Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Progettazione e sviluppo di Widget per Geo-querying ed inserimento massivo di dati

Tesi di laurea

Relatore

Prof.ssa Ombretta Gaggi

Laureando

Francesco Bacchin
Matricola 1227269

Anno Accademico 2021-2022

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecento ore, dal laureando Francesco Bacchin presso l'azienda RiskAPP s.r.l con sede a Conselve (PD).
L'obbiettivo del progetto di stage è stato quello di progettare, per poi sviluppare, un *widget* che permetta all'utente di effettuare l'inserimento massivo di oggetti, in una mappa presente all'interno della piattaforma RiskAPP.
Innanzitutto si è svolta l'analisi dei requisiti, si è passato poi alla progettazione ed infine lo sviluppo e la verifica della funzionalità implementata. Tali fasi vengono analizzate nel dettaglio nei capitoli che compongono la presente tesi.

"Dilexit Veritatem."

— Marc Bloch

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine alla Professoressa Ombretta Gaggi per la disponibilità e professionalità che mi ha fornito durante lo svolgimento della tesi.

Desidero ringraziare con affetto i miei genitori, mio fratello, mia sorella e tutti i parenti che mi hanno sostenuto e motivato durante gli anni di studio.

Ho desiderio di ringraziare la mia ragazza, Giovanna, che mi è stata sempre vicina ed i miei amici, per tutti i bellissimi anni passati insieme e le mille avventure vissute.

Padova, Settembre 2022 Francesco Bacchin

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 L'azienda

Fondata nel 2016, RiskApp (vedasi logo in Figura 1.1) è una azienda con sede a Conselve (PD) che si occupa di sviluppo software per il mondo assicurativo. Il *core business* dell'azienda è rappresentato dal mantenimento della piattaforma RiskAPP, pensata per intermediari del mondo assicurativo come agenti e *broker*. Il compito di tale piattaforma è quello di fornire un quadro più completo possibile di tutti i rischi a cui possono andare incontro le compagnie assicurative.
Il principale punto di forza di questa piattaforma è essere in grado di stimare le possibili perdite economiche di un'impresa, attraverso un algoritmo proprietario in grado di valutare il rischio basandosi su dati raccolti da diverse fonti.
Inoltre vengono messe a disposizione funzionalità di consulenza e gestione dei clienti. RiskAPP è distribuita come Software as a Service[g] a clienti dislocati in tutto il Mondo e, ad oggi, l'azienda ha intrapreso numerose relazioni commerciali con importati *player* del mondo della finanza come UnipolSai, PwC e Swiss Re [26].

Figura 1.1: Logo dell'azienda RiskAPP

1.1.1 Processi aziendali

L'azienda adotta un approccio *Agile* nello sviluppo software secondo la metodologia Scrum[g] che è stata riadattata per le sue esigenze, vengono quindi ripresi i principi base quali trasparenza, controllo ed adattamento, ma non si è aderito in modo completamente formale ad essa. Ad esempio non vengono eseguiti giornalmente i cosiddetti *daily scrum*, sostituiti da una comunicazione continua in ufficio o un paio di riunioni settimanali più approfondite.
RiskApp adotta inoltre gli approcci di Continous Integration[g] e Continous Delivery[g] in modo da fornire ai propri clienti un software testato e funzionante attraverso rilasci frequenti e continui.

1.1.2 Strumenti

Tra gli strumenti usati all'interno dell'azienda vi sono:

* **Slack**: usato per la messaggistica istantanea;
* **Gmail**: usato per la posta elettronica;
* **Teams**: usato per le videochiamate;
* **Jira**: usato per la gestione dei progetti secondo la metodologia *Agile* e l'*issue tracking*;
* **GitHub**: usato come sistema di versionamento del codice;
* **Circle CI**: usato per la Continous Integration e Continous Delivery;
* **Figma**: usato come *tool* per sviluppare mockup grafici;

Per lo sviluppo software non vengono date indicazioni per l'ambiente di sviluppo da utilizzare, pertanto si è liberi di scegliere l'IDE[g] più adatto per lo svolgimento del proprio lavoro.

1.2 Il progetto di stage

1.2.1 Dominio applicativo

Il progetto di stage proposto riguarda lo sviluppo della parte di *front-end* per due funzionalità da implementare nella mappa della piattaforma RiskAPP.
Ogni mappa fa rifermento ad un determinato cliente dell'utente e permette di visualizzare la posizione geografica di vari oggetti appartenenti al cliente a cui fa riferimento la mappa. Tali oggetti possono essere:

* Ubicazione: il terreno su cui è costruito un determinato edificio del cliente;
* asset: l'edificio del cliente;
* nodi: macchinari, mercati, code e fornitori che fanno riferimento al cliente.

L'inserimento di ubicazioni e asset avveniva uno per volta ma avendo riscontrato la necessità di velocizzare il processo, soprattutto nei casi in cui vi sono un numero elevato di elementi da inserire, si è deciso di implementare la funzionalità di inserimento massivo.
Il cliente, che deve aver eseguito l'autenticazione, potrà quindi specificare gli indirizzi degli elementi che ha la necessità di inserire. Vi saranno due modi per inserire gli elementi:

* direttamente nel sito tramite un'apposita GUI;
* tramite il caricamento di un foglio Excel precedentemente formattato.

Concluso l'inserimento il cliente verrà reindirizzato in una pagina dove visualizzerà una mappa che conterrà gli elementi appena inseriti, ottenuti tramite geocoding[g]. Gli verrà quindi richiesto di confermare che gli elementi visualizzati in essa corrispondano con gli asset e le ubicazioni aspettate. Conclusa la procedura di conferma gli asset e le ubicazioni inseriti e convalidati verranno salvati e l'utente verrà riportato nella mappa

iniziale dove potrà visualizzarli.
Oltre alla funzionalità di inserimento massivo è facoltativo lo sviluppo di un'altra funzionalità; essa nasce dall'esigenza di monitorare i rischi naturali relativi ad un'ubicazione o asset presente nella mappa. L'obbiettivo è quello di far visualizzare all'utente dei grafici relativi ai rischi delle pericolosità naturali per una ubicazione o asset, dove i dati relativi a quest'ultime vengono estrapolati tramite query dal database aziendale.

1.2.2 Obiettivi

Nella Tabella 1.1 vengono riportati gli obiettivi dello stage. Essi sono identificati dalla seguente sigla:

* **O** per gli obiettivi obbligatori, vincolanti in quanto obiettivo primario richiesto dal committente;
* **D** per gli obiettivi desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
* **F** per gli obiettivi facoltativi, rappresentanti valore aggiunto non strettamente competitivo.

Le sigle precedentemente indicate saranno seguite da una coppia sequenziale di numeri, identificativo dell'obiettivo.

Tabella 1.1: Tabella degli obiettivi dello stage.

Codice	Descrizione
O1	Stesura documento analisi dei requisiti
O2	Sviluppo widget per inserimento massivo tramite foglio excel
O3	Sviluppo widget per inserimento massivo tramite interfaccia grafica
O4	Sviluppo schermata di conferma dei dati inseriti
D1	L'interfaccia grafica implementata deve essere accessibile a tutte le categorie di utenti
D2	Stesura documento manuale sviluppatore
F1	Sviluppo widget per la visualizzazione di grafici relativi ai rischi delle pericolosità naturali
F2	Deve essere raggiunta tramite i test una code coverage maggiore o uguale all'80%;

1.2.3 Tecnologie utilizzate

1.2.3.1 Linguaggi

1.2.3.1.1 JavaScript

JavaScript (vedasi logo in Figura 1.2) è un linguaggio di *scripting* lato client utilizzato per rendere interattive le pagine web. Questo linguaggio è una delle tecnologie principali della programmazione *web front-end* poichè permette di gestire in modo dinamico il contenuto (HTML) e lo stile grafico (CSS) di un sito.
Il report di PYPL datato Agosto 2022 gli conferisce il terzo posto nei linguaggi di programmazione più popolari dopo Java (secondo) e Python (primo)[2].

Figura 1.2: Logo del linguaggio JavaScript

Nella webapp RiskAPP JavaScript è ampiamente utilizzato nel *front-end* e in parte nel *back-end*, attraverso Node.js. Viene utilizzata la versione ECMAscript 6, supportata da qualsiasi browser moderno [1], che permette una scrittura del codice più dichiarativa rispetto alle versioni precedenti.

1.2.3.1.2 JSX

JSX (vedasi logo in Figura 1.3) è un'estensione sintattica di JavaScript che permette di scrivere la struttura dei componenti usando una sintassi simile al linguaggio HTML al cui interno possono comparire espressioni JavaScript delimitate da parentesi graffe (vedasi esempio in Snippet di codice 1.1).
Nella webapp RiskApp la sintassi JSX viene utilizzata per descrivere gli elementi che compongono la UI.

```
1 const name = 'Giuseppe Verdi';
2 const element = <h1>Hello, {name}</h1>;
```

Snippet di codice 1.1: Esempio sintassi JSX

Figura 1.3: Logo dell'estensione sintattica JSX

1.2.3.1.3 HTML5

HTML (vedasi logo in Figura 1.4) è un linguaggio di markup che serve a strutturare le pagine web. La versione utilizzata in RiskAPP è l'ultima disponibile, ossia HTML5.
I nuovi tag introdotti in HTML5 permettono una sintassi più veloce e semplice, offrendo comunque la retrocompatibilità per le sue precedenti versioni.

Figura 1.4: Logo di HTML5

1.2.3.1.4 CSS3

CSS (vedasi logo in Figura 1.5) è un linguaggio di programmazione web utilizzato per definire l'aspetto e la formattazione di un sito web.
La versione utilizzata in RiskAPP è CSS3, ultima disponibile, aggiunge nuove importanti novità come nuovi modi per formattare il contenuto (*flex* e *grid*) e per gestire il ridimensionamento della pagina web (*media queries*).

Figura 1.5: Logo di CSS3

1.2.3.1.5 Overpass QL

Overpass QL è un linguaggio utilizzato per interrogare tramite *queries* il database di OpenStreetMap che contiene dati geografici.
In RiskAPP questo linguaggio è utilizzato per recuperare le coordinate geografiche di edifici e proprietà.

1.2.3.2 Framework

1.2.3.2.1 React

React (vedasi logo in Figura 1.6) è una libreria di JavaScript *open-source* per la creazione di interfacce utente (UI). Essa consente di sviluppare applicazioni dinamiche che non necessitano ricaricare la pagina per visualizzare un cambiamento nei dati visualizzati.
Le applicazioni create in React si basano sull'uso dei componenti, pezzi di codice indipendenti e riusabili che restituiscono codice HTML, definiscono la logica e il comportamento dell'interfaccia grafica.
Ci sono due tecniche che si possono adottare per descrivere un componente:

- sviluppo funzionale: descrive i componenti come funzioni JavaScript;
- sviluppo a componenti: descrive i componenti come classi.

Il *front-end* di RiskAPP è sviluppato a componenti.

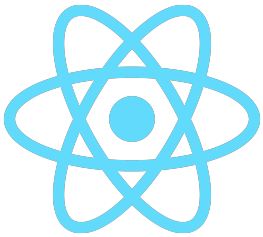

Figura 1.6: Logo di React

1.2.3.2.2 Node.js

Node.js (vedasi logo in Figura 1.7) è un *framework open-source* che permette di eseguire lato server codice JavaScript in maniera asincrona.

Figura 1.7: Logo di Node.js

1.2.3.3 Bootstrap

Bootstrap (vedasi logo in Figura 1.8) è un *framework* CSS *opensource* per lo sviluppo di applicazioni web Responsive[g] ed orientate al Mobile-first[g].
In RiskAPP è utilizzato react-boostrap che fornisce *templates* HTML, CSS e JavaScript per la maggior parte dei componenti presenti nella UI.

Figura 1.8: Logo di Bootstrap

1.2.4 Strumenti utilizzati

1.2.4.1 Ambiente di lavoro

1.2.4.1.1 Mac OS Monterey

Mac OS Monterey (vedasi logo in Figura 1.9) è l'ultima versione del sistema operativo di Apple per MacBook.
Poichè il PC fornito dall'azienda a dipendenti e stagisti è un MacBook si è deciso di utilizzare questo sistema operativo.

Figura 1.9: Logo di Mac OS Monterey

1.2.4.1.2 Visual Studio Code

Visual Studio Code (vedasi logo in Figura 1.10) è un editor di codice *cross-platform* che permette di:

- evidenziare la sintassi per ciascun linguaggio di programmazione;
- avere un supporto integrato per il *debugging*;
- avere un supporto integrato a Git;
- effettuare *refactoring* del codice;
- scaricare ed installare estensioni per ampliare le funzionalità del programma.

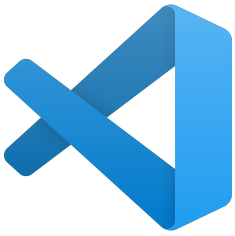

Figura 1.10: Logo di Visual Studio Code

1.2.4.1.3 npm

Npm (vedasi logo in Figura 1.11) è una *repository* online per la pubblicazione di progetti Node.js *open-source*. Installando Node.js, npm viene installato di default e permette, tramite linea di comando, di interagire con la sua *repository* online come gestore di pacchetti. Tutti i pacchetti presenti in tale *repository* sono consultabili in www.npmjs.com [17].

