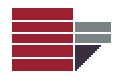
**Università della Calabria**

**Dipartimento di Matematica e Informatica**



**Corso di Laurea Triennale/Magistrale in Informatica**

Tesi di Laurea

Estensioni avanzate di un

IDE web per formalismi logici

|  |  |  |
| --- | --- | --- |
|  | Relatori | Candidato |
|  | Prof. Francesco Calimeri | Carmine Blefari |
|  | Dott. Stefano Germano | Matricola 213535 |

Anno Accademico 2019/2020

INDICE

**Introduzione** ………………………………………………………………………..3

**Il Progetto LoIDE** ………………………………………………………………….4

**Capitolo 1** **Tutorial Iniziale** ……………………………………………………8

**Capitolo 2** **Esempi** …………………………………………………………….13

**Capitolo 3** **Editor Collaborativo** ……………………………………………..21

**Conclusioni** ……………………………………………………………………….29

**Ringraziamenti** …………………………………………………………………..32

**Bibliografia** ………………………………………………………………………33

1. **INTRODUZIONE**

Il lavoro di tesi svolto presso il dipartimento di Informatica dell’Università della Calabria nasce con l’idea di apportare modifiche al progetto LoIDE, un IDE web per la programmazione logica.

Il contesto in cui si colloca il lavoro di tesi coinvolge L’Editor di LoIDE, utilizzato per progettare codice di intelligenza artificiale con approcci deduttivi al fine di produrre regole e fatti che permettano di risolvere problemi reali.

L’Editor di LoIDE mette a disposizione diverse funzionalità per agevolare l’utente nella realizzazione di diversi progetti; tuttavia, analizzando le esigenze mutate nel corso degli anni e dell’esperienza diretta sull’Editor, è sorta la necessità di attuare ulteriori modifiche.

Il lavoro centrale è stato svolto sull’Editor che mette a disposizione LoIDE ed il linguaggio principalmente utilizzato è TypeScript[[1]](#footnote-1).

Gli obiettivi principali della tesi fin dall’inizio sono stati quelli di imparare a contribuire su un medio/grosso progetto e relazionarsi con altri contributori e colleghi attraverso il servizio GitHub[[2]](#footnote-2) e Trello[[3]](#footnote-3).

Quindi è stato necessario imparare a lavorare su nuovi software con l’intento di interfacciarsi al mondo del lavoro aggiungendo al proprio curriculum quelle soft skills che si apprendono solo con l’esperienza e la costanza.

La necessità di apportare modifiche al progetto si concretizza dopo aver studiato le caratteristiche principali dell’IDE, ed il lavoro di seguito programmato è volto non ad aggiungere futili funzionalità per il solo intento di riempire l’Editor, ma a contribuire alla realizzazione di funzionalità che possano soddisfare qualsiasi tipologia di utente e andare di pari passo con gli Editor più utilizzati.

Il progetto quindi risulta essere aperto a qualsiasi forma di discussione su funzionalità che possono essere implementate al fine di rendere l’esperienza dell’utente migliore, quindi è stato possibile fin da subito dialogare e avere uno scambio di idee interessanti che hanno agevolato il contributo.

In questo lavoro di tesi si andranno a visionare nel dettaglio i seguenti capitoli:

* Capitolo 1: *Tutorial Iniziale*
* Capitolo 2: *Esempi*
* Capitolo 3: *Editor Collaborativo*

Si conclude l’introduzione mettendo in evidenza l’utilità di una tesi al fine di immedesimarsi in un sistema lavorativo già con una struttura non banale, poiché dalle difficoltà iniziali a causa della scarsa conoscenza di quelle soft skills sopra citate si arriva ad un punto finale in cui si è pronti per relazionarsi al mondo del lavoro.

1. **IL PROGETTO LoIDE**

In questo capitolo viene analizzato nel dettaglio il web IDE LoIDE.

Un IDE è un ambiente di sviluppo integrato, cioè un software che viene progettato per realizzare applicazione in modo tale da mettere insieme degli strumenti comuni di sviluppo e fornire un'unica interfaccia grafica all’utente.

In questa maniera si accorciano i tempi di apprendimento di tutte le funzionalità offerte (nel caso di LoIDE) dall’Editor e si agevola l’utente nell’apprendimento e nella praticità di utilizzo.

Per programmazione logica invece si fa riferimento ad un paradigma di programmazione che enfatizza l’uso del ragionamento logico per risolvere i problemi.

Viene utilizzato di seguito un modello di programmazione dichiarativo e dichiarazioni logiche per rappresentare fatti e regole che consentono al sistema di capire automaticamente come risolvere il problema.

La programmazione logica è usata principalmente nell’intelligenza artificiale e nel machine learning.

Inoltre, LoIDE è un progetto open source, ovvero un codice progettato per essere disponibile pubblicamente, rilasciato sotto licenza MIT e disponibile su GitHub.

LoIDE mette a disposizione un Editor in grado di interpretare formalismi logici e fornire all’utente un output di risposta alle regole e ai fatti inseriti.

L’Editor di LoIDE è stato sviluppato e implementato attraverso l’utilizzo di una tecnologia principale: TypeScript.

TypeScript è un linguaggio di programmazione open source sviluppato da Microsoft, nello specifico può essere descritto come un superset di JavaScript, ovvero vengono aggiunti tipi, classi e interfacce allo JavaScript tradizionale.

Nella sostanza può essere descritto come un’estensione di JavaScript.

Il motivo principale dell’utilizzo del TypeScript in LoIDE lo si può riscontrare in uno dei suoi punti fondamentali: esso è un linguaggio tipizzato, ovvero aggiunge definizioni di tipo statico.

Questi tipi consentono di descrivere la forma di un oggetto e permettono al TypeScript di verificare che il codice funzioni correttamente.

Come visto in precedenza, LoIDE offre un’interfaccia con molte funzionalità e quindi risulta di fondamentale importanza tipizzare queste ultime per snellire il codice, renderlo più intuitivo e funzionale possibile.

