



Informatica e Telecomunicazioni, articolazione Informatica

Progetto e sviluppo di un'applicazione Web per la gestione delle prenotazioni in ambito ristorativo: front-end.

Il candidato metta in evidenza gli aspetti inerenti alle discipline di Informatica, Sistemi e Reti ed eventualmente collegamenti con altre discipline o con l'esperienza di PCTO

Candidato: Enrico Dalla Croce

Classe: 5BII

Anno Scolastico 2020/2021

Introduzione

All'istituto Alessandro Rossi da alcuni anni gli studenti possono acquistare la merenda prima delle lezioni o durante l'intervallo presso i punti prestabiliti. Gli alimenti vengono preparati quotidianamente e poi venduti sino ad esaurimento. Ogni giorno si assiste al caos degli studenti che freneticamente cercano di comperare la merenda. Questo servizio è molto utilizzato dagli alunni, ma con l'arrivo dell'emergenza COVID-19 ha dovuto subire delle modifiche.

Ora che gli assembramenti sono da evitare la priorità è impedire l'affollamento nei punti vendita e lo spostamento di eccessive persone nella scuola.

Si è quindi deciso di adottare un sistema di ordinazione cartaceo.

Ogni mattina gli studenti segnano su un foglio cosa desiderano e il rappresentante di classe consegna l'ordine ai gestori del bar. All'intervallo un incaricato ritira l'ordine e lo distribuisce ai compagni.

Per migliorare ulteriormente questo servizio è stato chiesto agli studenti dell'indirizzo informatico di progettare e realizzare un sistema telematico per le ordinazioni.

L'obiettivo è quello di evitare l'uso della carta, gli spostamenti nella scuola e digitalizzare il processo rendendolo più moderno e comodo per tutti gli stakeholders.

Durante la scrittura del seguente documento si è assunto che il lettore abbia un bagaglio base di conoscenze informatiche, perciò alcuni concetti sono dati come assunti e non vengono spiegati.

Introduction

At the Alessandro Rossi Institute, for some years, students have been able to buy snacks before lessons or during the break at the established points. The snacks are prepared daily and then sold until they are out of stock. Every day we witness the chaos of students frantically trying to buy a snack. This service is widely used by students, but with the arrival of the COVID-19 emergency it has undergone changes.

Now that the crowds have to be avoided, the priority is to prevent crowding in the stores and the movement of excessive people in the school.

It was so decided to adopt a paper ordering system.

Each morning the students cross what they want on a piece of paper and the class representative delivers the order to the bar managers. At the break, a person in charge withdraws the order and distributes it to his teammates.

To further improve this service, the students of the IT department were asked to design and

implement a telematic system for ordering.

The goal is to avoid the use of paper, movements around the school and digitalize the process making it more modern and convenient for all stakeholders.

During the writing of the following document it has been assumed that the reader has a basic IT knowledge base, so some concepts are taken for granted and are not explained.

Indice

- Introduzione
- Introduction
- Indice
- Analisi
 - Richiesta dell' applicativo
 - Protocollo Coronavirus (COVID-19)
 - Discussione tecnica
 - Considerazioni aggiuntive
- Progettazione
 - Considerazioni generali
 - Dipendenze
 - Interfaccia | UI
 - Palette
 - Pagine & flow
 - Linguaggi di programmazione
 - Javascript
 - Panoramica del linguaggio
 - Browser Event Loop
 - Asincronicità in un singolo Thread
 - Async | Await
 - TypeScript
 - Panoramica del linguaggio
 - Vantaggi
 - React JSX | JavaScript XML
 - Panoramica del linguaggio
 - Frameworks
 - Next.js
 - Il framework React per la produzione
 - React
 - Vantaggi e Concetti fondamentali
 - Gestione pacchetti
 - npm
 - yarn
 - Pacchetti | Librerie
 - clsx

- notistack
- DevExtreme Reactive React Chart
- Login
- API
 - Idempotenza
- JWT
 - Panoramica sul JSON Web Token
 - Autenticazione tramite JWT
 - Utilizzo nel nostro applicativo
 - Offuscamento tramite WebAssembly
- WebSockets
 - Panoramica sui WebSockets
 - Utilizzo
- Implementazione
 - Codice
 - Funzioni
 - Arrow Functions
 - Const Functions
 - Snippets di codice rilevante
 - Versionamento | Sviluppo collaborativo | Git | Github
- Deploy
- Testing
 - Performance
- Manutenzione
 - Considerazioni aggiuntive
 - Autenticazione del sito
 - EditorConfig
 - Internazionalizzazione
 - Il debito tecnico
 - Debito nel nostro progetto
- Annotazioni^[1]

Analisi

Richiesta dell' applicativo

All'inizio dell'anno scolastico i rappresentanti degli studenti hanno chiesto alle classi di informatica la disponibilità a sviluppare un applicativo in grado di sostituire l'attuale sistema cartaceo.

Io e il mio compagno Francesco Marchetto abbiamo accettato l'incarico.

ITIS A. ROSSI - VICENZA		PRENOTAZIONI PANINI E BIBITE DEL GIORNO ____/____/____					
CLASSE ____		N. AULA ____		PIANO ____		I° INTERVALLO	
						II° INTERVALLO	
PANINI	PREZZO						TOTALE
NUTELLA	€1.00						
SALAME	€1.30						
PROSCIUTTO COTTO	€1.30						
PANCETTA	€1.30						
MORTADELLA	€1.30						
FORMAGGIO	€1.30						
PORCHETTA FUNGHI	€1.50						
COTTO FUNGHI MAIONESE	€1.50						
TONNO INSALATA MAIONESE	€1.50						
VEGETARIANO	€1.50						
HOT DOG MAIONESE	€1.50						
HOT DOG KETCHUP	€1.50						
HOT DOG MAIONESE KETCHUP	€1.50						
PIZZA BARCHETTA	€2.00						
BIBITE	PREZZO						
COCA-COLA	€0.80						
PEPSI TWIST	€0.80						
THE LIMONE	€0.80						
THE PESCA	€0.80						
THE VERDE	€0.80						
ACQUA NATURALE	€0.40						
ACQUA GASSATA	€0.40						
SUCCHI BRICK PESCA	€0.40						
SUCCHI BRICK PERA	€0.40						
SUCCHI BRICK ALBICOCCA	€0.40						

Attuale sistema per le ordinazioni.

Protocollo Coronavirus (COVID-19)

L'attuale sistema è in contrasto con le indicazioni del protocollo COVID che vieta gli assembramenti. Gli studenti si affollano all'ora prestabilita per consegnare le ordinazioni. Avviene inoltre uno scambio continuo di denaro, sia in classe che con i gestori.

Discussione tecnica

Dopo aver accolto la richiesta abbiamo discusso con il professore A.Viggi, referente per le reti informatiche, riguardo i mezzi a nostra disposizione e i vincoli imposti. I vincoli sono rappresentati dalla soluzione hosting fornitaci dalla scuola. La macchina che ci

è stata affidata è un QNAP (<https://www.qnap.com/en/product/ts-431>) in disuso. Il modello è obsoleto e pertanto ha limitata capacità computazionale e non supporta tecnologie moderne.

Considerazioni aggiuntive

Abbiamo fin da subito valutato la possibilità di utilizzare sistemi di pagamento online come Shopify (<https://it.shopify.com/>) o Stripe (<https://stripe.com/it>).

Ci è stato però segnalato un problema di responsabilità riguardo i pagamenti elettronici online e non da parte di minorenni.

A causa della minore età della maggior parte degli studenti non è stato possibile implementare questa funzione.

Progettazione

Considerazioni generali

Dipendenze

L'obiettivo che mi sono prefissato è di utilizzare il minor numero di librerie possibile. I pacchetti Node.js sono ricchi di dipendenze per la loro tendenza a creare librerie per le più piccole necessità. Avere molte dipendenze può portare a gravi conseguenze in futuro. L'obiettivo è quello di mantenere una bassa complessità per rendere lo sviluppo più semplice e accessibile a nuovi sviluppatori. Inoltre bisogna assicurarsi che le librerie scelte siano conosciute e supportate. Questo per evitare di ritrovarsi librerie che non verranno ulteriormente sviluppate o breaking updates.

Allo stesso tempo però si deve porre attenzione a non “reinventare la ruota”. È quindi necessario bilanciare bene dipendenze e comodità.

Interfaccia | UI

Palette



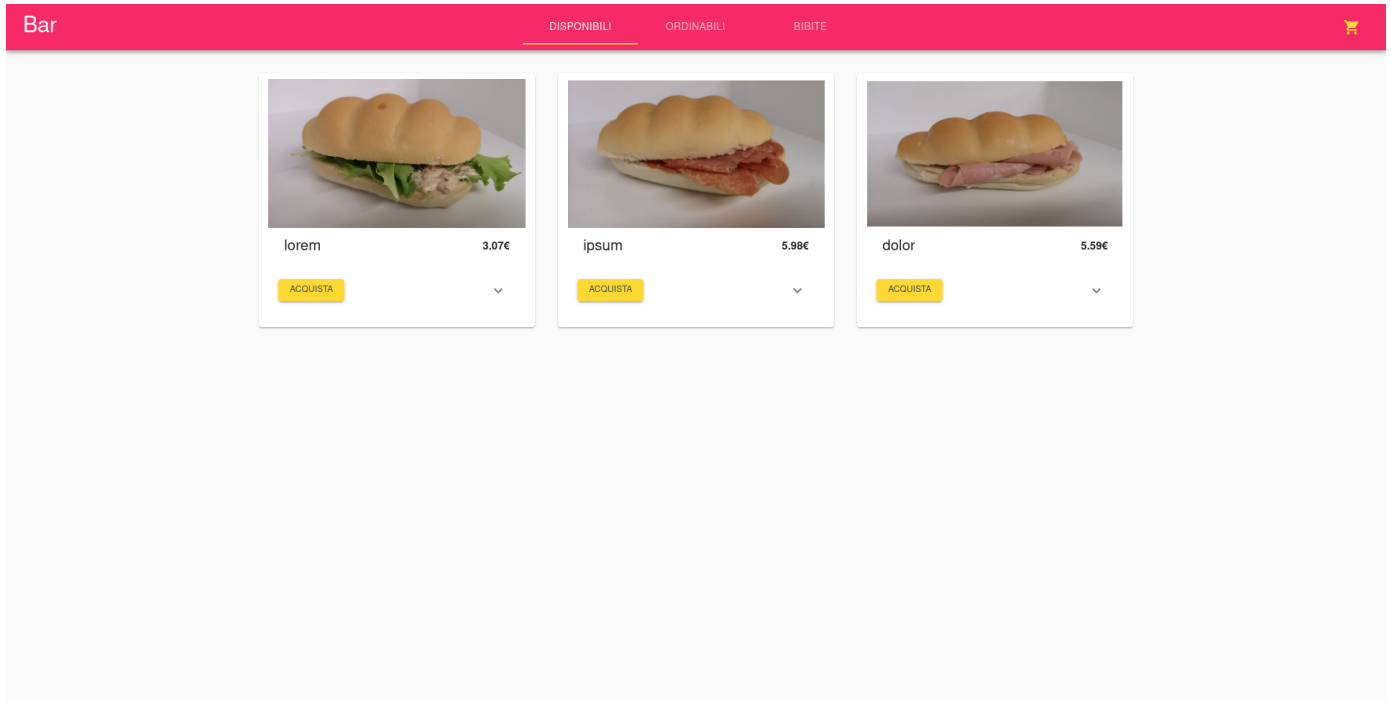
Palette

Palette di colori per l'interfaccia

Pagine & flow



È la pagina di login, che consente l'accesso alle altre pagine.





È l'unica pagina utilizzabile dagli studenti dove è possibile effettuare gli ordini.


Bar

DISPONIBILIORDINABILIBIBITE

Carrello

lorem
3.07€

ipsum
5.98€

dolor
5.59€

Totale26.60€

Oggettix5

TurnoPrimo turno

CONFERMA ORDINE

Attraverso questo form gli studenti possono gestire ed inviare i loro ordini.