Figura 1.11: Logo di npm

1.2.4.2 Versionamento

1.2.4.2.1 Git

Git (vedasi logo in Figura 1.12) è un sistema di controllo di versione distribuito ed *open-source* pensato per gestire qualsiasi tipo di progetto garantendo velocità ed efficienza. [11]
Nello sviluppo di RiskAPP viene utilizzata la tecnica *Git Feature Branch Workflow*. Essa predilige la creazione di un nuovo *branch* ogni qualvolta si debba inserire una nuova funzionalità all'interno della webapp. Nel nuovo *branch* creato verrà sviluppata la funzionalità e quando sarà conclusa e funzionante verrà effettuato il Merge[g] nel *branch* principale (*main*)[10].

Figura 1.12: Logo di Git

1.2.4.2.2 GitHub

GitHub (vedasi logo in Figura 1.13) è una piattaforma di *hosting* per *repository* git. Fornisce agli sviluppatori strumenti per migliorare e mantenere il codice come:

* *features* utilizzabili dalla linea di comando;
* gestione delle *pull request* e *code review*;
* strumenti per l'*issue traking*;

[12]
La webapp RiskAPP è suddivisa in varie *repository* su GitHub.

Figura 1.13: Logo di GitHub

1.2.4.3 Comunicazione

1.2.4.3.1 Slack

Slack (vedasi logo in Figura 1.14) è l'app di messaggistica istantanea, ideata per un ambiente lavorativo, utilizzata in RiskApp.

Figura 1.14: Logo di Slack

Organizzazione della tesi

Il secondo capitolo descrive l'analisi dei requisiti. Presenta gli attori coinvolti nell'interazione con le funzionalità implementate ed elenca i casi d'uso individuati. Vengono riportati in tabella i requisiti che vengono classificati in base alla tipologia e all'importanza.

Il terzo capitolo descrive la fase di progettazione, l'architettura di RiskAPP e l'organizzazione dei file. Viene riportato e descritto il diagramma di attività riguardante l'inserimento massivo.

Il quarto capitolo descrive la fase di codifica, vengono mostrate alcune schermate del prodotto e sono riportate le funzioni più rilevanti.

Il quinto capitolo descrive la fase di verifica e di validazione. Vengono elencate le principali attività svolte.

Il sesto capitolo riporta gli obbiettivi completati ed i requisiti soddisfatti. Viene data una breve valutazione personale dello stage svolto.

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

* gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
* per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: **parola**[g];
* i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*;
* il testo che si riferisce al codice o ai file delle funzionalità implementate viene evidenziato con il font `spaziato`.

Capitolo 2

Analisi dei requisiti

2.1 Attori

Sebbene la piattaforma RiskAPP possa essere utilizzata anche da un utente non autenticato, il progetto di stage interessa esclusivamente funzioni che riguardano utenti che hanno eseguito l'autenticazione tramite la pagina di login. É stato scelto di utilizzare il termine utente per identificare un utente che ha eseguito l'autenticazione. Sono presenti due attori:

* **utente**;
* **database *back-end***: dove verranno salvati gli oggetti inseriti.

2.2 Casi d'uso

Per lo studio dei casi di utilizzo del prodotto sono stati creati dei diagrammi. I diagrammi dei casi d'uso (in inglese *Use Case Diagram*) sono diagrammi di tipo Unified Modeling Language (UML) dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso.

2.2.1 Classificazione

I casi d'uso avranno la seguente classificazione:

UC[Codice padre](.[Codice figlio])

dove:

* **Codice padre** è il codice identificativo numerico del dato caso d'uso;
* **Codice figlio** è il codice identificativo di un eventuale sotto caso d'uso;

2.2.2 Descrizione

Ogni caso d'uso verrà descritto secondo la seguente struttura:

* **Attori principali**: gli attori che interagiranno nello UC;

* **Precondizioni**: le condizioni che devono essere vere prima dello svolgimento dello UC;

* **Scenario principale**: il flusso principale degli eventi;

* **Estensioni**: eventuali estensioni dello UC (opzionale);

* **Postcondizioni**: le condizioni che devono essere vere dopo lo svolgimento dello UC.

2.2.3 Elenco casi d'uso

2.2.3.1 Use Case principale

La Figura 2.1 riporta il caso d'uso relativo all'inserimento massivo.

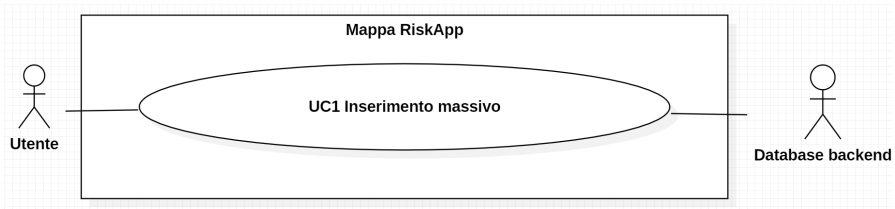

Figura 2.1: Use Case: Caso d'uso principale

UC1: Inserimento massivo

Attori Principali: Utente e database *back-end*.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente.

Scenario principale: L'utente effettua l'inserimento massivo di ubicazioni e asset nella mappa.

Postcondizioni: La mappa contiene le ubicazioni e asset che l'utente ha inserito.

2.2.3.2 Use Case 1: inserimento massivo

La Figura 2.2 riporta i sotto casi d'uso relativi all'inserimento massivo (vedasi Figura 2.1).

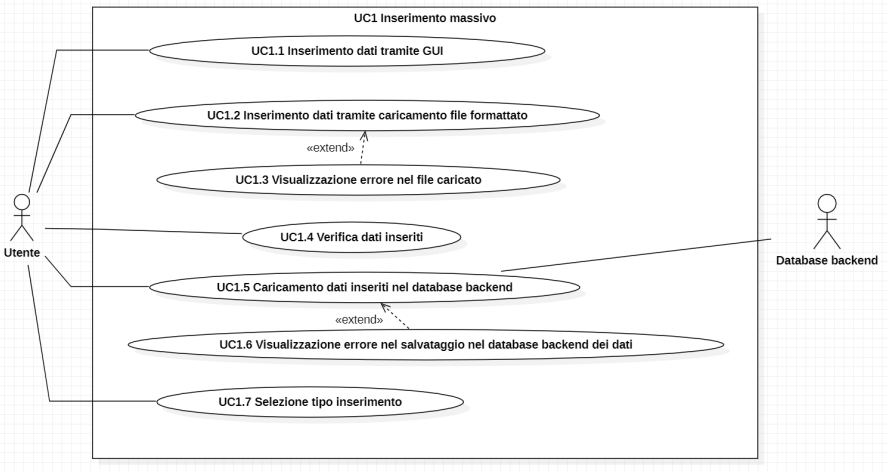

Figura 2.2: Use Case - UC1: Inserimento massivo

UC1.1: Inserimento dati tramite GUI

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo e ha selezionato "inserimento tramite GUI" come metodo con il quale effettuarlo.

Scenario principale: L'utente inserisce i dati relativi alle ubicazioni e asset che vuole inserire nell'interfaccia dedicata.

Postcondizioni: L'utente ha inserito i dati relativi alle ubicazioni e asset che voleva inserire.

UC1.2: Inserimento dati tramite caricamento file formattato

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo e ha selezionato "inserimento tramite file formattato" come metodo con quale vuole effettuarlo.

Scenario principale: L'utente carica un file, debitamente formattato, contenente i dati relativi alle ubicazioni e asset che vuole inserire.

Estensioni: UC1.3.

Postcondizioni: L'utente ha caricato un file, debitamente formattato, contenente i dati relativi alle ubicazioni e asset che voleva inserire.

UC1.3: Visualizzazione errore nel file caricato

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard

ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato "inserimento tramite file formattato" come metodo con il quale vuole effettuarlo, e ha inserito i dati tramite un file Excel non formattato correttamente.

Scenario principale: L'utente visualizza un messaggio di errore.

Postcondizioni: L'utente non ha caricato nessun dato.

UC1.4: Verifica dati inseriti

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo con il quale vuole effettuarlo, ed ha inserito i dati.

Scenario principale: L'utente verifica e convalida che i dati inseriti corrispondano alle ubicazioni e asset aspettati.

Postcondizioni: L'utente ha convalidato i dati inseriti.

UC1.5: Salvataggio dati inseriti nel database *back-end*

Attori Principali: Utente e database *back-end*.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo con il quale vuole effettuarlo, ha inserito i dati e li ha convalidati.

Scenario principale: L'utente carica i dati che ha convalidato nel database *back-end*.

Estensioni: UC1.6.

Postcondizioni: L'utente ha caricato i dati nel database *back-end*.

UC1.6: Visualizzazione errore nel salvataggio nel database *back-end* dei dati

Attori Principali: Utente e database *back-end*.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo con il quale vuole effettuarlo, ha inserito i dati, li ha convalidati e li sta caricando nel database *back-end* ma avviene un errore nel caricamento.

Scenario principale: L'utente visualizza un messaggio di errore del caricamento dei dati nel database *back-end*.

Postcondizioni: L'utente non ha caricato i dati nel database *back-end*.

UC1.7: Selezione tipo inserimento

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo.

Scenario principale: L'utente decide il metodo con il quale vuole effettuare l'inserimento massivo.

Postcondizioni: L'utente ha scelto il metodo con il quale vuole effettuare l'inserimento massivo.

2.2.3.3 Use Case 1.1: inserimento dati tramite GUI

La Figura 2.3 riporta i sotto casi d'uso relativi all'inserimento massivo dei dati tramite GUI (vedasi Figura 2.2 UC1.1).

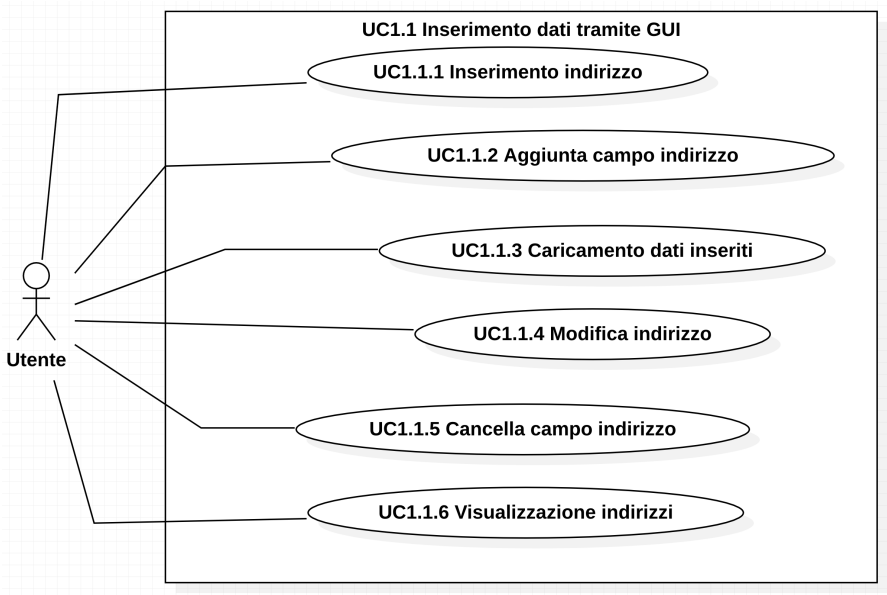

Figura 2.3: Use Case - UC1.1: Inserimento dati tramite GUI

UC1.1.1: Inserimento indirizzo

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo e ha selezionato il metodo "Inserimento tramite GUI".

Scenario principale: L'utente inserisce l'indirizzo.

Postcondizioni: L'utente ha inserito l'indirizzo.

UC1.1.2: Aggiunta campo indirizzo

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo e ha selezionato il metodo "Inserimento tramite GUI".

Scenario principale: L'utente aggiunge un campo indirizzo.

Postcondizioni: L'utente ha aggiunto un campo indirizzo.

UC1.1.3: Caricamento dati inseriti

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo "Inserimento tramite GUI" e ha inserito dei dati.

Scenario principale: L'utente carica i dati che ha inserito.

Postcondizioni: L'utente ha caricato i dati che ha inserito e passa alla fase di convalida di questi ultimi.

UC1.1.4: Modifica indirizzo

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo "Inserimento tramite GUI" e ha inserito un indirizzo.

Scenario principale: L'utente modifica l'indirizzo.

Postcondizioni: L'utente ha modificato l'indirizzo.

UC1.1.5: Cancella campo indirizzo

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo "Inserimento tramite GUI" e ha aggiunto un campo indirizzo.

Scenario principale: L'utente cancella il campo indirizzo.

Postcondizioni: L'utente ha cancellato il campo indirizzo.

UC1.1.6: Visualizzazione indirizzi

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo "Inserimento tramite GUI" e ha inserito uno o più indirizzi.

Scenario principale: L'utente visualizza gli indirizzi che ha inserito.

Postcondizioni: L'utente ha visualizzato gli indirizzi che ha inserito.

2.2.3.4 Use Case 1.1.6: visualizzazione indirizzi

La Figura 2.4 riporta il sotto casi d'uso relativo alla visualizzazione degli indirizzi (vedasi Figura 2.3 UC1.1.6).

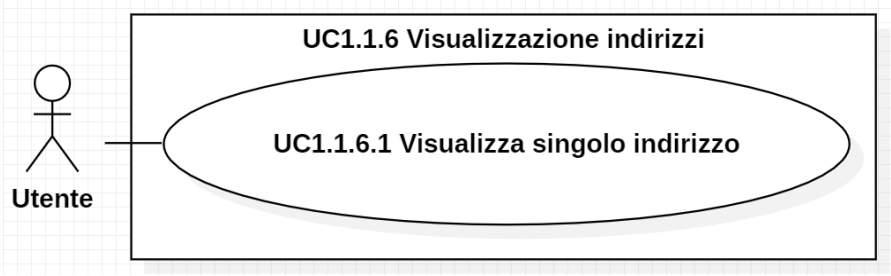

Figura 2.4: Use Case - UC1.1.6: Visualizzazione indirizzi

UC1.1.6.1: Visualizzazione singolo indirizzo

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo "Inserimento tramite GUI" e ha inserito uno o più indirizzi.

