Immagine che contiene schermata, nero, testo

Descrizione generata automaticamente

Relazione Tirocinio per il *Corso di Laurea Magistrale*

*in Ingegneria Informatica Sistemi distribuiti, Cloud ed Intelligenti*

**Tecniche e tecnologie per la fruizione di contenuti 3D**

Relatore Studente

*Prof. Salvatore Venticinque Giuseppe Diana A13/330*

A.A. 2022/2023

**Sommario**

[**Introduzione** 3](#_Toc147847306)

[**1.Mirador - Una Piattaforma per la Visualizzazione di Risorse Digitali** 3](#_Toc147847307)

[**1.1. Funzionalità di Mirador** 3](#_Toc147847308)

[**1.2. Utilizzo di Mirador in Locale** 4](#_Toc147847309)

[**2. Il Plugin Mirador** 4](#_Toc147847310)

[**2.1. Esempi di utilizzo del plugin** 4](#_Toc147847311)

[**2.2. Monitoraggio degli Eventi** 5](#_Toc147847312)

[**3. MongoDB** 5](#_Toc147847313)

[**3.1.1. Modello per gli Eventi di Clic** 6](#_Toc147847314)

[**3.1.2. Modello per gli Eventi di Zoom** 6](#_Toc147847315)

[**3.1.3. Modello per gli Eventi di Aggiunta di Finestre** 6](#_Toc147847316)

[**3.3. Architettura e Applicazione Server** 6](#_Toc147847317)

[**3.4. Configurazione del Server** 7](#_Toc147847318)

[**3.5. Rotte e API RESTful** 7](#_Toc147847319)

[**3.6. Gestione delle Rotte** 7](#_Toc147847320)

[**3.6.1. Rotta per gli Eventi di Clic** 7](#_Toc147847321)

[**3.6.2. Rotta per gli Eventi di Zoom** 7](#_Toc147847322)

[**3.6.3. Rotta per gli Eventi di Aggiunta di Finestre** 7](#_Toc147847323)

[**4.Codice Plugin** 7](#_Toc147847324)

# **Introduzione**

La presente relazione offre una descrizione del lavoro che ha coinvolto lo sviluppo di un plugin per Mirador3, un visualizzatore di risorse digitali, con lo scopo di profilare un utente in base alle sue azioni. È stato utilizzato MongoDB come database per la memorizzazione dei dati relativi agli eventi.

# **1.Mirador - Una Piattaforma per la Visualizzazione di Risorse Digitali**

Mirador è una piattaforma open-source per la visualizzazione e l'esplorazione di risorse digitali, documenti, immagini e collezioni culturali digitalizzate. Permette agli utenti di esplorare, visualizzare e annotare le risorse in una visione multipla, consentendo una profonda interazione con il contenuto digitale. Lo scopo del suo sviluppo è stato quello di soddisfare le esigenze di archivi, biblioteche, musei e istituzioni culturali che desiderano rendere disponibili al pubblico le loro risorse digitali in modo interattivo e intuitivo e accessibili da qualsiasi parte del mondo. È basato su IIIF, uno standard internazionale per la consegna di immagini interoperabili. La sua versatilità e la sua capacità di personalizzazione lo rendono uno strumento prezioso per l'esplorazione e la condivisione di risorse digitali di valore storico e culturale.

È implementato in JavaScript e si basa su una serie di librerie e framework, tra cui React, Redux e OpenSeadragon.

## **1.1. Funzionalità di Mirador**

Mirador offre numerose funzionalità chiave, tra cui:

* **Visualizzazione Multipla**: Gli utenti possono visualizzare contemporaneamente più immagini o pagine per una migliore comparazione.
* **Zoom**: Gli utenti possono interagire con le risorse digitali utilizzando funzionalità come zoom e annotazioni. Queste azioni sono facilitate dall'integrazione di OpenSeadragon, che gestisce il rendering e la manipolazione delle immagini ad alta risoluzione
* **Annotazioni**: Mirador consente agli utenti di visualizzare le annotazioni delle risorse digitali. Queste possono includere testo, marcature, collegamenti e commenti. La collaborazione è supportata attraverso la condivisione di annotazioni tra gli utenti
* **Gestione delle Finestre**: Supporto per l'apertura e la gestione di diverse finestre contenenti risorse diverse.
* **Interoperabilità**: Mirador è altamente personalizzabile e può essere adattato alle esigenze specifiche di un'organizzazione o di un progetto. Inoltre, può essere integrato in siti web esistenti o applicazioni tramite API e componenti personalizzati