Amministrazione

ORDINIMENU

Ordini

		Cliente	Turno	Quantità	Prezzo	Pagato
<input type="checkbox"/>	▼	mario 1234567	Primo turno	11	59.89	<input type="radio"/>
<input type="checkbox"/>	▼	lucia 2231253	Secondo turno	20	87.41	<input type="radio"/>

Totale ordinazioni



Item	Count
adipisci	8
dolor	8
sit	9
ipsum	6

Ordini



		Cliente	Turno	Quantità	Prezzo	Pagato
<input type="checkbox"/>	^	mario 1234567	Primo turno	11	59.89	<input type="radio"/>
Immagine		Nome	Quantità	Prezzo	Totale	
		adipisci	8	5.39€	43.12€	
		dolor	3	5.59€	16.77€	
<input type="checkbox"/>	v	lucia 2231253	Secondo turno	20	87.41	<input type="radio"/>

Totale ordinazioni



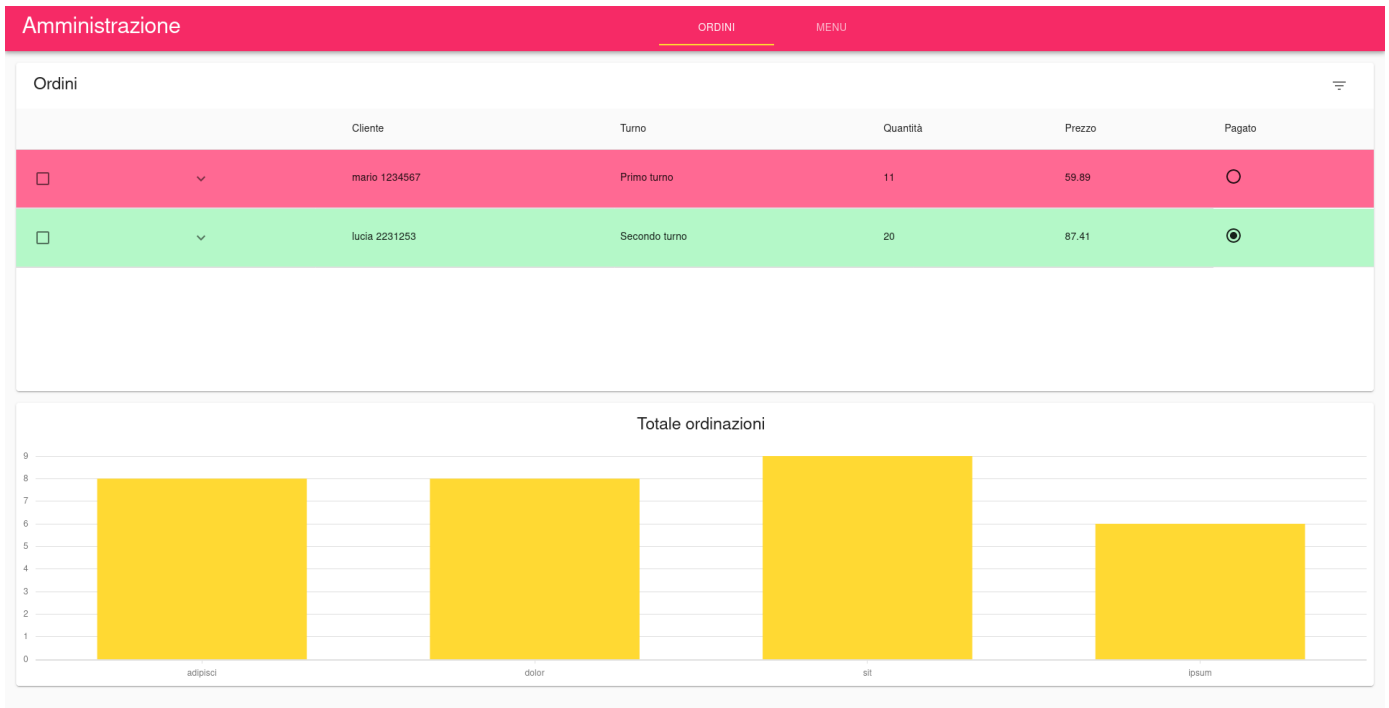
1 selezionati



		Cliente	Turno	Quantità	Prezzo	Pagato
<input checked="" type="checkbox"/>	v	mario 1234567	Primo turno	11	59.89	<input type="radio"/>
<input type="checkbox"/>	v	lucia 2231253	Secondo turno	20	87.41	<input type="radio"/>

Totale ordinazioni





È una delle due pagine dedicate agli operatori del bar. Qui possono monitorare e gestire gli ordini.

Amministratore ORDINI MENU

Ordini

		Immagine	Nome	Tipologia	Prezzo	Quantità massima
<input type="checkbox"/>	▼		lorem	available	3.07€	5
<input type="checkbox"/>	▲		ipsum	available	5.98€	5
			ipsum			
			ipsum			
<input type="checkbox"/>	▼		dolor	available	5.59€	5
<input type="checkbox"/>	▼		sit	orderable	2.62€	5
<input type="checkbox"/>	▼		amet	orderable	5.41€	5
<input type="checkbox"/>	▼		consectetur	orderable	2.40€	5
<input type="checkbox"/>	▼		adipisci	beverage	5.39€	5
<input type="checkbox"/>	▼		elit	beverage	2.52€	5

È la seconda pagina dedicata agli operatori del bar. Qui possono gestire il menù.

The screenshot shows a web application with a dark red header bar containing 'Amministrazione', 'ORDINI', and 'MENU'. A sidebar on the left is titled 'Ordini' and contains a list of items with checkboxes and dropdown arrows. A modal window titled 'Aggiungi prodotto' is open in the center. The modal contains the following fields: 'Nome *' (text input with placeholder 'Nome prodotto'), 'Image Upload' (button with a camera icon and text 'Select a file to show details'), 'Ingredienti' (text input with placeholder 'Nome ingrediente' and a '+' button), 'Tipologia *' (dropdown menu with 'Disponibile' selected), 'Numero massimo' (text input with '99' and a spinner), and 'Prezzo *' (text input with '€ 1.50' and a spinner). A yellow 'CONFERMA' button with a right arrow is at the bottom right of the modal.

Attraverso questo form i gestori possono aggiungere nuovi prodotti al menù

Linguaggi di programmazione

È stato necessario utilizzare i seguenti linguaggi non per scelta, ma per necessità.

JS^[2] ha da poco festeggiato i 25 anni di utilizzo come linguaggio di programmazione per il web e rappresenta quindi lo standard di fatto.

TS^[3] è invece una sorta di rivisitazione di JavaScript con lo scopo di renderlo un linguaggio tipizzato. È uguale in tutto e per tutto al suo antenato se non per l'aggiunta dei tipi.

L'ho dunque utilizzato per comodità nello sviluppo.

JSX^[4] deriva allo stesso modo da JS e ne espande le potenzialità.

Ho dovuto utilizzarlo in seguito alla scelta di usare la libreria React che si appoggia su questo linguaggio.

Esistono altre soluzioni più esotiche che stanno prendendo piede nel mondo del web developing ma che per mia inesperienza ho preferito evitare per restare nell'ambito dei linguaggi a me familiari.

Javascript (<https://it.wikipedia.org/wiki/JavaScript>)

Panoramica del linguaggio

JavaScript è un linguaggio di programmazione Open Source conforme alla specifica ECMAScript^[5].

JS è un linguaggio di alto livello la cui implementazione comune lo porta a essere compilato just-in-time^[6].

Non è caratterizzato da un paradigma preciso perciò supporta ovviamente il paradigma^[7] imperativo^[8] oltre che quello funzionale^[9] ed event-driven^[10].

Il type system è dinamico^[11].

Supporta le First-class function^[12], tratto caratteristico della programmazione funzionale e del suo type system.

Supporta l'OOP^[13], in particolare il Prototype-based programming^[14].

Browser Event Loop

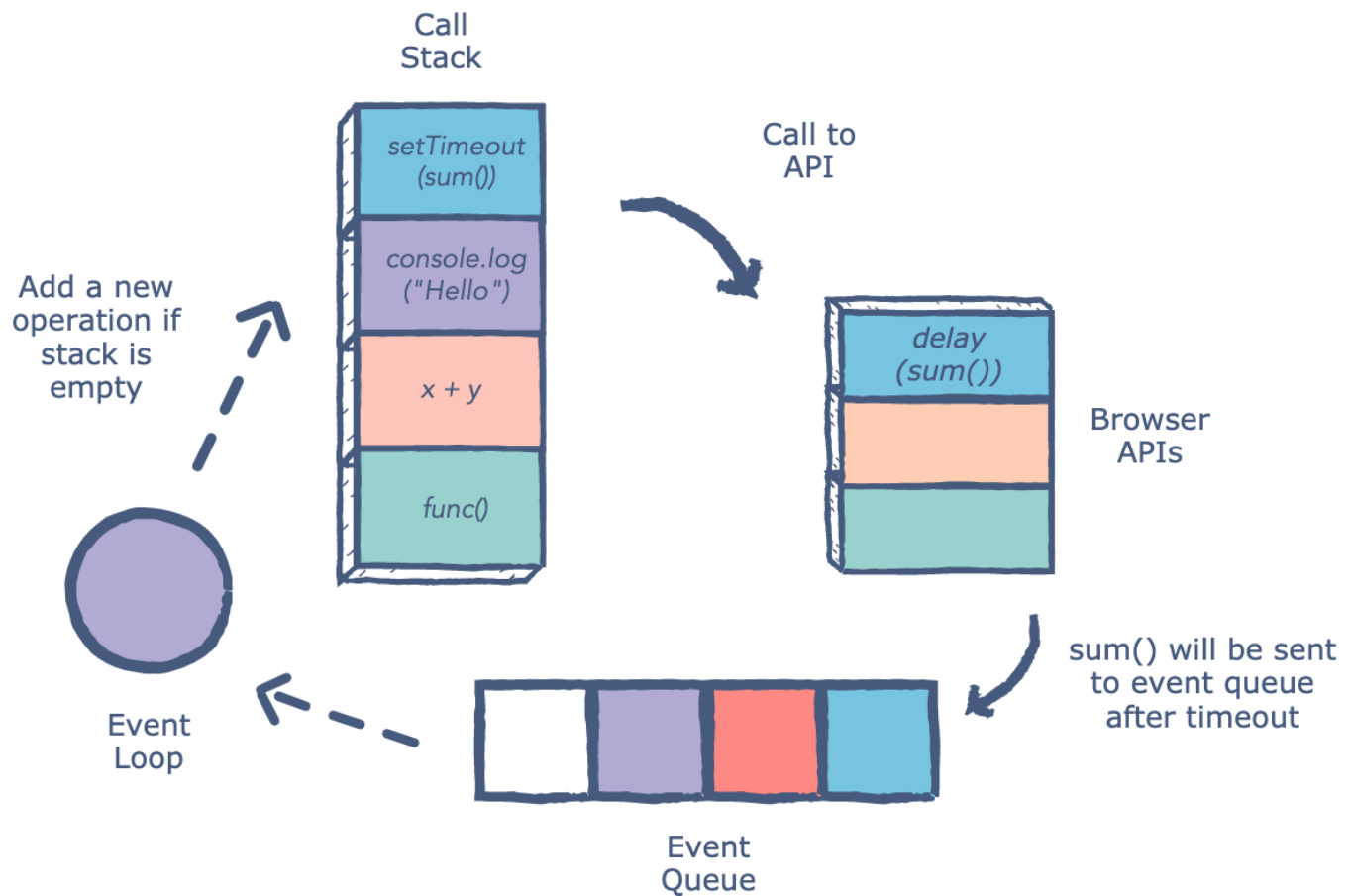
Il concetto di Event Loop non si applica semplicemente a JavaScript, ma a come il front-end di un browser funzioni e ancora più in generale al funzionamento delle interfacce grafiche.

L'Event Loop è piuttosto semplice, si tratta di un ciclo infinito all'interno del JS engine che attende istruzioni, le esegue e rimane in attesa. L'engine la maggior parte del tempo rimane in attesa e si attiva solamente quando necessario, per esempio in presenza di un tag `<script src="...">` o a causa di un evento.

Asincronicità in un singolo Thread

JavaScript non ha seguito lo sviluppo dell'informatica verso architetture multi-core rimanendo un linguaggio a singolo thread; ciò significa che può accedere ad un solo core.

Per ovviare a questo problema si utilizzano alcuni stratagemmi per ottenere l'illusione di un'esecuzione asincrona. L'asincronicità in JS non fa parte del linguaggio stesso ma è costruita al di sopra dell'engine attraverso le browser APIs. Keyword come `async` e `await` fanno parte del linguaggio, ma non la loro implementazione.