Scenario principale: L'utente visualizza un singolo indirizzo degli indirizzi che ha inserito.

Postcondizioni: L'utente ha visualizzato un singolo indirizzo degli indirizzi che ha inserito.

2.2.3.5 Use Case 1.2: Inserimento dati tramite caricamento file formattato

La Figura 2.5 riporta i sotto casi d'uso relativi all'inserimento massivo di dati tramite file formattato (vedasi Figura 2.2 UC1.2).

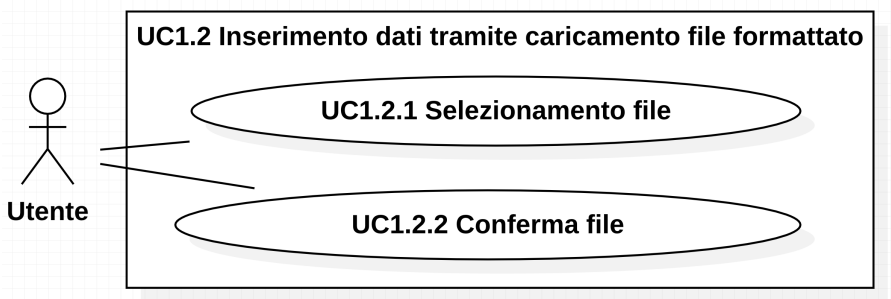

Figura 2.5: Use Case - UC1.2: Inserimento dati tramite caricamento file formattato

UC1.2.1: Selezionamento file

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo e ha selezionato il metodo "Inserimento tramite file formattato".

Scenario principale: L'utente seleziona un file.

Postcondizioni: L'utente ha selezionato un file.

UC1.2.2: Conferma file

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo "Inserimento tramite file formattato" e ha selezionato un file.

Scenario principale: L'utente conferma il file selezionato.

Postcondizioni: L'utente ha confermato il file selezionato ed il file è stato caricato. Passa alla fase di convalida dei dati appena inseriti.

2.2.3.6 Use Case 1.3: verifica dati inseriti

La Figura 2.6 riporta i sotto casi d'uso relativi alla verifica e convalidazione dei dati inseriti (vedasi Figura 2.2 UC1.3).

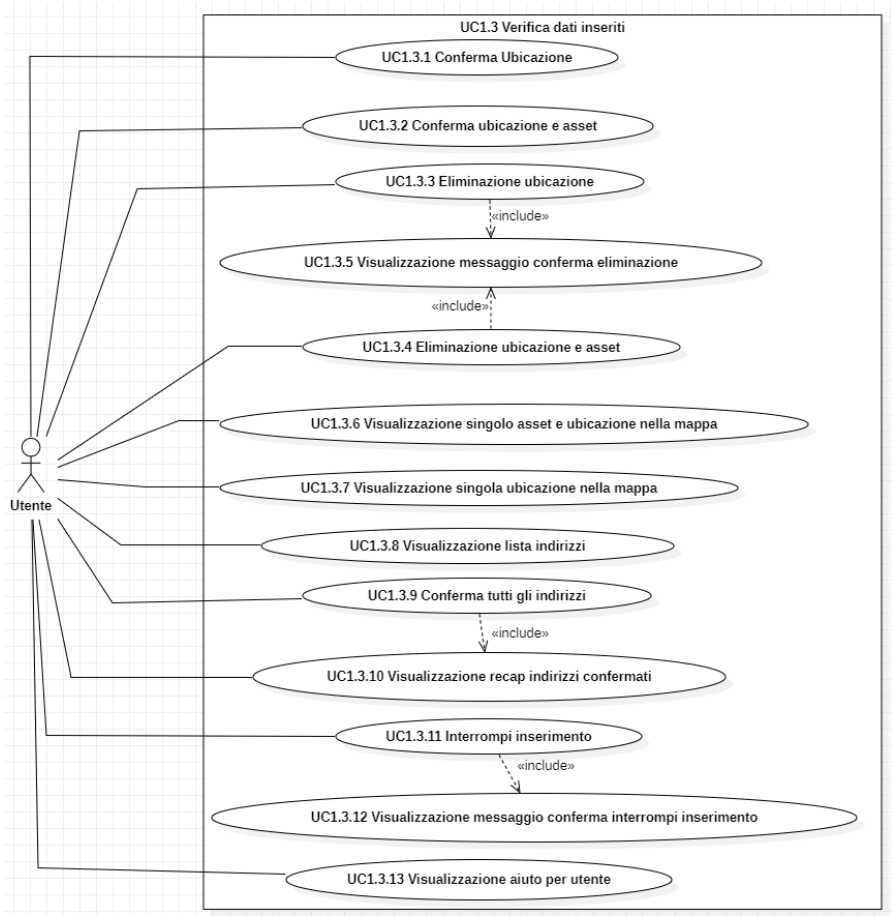

Figura 2.6: Use Case - UC1.3: Verifica dati inseriti

UC1.3.1: Conferma Ubicazione

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo per effettuarlo e ha caricato i dati.

Scenario principale: L'utente conferma l'ubicazione corrispondente all'indirizzo inserito.

Postcondizioni: L'utente ha confermato l'ubicazione corrispondente all'indirizzo inserito.

UC1.3.2: Conferma Ubicazione e asset

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni,

gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo per effettuarlo e ha caricato i dati.

Scenario principale: L'utente conferma l'ubicazione e l'asset corrispondente all'indirizzo inserito.

Postcondizioni: L'utente ha confermato l'ubicazione e l'asset corrispondente all'indirizzo inserito.

UC1.3.3: Eliminazione ubicazione

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo per effettuarlo e ha caricato i dati.

Scenario principale: L'utente elimina l'ubicazione corrispondente all'indirizzo inserito.

Estensioni: UC1.3.5.

Postcondizioni: L'utente ha eliminato l'ubicazione corrispondente all'indirizzo inserito.

UC1.3.4: Eliminazione ubicazione e asset

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo per effettuarlo e ha caricato i dati.

Scenario principale: L'utente elimina l'ubicazione e l'asset corrispondente all'indirizzo inserito.

Estensioni: UC1.3.5.

Postcondizioni: L'utente ha eliminato l'ubicazione e l'asset corrispondente all'indirizzo inserito.

UC1.3.5: Visualizzazione messaggio conferma eliminazione

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo per effettuarlo, ha caricato i dati e sta eliminando un'ubicazione e un asset.

Scenario principale: L'utente visualizza un messaggio che chiede di confermare eliminazione dell'ubicazione e dell'asset corrispondente all'indirizzo inserito.

Postcondizioni: L'utente ha visualizzato un messaggio che chiede di confermare eliminazione dell'ubicazione e dell'asset corrispondente all'indirizzo inserito.

UC1.3.6: Visualizzazione singolo asset e ubicazione nella mappa

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo per effettuarlo, ha caricato i dati e sta visualizzando la mappa di convalidazione.

Scenario principale: L'utente visualizza un singolo asset e ubicazione nella mappa.

Postcondizioni: L'utente ha visualizzato un singolo asset e ubicazione nella mappa.

UC1.3.7: Visualizzazione una singola ubicazione nella mappa

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo per effettuarlo, ha caricato i dati e sta visualizzando la mappa di convalidazione.

Scenario principale: L'utente visualizza una singola ubicazione nella mappa.

Postcondizioni: L'utente ha visualizzato una singola ubicazione nella mappa.

UC1.3.8: Visualizzazione lista indirizzi di ubicazioni e asset

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo per effettuarlo, ha caricato i dati e sta visualizzando la mappa di convalidazione.

Scenario principale: L'utente visualizza la lista degli indirizzi di ubicazioni e asset.

Postcondizioni: L'utente ha visualizzato la lista degli indirizzi di ubicazioni e asset.

UC1.3.9: Conferma tutti gli indirizzi

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo per effettuarlo, ha caricato i dati e sta visualizzando la mappa di convalidazione.

Scenario principale: L'utente convalida tutta la lista degli indirizzi di ubicazioni e asset da lui inseriti.

Estensioni: UC1.3.10.

Postcondizioni: L'utente ha convalidato tutta la lista degli indirizzi di ubicazioni e asset da lui inseriti.

UC1.3.10: Visualizzazione recap indirizzi confermati

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo per effettuarlo, ha caricato i dati e sta visualizzando la mappa di convalidazione.

Scenario principale: L'utente visualizza la lista degli indirizzi di ubicazioni e asset da lui convalidati.

Postcondizioni: L'utente ha visualizzato la lista degli indirizzi di ubicazioni e asset da lui convalidati.

UC1.3.11: Interruzione inserimento

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo per effettuarlo, ha caricato i dati e sta visualizzando la mappa di convalidazione.

Scenario principale: L'utente interrompe la fase di convalidazione degli indirizzi inseriti.

Estensioni: UC1.3.12.

Postcondizioni: L'utente ha interrotto la fase di convalidazione degli indirizzi inseriti e torna nella mappa di RiskAPP.

UC1.3.12: Visualizzazione messaggio conferma interrompi inserimento

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo per effettuarlo, ha caricato i dati e ha interrotto la fase di convalida degli inderizzi inseriti.

Scenario principale: L'utente visualizza il messaggio che chiede di confermare di voler interrompere l'inserimento massivo.

Postcondizioni: L'utente ha visualizzato il messaggio che chiede di confermare di voler interrompere l'inserimento massivo.

UC1.3.13: Visualizzazione aiuto per utente

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo per effettuarlo, ha caricato i dati e visualizza la mappa di conferma degli indirizzi.

Scenario principale: L'utente visualizza il messaggio che spiega il significato dei vari elementi presenti nella mappa di convalidazione.

Postcondizioni: L'utente ha visualizzato il messaggio che spiega il significato dei vari elementi presenti nella mappa di convalidazione.

2.2.3.7 Use Case 1.3.5: visualizzazione messaggio conferma eliminazione

La Figura 2.7 riporta i sotto casi d'uso relativi alla visualizzazione del messaggio di conferma dell'eliminazione di una ubicazione e asset (vedasi Figura 2.6 UC1.3.5).

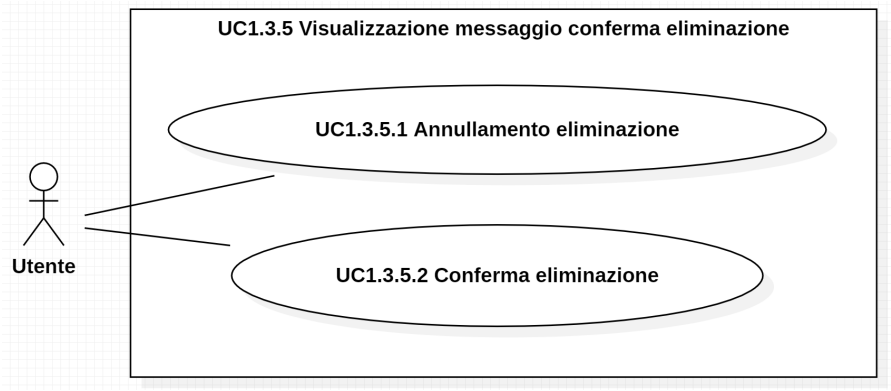

Figura 2.7: Use Case - UC1.3.5: Visualizzazione messaggio conferma eliminazione

UC1.3.5.1: Annullamento eliminazione

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo per effettuarlo, ha caricato i dati e ha visualizzato il messaggio di conferma dell'eliminazione di un'ubicazione e un asset.

Scenario principale: L'utente annulla l'eliminazione dell'ubicazione e dell'asset.

Postcondizioni: L'utente ha annullato l'eliminazione dell'ubicazione e dell'asset e quest'ultimo non è stato eliminato.

UC1.3.5.2: Conferma eliminazione

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni,

gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo per effettuarlo, ha caricato i dati e ha visualizzato il messaggio di conferma dell'eliminazione di un'ubicazione e un asset.

Scenario principale: L'utente conferma l'eliminazione dell'ubicazione e dell'asset.

Postcondizioni: L'utente ha confermato l'eliminazione dell'ubicazione e dell'asset e quest'ultimo è stato eliminato.

2.2.3.8 Use Case 1.3.8: visualizzazione lista indirizzi

La Figura 2.8 riporta il sotto caso d'uso relativo alla visualizzazione della lista degli indirizzi inseriti (vedasi Figura 2.6 UC1.3.8).

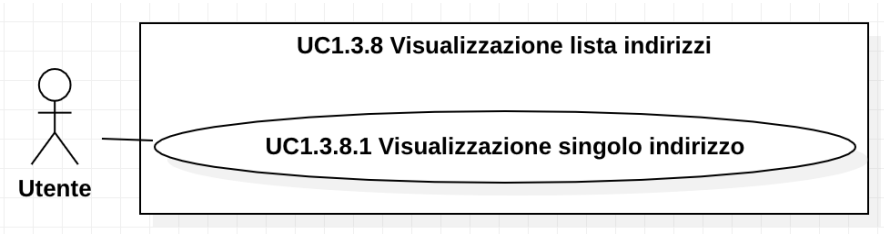

Figura 2.8: Use Case - UC1.3.8: VIsualizzazione lista indirizzi

UC1.3.8.1: Visualizzazione singolo indirizzo

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo per effettuarlo, ha caricato i dati e sta visualizzando la mappa di convalidazione.