Inoltre, una caratteristica rilevante è che il JavaScript risulta essere compatibile con la semantica e la sintassi del TypeScript, quindi i due linguaggi possono essere utilizzati insieme nello sviluppo di progetti.

Prima di parlare però delle modifiche apportate è necessario avere una panoramica più amplia del funzionamento dell’Editor di LoIDE, focalizzandosi sulle caratteristiche utili ad affrontare gli argomenti esposti nella tesi.

Nel progetto LoIDE il concetto fondamentale introdotto è quello della Tab, ovvero una pagina o una componente dell’Editor che contribuisce a svolgere attività differenti.

L’Editor si presenta con una Main Tab, che contiene le caratteristiche principali, le funzionalità sviluppate finora, e l’area di lavoro dedicata.

Su questa Main Tab viene svolto la maggior parte del lavoro.

Essa rappresenta la pagina principale che permette all’utente di svolgere diverse attività, tra cui: aprire progetti, condividere progetti, eliminare progetti, apportare modifiche al proprio progetto che si sta sviluppando.

Nell’Editor inoltre sono presenti altre Tab, ciascuna con ruoli precisi e diversi:

* La Tab Run Settings.
  + che contribuisce a selezionare e configurare i parametri di esecuzione del programma logico.
* La Tab Output.
  + che permette di visualizzare il contenuto ottenuto dall’esecuzione del progetto nella Main Tab.
* La Tab Appearance.
  + che mette a disposizione diversi settaggi per modellare dal punto di vista estetico l’Editor.
* La Tab About.
  + che invece ha lo scopo di fornire all’utente informazioni su LoIDE.

Le modifiche principali apportate al progetto LoIDE riguardano:

* Un tutorial iniziale con l’obiettivo di rendere più interattiva l’esperienza di un nuovo utente al suo primo utilizzo dell’Editor, e che implementi una sorta di guida conoscitiva di quest’ultimo.
* La funzionalità degli esempi, in modo tale da mettere a disposizione dell’utente nell’Editor principale del codice già pronto, da cui prendere esempio e con il quale avere un primo impatto sulle funzionalità principali.
* Una funzionalità collaborativa, che permette a due o più utenti di lavorare in contemporanea sullo stesso progetto.

1. **TUTORIAL INIZIALE**

Il *Tutorial Iniziale* nasce con l’idea di spiegare agli utenti come funziona LoIDE, attraverso un metodo di interazione e un linguaggio semplice e comprensibile.

Un Editor come LoIDE è un programma usato per creare file di testo e modificarne il contenuto, di conseguenze le funzionalità offerte sono molteplici e quindi è importante considerare l’eventualità che gli utenti possano interpretare queste funzionalità in maniera scorretta e differente, o utilizzarle per scopi che non sono conformi ai motivi che hanno generato la necessità di importare una determinata funzionalità nel programma.

Diventa necessario offrire agli utenti le motivazioni per le quali un team ha aggiunto una funzionalità, rendere chiaro ed evidente quali sono queste funzionalità ma soprattutto come utilizzarle al meglio possibile.

Quindi, dopo un lavoro di ricerca su altri Editor già esistenti e dopo una serie di condivisioni di idee con gli altri contributori al progetto, si è deciso che un *Tutorial Iniziale* era un qualcosa di necessario al fine di accogliere qualcuno sull’Editor nella miglior maniera possibile.

In questo periodo di ricerca sono stati considerati Editor che fossero il più simili possibile a LoIDE in modo tale da prendere in considerazione esempi validi e che hanno già ricevuto una risposta positiva da parte degli utenti.

Ma nel complesso, immedesimandosi nei panni di un utente che affronta per la prima volta un Editor, sono sempre diversi i dubbi riguardo alcune caratteristiche.

Di conseguenza si cerca di andare per intuizione, o di fare mente locale su Editor visti in precedenza.

L’intento di LoIDE è quello di evitare di creare confusione in un utente.

Durante la fase di analisi di questa funzionalità sono state considerate le più svariate possibilità di implementazione.

Infatti, è stato possibile constatare durante questo lavoro di tesi che l’idea generale fissata inizialmente può cambiare in qualsiasi momento, in seguito al dialogo e al confronto con i membri del team di sviluppo, ma anche grazie a constatazioni palesi dopo tentativi di approccio falliti.

Il problema principale riscontrato durante l’implementazione di questa funzionalità non è stato riscontrato nello sviluppo di quest’ultima, bensì nella fase iniziale.

Il *Tutorial Iniziale* inizialmente è stato immagino come inseribile all’interno di LoIDE attraverso un video tutorial iniziale, che avrebbe spiegato nella pratica di pochi secondi a video quale è il ruolo delle funzionalità offerte dall’Editor.

Tuttavia, questo risultava essere troppo arretrato con l’idea portata avanti dal progetto LoIDE e tuttavia un po’ pesante per un utente.

Si è cercato quindi di pensare ad un metodo che coinvolgesse l’utente nell’utilizzo di questa funzionalità, anche attraverso semplici passaggi, ma che almeno non gli designasse un ruolo marginale.

Alla fine, tutte le esigenze e i dubbi sorti sono stati annullati attraverso un Tool particolare che soddisfava tutti.

La funzionalità è stata implementata con l’intento di offrire uno Step-by-Step tutorial che fosse il più interattivo possibile.

Da qui nasce l’esigenza di introdurre nel progetto il Tool Intro.js[[4]](#footnote-4) per creare un vero e proprio tour Step-by-Step all’utente.

La scelta di Intro.js non è stata banale, ma dettata anche dalla sua facilità di utilizzo, dalla possibilità di customizzare il prodotto finale in base alle esigenze e infine poiché conforme alla politica di utilizzo e commercializzazione di LoIDE.

Per ogni step è stato possibile mettere a disposizione una descrizione delle singole funzionalità e abilitare il prosieguo di questo tutorial potendo tornare indietro anche in caso fosse sfuggito qualcosa.

Alla fine, si dà la scelta di non mostrare più questa funzionalità e proseguire nel programming.