Architettura di un browser.

- Call Stack
 - Si tratta del classico stack delle chiamate con struttura LIFO.
- Browser APIs
 - Sono integrate nel browser e permettono operazioni aggiuntive come la comunicazione con l'hardware della macchina. Un esempio è la Geolocation API che permette di ottenere la posizione del client in modo semplice per il programmatore. L'implementazione dell'API è però rappresentata da codice di basso livello come C++ che si occupa di comunicare con il GPS dell'elaboratore.
- Event Queue
 - Ha la responsabilità di inoltrare nuove funzioni alla Call Stack perchè vengano processate. Utilizza la struttura dati della coda.

Quando una funzione asincrona viene chiamata interviene il run-time che si occupa di gestirne l'esecuzione. L'API pensa poi a reinoltrare il processo alla Call Stack a tempo debito. Questo viene fatto tramite la Event Queue. Si può quindi notare che JS lavora su un singolo thread ma le API operano su thread separati.

È l'Event Loop ad occuparsi della gestione di questo meccanismo:

quando la Stack è vuota preleva il primo elemento dalla Event Queue e lo inserisce nella Call

Stack per essere eseguito.

Un ottimo esempio è la funzione `setTimeout()` che esegue un'operazione ad intervalli di tempo prestabilito.

```
1  const foo = () => console.log("Primo");
2  const bar = () => setTimeout(() => console.log("Secondo"), 500);
3  const baz = () => console.log("Terzo");
4
5  bar();
6  foo();
7  baz();
8
9  /*
10 Output:
11 Primo
12 Terzo
13 Secondo
14 */
```

Quando invochiamo la funzione `bar()` essa viene passata alla relativa API. Il timer inizia a scorrere e quando il tempo è terminato la callback viene inserita nella Event Queue. Dopo l'esecuzione di `bar()` e `foo()` la stack è vuota e quindi "Second" viene stampato in Console.

La struttura è in realtà però più complessa. La Event Queue si compone infatti di due queue minori appartenenti a due differenti categorie di operazioni:

- Micro-tasks
 - Sono operazioni che solitamente devono essere eseguite immediatamente.
 - Hanno maggiore priorità in quanto l'event loop non esce dalla relativa queue finchè essa non è vuota.
 - Esempi:
 - Promises
- Macro-tasks
 - Sono operazioni distinte e indipendenti.
 - Hanno minore priorità e la relativa queue lavora allo stesso modo della call stack ma per le chiamate asincrone.
 - Esempi:
 - `setTimeout`
 - UI Rendering
 - HTML parsing

- generazione del DOM

Questa distinzione viene fatta per migliorare l'esperienza utente.

Async | Await

`async` e `await` sono le due keyword base usate in JS e dai programmatori in generale come standard di fatto per la programmazione asincrona.

- `async` Esplicita l'asincronicità di una funzione.
- `await` Ordina di sospendere l'esecuzione in attesa di una Promise.
- `then` Permette di specificare una Callback da chiamare alla risoluzione di una Promise.

La differenza importante da notare tra `await` e `then` è che la prima sospende l'esecuzione per riportarla al contesto globale mentre la seconda fa procedere l'esecuzione della funzione asincrona normalmente.

Ciò dipende da come l'asincronicità è gestita in JavaScript.

TypeScript (<https://www.typescriptlang.org/>)

Panoramica del linguaggio

TypeScript è un linguaggio open-source creato da Microsoft (<https://it.wikipedia.org/wiki/Microsoft>) costruito su JS che aggiunge un type system statico^[17].

Utilizzare i tipi rimane comunque opzionale anche se è fortemente raccomandato. Il codice JS rimane valido e perciò non è necessario convertire subito la code base^[18].

TypeScript viene transcompilato in JavaScript tramite strumenti come Babel (<https://babeljs.io/>). In questo modo otteniamo semplice codice JS compatibile tra browsers.

Vantaggi

I tipi vengono controllati prima dell'esecuzione e se vi è qualche errore è impossibile eseguire il programma. Questa procedura elimina problemi in fase di esecuzione che in JS sono molto comuni.

I linguaggi tipizzati offrono anche una fase di debug^[19] semplificata, proprio per la presenza dei tipi. Grazie ad essi oggetti, funzioni ecc... possono essere definiti più accuratamente aiutando notevolmente il Debugger^[20] nel trovare errori logici.

Il lavoro del programmatore è così facilitato dalla maggiore efficienza del Debugger e dalla migliore leggibilità e mantenibilità del codice in quanto i tipi offrono una sorta di documentazione attraverso cui si può capire meglio come funzionano le funzioni e il codice che si sta leggendo.

Esempio

```
1  const user = {
2    firstName: "Mario",
3    lastName: "Rossi",
4    role: "Professore"
5  }
6
7  console.log(user.name)
8  Property 'name' does not exist on type '{ firstName: string; lastName: str
```

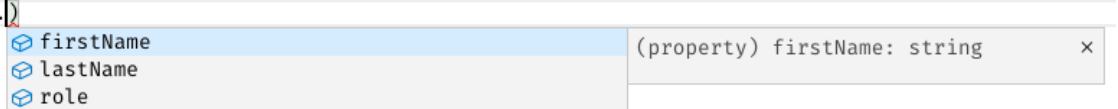
Il codice viene validato più efficacemente, rendendone l'esecuzione molto più prevedibile. Questo può essere molto utile quando si fa utilizzo di JSX.

Esempio

```
1  import * as React from "react";
2
3  interface UserThumbnailProps {
4    img: string;
5    alt: string;
6    url: string;
7  }
8
9  export const UserThumbnail = (props: UserThumbnailProps) =>
10    <a href={props.url}>
11      <img src={props.img} alt={props.alt} />
12    </a>
```

Una più accurata definizione aiuta anche l'intellisense^[21] che facilita la scrittura del codice.

```
1  const user = {
2    firstName: "Mario",
3    lastName: "Rossi",
4    role: "Professore"
5  }
6
7  console.log(user.)
```



React JSX | JavaScript XML (<https://it.reactjs.org/docs/introducing-jsx.html>)

Panoramica del linguaggio

JavaScript XML è un'estensione della sintassi JavaScript utilizzata per descrivere la UI, che

non va confusa con una stringa o con HTML.

Lo scopo è quello di creare componenti che inglobino contemporaneamente la logica e la descrizione dell'interfaccia. Non è in alcun modo obbligatorio utilizzare questa metodologia che però è fortemente raccomandata.

JSX quando compilato produce React nodes, costituiti da funzioni JS capaci di generare oggetti JS.

Esempio

```
1  const name = 'Mario Rossi';
2  const element = <h1>Ciao, {name}</h1>;
3  ReactDOM.render(
4    element,
5    document.getElementById('root')
6  );
```

JSX esegue l'escape dei valori iniettati prima del rendering, rendendo il linguaggio sicuro da attacchi basati su XSS (https://it.wikipedia.org/wiki/Cross-site_scripting).

Esempio

```
1  const title = response.contenutoPotenzialmentePericoloso;
2  const element = <h1>{title}</h1>; // Pratica sicura
```

Frameworks

Next.js (<https://nextjs.org/>)

Il framework React per la produzione

Next.js si impegna a risolvere una serie di problemi attraverso un bilanciato livello di astrazione:

- Compilazione del codice
- Ottimizzazione tramite il code-splitting^[22].
- Pre-rendering delle pagine per migliorare performance e SEO^[23]

Oltre che risolvere problemi comuni implementa alcune importanti funzionalità:

- Sistema di routing statico, dinamico e client-side con prefetching.

- Per client-side routing si intende la tecnica con cui si raggiunge l'illusione di utilizzare più pagine separate mentre in realtà tutto il contenuto è racchiuso in un'unica pagina. Questo è possibile tramite la manipolazione del DOM e la libreria react-router che permette di visualizzare diversi componenti in base all'URL. Questa feature consente di sgravare il server di una parte del carico computazionale e rende la navigazione quanto più fluida possibile.
- Fare prefetching significa precaricare le risorse che l'utente potrebbe richiedere per poterle fornire con maggiore velocità.
- Supporto per CSS e Sass nativo.
 - Sass è un linguaggio per definire lo stile basato su CSS che punta ad espanderne le capacità e risolverne i problemi.
- Development environment con fast refresh per uno sviluppo più facile.
 - Next.js supporta due diverse modalità: production e development. In modalità development sono attivati il code reporting e l'hot reloading. In pratica ogni volta che avviene una modifica e viene salvata Next.js automaticamente ricompila il codice rendendola immediatamente visibile.
- Server-side Rendering e Static generation.
 - Server-side: L'HTML viene generato ad ogni richiesta.
 - Static generation (raccomandato): L'HTML è generato durante la fase di build e viene riutilizzato ad ogni richiesta.
 - Quello che viene servito in entrambi i casi rappresenta solamente il primo render HTML. Successivamente la pagina verrà *idratata* ([https://en.wikipedia.org/wiki/Hydration_\(web_development\)](https://en.wikipedia.org/wiki/Hydration_(web_development))) dal client. Per idratazione si intende la conversione di una pagina HTML statica ad una dinamica.

React (<https://reactjs.org/>)

React.js conosciuto anche come React è una libreria pensata per la costruzione di interfacce Reattive cioè dinamiche. È sviluppato e mantenuto da Facebook; i socials Facebook ed Instagram sono esempi dell'utilizzo di React.

L'idea alla base è di non modificare mai direttamente il Document Object Model. È React ad occuparsi di modificare il DOM per fare in modo che combaci con l'interfaccia da noi creata. Questa modalità evita pratiche errate ed errori logici.

Ho deciso di utilizzare la nuova API denominata Hook API che personalmente trovo molto più semplice, snella ed intuitiva.

Inoltre mi aiuta nella gestione dello stato, un problema molto comune in React.

Ho deciso di non appoggiarmi a sistemi come Redux (<https://redux.js.org/>) per la gestione dello stato data la bassa complessità dell'applicativo.

Non vi sono comunque controindicazioni sull'utilizzo della “vecchia” API basata sull'OOP.

Vantaggi e Concetti fondamentali

Possiamo di seguito vedere un esempio di rendering tramite React.

Abbiamo bisogno di un nodo root al cui interno tutto sarà gestito da ReactDOM.

Esempio

```
1 <div id="root"></div>
2
3 const element = <h1>Hello world!</h1>;
4 ReactDOM.render(element, document.getElementById('root'));
```

React rappresenta i componenti attraverso elementi organizzati in un DOM virtuale.

Gli elementi React sono immutabili e questo porterebbe a pensare che l'unico modo per aggiornare l'interfaccia sia quello di chiamare la funzione `render()`. Nella pratica però si opera diversamente poichè i componenti React sono dotati di stato.

Limitarci ad utilizzare la funzione `render()` è quindi un ottimo esempio per mostrare la capacità di React di aggiornare l'interfaccia solo dove e quando necessario.

Ad ogni modifica viene generato un nuovo DOM virtuale che viene confrontato con quello precedente. React compara quindi gli elementi ed i relativi figli con quelli antecedenti e applica i cambiamenti.

Esempio

```
1 function tick() {
2   const element = (
3     <div>
4       <h1>Hello world!</h1>
5       <h2>Oggi è il {new Date().toLocaleTimeString()}</h2>
6     </div>
7   );
8   ReactDOM.render(element, document.getElementById('root'));
9
10  setInterval(tick, 1000);
```

Hello, world!

It is 12:26:46 PM.

```
Console Sources Network Timeline
▼<div id="root">
  ▼<div data-reactroot>
    <h1>Hello, world!</h1>
    ▼<h2>
      <!-- react-text: 4 -->
      "It is "
      <!-- /react-text -->
      <!-- react-text: 5 -->
      "12:26:46 PM"
      <!-- /react-text -->
      <!-- react-text: 6 -->
      "."
      <!-- /react-text -->
    </h2>
  </div>
</div>
```

React si basa sulla creazione di componenti ed il passaggio di props.

I primi sono innestabili e accettano come input delle props.

Esse non sono altro che dati in read-only che possono essere passati dal componente padre a quello figlio.

Tutti i componenti devono essere puri e vengono definiti tali quando non tentano mai di modificare l'input che ricevono e ritornano sempre lo stesso risultato con input identici.