Scenario principale: L'utente visualizza il singolo indirizzo nella lista degli indirizzi di ubicazioni e asset.

Postcondizioni: L'utente ha visualizzato il singolo indirizzo nella lista degli indirizzi di ubicazioni e asset.

2.2.3.9 Use Case 1.3.12: visualizzazione messaggio conferma interrompi inserimento

La Figura 2.9 riporta i sotto casi d'uso relativi alla visualizzazione del messaggio di conferma dell'interruzione dell'inserimento massivo (vedasi Figura 2.6 UC1.3.12).

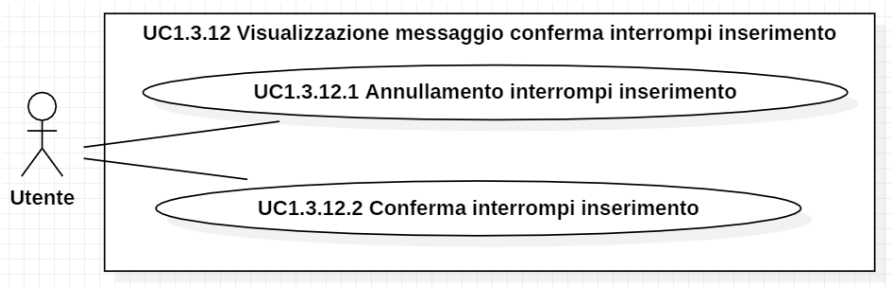

Figura 2.9: Use Case - UC1.3.11: Visualizzazione messaggio conferma interrompi inserimento

UC1.3.12.1: Annullamento interrompi inserimento

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo per effettuarlo, ha caricato i dati, ha interrotto la fase di convalida degli indirizzi inseriti e visualizza il messaggio di conferma dell'interruzione.

Scenario principale: L'utente annulla l'interruzione della convalida degli indirizzi.

Postcondizioni: L'utente ha annullato l'interruzione della convalida degli indirizzi e visualizza la mappa di convalidazione.

UC1.3.12.2: Conferma interrompi inserimento

Attori Principali: Utente.

Precondizioni: L'utente si è autenticato, ha selezionato un cliente nella dashboard ed è entrato nella mappa associata a quest'ultimo. La mappa contiene le ubicazioni, gli asset e i nodi del cliente. L'utente ha iniziato l'inserimento massivo, ha selezionato il metodo per effettuarlo, ha caricato i dati, ha interrotto la fase di convalida degli indirizzi inseriti e visualizza il messaggio di conferma dell'interruzione.

Scenario principale: L'utente conferma l'interruzione della convalida degli indirizzi.

Postcondizioni: L'utente ha confermato l'interruzione della convalida degli indirizzi e ed è tornato nella mappa del di RiskAPP.

2.3 Tracciamento dei requisiti

2.3.1 Classificazione

Da un'attenta analisi dei requisiti e degli use case effettuata sul progetto è stata stilata la tabella che traccia i requisiti in rapporto agli use case.
Sono stati individuati diversi tipi di requisiti e si è quindi fatto utilizzo di un codice identificativo per distinguerli.
Il codice dei requisiti è così strutturato:

R[Tipologia][Importanza][Codice]

dove:

* **Tipologia** può essere:
  - **F**: indica un requisito funzionale;
  - **Q**: indica un requisito qualitativo;
  - **V**: indica un requisito di vincolo.
* **Importanza** può essere:
  - **O**: indica un requisito obbligatorio;
  - **D**: indica un requisito desiderabile;
  - **F**: indica un requisito facoltativo;
* **Codice**: codice identificativo univoco per ogni requisito.

2.3.2 Elenco requisiti

2.3.2.1 Requisiti funzionali

Nella Tabella 2.1 sono elencati i requisiti funzionali relativi ai due widget per effettuare inserimento massivo.

Tabella 2.1: Tabella dei requisiti funzionali.

Requisito	Descrizione	Use Case
RFO1	L'utente, per effettuare l'inserimento di uno o più ubicazioni e asset, deve essere autenticato	
RFO2	L'utente può decidere il metodo con il quale effettuare l'inserimento massivo	UC1.7
RFO3	L'utente può inserire gli indirizzi di una o più ubicazioni e asset tramite l'interfaccia dedicata	UC1.1
RFO4	L'utente può inserire gli indirizzi di una o più ubicazioni e asset tramite il caricamento di un foglio excel debitamente formattato	UC1.2
RFO5	L'utente deve essere avvisato se il file caricato non è formattato correttamente o è avvenuto un errore durante il caricamento	UC1.3
RFO7	L'utente deve convalidare i dati inseriti	UC1.4
RFO8	Se durante la fase di convalida delle ubicazioni e asset caricati l'utente si accorge che una di queste è sbagliata può eliminarla	UC1.3.3 UC1.3.4
RFO9	L'utente deve poter convalidare un'ubicazione e asset	UC1.3.1 UC1.3.2
RFO10	L'utente deve poter visualizzare la lista di tutti gli indirizzi convalidati al termine della convalidazione	UC1.3.10
RFO11	Le ubicazioni e asset convalidati devono essere caricati nel database *back-end* e visualizzati nella mappa associata al cliente	UC1.5

RFO12	L'utente deve essere informato se è avvenuto un errore durante il caricamento dei dati nel database	UC1.6
RFO13	L'utente deve poter visualizzare un messaggio in cui si spiega che cosa deve fare nella fase di convalida dell'inserimento massivo	UC1.3.13
RFO14	L'utente, nella mappa di convalida, deve poter visualizzare le ubicazioni e asset che ha inserito	UC1.3.6 UC1.3.7
RFO15	L'utente deve poter visualizzare la lista degli indirizzi delle ubicazioni e asset che sta inserendo	UC1.3.8
RFO16	L'utente può interrompere la fase di convalida dell'inserimento massivo	UC1.3.11 UC1.3.12
RFD1	Le funzionalità implementate devono essere responsive	
RFD2	Le funzionalità implementate devono essere accessibili	

2.3.2.2 Requisiti qualitativi

Nella Tabella 2.2 sono elencati i requisiti qualitativi relativi ai due widget per effettuare inserimento massivo.

Tabella 2.2: Tabella dei requisiti qualitativi.

Requisito	Descrizione	Use Case
RQO1	Le funzionalità implementate devono essere traducibili in inglese e italiano	
RQO2	Stesura documento di analisi dei requisiti	
RQD1	Stesura manuale sviluppatore	
RQF1	Copertura test dell'80% nelle funzionalità implementate	

2.3.2.3 Requisiti di vincolo

Nella Tabella 2.3 sono elencati i requisiti di vincolo relativi ai due widget per effettuare inserimento massivo.

Tabella 2.3: Tabella dei requisiti vincolo.

Requisito	Descrizione	Use Case
RVO1	Utilizzo dei servizi API di Google per effettuare geocoding	
RVO2	Utilizzo dei servizi API di Overpass per eseguire le query sulla mappa di OpenStreetMap	
RVO3	Le funzionalità implementate devono funzionare con i browser: Google Chrome (v. 97.0.4692), Mozilla Firefox(v. 95.0) e Microsoft Edge (97.0.1072.69)	

RVO4	Utilizzo del Framework React 16.8.6 per lo sviluppo delle funzionalità	

Capitolo 3

Progettazione

3.1 Utilizzo dei servizi di Google Maps Geocoding e Overpass API

L'utente per effettuare l'inserimento massivo ha, come unico dato da inserire, gli indirizzi che corrispondono alle ubicazioni e gli asset che vuole aggiungere alla mappa. Durante l'analisi dei requisiti è quindi emersa la necessità, dato uno o più indirizzi, di individuare:

1. la loro posizione geografica, ossia effettuare geocoding;

2. trovare l'ubicazione, ed eventuale asset, che tali indirizzi rappresentano tramite geoquerying[g].

Sono stati individuati due strumenti per svolgere i compiti appena elencati:

* **Google Maps Geocoding API** [13]: utilizzato per effettuare geocoding, è un servizio web offerto da Google Maps facilmente integrabile nei progetti react tramite apposite librerie come *react-geocode* [23];

* **Overpass API** [18]: utilizzato per effettuate geoquerying, è un servizio offerto da OpenStreetMap e permette di comunicare con il database di quest'ultimo per recuperare informazioni geografiche tramite *queries*. Permette di ottenere le coordinate che delimitano il perimetro di un asset. Integrabile nei progetti react con la libreria *query-overpass* [20].

3.1.0.1 Ricerca e visualizzazione di un asset e ubicazione

Per mostrare nella mappa di convalida l'asset associato ad un indirizzo inserito, verrà eseguito il processo di geocoding che restituirà delle coordinate geografiche (nella forma [latitudine, longitudine]). Verranno poi utilizzate le coordinate trovate per effettuare geoquerying che ritornerà le coordinate che costituiscono la forma dell'asset, se contenute nel database di OpenStreetMap.
Ottenute le coordinate dell'asset, esso potrà essere evidenziato nella mappa di convalida. L'ubicazione sarà un rettangolo che contiene completamente l'asset.

3.1.0.2 Ricerca e visualizzazione di una ubicazione

Se l'indirizzo inserito non dovesse corrispondere ad un asset ma solamente ad una ubicazione, come ad esempio una proprietà terriera, la richiesta di geoquerying non ritornerà nessun risultato poichè nel database di OpenStreetMap non sono presenti questo tipo di dati. Per questo motivo, in accordo con il proponente, verrà visualizzato nella mappa un quadrato, rappresentante l'ubicazione, di dimensioni standard e centrato nelle coordinate trovate tramite geocoding.

3.1.0.3 Fallimento nella richiesta di Geocoding

Nel caso in cui il processo di geocoding non producesse nessun risultato, verrà mostrato all'utente che l'indirizzo inserito non è stato trovato.

3.2 Architettura di RiskAPP

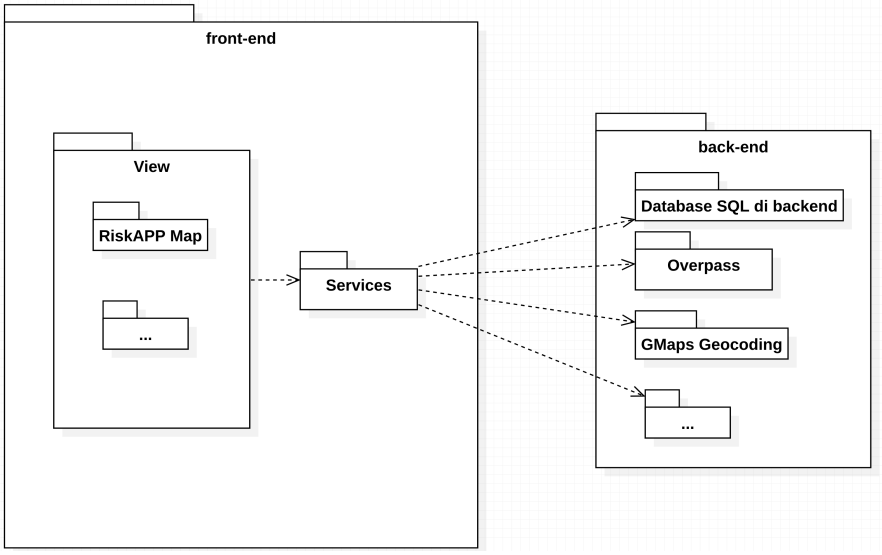

Figura 3.1: Architettura della piattaforma RiskAPP

In Figura 3.1 è riportata l'architettura della webapp RiskAPP.
Poichè l'inserimento massivo è una funzionalità che interessa unicamente la mappa di RiskAPP, tutti gli alti servizi offerti non sono presenti nel diagramma. Lo stesso vale per la parte di *back-end* dove il database, il servizio di geocoding e Overpass sono i servizi utilizzati dalla mappa, e per questo gli unici rappresentati.
Il package view contiene tutti i servizi che la piattaforma RiskAPP offre. Al suo interno, ogni servizio è così organizzato:

* `actions`: contiene i file che interessano le action di Redux;
* `asset`: contiene le immagini che vengono utilizzate;
* `components`: contiene i vari componenti da cui un servizio è composto;

- `css`: contiene i fogli di stile;
- `lib`: contiene file con funzioni di utilità o funzioni che interagiscono con le funzioni presenti nel package services.
- `reducers`: contiene i file che interassano i reducer di Redux;

Il database di *back-end* è un database relazionale[g] di tipo MySQL[g].
Per la gestione dello stato interno dell'applicazione viene utilizzato Redux[g].

3.2.1 Architettura della funzionalità di inserimento massivo

Per lo sviluppo dei due Widget per l'inserimento massivo si è seguita l'architettura presentata nella sezione precedente (vedasi sezione 3.2).
Nella cartella `services`, contenente le funzioni che permettono di utilizzare i servizi *back-end*, sono stati aggiunti i seguenti file:

- `geocoding.jsx`: che contiene le funzioni che riguardano il geocoding;
- `overpass.jsx`: che contiene le funzioni che riguardano l'interrogazione tramite *queries* al database di OpenStreetMap (geoquerying);

Lo Snippet di codice 3.1 mostra in maniera grafica come sono stati organizzati i file.

```
-src
|-riskappmap
| |-actions
| | |-massiveinsertionactions.jsx
| | |-...
| |-asset
| | |-...
| |-components
| | |-massiveinsertion
| | | |-massiveinsertionform.jsx
| | | |-massiveinsertionformgui.jsx
| | | |-massiveinsertionforminputfile.jsx
| | | |-massiveinsertionconfirmmap.jsx
| | | |-massiveinsertionconfirmmapitem.jsx
| | |-...
| |-css
| | |-massiveinsertion.css
| | |-...
| |-lib
| | |-geocoding.jsx
| | |-overpass.jsx
| | |-httppost.jsx
| | |-...
| |-reducers
| | |- massiveinsertionstore.jsx
| | |- ...
|-services
| |-geocoding.jsx
| |-overpass.jsx
| |-...
|-...
```

Snippet di codice 3.1: Organizzazione dei file relativi all'inserimento massivo

3.2.1.1 Cartella `actions`

Nella cartella `actions`, contenente i file riguardanti le action di Redux, è stato aggiunto il file `massiveinsertionactions.jsx`. Tale file contiene le action riguardanti l'inserimento massivo.