## **1.2. Utilizzo di Mirador in Locale**

Per iniziare:

* È necessario scaricare l'ultima versione di Mirador dal repository GitHub ufficiale: <https://github.com/ProjectMirador/mirador>.
* Cliccare sul pulsante "Code" e seleziona "Download ZIP" per scaricare il repository come file ZIP.
* Estrarre il contenuto del file ZIP in una directory.

Una volta scaricato Mirador è necessario configurarlo.

* Dopo aver aggiunto le dipendenze all’interno del file **package.json** e **package-lock.json,**con il comando **npm install** si vanno ad installare.
* All’interno della cartella **src** è presente il file di configurazione Mirador, **index.js,** dove vengono specificate le immagini da importare nella visualizzazione, il catalogo da cui scegliere ulteriori immagini da visualizzare e, sempre all’interno di questo file, con il comando **import** è possibile importare il plugin che si vuole integrare a Mirador.
* Una volta configurato il tutto, con il comando **npm run parcel** viene avviato un server web locale per visualizzare Mirador all'indirizzo **http://localhost:1234;**

# **2. Il Plugin Mirador**

Il plugin è stato implementato come un componente React integrato nell'applicazione Mirador. Lo scopo dello sviluppo del plugin è quello di monitorare e registrare eventi specifici da parte di un utente all'interno dell'applicazione. Una volta intercettato l’evento, tutte le informazioni relative a quest’ultimo vengono inviate ad un server Restful locale tramite richieste HTTP.

## **2.1. Esempi di utilizzo del plugin**

il plugin può essere utilizzato per:

* Analizzare quali immagini sono più popolari tra gli utenti, raccogliere dati sul numero di volte in cui ogni immagine viene cliccata, zoomata o aggiunta a una finestra di visualizzazione. Questi dati possono essere utilizzati per identificare le immagini più popolari tra gli utenti e per migliorare l'esperienza utente di Mirador.
* Identificare le immagini che gli utenti trovano difficili da capire. Il plugin può essere utilizzato per raccogliere dati sulle immagini che gli utenti zoomano o aggiungono a una finestra di visualizzazione più volte. Questi dati possono essere utilizzati per identificare le immagini che gli utenti trovano difficili da capire e per sviluppare nuove funzionalità di Mirador che aiutino gli utenti a capire meglio queste immagini.

## **2.2. Monitoraggio degli Eventi**

Per lo sviluppo del plugin sono state utilizzate le seguenti librerie

* **React**: una libreria JavaScript per la creazione di interfacce utente.
* **PropTypes**: una libreria JavaScript per la validazione dei tipi di dati.
* **Mirador**: una libreria JavaScript per la visualizzazione di immagini.
* **Axios**: una libreria JavaScript per effettuare richieste HTTP.

Il plugin monitora tre tipi principali di eventi all'interno di Mirador:

* **handleClick**: Il plugin rileva i clic su elementi specifici all'interno delle finestre di Mirador e registra queste interazioni, inclusi i dettagli dell'elemento cliccato.
* **handleZoom**: Il plugin rileva gli eventi di zoom e ne registra i dettagli, compreso l'elemento su cui è stato eseguito lo zoom.
* **windowAdded**: Il plugin rileva quando viene aggiunta una nuova finestra a Mirador e registra i dettagli dell'evento evidenziando le informazioni della nuova immagine.
* La funzione **mapStateToProps** è utilizzata per mappare lo stato globale di Mirador 3 alle proprietà necessarie per il componente. Questo consente di accedere alle informazioni sui canvas, finestre e relativi ID
* Per definire il comportamento del componente in diverse fasi del suo ciclo di vita sono stati utilizzati i metodi di React: ‘**componentDidMount’, ‘componentDidUpdate’ e ‘componentWillUnmount’**
* **componentDidMoun**t: Questo metodo viene chiamato dopo che il componente è stato inserito nel DOM (Document Object Model).
* **componentDidUpdate**: Questo metodo viene chiamato ogni volta che il componente viene aggiornato, cioè quando riceve nuove props o viene aggiornato lo stato interno.
* **componentWillUnmount**: Questo metodo viene chiamato prima che il componente venga rimosso dal DOM e viene utilizzato per la rimozione di eventi creati all’interno di componentDidMount