Per l’implementazione del *Tutorial Iniziale* è stato necessario introdurre il package Intro.js nel progetto.

Una volta importato il tutto si è passati a modellare il le funzionalità di Intro.js in risposta alle nostre necessità:

* Lingua Inglese, scelta per estendere l’utilizzo di LoIDE ad ogni fascia di utenti.
* Accogliere l’utente, mostrando un primo riquadro di benvenuto e spiegando quali saranno i passi successivi.
* Identificare gli oggetti compresi nel tutorial, associando così nuovi id ai diretti interessati per richiamarli nell’implementazione del Tool.
* Inserire un Cookie[[5]](#footnote-5) per evitare che questo tutorial venga mostrato ad ogni accesso a LoIDE e dare la possibilità all’utente di non ripetere più il tutorial.

Lo stile del tutorial è stato modificato al fine di offrire un'interfaccia leggera e piacevole al cliente: per questo è stato necessario modificare il foglio di stile offerto dal Tool in base alle nostre esigenze.

Alla fine, il risultato ottenuto è il seguente:

|  |
| --- |
| https://lh6.googleusercontent.com/cpleDCUpp3heMa77Zav0UnSzhVeRGPRjOFj2qccpiUCIkJLJKwIIuMjPdwHRBFmz-4hKntxP4WT5zD1LHPuTPGabFfpMjYTIjct9plAUhb6OT969Dzg7lOs9N0CTqP4J1C-xuL8hcrdr4ThfIb1GFA |
| *Figura dimostrativa di uno step del Tutorial iniziale.* |

Come si può notare nell’immagine è presente una check box con la voce ‘Dont show this again’.

Questo ci permette di offrire all’utente la possibilità di abilitare questa voce una volta ci si senta sicuri di aver capito le funzionalità dell’Editor, così facendo si evita che il tutorial implementato venga presentato ogni volta che si acceda a quest’ultimo. Per assegnare tutti gli attributi necessari ad ogni step del tutorial si è passati all’implementazione della logica del Tool nel seguente modo.

{

element: ‘#resetButton’,

title: ‘Reset Button’,

intro:‘Very useful…’  
},

Ovvero richiamando l’oggetto a cui riferisce lo step corrente del tutorial, e assegnandogli un titolo e un corpo di testo.

Per richiedere l’eventuale implementazione del tutorial implementato, il parere del resto del team di sviluppo e la correzione di eventuali errori, è stata creata su GitHub una pull-request a riguardo.

Una skill chiave appresa durante lo sviluppo di questa funzionalità la si è raggiunti attraverso la pulizia del codice.

Grazie all’esperienza e alla conoscenza dei relatori è stato importante capire che bisogna evitare di appesantire il codice di una singola componente inserendo tutti gli elementi fondamentali che contribuiscono allo sviluppo di una funzionalità, bensì organizzare il lavoro in cartelle, con nomi che mandassero all’utilità della corrispondente, al fine di avere un progetto ben organizzato, pulito e semplice da intraprendere da un occhio esterno.

Un esempio palese è stato quello della definizione degli steps del *Tutorial Iniziale*, che prevede una funzione che richiama gli oggetti su da attribuire ad ogni step del tutorial, e che quindi risulta essere ripetitiva e lunga.

Lasciare che questa parte contribuisca a comporre il codice di una componente solo perché la funzionalità introdotta appartiene ad esso è ingombrante, quindi è stato creato un file .js (JavaScript[[6]](#footnote-6)) a parte all’interno del quale è presente questa funzione.

Così facendo si sfruttano al meglio le potenzialità del TypeScript e una volta importata la funzione la si può utilizzare come se facesse parte della componente stessa.

Anche la cura del design del Tutorial Iniziale non è stata casuale.

Importante è stato il confronto con il team di sviluppo affinché la funzionalità fosse implementata seguendo lo stile già applicato all’Editor.

Lo stile messo a disposizione dalla componente di Intro.js risultava infatti essere troppo astratto e andava sicuramente modificato. Per evitare di modificare ogni singola componente che componeva l’intero tutorial all’interno dei vari file dove essa veniva chiamata, è stato creato appositamente un file introjs.css, ovvero un foglio di stile che contiene tutte le caratteristiche che vanno d’accordo con l’aspetto già offerto dall’Editor.

1. **ESEMPI**

Una delle operazioni più diffuse tra gli utenti durante le loro prime esperienze su un Editor è quella di eseguire del codice semplice e intuitivo, spesso associato a esperienze della vita reale oppure a esempi comprensibili facilmente.

L’utente cerca di comprendere l’Editor o un nuovo linguaggio che sta utilizzando non approfondendo la documentazione ad esso associata, ma avendo un responso immediato ad una idea da lui concepita o, come nel nostro caso, a esempi già pronti da eseguire.

Studiando gli Editor già diffusi e analizzando soprattutto quelli che più rispecchiano LoIDE, le sue funzionalità e il suo scopo, è subito sorto il dubbio se una funzionalità di esempi già presenti e pronti all’esecuzione fosse necessaria e sufficiente.

La risposta ottenuta dalle diverse implementazioni di questa idea e da un’immersione lato utente nell’Editor LoIDE è stata positiva.

Nasce così l’idea di *Esempi*, una semplice funzionalità posta sull’Editor che permette all’utente di scegliere uno tra gli esempi proposti, visualizzarne il contenuto ed eseguirlo correttamente.

Alla base della scelta implementativa sussiste l’idea di rispecchiare lo stile già presente su LoIDE: ogni scelta di design e di funzionalità associata ad essa si sposa perfettamente con le altre funzionalità già presenti al fine di non alterare la familiarità del progetto e sfruttare al meglio ciò che si ha a disposizione.

Per implementare la funzionalità *Esempi* ovviamente non si è partiti dall’idea finale ben impostata, ma è stato necessario sciogliere alcuni dubbi ed effettuare prove che poi si sono rivelate insufficienti a risolvere l’intero problema.