3.2.1.2 Cartella `components`

Nella cartella `components`, contenente i file relativi ai componenti da cui un servizio è composto, sono stati aggiunti i seguenti file:

* `massiveinsertionform.jsx`: che permette all'utente di selezionare il metodo con il quale effettuare l'inserimento massivo;
* `massiveinsertionformgui.jsx`: che permette all'utente di effettuare l'inserimento massivo tramite interfaccia grafica e direttamente dalla webapp;
* `massiveinsertionforminputfile.jsx`: che permette all'utente di effettuare l'inseriemento massivo tramite il caricamento di un foglio Excel debitamente formattato;
* `massiveinsertionconfirmmap.jsx`: che permette all'utente di eseguire la convalida degli indirizzi inseriti e visualizzare gli asset e ubicazioni relativi ad essi in una mappa;
* `massiveinsertionconfirmmapitem.jsx`: che permette di visualizzare nella mappa di convalida l'ubicazione e l'asset associati ad ogni indirizzo.

I file appena elencati sono raggruppati nella cartella `massiveinsertion` contenuta in `components`.

3.2.1.3 Cartella `css`

Nella cartella `css`, contenente i fogli di stile, è stato aggiunto il file `massiveinsertion.css` per definire lo stile delle componenti per l'inserimento massivo.

3.2.1.4 Cartella `lib`

Nella cartella `lib`, contenente le funzioni di utilità o che interagiscono con le funzioni presenti nel package `services`, sono stati aggiunti i seguenti file:

* `geocoding.jsx`: che chiama le funzioni contenute nel file `geocoding.jsx` della cartella `services`;
* `overpass.jsx`: che chiama le funzioni contenute nel file `overpass.jsx` della cartella `services`;
* `httppost.jsx`: contiene le funzioni che formattano i dati relativi ad asset e ubicazioni per poi chiamare le funzioni della cartella `services` per effettuare l'inserimento nel database *back-end*.

I file `geocoding.jsx` e `overpass.jsx` nella cartella `lib` sono utilizzati per far sì che i file contenuti nella cartella `components` utilizzino unicamente funzioni contenute nella cartella `lib` per interagire con il *back-end*. Questo mantiene l'architettura coerente.

3.2.1.5 Cartella `reducers`

Nella cartella `reducers`, contenente i file dei reducer di Redux, è stato aggiunto il file `massiveinsertionstore.jsx`. Tale file contiene i reducer dell'inserimento massivo.

3.3 Diagramma di attività dell'inserimento massivo

Nella Figura 3.2 viene riportato il diagramma di attività riguardante l'inserimento massivo effettuato dall'utente.

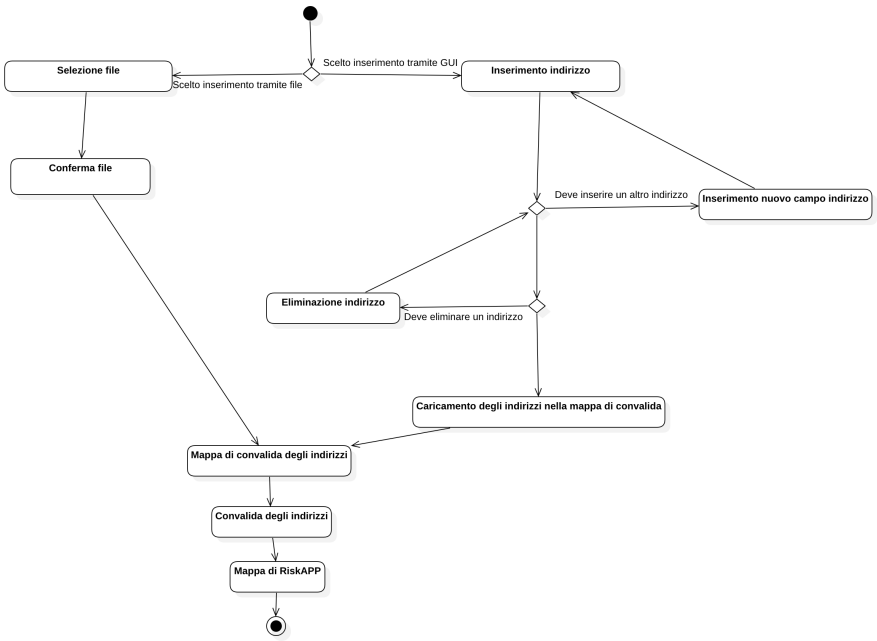

Figura 3.2: Diagramma di attività per l'inserimento massivo

L'utente sceglie la modalità con cui vuole effettuare l'inserimento massivo.

* Scegliendo la modalità "inserimento tramite file" l'utente:
  1. Seleziona il file;
  2. Conferma e carica gli indirizzi del file selezionato nella mappa di convalida;
  3. Viene reindirizzato alla mappa di convalida degli indirizzi;
  4. Convalida gli indirizzi inseriti;
  5. Viene reindirizzato alla mappa di RiskAPP nel quale visualizzerà gli indirizzi appena inseriti.
* Scegliendo la modalità "inserimento tramite GUI" l'utente:

1. Inserisce un indirizzo;
2. Se ha bisogno di inserire altri indirizzi, aggiunge un nuovo campo indirizzo e torna ad punto 1, altrimenti passa a punto 3;
3. Se vuole cancellare un campo indirizzo lo cancella e torna al punto 2 altrimenti passa al punto 4;
4. Carica gli indirizzi inseriti nella mappa di convalida;
5. Viene reindirizzato alla mappa di convalida degli indirizzi;
6. Convalida gli indirizzi inseriti;
7. Viene reindirizzato alla mappa di RiskAPP.

Nella Figura 3.3 viene analizzata in dettaglio e riportata in un diagramma l'attività di "convalida degli indirizzi".

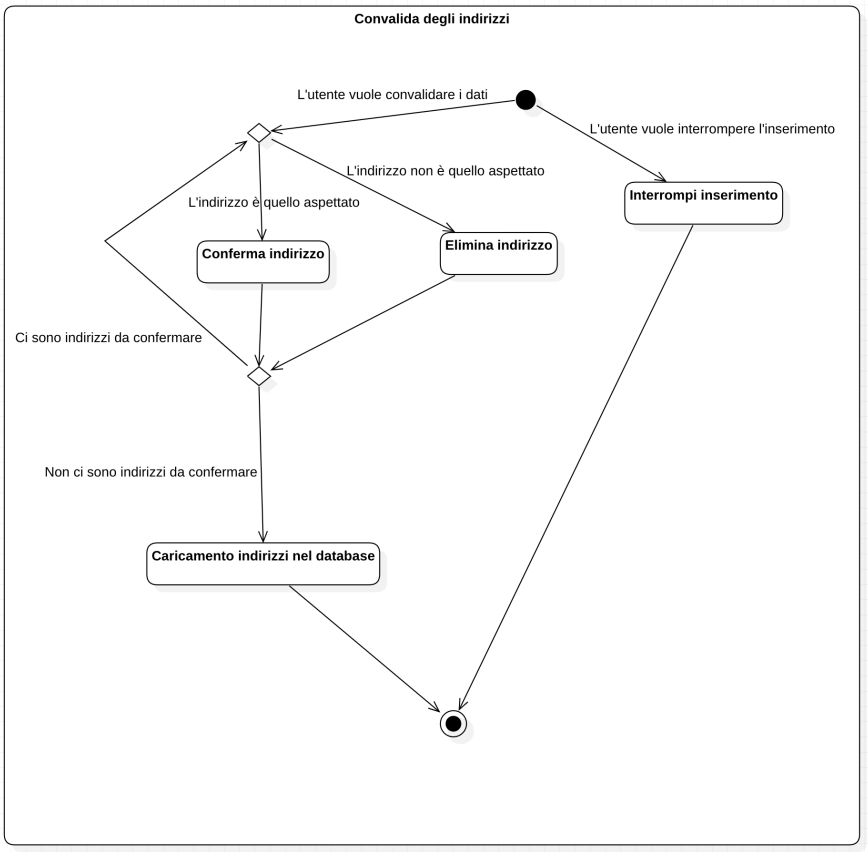

Figura 3.3: Diagramma di attività per la convalida degli indirizzi

L'utente viene reindirizzato nella pagina della mappa di convalida degli indirizzi.

∗ Volendo convalidare i dati, l'utente:

1. verifica che l'ubicazione e l'asset, associati all'indirizzo e visualizzati nella mappa di convalida, siano quelli aspettati. Se vero, l'utente conferma l'indirizzo, altrimenti elimina l'indirizzo.

2. se rimangono altri indirizzi da confermare ripete il punto 1, altrimenti tutti gli indirizzi che sono stati confermati vengono caricati dall'utente nel database.

* Non volendo convalidare i dati, l'utente interrompe l'inserimento.

Infine l'utente viene reindirizzato nella mappa di RiskAPP da cui aveva iniziato l'inserimento massivo.

Capitolo 4

Codifica

4.1 Codifica utilizzata

La UI di un'applicazione sviluppata in ReactJS è composta da componenti, pezzi di codice riutilizzabili e indipendenti. Per lo sviluppo di un componente vi sono due approcci da poter seguire:

* sviluppo a *class-component*;
* sviluppo funzionale.

La prima differenza tra i due approcci si può osservare nella sintassi della definizione di un componente. Lo Snippet di codice 4.1 presenta un componente definito secondo lo sviluppo funzionale, una semplice funzione che accetta delle props[g], e ritorna un *React element* definito dalla sintassi JSX.
Lo stesso componente si può definire utilizzando lo sviluppo a *class-component* (vedasi Snippet di codice 4.2), nel quale viene definita una classe estendendo `React.Component` al cui interno la funzione `render()` ritorna un *React element*.

```
1 function Welcome(props) {
2   return <h1>Hello, {props.name}</h1>;
3 }
```

Snippet di codice 4.1: Esempio di un componente utilizzando sviluppo funzionale

```
1 class Welcome extends React.Component {
2   render() {
3     return <h1>Hello, {this.props.name}</h1>;
4   }
5 }
```

Snippet di codice 4.2: Esempio di un componente utilizzando sviluppo a class-component

Sebbene abbiano una diversa sintassi, i due approcci sono equivalenti e restituiscono lo stesso output. Le principali differenze che i due metodi presentano sono le seguenti:

* lo sviluppo funzionale richiede meno codice [8];
* fino a React 16.8 lo sviluppo funzionale permetteva unicamente lo sviluppo di componenti stateless[g] e senza l'utilizzo dei lifecycle[g] [29];

* lo sviluppo a *class-component* dichiara gli stati di un componente all'interno del costruttore della classe, da React 16.8 in poi lo sviluppo funzionale utilizza gli *useState hooks* [29];

* lo sviluppo funzionale permette di dichiarere componenti React che vengono eseguiti dall'alto verso il basso, lo sviluppo a *class-component* instanzializza la classe.

La codifica delle delle funzionalità relative all'inserimento massivo è stata fatta utilizzando l'approccio a *class-component*. Questo approccio è stato richiesto dal tutor per uniformare il codice prodotto con il resto del codice riguardante il *frond-end* della webapp RiskAPP.

4.2 Schermata per la scelta del tipo di inserimento

La schermata per la scelta del tipo di inserimento permette all'utente di scegliere la modalità con la quale effettuarlo. Ci sono due modalità con le quali effettuare l'inserimento massivo:

* Caricare di un file formattato contenente gli indirizzi;

* inserire gli indirizzi tramite un'interfaccia dedicata.

Nella Figura 4.1 è presente la schermata per la scelta del tipo di inserimento. Viene richiesto all'utente di scegliere una delle due modalità.

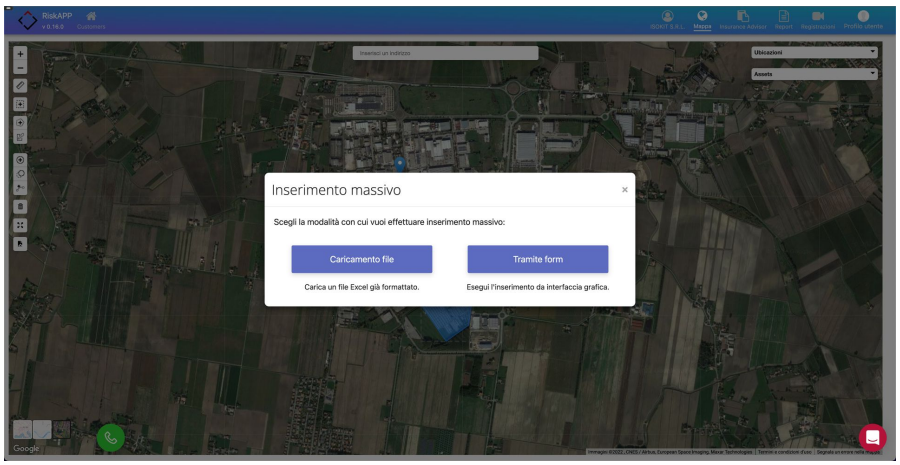

Figura 4.1: Schermata per la scelta scelta del tipo di inserimento massivo (Desktop)

Per la realizzazione di questa interfaccia è stata ampiamente utilizzata la libreria *react-bootstrap* [22] che ha fornito i *templates* HTML, CSS e JavaScript per i seguenti componenti:

* Finestra modale: la finestra "figlia" che richiede all'utente il tipo di inserimento con cui vuole effettuare l'inserimento massivo. Essa impedisce qualsiasi azione nella finestra "madre" ossia quella contenente la mappa di RiskAPP.