Esempio

```
1  function Avatar(props) {
2    return (
3      <img
4        className="Avatar"
5        src={props.user.avatarUrl}
6        alt={props.user.name}
7      />
8    );
9  }
10
11 function UserInfo(props) {
12   return (
13     <div className="UserInfo">
14       <Avatar user={props.user} />
15       <div>{props.user.name}</div>
16     </div>
17   );
18 }
19
20 function Comment(props) {
21   return (
22     <div className="Comment">
23       <UserInfo user={props.author} />
24       <div>{props.text}</div>
25       <div>
26         {props.date.toLocaleDateString()}
27       </div>
28     </div>
29   );
30 }
31
32 const comment = {
33   date: new Date(),
34   text: 'Hello world!',
35   author: {
36     name: 'Mario Rossi',
37     avatarUrl: 'https://image-link.png',
38   },
39 };
40 ReactDOM.render(
41   <Comment
42     date={comment.date}
43     text={comment.text}
44     author={comment.author}
45   />,
46   document.getElementById('root')
47 );
```

Il concetto cardine dietro all'interfaccia reattiva è quello di stato.

Il metodo `render()` in realtà è necessario chiamarlo solo una volta.

È attraverso lo stato dei componenti che viene cambiato l'aspetto della pagina. Gli stati sono locali e possono essere condivisi con i componenti figli.

Un altro importante concetto è il Lifecycle di un componente.

I metodi di Lifecycle vengono utilizzati ad esempio per liberare risorse non più necessarie alla distruzione di un componente o per eseguire routines alla sua creazione.

Esempio


```

1  class Clock extends React.Component {
2      constructor(props) {
3          super(props);
4          this.state = {date: new Date()};
5      }
6
7      componentDidMount() { //metodo di Lifecycle
8          this.timerID = setInterval(
9              () => this.tick(),
10             1000
11         );
12     }
13
14     componentWillUnmount() { //metodo di Lifecycle
15         clearInterval(this.timerID);
16     }
17
18
19     tick() {
20         this.setState({
21             date: new Date()
22         });
23     }
24
25     render() {
26         return (
27             <div>
28                 <h1>Hello world!</h1>
29                 <h2>Oggi è il {this.state.date.toLocaleTimeString()}.</h2>
30             </div>
31         );
32     }
33 }
34
35 ReactDOM.render(
36     <Clock />,
37     document.getElementById('root')
38 );

```

React assieme a JS permette di implementare complesse logiche che comprendono anche la renderizzazione condizionale.

Esempio

```

1  class LoginControl extends React.Component {
2      constructor(props) {
3          super(props);
4      }
5
6      render() {
7          return (
8              <div>
9                  {
10                     if (props.isLoggedIn) {
11                         <LogoutButton onClick={this.function} />;
12                     } else {
13                         <LoginButton onClick={this.function} />;
14                     }
15                 }
16             </div>
17         );
18     }
19 }
20
21 ReactDOM.render(
22     <LoginControl isLoggedIn={false}/>,
23     document.getElementById('root')
24 );

```

Grazie alla renderizzazione condizionale ci è possibile creare un numero di elementi variabile. React ha bisogno di identificare ogni elemento per poterlo creare, modificare ed eliminare.

L'identificazione automatica non è attuabile quando il numero di elementi è dinamico e non è possibile associare ad ognuno di essi un identificatore a priori.

Il programmatore è chiamato a risolvere questo problema.

Attraverso la keyword `key` può infatti specificare un valore che indichi univocamente un elemento in una pagina. È quindi importante che la `key` assegnata sia unica e consistente.

Esempio

```
1  function NumberList(props) {
2    const numbers = props.numbers;
3    return (
4      <ul>
5        {
6          numbers.map((number) =>
7            <li key={number.toString()}>{number}</li>
8          )
9        }
10     </ul>
11   );
12 }
13
14 const numbers = [1, 2, 3, 4, 5];
15 ReactDOM.render(
16   <NumberList numbers={numbers} />,
17   document.getElementById('root')
18 );
```

Le linee guida React indicano di evitare la pratica dell'ereditarietà proponendo come alternative Containment e Specialization.

Questo stile di programmazione rende il codice più semplice da leggere.

Props e Composizione permettono di personalizzare un componente in modo esplicito e sicuro.

Esempio

```

1  function Dialog(props){
2      return(
3          <div>
4              <h1>
5                  {props.title}
6              </h1>
7              <p>
8                  {props.message}
9              </p>
10             {props.children}
11         </div>
12     );
13 }
14
15 function WelcomeDialog() {
16     return (
17         <Dialog
18             title="Hello"
19             message="Hello world!"
20         >
21             <p>lorem ipsum dolor sit amet</p>
22         </Dialog>
23     );
24 }

```

Gli Hook sono una nuova API per la gestione dello stato aggiunta in React 16.8 (allo stato attuale l'ultima versione è la 17.0.2).

Gli Hook sono speciali contenitori di dati che al loro variare dettano un ridisegno dell'interfaccia dove necessario. Rendono il codice più leggibile e facile da gestire semplificando il riutilizzo della stessa logica tra più componenti.

Esempio

```

1  function Example() {
2      const [count, setCount] = useState(0);
3      return (
4          <div>
5              <p>You clicked {count} times</p>
6              <button onClick={() => setCount(count + 1)}>
7                  Click me
8              </button>
9          </div>
10     );
11 }

```

Sino ad ora non è mai stata considerata la possibilità per un componente figlio di passare valori al componente padre.

Ciò è però possibile in vari modi. La pratica del Lifting State Up è l'esempio più comune. Spesso succede che un componente figlio debba passare un valore al componente padre per condividerlo con altri figli affinché le modifiche si riflettano anche su altri nodi. Proporre un esempio appropriato sarebbe complesso ma è bene conoscere l'esistenza di questa pratica.

Esempio

```
1  function Counter() {
2    const [count, setCount] = useState(0);
3    function handleClick()
4    {
5      setCount(count + 1);
6    }
7
8    return (
9      <div>
10        <p>Hai cliccato {count} volte</p>
11        <ClickButton handleClick={handleClick}></ClickButton>
12      </div>
13    );
14  }
15
16  function ClickButton(props) {
17    return (
18      <div>
19        <button onClick={props.handleClick}>
20          Cliccami
21        </button>
22      </div>
23    );
24  }
```

Material-UI (<https://material-ui.com/>)

È una libreria di componenti già pronti per la costruzione di interfacce grafiche. Viene utilizzata per accelerare lo sviluppo ed utilizzare codice già testato e documentato.

È definita Material in quanto segue le linee per il design Material (<https://material.io/design>) ideato da Google che tutti conosciamo. Permette comunque una profonda personalizzazione tramite la creazione di temi.

Gestione pacchetti

npm (<https://www.npmjs.com/>)

Node Package Manager è il sistema di pacchettizzazione utilizzato da Node.js.

Viene utilizzato per la gestione delle dipendenze ovvero delle librerie.

Il suo funzionamento è legato ad un file chiamato `packages.lock` che registra tutti i pacchetti installati e la loro versione.

yarn (<https://classic.yarnpkg.com/en/>)

Yarn é un altro gestore di pacchetti per Node.js che punta a risolvere i problemi di npm ed introduce alcune funzionalità.

- Velocità
 - yarn esegue il caching dei pacchetti scaricati, eliminando la necessità di scaricarli nuovamente ad ogni installazione. Inoltre parallelizza le operazioni rendendo le installazioni più veloci, riducendo il bisogno di risorse necessarie.
- Affidabilità
 - Tramite il suo lockfile che imita il comportamento di `packages.lock` e un algoritmo deterministico il processo garantisce l'installazione dei pacchetti in modo consistente anche su macchine diverse. Una richiesta fallita viene ritentata senza interrompere l'installazione. Inoltre risolve autonomamente gli errori di versione delle dipendenze unificandole per evitare duplicati.
- Garanzia:
 - Su ogni pacchetto viene effettuato il checksum per controllarne l'integrità prima che il suo codice venga eseguito.

Pacchetti | Librerie

clsx (<https://github.com/lukeed/clsx>)

È una libreria molto piccola che permette di assegnare classi ai componenti in modo condizionale.

Il suo punto di forza sono le performance ottenute anche grazie al suo ridotto peso di appena 228byte. Questo non sacrifica la compatibilità che è mantenuta con tutti i moderni browsers.

Nell'esempio la classe `expandOpen` viene assegnata quando `expanded` assume valore `true`.

Esempio

```
1   className={clsx(classes.expand, {  
2     [classes.expandOpen]: expanded,  
3   })}
```

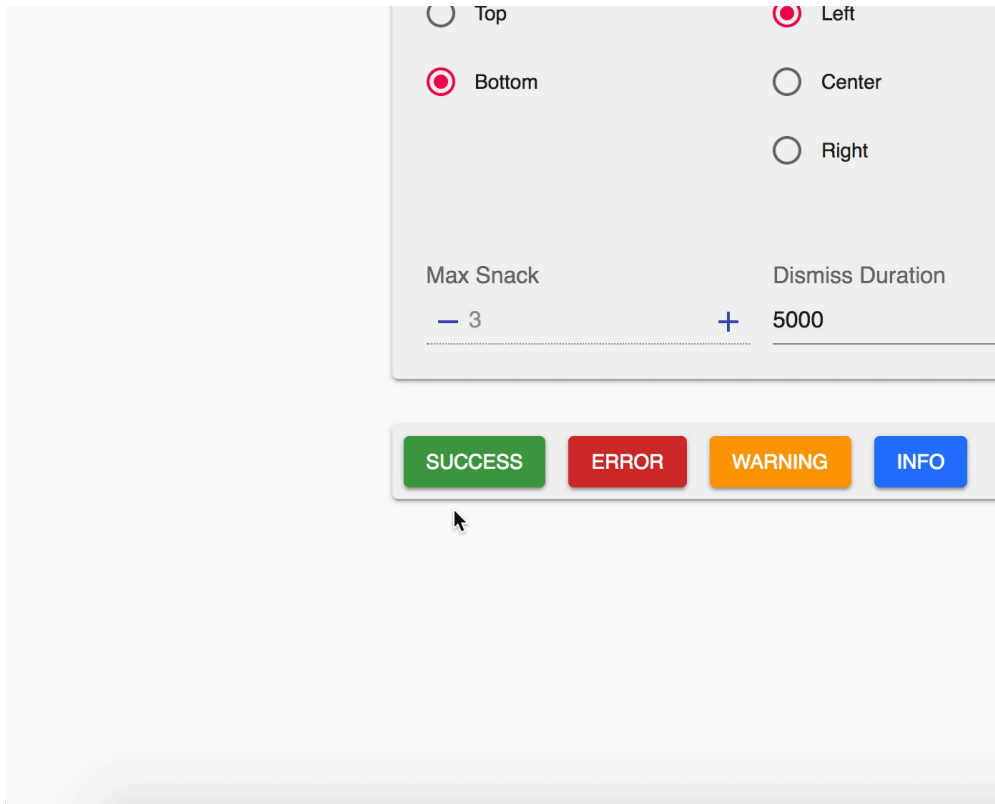
notistack (<https://github.com/iamhosseindhv/notistack>)

Notistack è una libreria che punta a rendere facile la gestione e visualizzazione delle notifiche in stile Snackbar. È altamente configurabile e permette di visualizzare le notifiche in colonna.

Quest'ultima feature molto utile è però sconsigliata dalle linee guida di Material-UI su cui si basa.

Esempio

```
1   const MyButton = () => {  
2     const { enqueueSnackbar, closeSnackbar } = useSnackbar();  
3  
4     const handleClick = () => {  
5       enqueueSnackbar('Hello world!');  
6     };  
7  
8     return (  
9       <Button onClick={handleClick}>Mostra snackbar</Button>  
10    );  
11  }
```



DevExtreme Reactive React Chart (<https://devexpress.github.io/devextreme-reactive/react/chart/>)

DevExtreme React Chart è un componente pensato per la visualizzazione di dati di diverso tipo. Propone varie possibilità di visualizzazione: barre, linee, area, scatter, torta ecc... Il componente è progettato per essere estendibile attraverso plugin che aggiungono funzioni come la legenda o una griglia. Supporta anche Typescript e l'integrazione con librerie grafiche come Material-UI.