L’idea iniziale era quella di creare una nuova Tab che si presentasse allo stesso modo del risultato finale poi raggiunto e successivamente sfruttare dei file già presenti nel nostro progetto per copiarli su un posto di comodo, in questo caso la Navigator.clipboard[[7]](#footnote-7) e poi permettere all’utente tramite un pulsante presente nella pagina principale dell’Editor di richiamare la funzione associata e visualizzare l’esempio scelto.

Come si può ben notare però lo sviluppo di questa opzione risulta essere molto complicato, non lineare con le idee già presenti su LoIDE e forse troppo confusionaria per un utente che magari potrebbe decidere di non affidarsi ad essa proprio per queste motivazioni.

Diventa quindi fondamentale capire come risparmiare operazioni al fine di ottenere un risultato anche più pulito dal punto di vista implementativo, e anche come rendere la funzionalità più accessibile ad un utente e più semplice da utilizzare.

Si decide così di basarsi sulle idee già presenti nell'Editor e di proseguire utilizzando il modello applicato nelle operazioni già presenti.

Dal punto di vista del design della funzionalità si abbatte la creazione di un’altra Tab e si passa a crea un nuovo popup nella pagina principale, mentre dal punto di vista di implementazione codice si evita di utilizzare un semplice copia e incolla ma si utilizza la funzionalità dell’Editor che permette di cambiare il valore della Tab corrente passando l’indice che riconosce su quale Tab stiamo lavorando e il nuovo valore da visualizzare su quest’ultima.

Per quanto riguarda il primo punto quindi si è passati a creare un nuovo modello, ovvero un nuovo componente che funziona come un popup che non va ad aumentare il carico finale del progetto e non va ad appesantire l’interfaccia con una nuova pagina.

Dai risultati inoltre risulta più soddisfacente e comodo per un utente utilizzare questa modalità piuttosto che recarsi su un’altra pagina ed eseguire le stesse operazioni.

Per aprire questo popup si utilizzano gli State di React[[8]](#footnote-8), ovvero attraverso lo stato di alcune variabili si può decidere se eseguire l’operazione di apertura e chiusura.

Inoltre, per essere coerenti con l’intero progetto è stato necessario abilitare anche l’operazione di utilizzo associata agli esempi tramite comandi di tastiera.

Di seguito viene riportata la scelta implementativa adottata.

useEffect (() => {

Mousetrap.bind(["ctrl+e", "command+e"], () =>

{

console.log(document.getElementById("main"));

setShowExamplesModal(!showExamplesModal);

return false;

});

}

Anche in questo caso si utilizzano gli State di React, nel particolare vediamo come l’utilizzo della variabile showExampleModal e che viene associata alla componente degli Esempi importata ci permette di decidere quando effettuare l’operazione di apertura o chiusura del popup digitando l’opportuno comando da tastiera.

Un ulteriore incentivo che ha spinto all'incorporata di questa funzionalità nell’Editor principale è rappresentato dal secondo punto espresso sopra.

Con una nuova pagina non si poteva accedere direttamente alle funzioni dell’Editor ma si dovevano o implementare nuove funzioni, o duplicare funzioni già presenti. In questa maniera invece si agisce direttamente sul diretto interessato.

La funzionalità si presenta su un popup che si apre cliccando sul pulsante aggiunto alla pagina principale dell’Editor.

Ecco un’anteprima di ciò che appare all’utente:

|  |
| --- |
|  |
| *Figura dimostrativa dell’utilizzo della funzionalità degli esempi prescelta* |

Per implementare questa funzionalità vengono sfruttati gli State di React, ovvero ad ogni esempio è associato uno stato che cambia in base alla scelta dell’utente.

Quando un utente seleziona un titolo di un esempio lo stato associato viene impostato a *vero* e di conseguenza si apre un'area sottostante che mostra la descrizione corrispondente ad un esempio e il relativo pulsante per la sua selezione.

Ecco il risultato di cambio di stato:

|  |
| --- |
|  |
| *Figura dimostrativa dell’utilizzo della funzionalità degli esempi post scelta* |

Quando viene selezionato un esempio tramite il pulsante “*Select* *Example*” basta chiudere il popup ed esso verrà mostrato sull’Editor:

|  |
| --- |
|  |
| *Figura dimostrativa dell’utilizzo della funzionalità degli esempi post scelta* |

Nel caso in cui si scelga un altro esempio da mostrare o nel caso in cui sull’Editor era presente già del codice questo viene sostituito con il nuovo esempio scelto, tutto attraverso la funzione utilizzata.

Per gestire la scelta di un esempio e quindi capire quando e come caricare il contenuto dell’esempio scelto nell'Editor è stato necessario adottare l’utilizzo delle variabili globali.

Quindi è stato creato un file con estensione .d.ts che contiene una variabile globale.

Il file con estensione .d.ts è chiamato file di dichiarazione del tipo, il cui unico scopo è quello di descrivere la forma di un modulo esistente e contenere informazioni relative soltanto al tipo utilizzato.

Il file si presenta come di conseguenza riportato.

declare global {

var example: string;

}

export {};

La variabile globale *example* viene esportata a qualsiasi altro componente e file presente nel progetto.

In questa maniera si risolve il problema di andare a creare nuove funzionalità da implementare all’interno della Main Page per far sì che essa possa operare con quelle già fornite dell’Editor di LoIDE.

Richiamando la variabile globale tramite la stringa di codice *(window as any).example* si effettua la chiamata alla funzione di collaborazione.

 Infine per abilitare nel progetto la possibilità di accedere a queste variabili globali è stato necessario apportare modifiche ad un altro file, ovvero tsconfig.json, che specifica i file radice e le opzioni del compilatore necessarie per compilare il progetto.

Quindi è stata inserita la seguenta linea di codice.

"typeRoots": ["./node\_modules/@types", "./src/types"]

Questo permette di aggiungere i percorsi di tipo @types al file tsconfig.json.

Il valore della variabile globale vieni modificato in seguito ad un'operazione effettuata dall’utente nel popup e viene associato il titolo dell’esempio.

Successivamente il valore nuovo associato alla variabile globale viene controllato tramite un useEffect all’interno della Main Page.