* Bottoni: permettono di far scegliere all'utente il tipo di inserimento.

La libreria *react-bootstrap* garantisce che i componenti creati con essa possano essere resi accessibili fino a soddisfare le WCAG[g] 2.0 (A/AA/AAA) [21].
Lo Snippet di codice 4.3 presenta il codice che renderizza la finestra modale.

```
<Modal
    show={this.state.isVisible}
    onHide={() => this.onHideHandler()}
    aria-label="Finestra modale per inserimento massivo"
    centered
>
    <Modal.Header closeButton>
        <Modal.Title>Inserimento massivo</Modal.Title>
    </Modal.Header>
    <Modal.Body>
        {this.state.isGuiFormVisible ? (
        <MassiveInsertionFormGUI
        /> //renderizza schermata di inserimento tramite form
         ) : this.state.isInputFormVisible ? (
            <MassiveInsertionFormInputFile
            /> //renderizza schermata di inserimento tramite caricamento file
          ) : (
            this.renderChooseInsertionType() //renderizza i bottoni
          )}
    </Modal.Body>
</Modal>
```

Snippet di codice 4.3: Finestra modale per la scelta della modalità con la quale effettuare l'inserimento massivo

Il contenuto del `Modal` cambia a seconda di cosa seleziona l'utente, se infatti viene premuto il tasto "Caricamento file" la finestra modale mostrerà la schermata di caricamento dati tramite file formattato. Se invece viene premuto il tasto "Tramite form", mostrerà l'interfaccia grafica nella quale inserire gli indirizzi.

4.3 Schermata di inserimento dati tramite il caricamento di un file

Scegliendo l'inserimento tramite il caricamento di un file, l'utente visualizzerà la schermata presente in Figura 4.2.

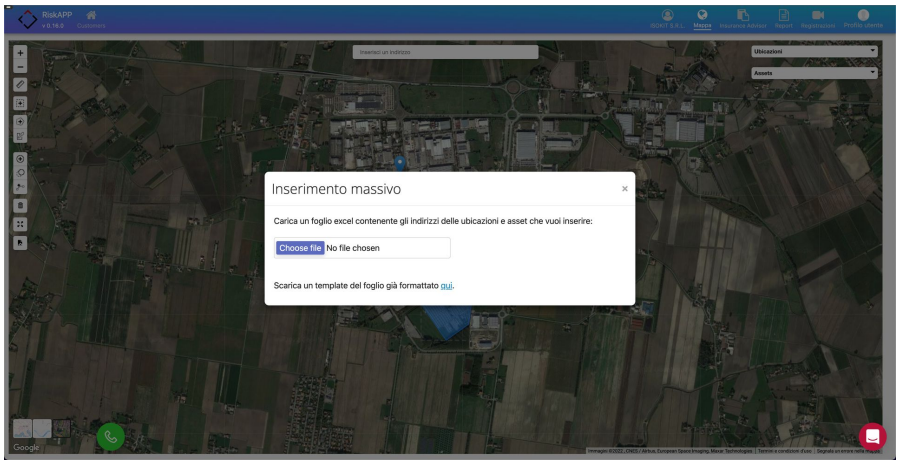

Figura 4.2: Interfaccia per il caricamento di un file formattato (Desktop)

Tale schermata è definita dal componente `<MassiveInsertionFormInputFile \>` (presente nello Snippet di codice 4.3 a riga 15). L'utente potrà caricare un file. Per evitare caricamenti errati vengono accettati unicamente file Excel (.xlsx).
Il file caricato viene letto: se il contenuto del file non è formattato correttamente, o il file è vuoto, verrà notificato un errore all'utente; se invece è correttamente formattato e non vuoto, l'utente verrà reindirizzato alla mappa di conferma degli indirizzi inseriti (vedasi sezione 4.5).
É possibile scaricare un template formattato in inglese o italiano, in base alla lingua selezionata dall'utente.
Nello Snippet di codice 4.4 è presente la funzione che, dato un file Excel, legge gli indirizzi contenuti in esso. Si basa sul completamento di una Promise[g] che rappresenta la lettura del file caricato dall'utente. Al completamento della lettura di tale file, se non si sono verificati errori, i dati vengono caricati nello store di Redux. Se non è presente nessun dato, viene notificato all'utente un messaggio di errore.

```
1  import * as XLSX from "xlsx";
2
3  readExcel(file) {
4      // lettura file Excel
5      const promise = new Promise((resolve, reject) => {
6        const fileReader = new FileReader();
7        fileReader.readAsArrayBuffer(file);
8
9        fileReader.onload = (e) => {
10         const bufferArray = e.target.result;
11         const wb = XLSX.read(bufferArray, { type: "buffer" });
12         const wsname = wb.SheetNames[0];
13         const ws = wb.Sheets[wsname];
14         const data = XLSX.utils.sheet_to_json(ws);
15
16         resolve(data);
17       };
18
19       fileReader.onerror = (error) => throw(error);
```

```
20      });
21      // conclusa la lettura del file, result rappresenta i dati letti
22      promise.then((result) => {
23        var data = [];
24        var isEmpty = true;
25        result.forEach((element) => {
26          data.push(element.ADDRESS); // viene salvato ogni elemento presente nella
27                  //colonna con l'intestazione "ADDRESS"
28          if (element.ADDRESS !== null) isEmpty = false;
29        });
30        if (isEmpty) this.renderAlert(); // mostra messaggio errore
31        else this.onSubmit(data); // carica dati in store Redux
32      });
33    }
```

Snippet di codice 4.4: Funzione per la lettura degli indirizzi dato un file Excel debitamente formattato

4.4 Schermata di inserimento tramite interfaccia grafica

Scegliendo l'inserimento tramite interfaccia grafica, l'utente visualizzerà la schermata presente in Figura 4.3.

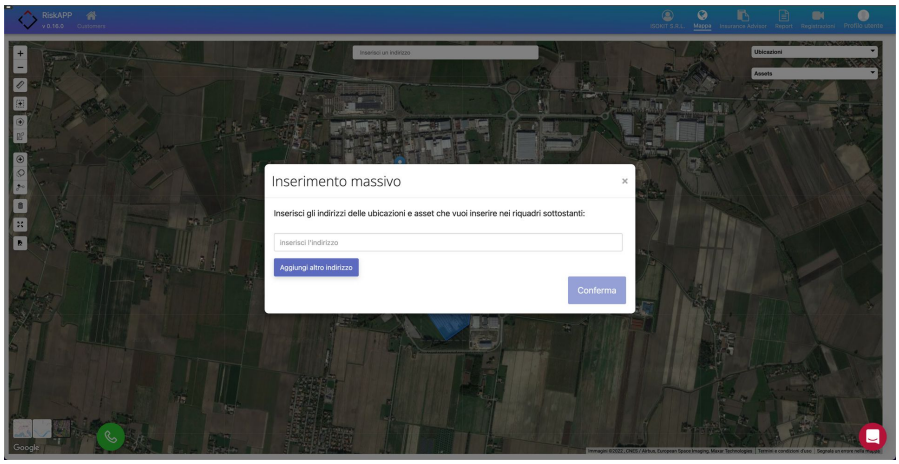

Figura 4.3: Interfaccia per l' inserimento di indirizzi (Desktop)

Tale schermata è definita dal componente `<MassiveInsertionFormGUI \>` (presente nello Snippet di codice 4.3 a riga 12).
L'utente potrà inserire gli indirizzi delle ubicazioni e degli asset che vuole caricare sulla mappa di RiskAPP.
L'utente avrà la possibilità di aggiungere e/o cancellare i campi per l'inserimento (vedasi Figura 4.4).
Concluso l'inserimento dei dati, cliccando il tasto "Conferma" l'utente verrà reindirizzato nella mappa di conferma e gli indirizzi inseriti verranno visualizzati nella mappa

con l'ubicazione e l'asset associato.
Se un campo per l'inserimento degli indirizzi rimane vuoto alla pressione del tasto conferma, tale campo verrà ignorato.

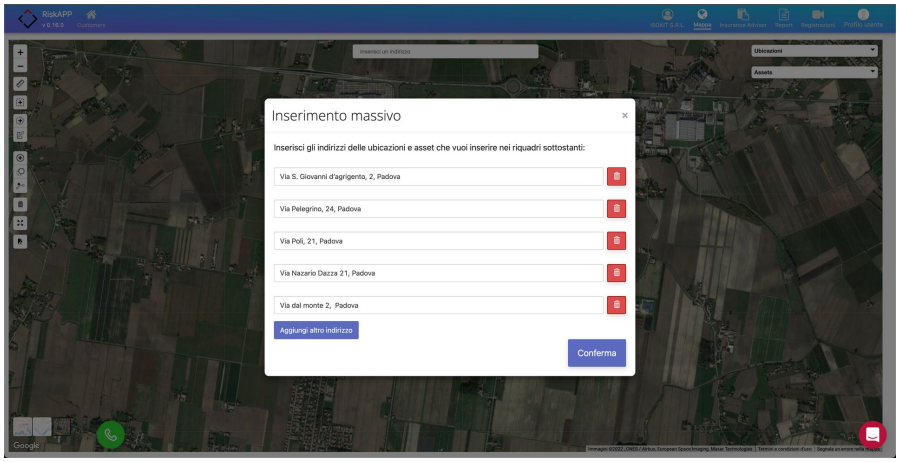

Figura 4.4: Interfaccia per l' inserimento di indirizzi con quattro campi aggiunti (Desktop)

4.4.1 Struttura di una `label`

Una `label`, ossia il campo dove inserire un indirizzo, è identificata come un oggetto JSON (vedasi Snippet di codice 4.5) composto da:

* id: numero univoco che identifica la label;
* value: valore che presenta la label.

```
{
    id: int, // diverso per ogni label inserita
    value: String,
}
```

Snippet di codice 4.5: Oggetto JSON rappresentante una label

4.4.2 Funzione `addLabelHandler`

Lo stato `labels`, presente nel costruttore di `<MassiveInsertionFormGUI \>`, è un *array* che contiene tutte le `label`.
Nello Snippet di codice 4.6 è presente la funzione che viene eseguita quando l'utente preme il tasto "Aggiungi indirizzo".
Viene creato un oggetto JSON rappresentante la `label` che poi è inserito nella copia dell'array `labels`. Infine viene settato il nuovo array contenente la nuova `label`. Poichè avviene un cambiamento di stato, il componente aggiorna la propria view mostrando la `label` appena inserita.

```
addLabelHandler() {
    var default_label = {
      id: this.state.labels.length,
```

```
      value: "",
    };

    var labelArray = this.state.labels.slice();

    labelArray.push(default_label);
    this.setState({
      labels: labelArray,
    });
  }
```

Snippet di codice 4.6: Funzione per l'inserimento di una nuova label

4.5 Mappa di conferma degli indirizzi

Dopo aver inserito gli indirizzi, tramite una delle due modalità discusse precedentemente, l'utente viene reindirizzato alla schermata di conferma degli indirizzi (vedasi Figura 4.5).

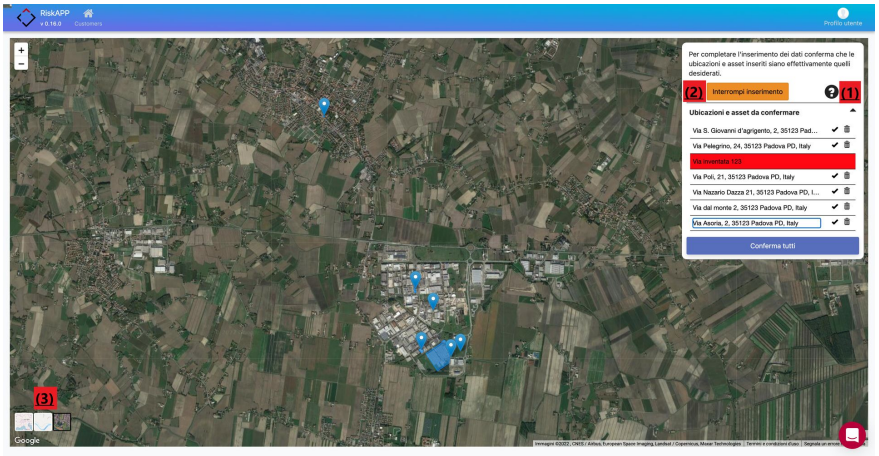

Figura 4.5: Schermata di conferma degli indirizzi (Desktop)

L'utente in questa schermata potrà verificare che gli indirizzi inseriti, e le relative ubicazioni ed asset associati, trovati tramite geocoding e geoquerying, siano quelli desiderati.
Quando avrà confermato tutti gli indirizzi visualizzerà una schermata di Recap dove potrà salvare le ubicazioni e gli asset confermati (vedasi Figura 4.7).
Cliccando su un indirizzo esso verrà evidenziato e messo in primo piano nella mappa (vedasi Figura 4.6).
L'utente potrà visualizzare un tutorial nel quale gli verrà spiegato cosa deve fare cliccando il tasto "?" **(1)** in Figura 4.5. Cliccando invece il tasto "interrompi inserimento" **(2)**, l'inserimento massivo sarà interrotto e i dati inseriti saranno persi.
L'utente può cambiare il tipo di mappa visualizzata scegliendo tra una delle tre possibili opzioni **(3)**.

