessari per il corretto aggiornamento dell'interfaccia utente;

I diagrammi prodotti che forniscono una visione generale delle dipendenze tra componenti sono:

- **StoreDiagram:** include lo *store*;
- **ReactComponentsHierarchy:** include i componenti React che rappresentano l'interfaccia utente e la scena 3D.

2.3.1 CartFeaturesDiagram

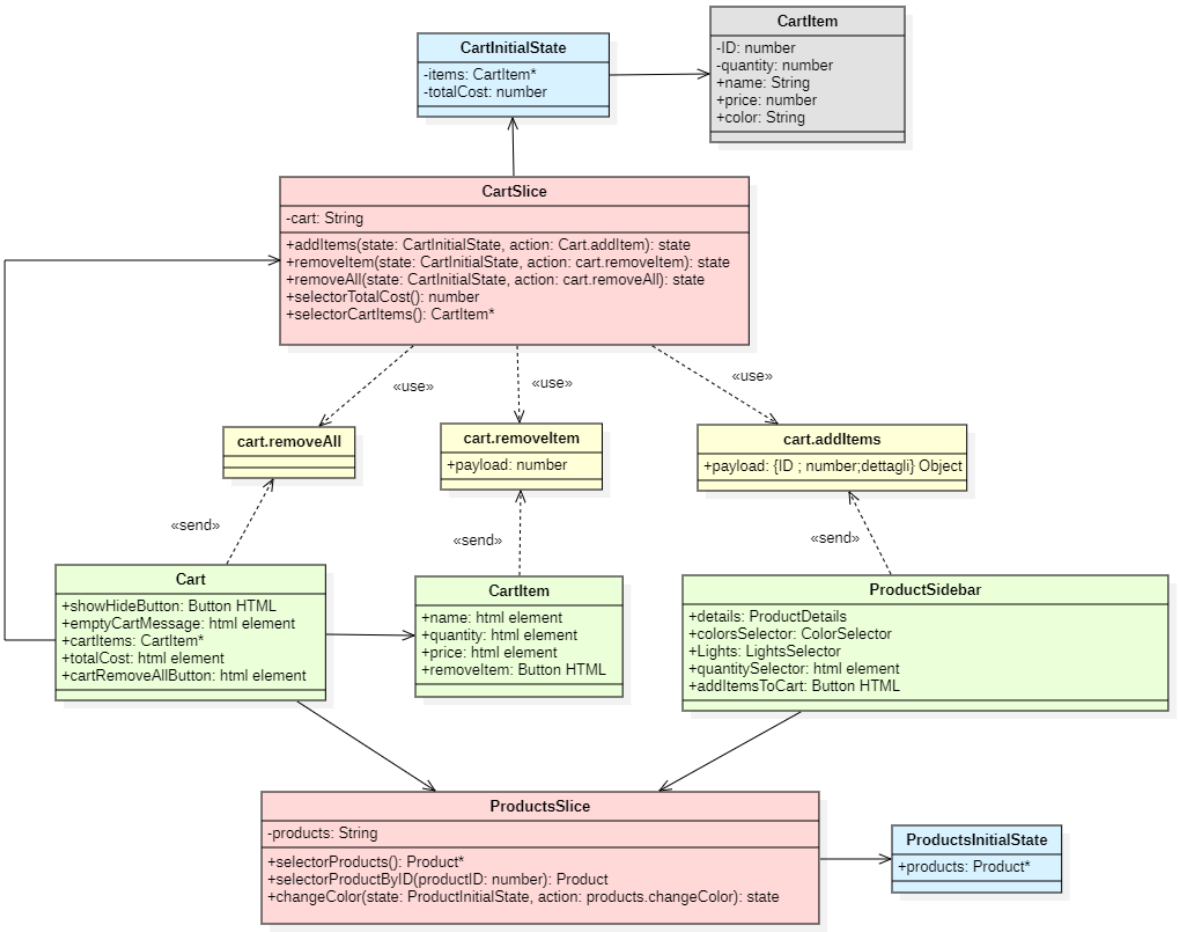

Figura 1: UML delle classi CartFeaturesDiagram.
Legenda: [*Slices*: rosso] - [*Actions*: giallo] - [*Model classes*: grigio] - [*Initial states*: azzurro] - [*UI React components*: verde]

Descrizione del diagramma: CartFeaturesDiagram contiene i componenti coinvolti nelle interazioni con il carrello.

- **CartSlice**
  - *Connessioni*:
  - *Interazioni*:
- **ProductsSlice**
  - *Connessioni*:
  - *Interazioni*:
- **CartInitialState**
  - *Connessioni*:
  - *Interazioni*:

- **ProductsInitialState**
  - *Connessioni*:
  - *Interazioni*:
- **cart.removeAll**
  - *Connessioni*:
  - *Interazioni*:
- **cart.removeItem**
  - *Connessioni*:
  - *Interazioni*:
- **cart.addItems**
  - *Connessioni*:
  - *Interazioni*:
- **CartItem**
  - *Connessioni*:
  - *Interazioni*:
- **Cart**
  - *Connessioni*:
  - *Interazioni*:
- **CartItem**
  - *Connessioni*:
  - *Interazioni*:
- **ProductSidebar**
  - *Connessioni*:
  - *Interazioni*:

2.3.2 PlayerFeaturesDiagram

2.3.3 SidebarFeaturesDiagram

2.3.4 InterfaceFeaturesDiagram

2.3.5 StoreDiagram

2.4 Architettura di deployment

2.5 Idiomi e pattern di livello più basso

2.6 Altri aspetti di design

2.7 Riferimenti

2.7.1 Riferimenti normativi

- Capitolato d'appalto C6: https://www.math.unipd.it/~tullio/IS-1/2022/Progetto/C6.pdf;

- *Norme di Progetto*;
- *Verbale esterno 17/11/22*;
- *Verbale interno 01/12/22*;
- *Verbale interno 07/12/22*;
- *Verbale esterno 11/01/23*;
- *Verbale esterno 18/01/23*;
- *Verbale esterno 17/02/23*;
- *Verbale interno 24/02/23*.

2.7.2 Riferimenti informativi

- Presentazione del capitolato: https://www.math.unipd.it/~tullio/IS-1/2022/Progetto/C6.pdf;
- Three.Js - Riferimenti:
  - Fondamenti di Three.js: https://threejs.org/manual/#en/fundamentals;
  - Documentazione Three.js: https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene;
  - Repository$_g$ informativa Three.js: https://github.com/mrdoob/three.js/.