Lo useEffect ci permette di effettuare delle operazioni ad ogni cambio di stato avvenuto all’interno delle nostre componenti.

Il codice riportato di seguito spiega la funzionalità.

const example = (window as any).example;

useEffect(() => {

if(example == "Battleship")

{

Utils.Editor.changeTabValue(1,battleshipExample.tabs[0].value);

}

if(example == "Saddlebags")

{

Utils.Editor.changeTabValue(1,saddlebagsExample.tabs[1].value);

}

}, [example]);

Ad ogni cambio del contenuto della variabile example dichiarata, che è possibile controllare passando la variabile tra parentesi quadre alla fine dello useEffect, si va a controllare a quale esempio l’utente si sta riferendo.

In questo piccolo esempio dimostrativo si può vedere come quando la scelta dell’utente ricade sul gioco Battaglia Navale, si procede a cambiare il valore dell’area dedicata alla descrizione e il valore dell’Editor 1 in questo caso con il contenuto del file. json battleshipExample.

Anche la scelta degli esempi da proporre agli utenti non è stata casuale, ma si basa sulle esperienze portate sul campo dalle personali esperienze universitarie.

Presentare esempi di grandi dimensioni, con complessità elevate o che prevedono l’uso di funzioni di linguaggio avanzante non è lo scopo che vuole raggiungere la funzionalità: l’obiettivo rimane offrire all’utente una marcia in più nei primi momenti

di approccio all’Editor, un piccolo reminder su semplici esempi di codice che in un momento qualsiasi può sfuggire.

Ecco perché gli esempi sono molto semplici da comprendere e replicare con altre problematiche di codice che possono presentarsi durante il lavoro su progetti.

Il valore lo si ottiene importando i file. json (JSON[[9]](#footnote-9)) che sono già presenti nel progetto.

1. **EDITOR COLLABORATIVO**

La necessità di implementare una funzionalità collaborativa nel progetto LoIDE nasce dopo un’attenta disamina delle modalità di studio tutt’oggi utilizzate ma anche dall’esperienza personale portata sul campo.

Utilizzando l’Editor di LoIDE sia in veste di studente che da diretto interessato all’implementazione si può notare come le sedute di coding siano spesso accompagnate da momenti di confronto e condivisione con altri colleghi, amici etc.

Anche per questo motivo quindi diventa utile avere un mezzo di collaborazione diretta che non va cercato al di fuori del contesto LoIDE, ma è esso stesso che lo diventa.

Le forme di collaborazione sul codice possono essere le più varie, ma quando si ha a disposizione una funzionalità che permette a due o più utilizzatori dell’Editor di lavorare in simbiosi e contemporanea sullo stesso progetto, tutto diventa più produttivo e soddisfacente.

Si può pensare anche a casi di utenti che non siano in grado di risolvere momentaneamente pezzi di codici, e in preda al dubbio e alla disperazione cercano aiuto online anche con scarsi risultati.

Diventa quindi compito e responsabilità di LoIDE evitare di stimolare queste reazioni nell’utente e di presentare una risposta, in parte, al problema.

Certo, un *Editor collaborativo* non rappresenta l’innovazione maggiore per ogni difficoltà di scrittura di codice, ma anche dall’esperienza portata in campo da altri Editor si percepisce l’esigenza sviluppata dagli utenti di collaborare per ottenere risultati migliori e precoci.

Infine, un tassello importante che ha spinto alla scelta di un Editor collaborativo è stato la crescita esponenziale del lavoro da casa.

La combinazione di lavoro in loco e lavoro da remota necessità di un mezzo che permetta ad un team di poter continuare a lavorare insieme.

Lo sviluppo di una funzionalità collaborativa all’interno di un Editor può essere supportato dalla presenza di framework già esistenti, o in caso contrario di una modalità di collaborazione basata su un sistema interno.

Nel progetto LoIDE è stato implementato un *Editor collaborativo* attraverso l’uso del framework Fluid Framework[[10]](#footnote-10).

La scelta tra le due nell’implementazione è stata dettata dalle risposte più che positive che Fluid Framework ha saputo dare nel presentare i suoi prodotti.

In primis si può notare che la scelta da parte di Microsoft di rilasciare il framework in maniera open source è stata sicuramente vincente e invitante.

Gli sviluppatori infatti hanno adottato subito questo metodo per creare contenuti interattivi e multiutenti.

Successivamente, un vantaggio principale è che il Fluid Framework permette di lavorare su documenti indipendenti: creare nuovi progetti e iniziare la collaborazione su ognuno di essi.

Con Fluid Framework i documenti non devono più essere passati avanti e indietro tra colleghi, ma si sfrutta la componente create e qualsiasi modifica si apporta verrà visualizzata.

Il primo passaggio per l’implementazione di questa funzionalità è stato capire come inserire all’interno di una componente già esistente come quella dell’Editor di LoIDE, una funzionalità di collaborazione.

Si è passati quindi a studiare come l’area di lavoro supportata dall’Editor veniva implementata, e come andare a modificare i parametri opportuni al fine di attribuire la capacità collaborativa.

È stato importante anche studiare la componente di default importata e utilizzata all’interno di LoIDE: quindi si è partiti dalla componente basilare, ovvero figlia, e a ritroso si è risaliti fino all’Editor che viene mostrato all’utente.

La prima idea sull’implementazione di questa funzionalità proposta al team era quella di creare una nuova Tab, distaccata da quelle già presenti nel progetto e denominata Collaborative Tab, che mettesse a disposizione un nuovo *Editor collaborativo*, che non avesse nulla a che fare con quello ufficiale utilizzato dall’utente e che permette l’esecuzione del codice, con il solo intento di realizzare una rete collaborativa tra più utenti.

In seguitò però al confronto con il team si è capito che così facendo si andava a perdere la reale natura di questa funzionalità.

L’idea di LoIDE è quella di utilizzare l’Editor che si ha a disposizione e dotarlo di tutte le caratteristiche necessarie per la collaborazione.