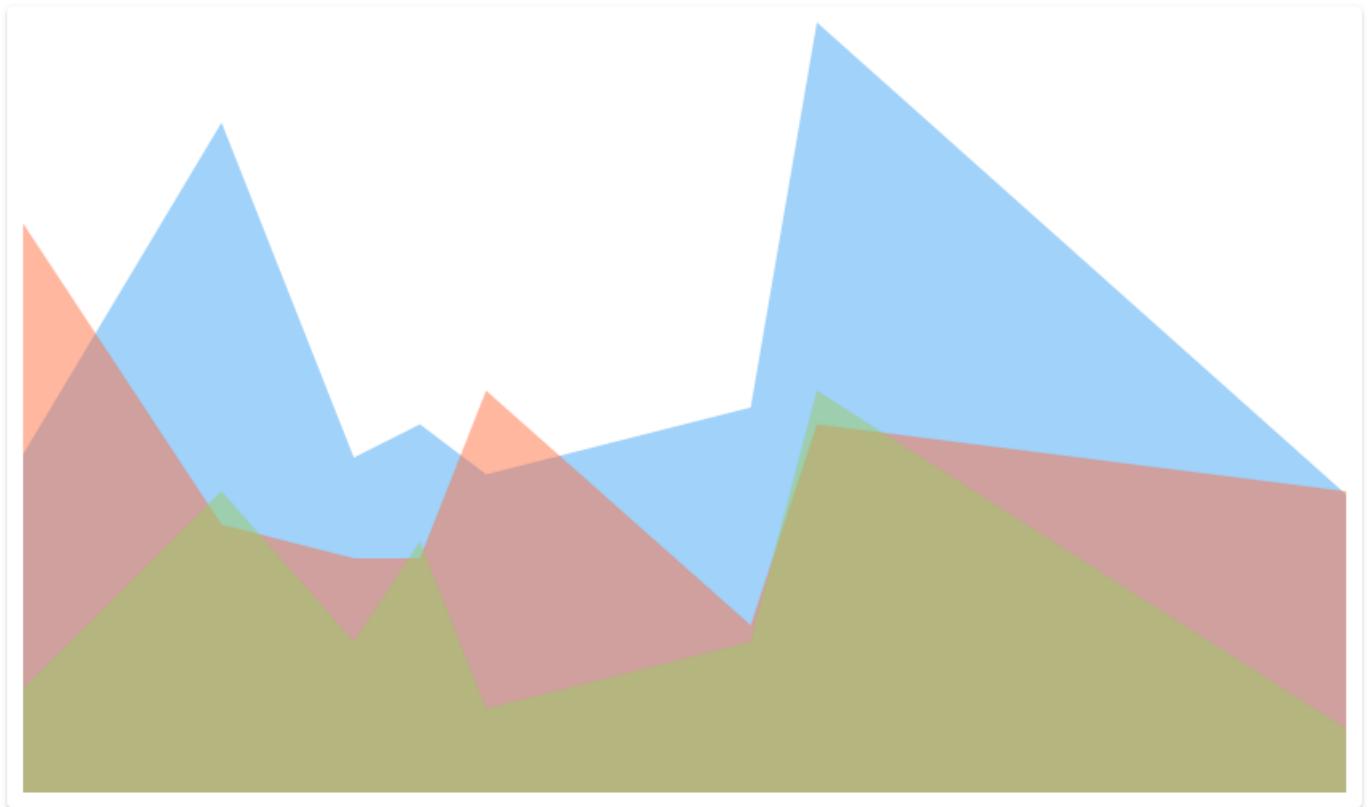
È compatibile con le ultime versioni dei browser più noti.

Esempio


```

1  <Chart
2    data={chartData}
3  >
4    <AreaSeries
5      valueField="it"
6      argumentField="year"
7    />
8    <AreaSeries
9      valueField="uk"
10     argumentField="year"
11   />
12   <AreaSeries
13     valueField="us"
14     argumentField="year"
15   />
16 </Chart>

```



Login

Ogni studente è munito di un account Google appartenente al dominio scolastico
 itisrossi.vi.it .

Affidarsi al sistema di account Google ci ha consentito una gestione semplificata degli utenti in quanto non abbiamo dovuto affrontare le relative problematiche di storage e sicurezza. Il login avviene come segue:

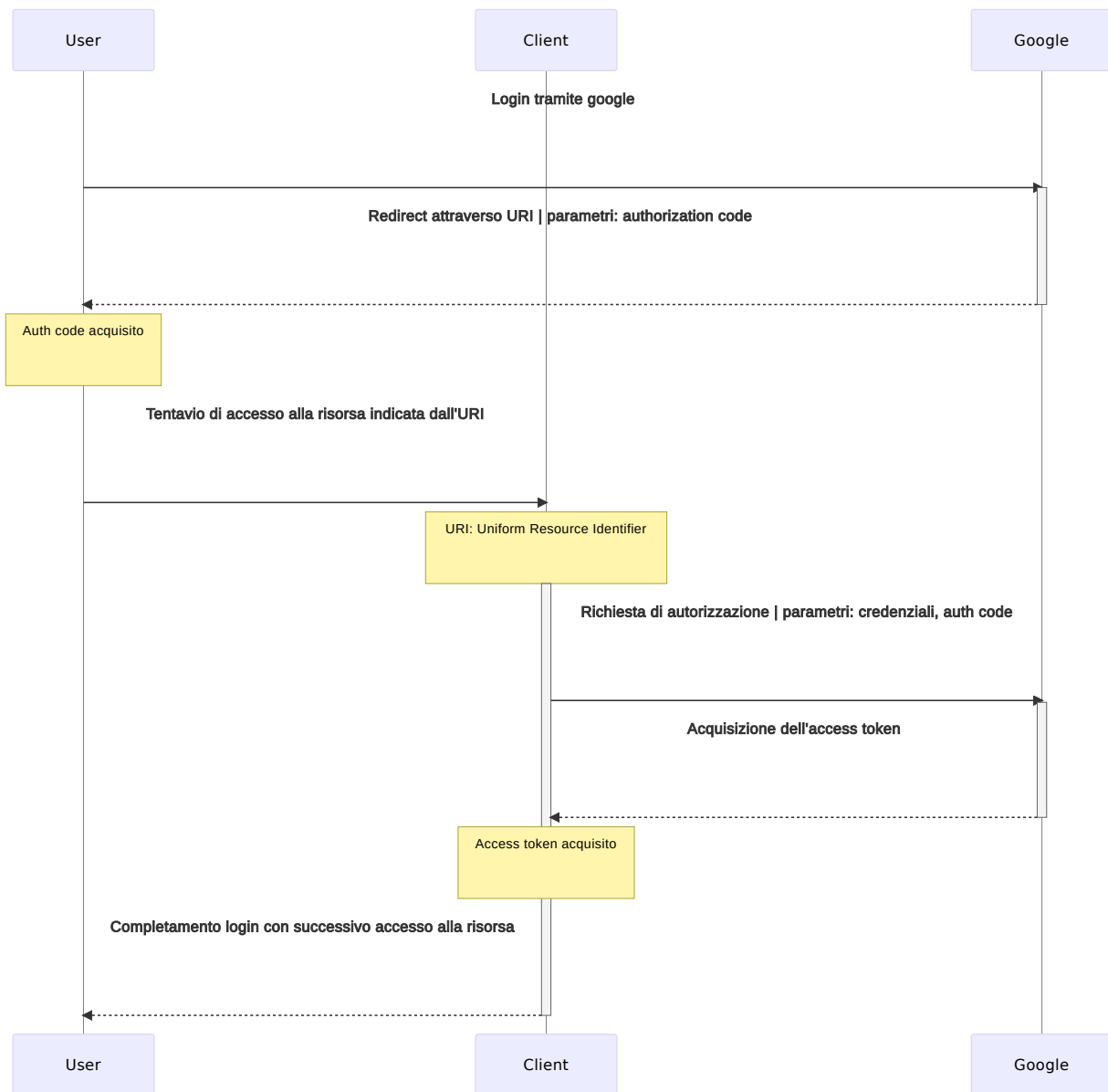
- Il Client contatta Google autenticandosi con un api key.
- Compare una finestra Pop-up gestita interamente da Google attraverso cui l'utente si autentica. Un filtro basato sul dominio scolastico consente l'ingresso solo agli studenti.
- Il Client riceve un token di accesso dalla durata limitata per il relativo account.

L'autenticazione segue lo standard OAuth2 (<https://oauth.net/2/>).

Attraverso le API Google è possibile recuperare informazioni quali l'ID, il nome e il cognome di registrazione dell'account. Questo serve ad identificare il mittente di un ordine.

OAuth2 è un protocollo diventato standard di fatto per l'autorizzazione anche non web.

OAuth2 è di facile utilizzo e può essere configurato per soddisfare requisiti specifici.



Schema dell'autenticazione nel nostro applicativo.

API

L'Application programming interface rappresenta un'interfaccia per impartire istruzioni. Lo scopo è di fornire un adeguato livello di astrazione per sgravare il programmatore dalle complicazioni sottostanti. Le API possono essere sia un'interfaccia tra client e server che un insieme di procedure appartenenti a librerie software.

Le API da noi create servono a mettere in comunicazione client e server, nel nostro caso rappresentati da sito web e back-end.

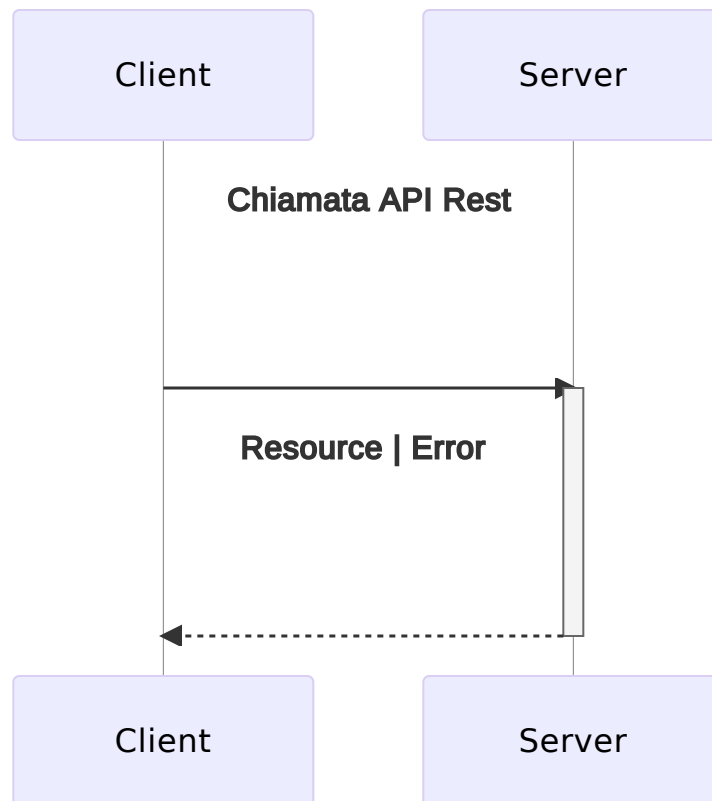
Abbiamo deciso di implementare API di tipo REST considerata la loro diffusione e comodità. Abbiamo anche scelto di basarci su CRUD senza però implementare l'operazione di Update

a noi non necessaria, evitando anche di affrontare il complesso calcolo dei delta delle modifiche.

Ci avvaliamo solamente delle operazioni: Create, Read e Delete.

Come codifica dati abbiamo scelto JSON per la sua popolarità e facilità d'uso in fase di progettazione. Avevamo preso in considerazione anche la più efficiente codifica FlatBuffers che però non abbiamo utilizzato perchè più complessa.

Schema dell'API del nostro applicativo



Metodi dell'API

```
1 interface API {
2     getMenu(): Promise<Product[] | Error>;
3     addMenuItem(prod: Omit<Product, 'id'>): Promise<Product | Error>;
4     removeMenuItem(prodId: number): Promise<Product | Error>;
5     getOrders(): Promise<Order[] | Error>;
6     getAllOrders(): Promise<Order[] | Error>;
7     addOrder(first_term: boolean, order: [number, number][]): Promise<Order | Error>;
8     setOrderAsDone(orderId: number): Promise<Order | Error>;
9 }
```

Esempio di un metodo:

```
1  async getMenu(): Promise<Product[] | Error> {
2      const auth_token = this.getAuthHeader(null);
3      const response = await this.fetch(
4          `menu`,
5          {
6              method: "GET",
7              headers: auth_token
8          }
9      );
10
11     const payload: RawProduct[] | Error = await response.json();
12     //... codice ...
13 }
```

Breve documentazione dei metodi:

- `getMenu()`
 - GET - Ritorna la lista dei prodotti in menù.
- `addMenuItem()`
 - PUT - Aggiunge un nuovo prodotto al menù.
- `removeMenuItem()`
 - DELETE - Rimuove un prodotto dal menù.
- `getOrders()`
 - GET - Ritorna gli ordini ancora da servire relativi all'account corrente.
- `getAllOrders()`
 - GET - Ritorna tutti gli ordini ancora da servire.
- `addOrder()`
 - PUT - Aggiunge un ordine per l'account corrente.
- `setOrderAsDone()`
 - DELETE - Marchia un ordine come servito senza però eliminarlo. La tabella degli ordini viene resettata ogni notte. In questo modo si possono recuperare ordini persi per un errore umano o informatico.