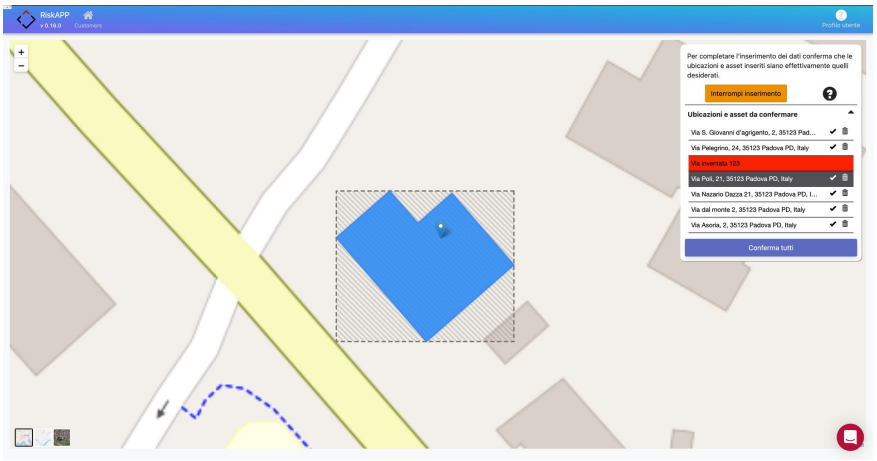

Figura 4.6: Schermata che presenta un asset un' ubicazioni evidenziati nella mappa di conferma (Desktop)

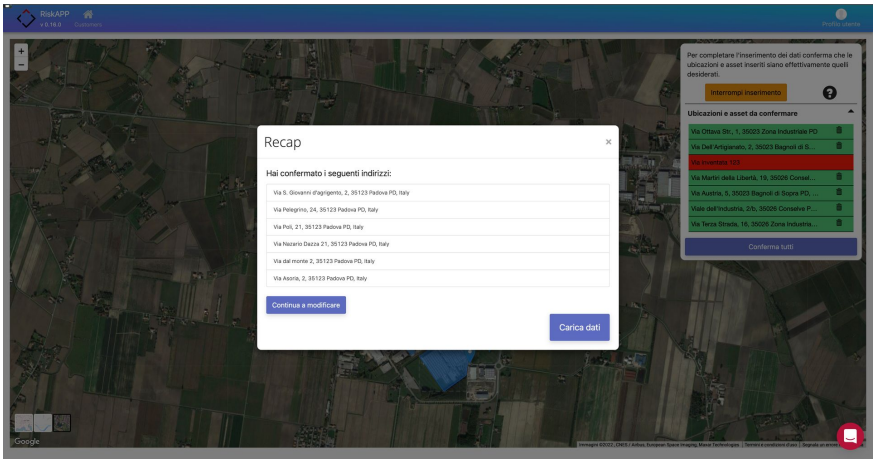

Figura 4.7: Schermata di Recap degli indirizzi confermati (Desktop)

4.5.1 Struttura di un indirizzo

Ogni indirizzo inserito dall'utente viene rappresentato nella mappa tramite un oggetto JSON (vedasi Snippet di codice 4.7).

```
{
    address: String, // indirizzo
    id: int, // identificativo univoco
    latlng: [double, double], // latitudine e longitudine
    isConfirmed: bool, // true se l'indirizzo è stato confermato, false altrimenti
    riskLocationCoordinates: [[double, double], ...], // coordinate ubicazione
    assetCoordinates: [[double, double], ...], // coordinate asset
}
```

Snippet di codice 4.7: Oggetto JSON rappresentante un indirizzo

4.5.2 Funzione `loadItems`

La funzione `loadItems` è presente nel lifecycle `componentDidMount()` che viene invocato quando il componente `<MassiveInsertionConfirmMap \>` è montato (cioè inserito nell'albero del DOM[g] [3]).
La funzione `loadItems` (vedasi Snippet di codice 4.8), per ogni indirizzo inserito dall'utente, effettua l'operazione di geocoding.
Se ha successo vengono assegnate:

* le coordinate geografiche;
* l'indirizzo formattato;
* l'id;

nell'oggetto JSON rappresentante l'indirizzo (vedasi Snippet di codice 4.7). Le coordinate dell'ubicazione e dell'asset verranno assegnate in un altro momento (vedasi sezione 4.6).
L'oggetto viene quindi inserito in un array e viene aggiornato lo stato del componente.
Se la richiesta di geocoding non ha successo viene inserito nell'array un oggetto con le coordinate nulle, rappresentati un indirizzo non esistente.
Poichè la richiesta di geocoding avviene in maniera asincrona, viene utilizzato il meccanismo delle Promise.

```
1  loadItems(data) {
2    var cnt = 0;
3    const itemArray = [];
4    data.forEach((element) => {
5      this.fromAddress(element).then((response) => {
6        if (response === null) {
7          var tmp = {
8            id: cnt++,
9            address: element,
10           latlng: null,
11           isConfirmed: false,
12           riskLocationCoordinates: null,
13           assetCoordinates: null,
14         };
15         itemArray.push(tmp);
16         this.setState({ item: itemArray });
17       } else {
18         var tmp = {
19           id: cnt++,
20           address: response.formattedAddress,
21           latlng: [response.lat, response.lng],
22           isConfirmed: false,
23           riskLocationCoordinates: null,
24           assetCoordinates: null,
25         };
26         itemArray.push(tmp);
27         this.setState({ item: itemArray});
28       }
29     });
```

```
    });
  }
```

Snippet di codice 4.8: Funzione per la formattazione e geocoding per gli indirizzi inseriti

4.5.3 Accessibilità di un Marker della mappa

La mappa è creata grazie alla libreria *react-leaflet* [24] e segue le linee guida per l' accessibilità elencate nella documentazione di *leaflet* [15].
Ogni elemento presente nella mappa è selezionabile tramite la tastiera, ogni marker presenta l'attributo `alt` che viene letto dallo *screen-reader* ogni qual volta il marker viene selezionato. Se un marker viene cliccato, viene mostrato a schermo un pop-up contenente l'indirizzo del marker, se lo screen-reader è attivo, l'indirizzo viene letto grazie all'attributo `aria-label`.
Il codice di un marker accessibile è presentato nello Snippet di codice 4.9.

```
renderMarker(address, latlng, key) {
    return latlng != null ? (
      <Marker
        alt={"Ubicazione: " + key}
        position={latlng}
        key={key}
        onclick={() => {
          this.clickMarkerHandler(key);
        }}
      >
        <Popup>
          <p aria-label="indirizzo">{address}</p>
        </Popup>
      </Marker>
    ) : null;
  }
```

Snippet di codice 4.9: Codifica di un Marker accessibile

4.6 Fase di Geoquerying

Il componente che si occupa di recuperare e visualizzare nella mappa l'ubicazione e l'asset dato un indirizzo è `<MapItem \>`, in `<MassiveInsertionConfirmMap \>` ne viene instanzializzato uno ad ogni indirizzo inserito. In tale componente viene effettuata la richiesta di geoquerying al server di Overpass. Nello Snippet di codice 4.10 è presente la query che viene svolta per ogni indirizzo.

```
const overpass = require("query-overpass");

queryRequest(lat, lng, range, overpassServer){
  const query =
        "[out:json];way(around:" +
        range +
        "," +
```

```
8           lat +
9           "," +
10          lng +
11          ")" +
12          '["building"]' +
13          ";out body;>;out skel qt;";
14        const options = {
15          flatProperties: true,
16          overpassUrl: overpassServer,
17        };
18        overpass(query, this.dataHandler, options);
19    }
```

Snippet di codice 4.10: Query per recuperare le coordinate di un asset date le coordinate di un indirizzo

Le variabili `lat` e `lng` rappresentano la latitudine e longitudine dell'indirizzo di cui si vuole trovare l'asset.
La query traccia un cerchio immaginario con raggio pari al valore (in metri) della variabile `range` e con centro nelle coordinate `[lat,lng]`. Restituisce come risultato, tramite un oggetto JSON, le coordinate di tutti gli elementi contrassegnati come `"building"` che sono completamente all'interno del cerchio tracciato.
Dovendo recuperare unicamente un asset la richiesta al server può avvenire più di una volta se il risultato trovato non fosse quello desiderato. I principali problemi che possono accadere sono i seguenti:

1. il raggio del cerchio non è abbastanza grande da includere completamente l'asset al suo interno, per questo motivo l'asset cercato non viene trovato;
2. il raggio del cerchio include più di un asset e quindi vengono restituiti come risultato più asset al posto che l'unico cercato.

Il primo problema è stato risolto ripetendo la *query* incrementando di dieci metri il valore della variabile `range` ogni volta fino a che non si trova l'asset. Per evitare *loop* di richieste verso il server, il valore massimo a cui può arrivare la variabile `range` è cento. Se nemmeno con questo valore venisse trovato un asset, viene creata un ubicazione di dimensioni standard con centro in `[lat, lng]`.
Il secondo problema, ossia trovare l'asset desiderato se vengono restituiti più asset come risposta dal server, è stato risolto definendo la funzione `isInside` (vedasi Snippet di codice 4.11) che verifica che le coordinate `[lat, lng]` corrispondino ad un punto interno all'ubicazione dell'asset trovato. Le coordinate di un' ubicazione sono i seguenti quattro punti geografici di un asset:

* latitudine e longitudine massima;
* latitudine massima e longitudine minima;
* latitudine e longitudine minima;
* latitudine minima e longitudine massima;

Nella Figura 4.6 l'asset è evidenziato in blu mentre l'ubicazione riferita all'asset è grigia.
Ne segue quindi che se le coordinate dell'indirizzo sono interne alle coordinate dell'ubicazione, l'asset trovato è quello cercato. Se le coordinate non dovessero essere

contenute nell'ubicazione trovata si passa ad un altro asset restituito dal risultato della *query*.

```
isInside(assetCordinates, lat, lng) {
  var maxLat = findMaxLat(assetCordinates); // coordinata con latitudine massima
  var minLat = findMinLat(assetCordinates); // coordinata con latitudine minima
  var maxLng = findMaxLng(assetCordinates); // coordinata con longitudine massima
  var minLng = findMinLng(assetCordinates); // coordinata con longitudine minima
  if (lat <= maxLat && lat >= minLat && lng <= maxLng && lng >= minLng)
  // se [lat, lng] contenute dentro ubicazione ritorna true
  // e coordinate dell'ubicazione
    return {
      isInside: true,
      riskLocationCordinates: [
        [maxLat, maxLng],
        [maxLat, minLng],
        [minLat, minLng],
        [minLat, maxLng],
      ],
    };
    // se [lat, lng] non contenute dentro ubicazione ritorna false
  else return { isInside: false, riskLocationCordinates: null };
}
```

Snippet di codice 4.11: Funzione per verificare che le coordinate di un indirizzo siano contenute all'interno dell'ubicazione relativa all'asset trovato

Capitolo 5

Verifica e validazione

5.1 Verifica

5.1.1 Analisi statica

L'analisi statica è un metodo di debug per il *software*. Tale metodo viene eseguito esaminando il codice senza richiederne l'esecuzione.
L'analisi statica è efficace per trovare problemi di codifica come:

* errori di programmazione;
* qualità del codice non adeguata;
* valori indefiniti;
* errori di sintassi;
* falle di sicurezza.

Durante tutto il periodo di codifica è stata eseguita l'attivita di analisi statica svolgendo le seguenti attività:

* lettura ripetitiva e attenta del codice prodotto al fine di trovare errori di programmazione, valori indefiniti o errori di sintassi;
* mantenimento di una qualità del codice adeguata effettuando code refactoring[g] e formattando in maniera automatica il codice tramite *Prettier* [19].

5.1.2 Analisi dinamica

Anche l'analisi dinamica è un metodo di debug per il *software* ma, al contrario dell'analisi statica, il *software* viene testato durante la sua esecuzione.
L'analisi dinamica è stata eseguita tramite un approccio manuale, sia per rimanere nei tempi previsti del Piano di Lavoro (vedasi sezione 6.1), sia perchè è stata data più importanza agli obiettivi obbligatori e desiderati (vedasi Tabella 1.1).
L'analisi dinamica manuale consiste nell'effettuare una sequenza di operazioni stabilite a priori per verificare che il *software* si comporti nella maniera attesa. I test sono stati eseguiti per ogni Use Case individuato nella sottosezione 2.2.3, sia simulando il comportamento atteso dell'utente, sia cercando appositamente di far fallire l'applicazione, in modo da verificare la presenza di *bug* o comportamenti imprevisti.

5.1.3 Test di accessibilità

Per garantire il livello di accessibilità richiesto sono stati eseguiti i seguenti test:

* verifica di un adeguato contrasto nei colori utilizzati. Per eseguire tale verifica è stato utilizzato WebAIM [31], verificando che fosse rispettato il livello WCAG AA;
* verifica che ogni elemento cliccabile sia selezionabile da tastiera;
* verifica che le funzionalità implementate siano utilizzabili anche attraverso l'uso esclusivo di uno *screen-reader* e di una tastiera. Lo *screen-reader* utilizzato per compiere questa verifica è stato VoiceOver, *screen-reader* predefinito di MacOS Monterey;
* verifica della correttezza semantica del codice HTML svolta tramite l'utilizzo di Markup Validation Service [30];
* verifica che ogni schermata relativa alle funzionalità implementate sia Responsive.