Il primo ostacolo incontrato nell’implementazione dell’*Editor Collaborativo* è rappresentato dalla tipologia di modello adottata da Fluid Framework, ovvero quello client server.

Il modello corrisponde perfettamente a ciò che il progetto LoIDE aspirava, ma si basa su un client che viene eseguito localmente sull’Editor in utilizzo, e su un server invece, come esplicitato nella documentazione associata, che va eseguito tramite un’altra Shell[[11]](#footnote-11).

Questo presentava un problema per LoIDE che invece vuole offrire all’utente un prodotto già finito, pronto all’utilizzo e senza necessità di coinvolgere l’utente in parti esterne a LoIDE.

La soluzione alla base di questo intoppo è stata trovata nell’utilizzo di un nuovo comando per l’avvio del server associato all’applicazione React.

Un’applicazione base React infatti tramite il file package. json effettua l’avvio server dei propri scripts tramite il seguente comando.

“script”: {

“start”: “React-scripts start”,

Quello che si è rilevato invece molto utile nel prodotto finale raggiunto è stato l’utilizzo di un nuovo comando, concurrently[[12]](#footnote-12), che è possibile utilizzare installandolo correttamente all’interno del progetto, e che prevede di poter avviare diversi server in concorrenza quando viene avviato il server principale.

Il risultato che ne segue, modificando l’opportuno file package. json è il seguente.

“scripts”: {

“start”: “concurrently \”React-scripts start\” \”npx tinylicious\””,

Così facendo, all’avvio dell’applicazione viene chiamato il comando concurrently che fa partire sia il server associato agli scripts della nostra applicazione, ma anche il server messo a disposizione da Fluid Framework.

Risolto il problema lato server, rimane quello di associare correttamente il client all’Editor Collaborativo utilizzato dall’utente.

Per prima cosa è stato dichiarato il client, importando la componente corretta da quelle installate del pacchetto Fluid Framework.

Un'opzione facoltativa è quella di dichiarare la porta dove dialogano i client connessi, ma omettendo questa dichiarazione si utilizza quella già presente all’avvio di LoIDE.

Successivamente è stato dichiarato uno schema, contenente gli oggetti su cui si lavorerà attraverso il client.

Il terzo passaggio ha previsto la dichiarazione di un container, con annesso id, in modo tale da riconoscere l’oggetto che dialoga con il client.

Infine, si ritorna con il container dichiarato e lo schema ad esso associato.

Tutto ciò è stato implementato all’interno di una funzione descritta nel seguente spezzone di codice.

|  |
| --- |
| const getFluidData = async () => {  const client = new TinyliciousClient ();  const containerSchema = {  initialObjects: {sharedText: SharedMap}  };  let container;  const containerId=window.location.hash.substring(1);  if (!containerId) {  ({container} = await.client.createContainer(containerSchema));  const id = await.container.attach();  window.location.hash = id;  } else {  ({container}=await.client.getContainer(containerId, containerSchema));  }  return container.initialObjects;  } |

Questa funzione viene successivamente richiamata all’interno di uno useEffect, ovvero una modalità di React per effettuare operazioni a seguito di aggiornamenti che avvengono nelle pagine, per andare a settare i dati associati al container che viene restituito.

Un ulteriore problema nell’implementazione dell’*Editor Collaborativo* è stato riscontrato sulla configurazione dell’Editor stesso utile per il framework utilizzato.

All’inizio si era pensato di utilizzare una semplice textarea modellata appositamente per permettere agli utenti di utilizzarla come un'area di lavoro il più simile possibile all’Editor della pagina principale.

Questo però avrebbe sminuito l’intento dell’implementazione, la sua importanza e anche la sua efficienza per gli utilizzatori.

Analizzando però la componente utilizzata dall’Editor principale si è andati alla ricerca di quella direttamente interessata alle modifiche.

Cestinata l’idea di utilizzare una nuova textarea, è stata individuata la componente principale per la stesura del codice su LoIDE: LoideAceEditor.

Lo scopo di questa componente è quello di implementare al suo interno la componente già esistente di React AceEditor che mette a disposizione una textarea avanzata utile a diverse esigenze, e modellarla e renderla utile nell’utilizzo all’interno di essa del linguaggio informatico per il quale l’Editor è stato realizzato.

L’Editor risultante risulta essere pronto a comporre il puzzle finale: l’unico tassello che rimane da inserire è la modalità di collaborazione.

All’interno del LoideAceEditor è stata individuata la funzione che si occupa di gestire il cambiamento della textarea, ovvero la funzione onChange().

Questa funzione effettua delle operazioni in seguito a dei cambiamenti che avvengono all’interno dell’Editor, e qui si è capito che andando ad effettuare operazioni all’interno si potevano richiamare le funzioni collaborative utilizzate da Fluid Framework.

L'obiettivo della funzione collaborativa infatti è quella di eseguire operazioni in seguito a cambi di stato dell’Editor, nel caso dell’*Editor Collaborativo* il cambio di stato è rappresentato da codice scritto sull’area di lavoro e che va riportato a chi visualizza l’Editor dall’altra parte.

Per effettuare i cambiamenti opportuni all’*Editor Collaborativo* è stata aggiunta la funzione sotto descritta.

function collaborate ()

{

fluidSharedObjects?sharedText.set("testo", ReactAce.current?.editor.getValue())

}

Questa funzione successivamente viene richiamata all’interno della funzione onChange() dell’Editor e che abilita la collaborazione.

All’interno dell’*Editor Collaborativo* è stata inserita una voce indispensabile, quella della referenza, in modo tale da evitare il circolo inutile e dispersivo del contenuto dell’Editor e passando all’interno del metodo set della funzione del Fluid Framework il valore ottenuto dalla referenza dell’Editor corrente ad esso associato.

La collaborazione su cui si basa L’*Editor Collaborativo* di LoIDE è possibile tra due o più utenti attraverso l’utilizzo di questa funzionalità sullo stesso URL: infatti basta semplicemente che essi abbiano aperto L’*Editor Collaborativo* sullo stesso URL e possono iniziare a sfruttare il tutto perfettamente.