Idempotenza

L'idempotenza è una proprietà delle funzioni per la quale applicando più volte la stessa funzione il risultato ottenuto è uguale a quello di una sola applicazione.

Il metodo HTTP PUT è idempotente a differenza di HTTP POST. Ripetere la stessa chiamata PUT non ha effetti collaterali. In pratica si traduce nell'impossibilità di effettuare lo stesso

ordine due volte.

Abbiamo deciso di implementare questa meccanica anche nelle nostre API per impedire che lo stesso ordine possa essere inoltrato più volte causando una perdita economica e lo spreco di cibo.

Questo risultato viene raggiunto per mezzo di un token contenente uno UUID^[24].

Lo UUID da noi usato è generato calcolando l'hash del JWT combinato al payload.

Quest'ultimo dipende dal metodo che viene chiamato.

Nel caso in cui l'idempotenza non sia importante il payload può essere semplicemente la funzione chiamata, come `getMenu()`.

Quando invece l'idempotenza è fondamentale come per il metodo `addOrder()` il payload deve essere più complesso e nello specifico caso è composto dall'ordine stesso. La complessità aggiuntiva serve ad aumentare l'entropia per assicurare uno UUID sempre differente.

JWT (<https://jwt.io/>)

Panoramica sul JSON Web Token

È uno standard aperto per la trasmissione di informazioni in modo sicuro. Grazie ad esso è possibile garantire l'autenticità e l'integrità delle informazioni trasmesse. I JWT permettono di gestire le sessioni e l'autenticazione in modo efficiente e riducono il bisogno di salvare dati sul server.

Esso è composto da:

- Header
 - contiene informazioni aggiuntive, le più importanti sono:
 - La tipologia del token. Nel caso in esame la tipologia è sempre JWT perchè segue la codifica JSON.
 - L'algoritmo di cifratura.
- Payload
 - Contiene i parametri che possono essere di 3 tipi:
 - Registrati
 - Si tratta di opzioni predefinite non obbligatorie ma consigliate, un esempio è `exp(expiration time)`.
 - Pubblici
 - Si tratta di opzioni codificate nello IANA JSON Web Token Registry, un registro pubblico per la definizione di parametri standardizzati.
 - Privati
 - In questa sezione si è liberi di specificare quello che si desidera in base

alle necessità dell'applicativo.

- Signature
 - la firma è composta dall'hash dell'header e dal payload codificati in base64 e da una chiave condivisa tra client e server. L'algoritmo più usato è il HMAC SHA256:

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  secret)
```

Esempio di header

```
1  {  
2    "alg": "HS256",  
3    "typ": "JWT"  
4  }
```

Esempio di payload

```
1  {  
2    "sub": "1234567890",  
3    "name": "John Doe",  
4    "admin": true  
5  }
```

Esempio di token

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4  
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.  
4pcPyMD09o1PSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Autenticazione tramite JWT

Quando un utente si autentica con successo riceve un token che potrà usare sino alla sua scadenza per effettuare l'accesso.

Il token viene inviato nell'Authorization header usando lo schema Bearer:

1 | Authorization: Bearer <token>

Questa modalità è uno standard di autenticazione HTTP per mezzo di JWT.

Utilizzo nel nostro applicativo

Diversamente da come si usa fare, abbiamo deciso di utilizzare il JSON Web Token per autenticare il sito. In questo modo possiamo identificare chi esegue le richieste migliorando la sicurezza. Questo fattore è molto importante in quanto il server è ospitato all'interno della rete scolastica che per sua natura è sensibile.

Questa impostazione implica che sia il client a forgiare il JWT tramite un modulo scritto in Rust e compilato in WebAssembly.

La firma del Json Web Token necessita di una chiave segreta condivisa tra client e server. Nel nostro applicativo la chiave non varia perchè abbiamo deciso di non utilizzare chiavi di sessione che sarebbero preferibili ma che richiederebbero l'implementazione di complessi meccanismi come l'algoritmo di Diffie-Hellman (https://it.wikipedia.org/wiki/Scambio_di_chiavi_Diffie-Hellman).

È di estrema importanza non salvarla in chiaro, pertanto abbiamo provveduto ad offuscarla per assicurarne la segretezza.

Offuscamento tramite WebAssembly

WebAssembly è uno standard che definisce un formato binario per la scrittura di codice veloce quasi quanto quello macchina eseguibile nelle pagine web.

È stato pensato per sopperire nelle parti critiche alle basse prestazioni di JavaScript e consentire lo sviluppo web in altri linguaggi di cui può essere il compilation target.

La chiave è salvata all'interno del codice compilato che rende il reverse engineering molto complesso. Questa però non rappresenta una soluzione definitiva.

L'analisi statica permette di osservare che all'interno del codice un valore rimane costante, che se interpretato correttamente può essere riconosciuto come chiave.

Per ovviare anche a questo problema la chiave non è salvata direttamente nel codice ma viene iniettata attraverso una macro Rust. Le macro vengono eseguite a compile-time e appartengono al metaprogramming^[25]. In pratica si tratta di codice che scrive altro codice. Nello specifico la macro combina due binari tramite XOR producendo la chiave.

Questo è possibile grazie alle proprietà dell'operazione XOR:


```
const key = 64;
const bytes = 16;

const result = key ^ bytes //"^" rappresenta l'operatore XOR in JS
console.log(key ^ bytes); //80

console.log(result ^ bytes); //64
```

Queste procedure non assicurano la totale segretezza della chiave ma mitigano notevolmente la possibilità di subire attacchi.

WebSockets

Panoramica sui WebSockets

I WebSockets sono il principale metodo di comunicazione bidirezionale supportato dai moderni browsers in alternativa al polling^[26].

Il protocollo permettere la trasmissione di dati in tempo reale senza che sia il client a richiederli.

WebSocket opera a livello 7 dello stack ISO/OSI e si basa su TCP, non direttamente utilizzabile dai browser, a livello 4.

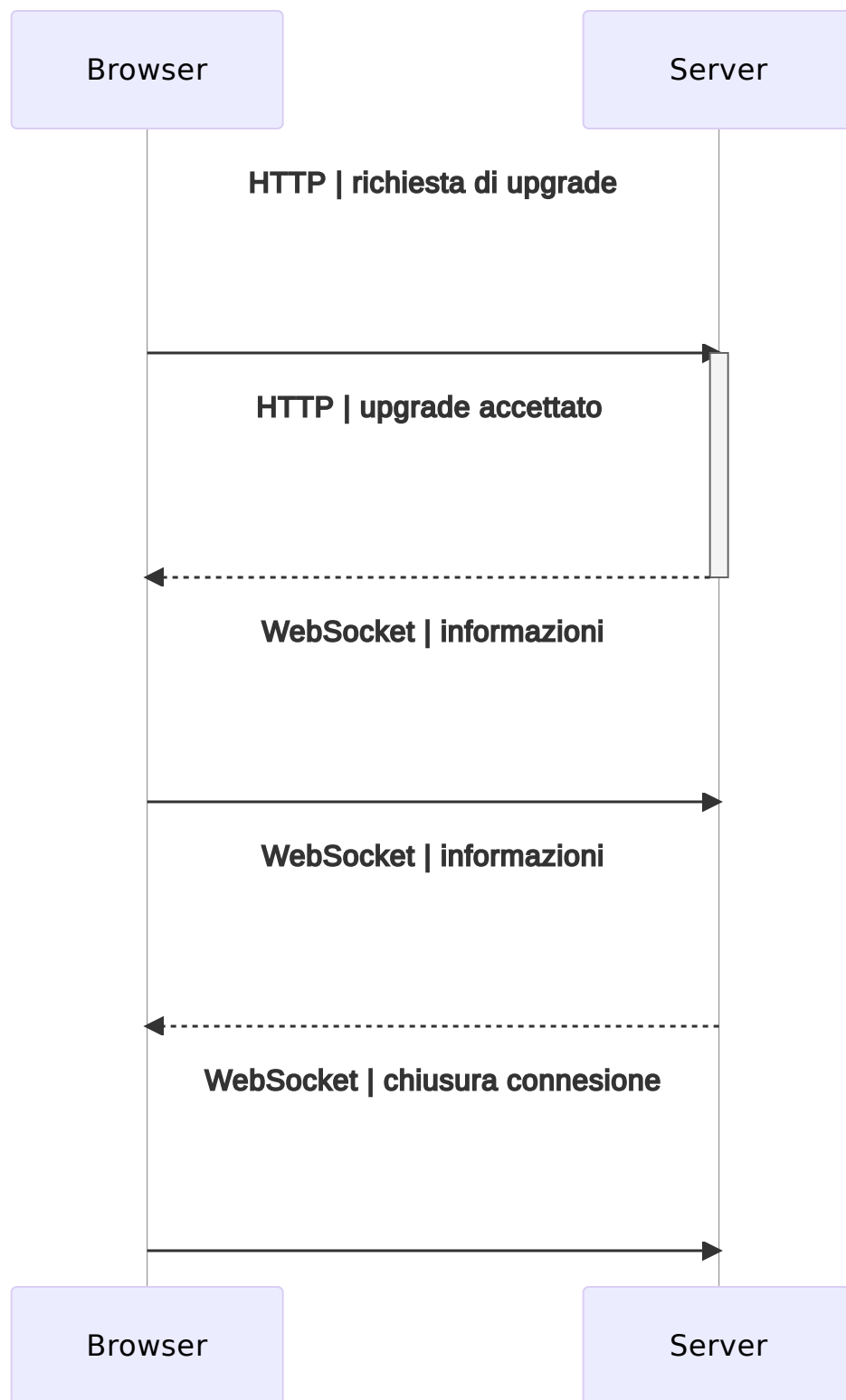
Il protocollo operando a livello applicativo offre maggiore praticità. I dati ricevuti rappresentano sempre l'intero messaggio, mentre TCP non fornisce alcuna garanzia riguardo i dati che potrebbero rappresentare l'intero messaggio oppure una parte di esso.

La comunicazione viene avviata tramite una richiesta HTTP contenente uno speciale header:

```
1 | Connection: Upgrade
2 | Upgrade: websocket
```

Se il destinatario accetta l'upgrade la comunicazione procede utilizzando il nuovo protocollo.

Trasmissione WebSocket



Utilizzo

I WebSockets rappresentano un'ottima soluzione per l'aggiornamento della tabella degli ordini nel pannello di amministrazione.

Il basso volume di traffico ci permetterebbe di utilizzare il polling ma risulterebbe una

soluzione poco elegante e scalabile.

Implementazione

Codice

Abbiamo creato su Github un'organizzazione per pubblicare e raggruppare tutte le repository sia di front-end (<https://github.com/SpaceNightstands/Waiter-client>) che back-end (<https://github.com/SpaceNightstands/Waiter-server>) in modo semplice.

Le repository possono essere esplorate al seguente indirizzo: Organizzazione (<https://github.com/SpaceNightstands>).

Funzioni

Arrow Functions

Le arrow functions, chiamate anche funzioni anonime o lambda, vengono spesso usate per comodità o in caso si utilizzi il paradigma funzionale.

Esempio

```
1  const materials = [  
2    'Oro',  
3    'Argento',  
4    'Rame',  
5    'Cristallo'  
6  ];  
7  
8  console.log(  
9    materials.map((material) => {  
10      material.length //output: Array [3, 7, 4, 9]  
11    })  
12  );
```

```
1  (parameters) => { code }  
2  
3  parameter => code  
4  
5  (parameter1, parameter2) => { return parameter1 + parameter2 }
```

Le funzioni anonime seguono una sintassi particolare. Non è obbligatorio utilizzare la keyword `return` e non è sempre necessario l'uso delle parentesi tonde e graffe.

Const Functions