# **3. MongoDB**

MongoDB è stato scelto come sistema di gestione del database per memorizzare i dati degli eventi registrati dal plugin. MongoDB è un database NoSQL orientato ai documenti che offre flessibilità nella memorizzazione dei dati strutturati in formato JSON.

Come prima operazione ho creato un account Atlas gratuito, creato il cluster, successivamente un Database nominato Mirador e ho generato la stringa di connessione alla banca dati. L'interazione con il database è stata gestita tramite il modulo Mongoose, che semplifica la creazione di modelli e la comunicazione con il database.

**3.1. Schema dei Dati**

Per memorizzare i dati degli eventi, sono stati definiti tre modelli di dati Mongoose:

### **3.1.1. Modello per gli Eventi di Clic**

javascriptCopy code

const mongSchema = {windowId: String, target: String, message: String,timestamp: String message: String,}

### **3.1.2. Modello per gli Eventi di Zoom**

javascriptCopy code

const mongSchema = {windowId: String, target: String, message: String, timestamp: String, }

### **3.1.3. Modello per gli Eventi di Aggiunta di Finestre**

javascriptCopy code

const mongSchema = {windowId: String, target: String, message: String, timestamp: String,}

Ciascuno di questi modelli definisce quattro campi fondamentali: **windowId**, **target**, **message** e **timestamp**, che contengono rispettivamente l'ID della finestra, URL della risorsa, un messaggio che descrive il tipo di evento e infine un timestamp.

## **3.3. Architettura e Applicazione Server**

Immagine che contiene diagramma, testo, Carattere, linea

Descrizione generata automaticamente

L'applicazione server è stata implementata utilizzando Node.js insieme a Express.js come framework per gestire le richieste HTTP provenienti dal plugin Mirador. L'applicazione server è stata configurata per ascoltare sulla porta 3000.

## **3.4. Configurazione del Server**

L'applicazione server è stata configurata per:

* Utilizzare il middleware **Cors** per consentire richieste da diverse origini.
* Utilizzare il middleware **express.json()** per consentire il parsing dei dati JSON dalle richieste HTTP.
* Connettersi al database MongoDB tramite la libreria Mongoose.

## **3.5. Rotte e API RESTful**

Per consentire al plugin di inviare dati al server locale, è stata creata un'API RESTful utilizzando Express.js. Le rotte dell'API gestiscono le richieste POST provenienti dal plugin Mirador e salvano i dati nel database MongoDB utilizzando i modelli Mongoose appropriati.

Per la comunicazione con il server locale è stata usata la libreria **Axios** per inviare i dati degli tramite richieste POST HTTP

## **3.6. Gestione delle Rotte**

L'applicazione server gestisce tre rotte principali:

### **3.6.1. Rotta per gli Eventi di Clic**

La rotta **/click** gestisce le richieste POST per i dati relativi agli eventi di clic inviati dal plugin Mirador. Estrae i dati dalla richiesta HTTP e li salva nel database MongoDB utilizzando il modello Mongoose **ClickInfo**.

### **3.6.2. Rotta per gli Eventi di Zoom**

La rotta **/zoom** gestisce i dati relativi agli eventi di zoom e li salva nel database utilizzando il modello Mongoose **ZoomInfo**.

### **3.6.3. Rotta per gli Eventi di Aggiunta di Finestre**

La rotta **/addwindow** gestisce i dati relativi agli eventi di aggiunta di finestre e li salva nel database utilizzando il modello Mongoose **WindowInfo**.

# **4.Codice Plugin**

<https://github.com/GiuseppeDiana95/Tirocinio>