Per permettere agli utenti di condividere l’URL della pagina su cui loro sono connessi è stato messo a disposizione un input field con accanto un pulsante per rendere l’URL visibile all’interno dell’input field e copiare il suo contenuto all’interno della Navigator.clipboard.

Il motivo principale alla base di questa scelta è il fatto che non bisogna dare per scontato che un utente debba intuire obbligatoriamente la concezione dello sviluppatore di rendere questa funzionalità disponibile semplicemente accedendo allo stesso URL e rendere veicolante il passaggio copiarlo dalla barra di ricerca e condividerlo successivamente.

Si prosegue quindi andando a modificare una funzionalità già presente all’interno del progetto LoIDE, ovvero Share.

Questa funzionalità è stata progettata con l’intento di permettere all’utente di copiare il progetto reale su cui si sta lavorando e poter condividerlo.

Con l’introduzione della modalità collaborativa è stato necessario estendere questa funzionalità introducendo un ulteriore campo che permettesse non di copiare l’URL associato al progetto di LoIDE, bensì di ottenere l’URL della pagina su cui si sta utilizzando l’Editor.

|  |
| --- |
|  |
| *Figura descrittiva della funzionalità Share estesa* |

Come si può notare dall’immagine ora risulta essere un nuovo input field che contiene l’URL della pagina ottenuto tramite il comando Window.location.href, e cliccando sul pulsante sottostante si può copiare l’URL sulla propria Navigator.clipboard.

Successivamente aprendo qualsiasi browser per la navigazione e incollando l’URL ottenuto si può collaborare con l’Editor della persona che si trova sul suo stesso URL.

1. **CONCLUSIONI**

Sulla base dei risultati ottenuti e del lavoro svolto durante il periodo di tirocinio e svolgimento della tesi è arrivato il momento di tirare le somme.

Le attività svolte all’interno del progetto LoIDE e le funzionalità sviluppate hanno rappresentato per questo lavoro di tesi dei punti di partenza importanti a livello di capacità di programmazione ma soprattutto a livello della modalità di contribuzione ad un progetto.

Nello specifico lo sviluppo della funzionalità *dell’Editor Collaborativo* ha presentato diversi dubbi e problematiche dal punto di vista implementativo, come trovare la giusta componente sulla quale estendere le funzionalità del Fluid Framework oppure come associare appunto queste funzionalità ad una componente già esistente.

Quindi è stato necessario fare un salto di qualità sulle tecniche utilizzate su React e sulla logica applicata.

D’altra parte, nello sviluppo delle funzionalità associate al *Tutorial Iniziale* e agli *Esempi* è stato importante capire quali sono i tratti distintivi di LoIDE e come contribuire in maniera giusta e in linea con il progetto.

Quindi è stato necessario adattarsi all’uso di diversi software esterni che hanno permesso di comunicare in maniera appropriata e professionale con gli altri contributori del progetto.

Spesso nasceva l’esigenza di introdurre dettagli che apparivano appropriati, salvo poi capire che è sempre meglio prima concordare un piano di sviluppo con gli altri membri del team e man mano presentare i propri progressi ed insieme andare avanti nella stessa direzione.

Il lavoro svolto quindi non è stato solo utile a capire come relazionarsi in un team di sviluppo, ma anche come sfruttare al meglio gli elementi messi a disposizione dal team stesso per rendere il lavoro più fluido e professionale possibile.

Interfacciarsi su un progetto di medie/grandi dimensioni, attraverso canali di comunicazione tipici e usuali nel mondo del lavoro quali GitHub e Trello nel mio caso, è stata un’esperienza formativa a 360 gradi.

Uno degli strumenti più funzionali e utilitari per lo sviluppo di questo progetto è GithHub, un servizio di hosting per progetti software che facilita, a differenza di Trello, le discussioni e la risoluzione di problemi relativi al codice.

Il tema infatti è stato già affrontato durante i corsi del percorso universitario, ma viverlo e capirlo direttamente durante questo lavoro di tesi è stato gratificante.

Sono state infatti effettuate delle pull request al fine di permettere agli altri contributori del progetto di capire il lavoro in corso e di poter dare consigli.

All’interno di GitHub sono state aperte anche discussioni relative ai problemi presenti nel lavoro presentato e alle possibili operazioni di modifica e correzione.

D’altra parte, è diventato indispensabile il supporto offerto da Trello, un software gestionale che ha consentito di avere una propria area personale dedicata al proprio contributo offerto al progetto LoIDE e che ha facilitato la comunicazione e lo scambio di idee con tutti i membri del team.

L’area personale di Trello offre la possibilità di creare nuove liste, all’interno di possono aggiungere delle card relative alle funzionalità da implementare.

Per il lavoro di tesi svolto le liste create sono: ToDo, Doing, Done.

Questo ha reso più facile lato tesista avere un lavoro organizzato nel migliore dei modi, e lato relatore comprendere più facilmente i progressi.

Risulta a volte troppo semplice immaginarsi come sarà la prima esperienza lavorativa, cosa aspettarsi e come relazionarsi, ma credo che una formazione come quella offerta dal Dipartimento di Informatica sia stata sufficiente a rispondere a qualsiasi dubbio possa avere mai avuto e rendermi più sicuro sulle scelte future, sulle modalità e sulle competenze da portare sul campo.

Posso affermare però di essere soddisfatto dei risultati ottenuti.

Sono partito da una conoscenza mediocre sugli strumenti utilizzati nel lavoro di tesi, e da una conoscenza molto scarsa relativa al modo in cui si lavora in un progetto, al modo in cui ci si interfaccia con un team di sviluppo, e alle pratiche da rafforzare ogni giorno sempre di più per eccellere in ciò che si fa.

Certamente ad oggi non mi credo di essere uno sviluppatore provetto, ma fortunatamente grazie a questa bellissima esperienza ho trovato ciò che realmente amo fare, e ciò che realmente voglio migliorare.