```
1  const handleExpandClick = (id: number) => {  
2      setId(id);  
3      setExpanded(!expanded);  
4  };
```

```
1  //in JS scrivere  
2  function foo(){}  
3  //equivale a scrivere  
4  let foo = () => {} //foo è una variabile mutabile  
5  
6  const foo = () => {} //foo è una costante  
7  //const costringe il run-time a trattare la funzione come una costante
```

Questa sintassi è preferibile in quanto rende il codice più elegante, evita errori e cattive pratiche come utilizzare una funzione prima della sua dichiarazione.

Snippets di codice rilevante

```

1  export default function App({ Component, pageProps }: AppProps) {
2    if(process.env.NODE_ENV === 'development'){
3      React.useEffect(() => {
4        // Remove the server-side injected CSS.
5        const jssStyles = document.querySelector('#jss-server-side');
6        if (jssStyles && jssStyles.parentElement) {
7          jssStyles.parentElement.removeChild(jssStyles);
8        }
9      }, []);
10   }
11
12   return (
13     <React.Fragment>
14       <Head>
15     </Head>
16     <ThemeProvider theme={responsiveFontSizes(theme)}>
17       <SnackbarProvider maxSnack={3}>
18         <CssBaseline />
19         <Component {...pageProps} />
20       </SnackbarProvider>
21     </ThemeProvider>
22   </React.Fragment>
23   );
24 }

```

Questo componente rappresenta l'elemento root. Next.js utilizza il componente App per inizializzare le pagine. Esse sono in questo caso rappresentate da `<Component {...pageProps} />`. Gli altri tag importanti sono invece i cosiddetti Providers. Essi si basano sulla feature Context di React. Vengono utilizzati per passare ai loro figli dei valori come per esempio il tema: `<ThemeProvider theme={responsiveFontSizes(theme)}>`. Proprio per questa loro proprietà vengono utilizzati nel componente App che rappresenta la root assoluta dei nodi.

```

1  import { useAPI, API as WaiterAPI, Product, Error, isError } from "../lib/
2  //... codice ...
3  const [api, setAPI] = React.useState<WaiterAPI | Error>();
4  const [menu, setMenu] = React.useState<Product[] | Error>();
5  //... codice ...
6  React.useEffect(
7      api === undefined ? () => {
8          if(typeof profile === 'object'){
9              useAPI(profile).then(setAPI);
10         } else if(typeof profile === 'string') {
11             router.replace("/")
12         }
13     } : () => {
14         if (menu === undefined && !isError(api)) {
15             api.getMenu().then(setMenu);
16         }
17         else if(isError(api)){
18             enqueueSnackbar(<Typography>Impossibile caricare la lista prodot
19                 preventDuplicate: true,
20                 variant: 'error',
21                 autoHideDuration: 2000,
22             });
23         }
24     }
25 );
26 // ... codice ...
27 <TabPanel value={value} index={0} style={classes.tabPanel}>
28     {
29         menu.filter((item: Product) => item.kind === "available")
30         .map((item: Product) =>
31             <ProductCard key={item.id} product={item} style={ProductBarStyle}
32                 <Collapse in={expanded && item.id == id} timeout="auto" unmount(
33                     <CardContent>
34                         {item.ingredients && <Ingredients ingredients={item.ingrec
35                     </CardContent>
36                 </Collapse>
37             }>
38                 <Button variant="contained" color="secondary" size="medium" onCl
39                 <ExpandButton expanded={expanded} itemId={item.id} selectedId={i
40             </ProductCard>
41         )
42     }
43 </TabPanel>

```

L'interfaccia dell'API espone i metodi utilizzati all'interno del codice.

Si può osservare come viene implementata la pagina bar. Durante il caricamento della

stessa si ottiene il menù dopo aver controllato che le API siano raggiungibili.

Ottenuto il menù rappresentato da un array di prodotti questi vengono filtrati per categoria e successivamente visualizzati nella pagina.

Versionamento | Sviluppo collaborativo | Git (<https://git-scm.com/>) | Github (<https://github.com/>)

Fin dall'inizio abbiamo deciso di utilizzare Git, un software creato da Linus Torvalds creatore del kernel Linux.

Questo strumento permette di versionare il codice tramite i commit. Ognuno di essi rappresenta uno stato della code base, una modifica. Si ottiene così uno storico delle modifiche che può essere usato per tornare ad uno stato precedente se necessario.

Si parla di sviluppo collaborativo in quanto Git offre gli strumenti per gestire i conflitti tra versioni che generalmente nascono quando più persone lavorano sullo stesso codice e ottenere una versione univoca del codice che si sta sviluppando.

Altrettanto importante è il concetto di repository che indica il luogo in cui tutte le modifiche vengono salvate. Solitamente si usa un server remoto spesso munito di un'interfaccia web per monitorare lo stato del repository.

Inizialmente abbiamo utilizzato un repository privato attraverso Gitea (<https://gitea.io/en-us/>).

Successivamente abbiamo deciso di spostare il codice sul più famoso e soprattutto pubblico Github.

Sottolineo che Git è un software molto complesso le cui potenzialità andrebbero approfondite.

Deploy

Nella prima fase di sviluppo ho utilizzato Vercel (<https://vercel.com/>), un servizio per il deploy di applicativi che supporta Next.js e il deploy automatico inizialmente molto utile.

Il deploy automatico consiste nella messa in produzione di ogni commit consentendo di rilevare immediatamente errori che in modalità development difficilmente vengono notati.

Rappresenta un'ottima scelta se si intende applicare la Continuous Integration.

Fin da subito ho dovuto necessariamente familiarizzare con gli env file e prevedere comportamenti differenti per le modalità production e development.

Ad esempio durante il testing è stato importante potere accedere a tutte le pagine senza effettuare il login.

Testing

Quando abbiamo raggiunto un prototipo soddisfacente abbiamo mostrato il software agli effettivi utilizzatori: gli studenti ed i gestori del bar.

Abbiamo riscontrato alcune mancanze che si sono tramutate in richieste. Questo dimostra l'importanza di confrontarsi con gli utenti in quanto i programmatori possono ignorare necessità o problematiche specifiche.

Le correzioni che ci sono state chieste dai gestori sono le seguenti:

- Risolvere il problema degli omonimi
 - Una visione esterna ha evidenziato un problema piuttosto ovvio. Fino ad allora un ordine veniva identificato solo con il nome dello studente.
 - Soluzione
 - Gli ordini vengono contrassegnati mediante nome, cognome ed ID dell'account dello studente
- Rendere facoltativo o variabile il numero massimo di ordinazioni per ogni articolo
 - Alcuni articoli come le lattine di Coca-Cola sono limitati. I ristoratori hanno la necessità di impostare il numero di pezzi disponibili giornalmente. In altri casi questo valore non è necessario, ad esempio per i prodotti su ordinazione.
- Impostare un orario limite entro cui inviare gli ordini
 - Preparare le ordinazioni richiede tempo. Alcuni studenti consegnano la prenotazione eccessivamente a ridosso della ricreazione, mettendo il bar in difficoltà.
- Indicare il numero di ordinazioni per singolo prodotto nella pagina degli ordini
 - Ai ristoratori è utile conoscere il numero totale di pezzi da preparare per singolo articolo.
 - Soluzione
 - Un grafico a barre permette di conoscere il numero di ordini per singolo prodotto
- Permettere di indicare lo stato di un ordine: pagato o evaso
 - Tenere traccia di tutti i pagamenti è complesso soprattutto perchè è possibile pagare alla consegna. È quindi necessario prevedere un modo per indicare se un ordine è stato pagato o meno prima di passarlo in stato evaso.
 - Soluzione
 - Gli operatori possono contrassegnare un ordine come pagato e rimuoverlo dalla lista degli ordini quando viene evaso.

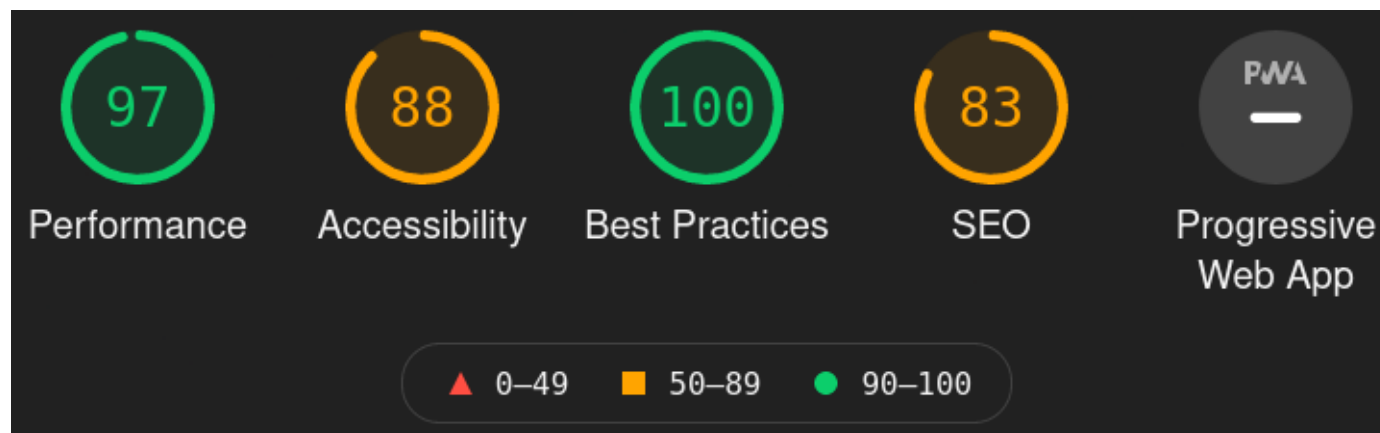
Gli studenti ci hanno fatto notare che quando si preme sul pulsante acquista non c'è alcun feedback che indichi se l'elemento è stato aggiunto o meno al carrello.

Performance

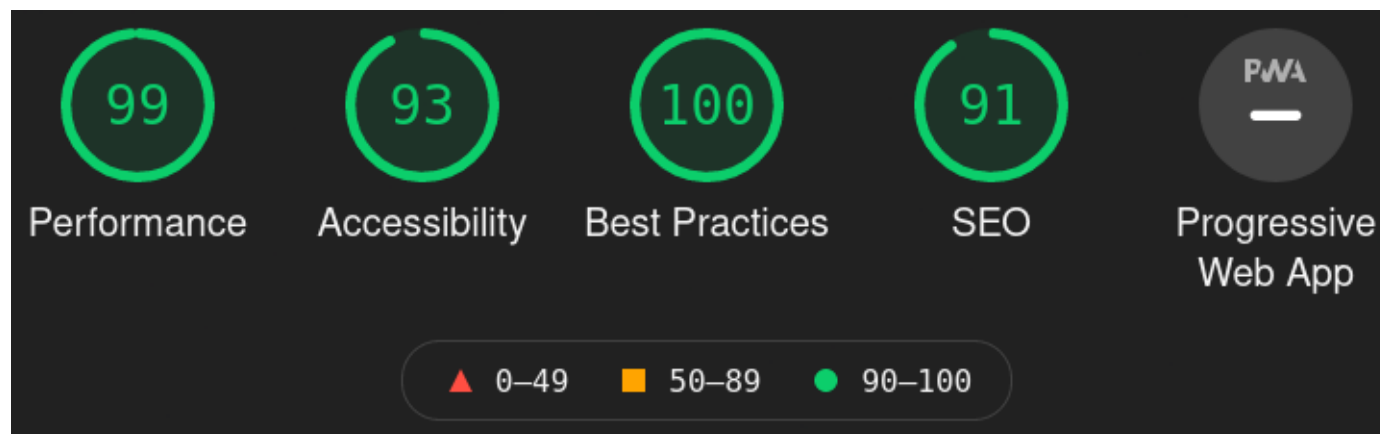
Il sito è stato analizzato tramite Lighthouse, un software Google per l'analisi delle performance dei siti web.

La procedura è completamente automatizzata, è solamente necessario un URL come input. In seguito viene generato un report delle performance con eventuali raccomandazioni sulle componenti da migliorare e le cause delle performance negative.

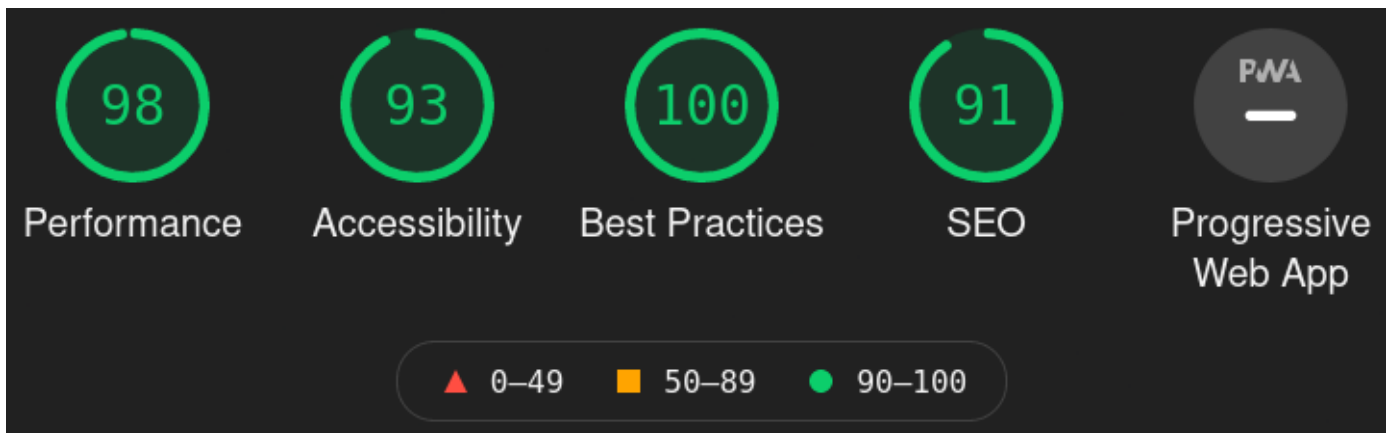
Il report dell'applicativo è il seguente:



Index



Bar



Admin

In generale il report è ottimo. L'assenza di metadati non permette di ottenere un alto valore SEO, il che non rappresenta un problema nel nostro progetto.

Manutenzione

Il progetto è stato pensato fin da subito per essere Open Source.

Con Open Source si indica un tipo di software la cui licenza prevede che i detentori dei diritti promuovano la modifica, lo studio, l'utilizzo e la redistribuzione del codice sorgente. La caratteristica fondamentale è quindi la pubblicazione del codice.

Per ragioni etiche ma non solo abbiamo scelto questa modalità di sviluppo. L'idea è sempre stata quella di poter affidare il progetto ai futuri studenti che avrebbero potuto mantenere il codice ed espanderlo. Le tecnologie scelte per il front-end sono battle-tested e molto famose quindi non richiedono conoscenze di nicchia. L'insieme di questi fattori dovrebbe garantire longevità al progetto.

Considerazioni aggiuntive

Autenticazione del sito

L'autenticazione del sito può sembrare una buona idea ma pone diversi problemi.

Il sistema attuale limita notevolmente sviluppi futuri. Al momento non sarebbe possibile creare un'app per cellulare in quanto solo il sito avrebbe accesso al server. Questo non è in contrasto con i principi dell'open source ma di fatto riduce l'espandibilità da parte di terzi.

Inoltre l'attuale livello di sicurezza è forse inferiore a quello che si potrebbe ottenere applicando rigidi controlli lato server.

EditorConfig

L'EditorConfig è un file in cui vengono descritte le regole del progetto che gli sviluppatori dovrebbero rispettare.

Lo scopo è quello di avere codice dallo stile e dalla formattazione consistenti.

Ovviamente si integra con Git e rappresenta un'ottima risorsa per lo sviluppo collaborativo.

L'esempio mostra come vengono codificate le regole.