I risultati ottenuti dal lavoro di tesi nel contesto LoIDE sono solidi, ma è importante rafforzare sempre di più.

Ci sono elementi che a volte vengono considerati troppo banali, quali il modo di comunicare e le tipologie di aggiornamento collettivo sul progetto in corso, ma durante questo periodo, proprio sulla base di errori commessi, sono entusiasta di aver appreso nozioni indispensabili per essere considerati validi.

Certo il lavoro non finisce qua, perché sicuramente in qualsiasi percorso a maggior ragione quello intrapreso da noi Informatici l’aggiornamento non finisce mai, e ho imparato anche grazie all’esperienza fatta che non bisogna mai accontentarsi di ciò che si è raggiunto ma guardare sempre a cos’altro può essere fatto per migliorare.

I risultati ottenuti diventano quindi la base di una valutazione personale per aspirare ad ottenere sempre di più ed un punto di partenza fondamentale per il futuro.

Lo sviluppo di LoIDE sicuramente non si ferma qui, anzi ciò che è stato prodotto può rappresentare un punto di partenza importante.

L’idea dell*’Editor Collaborativo* può essere fonte di diversi sbocchi: ampliare le modalità di collaborazione, filtrare le tipologie di collaborazione tra utenti, ma mantenendo sempre vivo il motivo che ha spinto l’implementazione di questa funzionalità, ovvero agevolare gli utenti nell’utilizzo di LoIDE e nello sviluppo di progetti.

La Collaborazione ormai è un elemento consolidato e presente nella maggior parte degli Editor presenti online, ma potrebbe essere anche migliorata inserendo una chat formale tra Utenti con la possibilità di avere una rubrica di Utenti; in quel caso però è necessario l’utilizzo di un framework diverso da Fluid Framework al fine di rispettare criteri diversi.

Anche il Tutorial Iniziale può essere motivo di sviluppo, inserendo la possibilità di eseguire azioni relative alle funzionalità durante il percorso di informazione.

1. **RINGRAZIAMENTI**

Parto dal ringraziare l’essenza di tutto ciò, la mia famiglia.

A voi devo tutto, anche se so che per quanto ancora posso e dovrò fare non sarò mai in grado di restituire quanto fate.

Non posso ripagare mia madre che mi ha messo al mondo e che ringrazio di avermi sempre appoggiato durante questo percorso, e non potrò mai ripagare mio padre che ha sempre dato l’anima per la sua famiglia e che mi insegna ogni giorno come essere e diventare uomo.

Grazie ai miei due angeli custodi, coloro che oggi dovevano essere qui fisicamente a prendersi i meriti di ciò che sono ma che sono sempre presenti nelle nostre vite: i miei nonni.

Potrei parlare non so per quanti giorni e dire poco e nulla su di loro, che mi hanno insegnato valori, rispetto, sentimenti e quanto altro ancora.

Grazie a tutti i miei amici che ormai rappresentano una famiglia parallela, i momenti migliori li ho passati con tutti voi.

Grazie a tutti i miei parenti che hanno sempre creduto in me, che hanno sempre preso parte a questo mio percorso e sono un punto fermo della mia vita.

Grazie alla persona che ha dato un senso in più a questo mio percorso universitario ma un senso più importante alla mia vita, Antonella.

Con te è tutto speciale, tu mi completi e continuare tutto questo con te è il mio più grande traguardo.

1. **BIBLIOGRAFIA**

1. TypeScript:

Linguaggio di programmazione open source sviluppato da Microsoft

URL: <https://www.typescriptlang.org/>

1. 2. GitHub:

Servizio che aiuta gli sviluppatori ad archiviare e gestire il codice e tracciare le modifiche

URL: <https://github.com/>

1. 3. Trello:

Strumento che consente a gruppi di gestire visivamente qualsiasi tipo di progetto

URL: <https://trello.com/>

1. 4. Intro.js:

Tool utile per la creazione step by step di un tutorial sui propri prodotti.

URL: <https://introjs.com/>

1. 5. Fluid Framework:

Framework per creare applicazioni Fluid e abilitare l’uso dell’Editor Collaborativo

URL: <https://fluidframework.com/docs/start/quick-start/>

1. 6. React:

React è una libreria open-source, front-end, JavaScript per la creazione di interfacce utente.

URL: [https://it.Reactjs.org/](https://it.reactjs.org/)

1. Concurrently:

Tool di React che permette di eseguire più comandi contemporaneamente.

URL: <https://www.npmjs.com/package/concurrently>

1. Shell:

La Shell è la componente fondamentale che permette ad un utente di avere il più alto livello di interazione con un sistema operativo.

1. Cookie:

File di testo utile al server per ottenere informazioni sulle attività compiute da un utente su una pagina.

1. JavaScript:

Linguaggio di programmazione multi-paradigma orientato agli eventi, utilizzato sia nella programmazione lato client web che lato server per la creazione di rest API.

URL: <https://www.javascript.com/>

1. JSON:

Acronimo di Javascript Object Notation, è un formato adatto all’interscambio di dati fra applicazioni client/server.

URL: <https://www.json.org/json-it.html>

1. Navigato.clipboard:

API che può essere utilizzata per implementare le funzionalità di taglia, copia e incolla all’interno di un’applicazione web.

1. (TypeScript, s.d.) [↑](#footnote-ref-1)
2. (GitHub, s.d.) [↑](#footnote-ref-2)
3. (Trello, s.d.) [↑](#footnote-ref-3)
4. (Intro.js, s.d.) [↑](#footnote-ref-4)
5. (Cookie, s.d.) [↑](#footnote-ref-5)
6. (JavaScript, s.d.) [↑](#footnote-ref-6)
7. (Navigator.clipboard) [↑](#footnote-ref-7)
8. (React, s.d.) [↑](#footnote-ref-8)
9. (JSON, s.d.) [↑](#footnote-ref-9)
10. (Fluid Framework, s.d.) [↑](#footnote-ref-10)
11. (Shell, s.d.) [↑](#footnote-ref-11)
12. (Concurrently, s.d.) [↑](#footnote-ref-12)