```
root = true //root EditorConfig file

//Override dell'indentazione per tutti i file
[*]
end_of_line = lf //Unix-style newline.
insert_final_newline = true //Ogni pagina termina con un newline

//Override dell'indentazione per tutti i file python
[* .py]
indent_style = space //Indentazione tramite spazi
indent_size = 4 //Indentazione di 4 spazi

//Override dell'indentazione per tutti i file JS contenuti nella cartella lib
[lib/**/*.js]
indent_style = space
indent_size = 2
```

Il progetto non presenta alcun EditorConfig che andrebbe però definito a vantaggio della mantenibilità del progetto.

Internazionalizzazione

Attualmente il sito è disponibile solo in lingua italiana. Dato il dominio di utilizzo molto ristretto ciò non rappresenta un problema.

Predisporre il codice all'internazionalizzazione sarebbe anche utile per la gestione del testo che sarebbe centralizzato in un file diventando così più semplice da modificare. Attualmente è possibile aggiornare il testo solo modificandolo nel codice.

Le librerie apposite permettono inoltre una formattazione più consistente.

La localizzazione viene chiamata i18n. Sono disponibili librerie apposite in grado anche di sfruttare i vantaggi di Next.js.

Implementare questa feature rappresenterebbe un'ottima prima issue. Un nuovo collaboratore sarebbe costretto ad esplorare tutto il codice per implementare la localizzazione familiarizzando con la code base.

Il debito tecnico

Per debito tecnico si intendono le complicazioni derivanti da uno sviluppo di software di scarsa qualità. È importante scrivere codice studiato accuratamente, facilmente mantenibile e dalla bassa complessità.

Il termine debito viene preso in prestito dal mondo finanziario in cui per sanare un debito occorre pagarne anche gli interessi.

Lo stesso meccanismo si riflette sul codice sviluppato con scarso impegno quando si deve modificarlo. Il debito tecnico aumenta nel tempo ed è bene estinguerlo il prima possibile o comunque mantenerlo a livelli minimi.

Debito nel nostro progetto

Sia io che il mio compagno ci siamo resi conto di aver fatto scelte errate durante lo sviluppo del progetto.

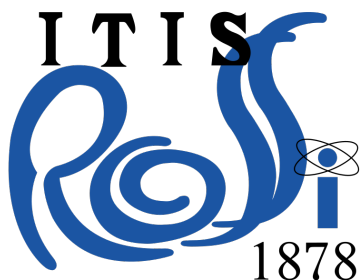
Lui ha capito in fase avanzata che avrebbe dovuto scrivere il back-end in un linguaggio più mantenibile e supportato.

Per quanto mi riguarda ho subito le conseguenze di uno sviluppo frettoloso dettato dal fatto che l'applicativo sembrava di necessità immediata. Puntavo a realizzare un prodotto funzionante da migliorare successivamente.

Ho utilizzato codice scritto da altri e/o che non capivo completamente.

L'insieme di questi fattori ha determinato una code base difficile da leggere ed espandere.

Ho pagato con gli interessi il mio debito che mi ha costretto ad una riscrittura quasi totale dell'applicativo.



Istituto Tecnico Industriale Statale

Alessandro Rossi

Anno Scolastico 2020/2021

Enrico Dalla Croce 5BII

enricodallacroce@gmail.com

1. Questa sezione raccoglie vocaboli e/o concetti che potrebbero necessitare una breve spiegazione. ←
2. JS: JavaScript ←
3. TS: TypeScript ←

4. JSX: JavaScript XML ↔
5. ECMAScript è lo standard di un linguaggio di scripting. L'implementazione più conosciuta è JavaScript. La specifica garantisce l'interoperabilità delle pagine web tra browser diversi. ↔
6. La compilazione just-in-time viene eseguita durante l'esecuzione. Tipicamente quello che accade è la traduzione in codice macchina di un linguaggio intermedio detto bytecode. L'engine JS più famoso è V8^[15]. ↔
7. I paradigmi sono stili fondamentali di programmazione. Costituiscono un insieme di strumenti concettuali che definiscono il modo in cui il programmatore concepisce la stesura del codice ed il programma. ↔
8. Il paradigma imperativo prevede che il programmatore sia in grado di dare istruzioni alla macchina su come cambiare il suo stato in modo procedurale o orientato agli oggetti. ↔
9. Il paradigma funzionale prevede che le istruzioni siano date per mezzo dell'applicazione e composizione di funzioni. Per funzioni si intendono espressioni che mappino valori ad altri valori e non una sequenza di istruzioni che modifichino lo stato della macchina. ↔
10. Il paradigma event-driven si basa sul fatto che il flusso del programma sia determinato da eventi come le azioni dell'utente. Questa modalità è comune in interfacce grafiche e applicazioni come le pagine web che sono focalizzate a compiere azioni in risposta all'input dell'utente. ↔
11. Lo scopo di un type system dinamico è quello di esonerare il programmatore dallo specificare il tipo di dato che vuole manipolare. Il compilatore o il parser capiranno al suo posto di che tipo di dato si tratta. Questa modalità comporta performance inferiori e altre problematiche ma permette di scrivere codice più facilmente e in modo più dinamico. ↔
12. Si dice che un linguaggio implementa le first-class functions quando le funzioni sono trattate come first-class citizens^[16]. ↔
13. La programmazione orientata agli oggetti (Object Oriented Programming, abbreviato OOP) è un paradigma basato sull'interazione tra oggetti astratti contenenti dati chiamati attributi o proprietà e codice sottoforma di metodi. ↔
14. Il Prototype-based programming è un'estensione dell'OOP in cui l'ereditarietà è implementata tramite il riutilizzo di oggetti già esistenti che svolgono il ruolo di prototipi. Esempio:

```
var foo = {name: "foo", one: 1, two: 2};  
var bar = {two: "due", three: 3};  
  
Object.setPrototypeOf(bar, foo); // foo é ora il prototipo di bar.  
  
bar.one // ritorna 1.  
bar.three // ritorna 3.  
bar.name (http://bar.name); // ritorna "foo"
```

↩

15. V8 è un JS engine open-source utilizzato per i web browsers basati su Chromium (<https://www.chromium.org/>) e più in generale per l'esecuzione di codice JS. V8 è utilizzato da Node.js (<https://nodejs.org/it/>) e Deno (<https://deno.land/>). ↩
16. Un first-class citizen in programmazione è un'entità che supporta tutte le operazioni applicabili alle altre entità. Questo significa che può essere passato come argomento, ritornato come valore, modificato e assegnato ad una variabile. ↩
17. Il type system statico si basa sul controllo statico dei tipi. I tipi e le loro interazioni sono controllati prima dell'esecuzione. Se il codice supera la fase di controllo non ci saranno errori relativi ai tipi durante l'esecuzione. ↩
18. La code base rappresenta la collezione dell'intero codice sorgente utilizzato per lo sviluppo di un software. ↩
19. Per fase di debug o debugging si intende la pratica di individuazione e correzione degli errori rilevati nel codice. ↩
20. Un Debugger è un programma progettato per l'analisi e l'eliminazione dei bug, cioè errori di programmazione interni al codice. ↩
21. L'IntelliSense è una tecnologia che permette il completamento automatico. Viene anche usata per fornire una rapida documentazione, ad esempio per visualizzare la descrizione delle funzioni. ↩
22. Durante la compilazione del codice viene effettuato il bundle, ovvero viene raccolto il sorgente presente nei file importati e viene inserito in un unico nuovo file. Si può così racchiudere una pagina web in unico file che necessita di un solo caricamento. Questa feature non comporta solo vantaggi. Infatti l'aggiunta di nuovo codice e librerie aumenta il peso del bundle sino al punto in cui caricarlo comporterà un rallentamento della pagina.
Per evitare che questo accada viene effettuato il code-splitting cioè la divisione del

bundle in altri più piccoli. ↵

23. Search Engine Optimization (https://en.wikipedia.org/wiki/Search_engine_optimization). Gli engine di ricerca come Google valutano anche la velocità di caricamento durante il ranking dei siti. ↵

24. UUID: Universally unique identifier è un identificativo basato su un sistema distribuito che permette l'identificazione ma non la garantisce.

Il numero finito delle sue combinazioni implica la possibilità che ci siano collisioni. La probabilità che avvenga una collisione dipende dalla sua lunghezza e dall'algoritmo usato.

Nonostante ciò può essere usato con sufficiente probabilità che esso sia unico. ↵

25. Il metaprogramming è una tecnica di programmazione in cui i software utilizzano altri programmi come dati su cui operare.

Un programma ne può quindi analizzare e trasformare un altro o modificare se stesso durante l'esecuzione. ↵

26. Il polling consiste nel campionamento attivo dello stato di un dispositivo esterno. Un esempio è quello di un client che effettua continuamente chiamate all'API per assicurarsi di avere la versione più aggiornata dei dati. ↵