5.2 Validazione

Il processo di validazione è stato svolto dal proponente, in questo caso il tutor aziendale, che ha verificato che le funzionalità implementate fossero quelle attese e funzionassero correttamente.
Sono stati redatti e consegnati i seguenti documenti:

* analisi dei requisiti;
* manuale sviluppatore.

Il codice prodotto è inoltre corredato da commenti in lingua inglese che ne permettono una comprensione più semplice e veloce.

Capitolo 6

Conclusioni

6.1 Raggiungimento degli obiettivi

Lo scopo del progetto di stage era quello di sviluppare la funzionalità di inserimento massivo all'interno della mappa presente nella piattaforma RiskAPP. Era prevista la realizzazione di diverse schermate:

* per la scelta del tipo di inserimento;
* per l'inserimento tramite il caricamento di un foglio formattato;
* per l'inserimento tramite interfaccia grafica;
* per la convalida degli indirizzi.

Inoltre era richiesto il caricamento delle informazioni inserite e validate nel *database* di *back-end*. In questo capitolo si analizzano gli obiettivi raggiunti relativi allo stage svolto.

6.1.1 Completamento degli obiettivi

Con riferimento alla Tabella 1.1 nella quale sono elencati gli obiettivi di inizio stage, in Tabella 6.1 vengono riportati gli obiettivi con il loro stato di completamento.

Tabella 6.1: Tabella presentante lo stato di completamento degli obiettivi dello stage.

Codice	Descrizione	Stato
O1	Stesura documento analisi dei requisiti	Completato
O2	Sviluppo widget per inserimento massivo tramite foglio excel	Completato
O3	Sviluppo widget per inserimento massivo tramite interfaccia grafica	Completato
O4	Sviluppo schermata di conferma dei dati inseriti	Completato
D1	L'interfaccia grafica implementata deve essere accessibile a tutte le categorie di utenti	Completato
D2	Stesura documento manuale sviluppatore	Completato

F1	Sviluppo widget per la visualizzazione di grafici relativi ai rischi delle pericolosità naturali	Non completato
F2	Deve essere raggiunta tramite i test una code coverage maggiore o uguale all'80%;	Non completato

Per ragioni dovute alle tempistiche dello stage non sono stati completati i due obiettivi facoltativi F1 ed F2.

6.1.2 Soddisfacimento dei requisiti

Con riferimento alla tabella Tabella 2.1, nella quale sono elencati i requisiti funzionali, alla Tabella 2.2, nella quale sono elencati i requisiti qualitativi e alla tabella Tabella 2.3, nella quale sono riportati i requisiti di vincolo, in Tabella 6.2 vengono riportati i requisiti con il loro stato di soddisfacimento.

Tabella 6.2: Tabella presentante lo stato di soddisfacimento dei requisiti funzionali, qualitativi e di vincolo.

Requisito	Descrizione	Stato
RFO1	L'utente, per effettuare l'inserimento di uno o più ubicazioni e asset, deve essere autenticato	Soddisfatto
RFO2	L'utente può decidere il metodo con il quale effettuare l'inserimento massivo	Soddisfatto
RFO3	L'utente può inserire gli indirizzi di una o più ubicazioni e asset tramite l'interfaccia dedicata	Soddisfatto
RFO4	L'utente può inserire gli indirizzi di una o più ubicazioni e asset tramite il caricamento di un foglio excel debitamente formattato	Soddisfatto
RFO5	L'utente deve essere avvisato se il file caricato non è formattato correttamente o è avvenuto un errore durante il caricamento	Soddisfatto
RFO7	L'utente deve convalidare i dati inseriti	Soddisfatto
RFO8	Se durante la fase di convalida delle ubicazioni e asset caricati l'utente si accorge che una di queste è sbagliata può eliminarla	Soddisfatto
RFO9	L'utente deve poter convalidare un'ubicazione e asset	Soddisfatto
RFO10	L'utente deve poter visualizzare la lista di tutti gli indirizzi convalidati al termine della convalidazione	Soddisfatto
RFO11	Le ubicazioni e asset convalidati devono essere caricati nel database *back-end* e visualizzati nella mappa associata al cliente	Soddisfatto
RFO12	L'utente deve essere informato se è avvenuto un errore durante il caricamento dei dati nel database	Soddisfatto
RFO13	L'utente deve poter visualizzare un messaggio in cui si spiega che cosa deve fare nella fase di convalida dell'inserimento massivo	Soddisfatto

RFO14	L'utente, nella mappa di convalida, deve poter visualizzare le ubicazioni e asset che ha inserito	Soddisfatto
RFO15	L'utente deve poter visualizzare la lista degli indirizzi delle ubicazioni e asset che sta inserendo	Soddisfatto
RFO16	L'utente può interrompere la fase di convalida dell'inserimento massivo	Soddisfatto
RFD1	Le funzionalità implementate devono essere responsive	Soddisfatto
RFD2	Le funzionalità implementate devono essere accessibili	Soddisfatto
RQO1	Le funzionalità implementate devono essere traducibili in inglese e italiano	Soddisfatto
RQO2	Stesura documento di analisi dei requisiti	Soddisfatto
RQD1	Stesura manuale sviluppatore	Soddisfatto
RQF1	Copertura test dell'80% nelle funzionalità implementate	Non soddisfatto
RVO1	Utilizzo dei servizi API di Google per effettuare geocoding	Soddisfatto
RVO2	Utilizzo dei servizi API di Overpass per eseguire le query sulla mappa di OpenStreetMap	Soddisfatto
RVO3	Le funzionalità implementate devono funzionare con i browser: Google Chrome (v. 97.0.4692), Mozilla Firefox(v. 95.0) e Microsoft Edge (97.0.1072.69)	Soddisfatto
RVO4	Utilizzo del Framework React 16.8.6 per lo sviluppo delle funzionalità	Soddisfatto

Al termine del periodo di stage, il prodotto soddisfa 24 dei 25 requisiti individuati tramite l'analisi dei requisiti svolta. Il requisito RQF1 non è stato soddisfatto per ragioni legate alla tempistiche dello stage e il suo compimento avrebbe richiesto un dispendio valutato di circa 20 ore lavorative.
Non è stata svolta l'analisi dei requisiti per l'obbiettivo F1 (vedasi Tabella 6.1) poichè si è preferito completare gli obiettivi e soddisfare i requisiti obbligatori e desiderabili prima di iniziare gli obiettivi facoltativi. Il tempo stimato per il completamento di tale obbiettivo è di 15 ore lavorative.
Le ragioni di questo ritardo sono le seguenti:

* non avendo esperienze con il *framework* React, il linguaggio è stato imparato durante la prima fase dello stage. La fase di formazione su questo *framework* ha richiesto circa 80 ore, 16 in più di quelle preventivate;

* il linguaggio per effettuare geoquerying ha necessitato di una formazione di 10 ore che non erano state preventivate;

* il tutor ha richiesto la sistemazione di alcuni *bug* che avevo fatto notare presenti nella mappa di RiskAPP. Per sistemarli sono state impiegate 10 ore lavorative.

Sebbene non siano stati completati tutti gli obiettivi previsti, l'azienda si è comunque detta molto soddisfatta del lavoro svolto e della documentazione fornita.

6.2 Valutazione personale

La principale tecnologia utilizzata, ossia ReactJS, è stata studiata appositamente per lo svolgimento dello stage. Ritengo che la conoscenza di tale tecnologia sia molto importante per uno sviluppatore essendo tra i *framework* più utilizzati per lo sviluppo di siti Web.
Un altro *framework* che ho avuto modo di utilizzare durante lo stage è Bootstrap, in grado di fornire componenti React già provvisti di uno stile ma altamente personalizzabili e accessibili. I componenti forniti da Bootstrap riducono in modo evidente i tempi per lo sviluppo del prodotto.
Infine ho avuto modo di utilizzare i servizi offerti da Google Maps che mette a disposizione degli sviluppatori un ampio raggio di funzioni in grado di interagire con la mappa e recuperare diverse informazioni da essa.
In generale ho avuto modo di gestire tutto il percorso per lo sviluppo delle funzionalità implementate: partendo dallo svolgimento dell'analisi dei requisiti con il proponente, passando alla presentazione di un *mockup* grafico, svolgendo la fase di progettazione e codifica, e concludendo con la redazione della documentazione e la fase di validazione.
Le 320 ore di stage svolte mi hanno permesso di affinare le mie conoscenze informatiche e organizzative, dandomi la possibilità di applicare le nozioni imparate durante il mio percorso universitario.

Acronimi e abbreviazioni

DOM Document Object Model. 47, 59

GUI Graphical User Interface. xi, 2, 17–19

IDE Integrated development environment. 2, 59

SaaS Software as a Service. 60

UC Use Case. 13, 14

UI User interface. 4–6, 39

UML Unified Modeling Language. 13, 60

Glossario

Code Refactoring è un'attività di modifica del codice nella quale si interviene nella struttura di esso senza modificare il comportamento esterno. L'obbiettivo è quello di migliorare alcune caratteristiche non funzionali del *software* quali la leggibilità, la riusabilità o la sua estensibilità. 51

Continous Delivery in ingegneria del software la *Continous Delivery* è una pratica nella quale i team producono *software* in cicli brevi, assicurandosi che il *software* possa essere rilasciato in modo affidabile, in ogni momento e in modo automatico. L'obbiettivo della *Continous Delivery* è creare, testare e rilasciare codice con maggiore velocità e frequenza riducendo i costi e i tempi [4] . 1, 2

Continous Integration in ingegneria del software la *Continous Integration* è una pratica che si applica a contesti in cui lo sviluppo del *software* avviene attraverso un sistema di controllo di versione. Consiste nell'allineamento frequente degli ambienti di lavoro degli sviluppatori verso l'ambiente condiviso [5] . 1, 2

Database MySQL è un *software open-source* per la gestione di un database relazionale composto da un client a riga di comando e un server [6]. 33

Database relazionale è una raccolta di elementi dati tra i quali sussistono relazioni predefinite. Questi elementi sono organizzati sotto forma di set di tabelle con righe e colonne. Le tabelle vengono usate per contenere le informazioni sugli oggetti da rappresentare nel database [7]. 33

DOM è la descrizione della struttura interna di una pagine Web vista come gerarchia di oggetti JavaScript. Tramite il DOM si può accedere alla struttura della pagina Web e modificarla in modo dinamico [9]. 57

Geocoding è il processo nel quale, dato un indirizzo, vengono restituite le coordinate geografiche che lo identificano nella superficie Terrestre. 2, 31–33, 45, 47

Geoquerying è il processo nel quale viene interrogato un database contenente dati geografici per estrarre informazioni spaziali di unità geografiche. Le unità geografiche possono essere costruzioni, parchi, strade, fiumi, città etc.. . 31–33, 45, 48, 55

IDE è un' applicazione *software* che fornisce servizi completi ai programmatori. Normalmente consiste in un editor per il codice sorgente, strumenti di automazione della build e un debugger [14] . 57

Lifecycle sono metodi che hanno lo scopo di gestire il ciclo di vita di un componente React. I principali sono render(), componentDidMount(), componentWillUnmount() etc.. . 39, 47

Merge è il processo nel quale il codice sviluppato in due *branch* differenti viene unito in un unico *branch*. 8

Mobile-first è un approccio al *web design* pensato per l'ottimizzazione dei siti sui dispositivi mobili. Quindi prima vengono create pagine ottimizzate per i dispositivi mobili, poi seguono successive estensioni per i siti per desktop. 6

Promise in JavaScript rappresentano operazioni incomplete nel momento corrente che saranno completate in futuro. Vengono quindi utilizzate per elaborazioni asincrone. Una Promise può avere tre stati: "pending" se è in attesa, "fulfilled" se l'operazione si è conclusa con successo e "rejected" se è fallita. 42, 47

Props Le props sono tutti quegli oggetti, funzioni, stringhe, numeri, array, ecc. che possono essere passati ad un componente React per essere utilizzati al suo interno . 39

Redux è una libreria per la gestione degli stati che viene comunemente utilizzata in React per mantenere i cambiamenti di stato più prevedibili e tracciabili. Essa è basata su uno store centrale, che memorizza le informazioni, al quale si può accedere da qualsiasi punto dell'app tramite selettore. Gli stati all'interno dello store possono essere modificati effettuando un *dispatch* di un'azione [25]. 33–35, 42

Responsive in ambito informatico rappresenta la capacità di un sito web di adattarsi graficamente e in modo automatico al dispositivo con il quale viene visualizzato, riducendo al minimo la necessità dell'utente di ridimensionare e scorrere i contenuti. 6, 52

SaaS è un modello di servizio del software applicativo realizzato da un produttore che mette a disposizione un programma, direttamente o tramite terze parti, con modalità telematica come ad esempio un'applicazione web. Il software utilizzato dal cliente non è installato localmente, ma viene messo a disposizione tramite una connessione internet (previo eventuale abbonamento). Il cliente paga l'utilizzo del software, non il suo possesso [28] . 1, 57

Scrum è un *framework agile* per la gestione del ciclo di sviluppo del software, iterativo ed incrementale, concepito per gestire progetti e prodotti sofware o applicazioni di sviluppo. Esso si basa sulla divisione del lavoro in *sprints* con durata fissa di generalmente quattro settimane [27] . 1

Componenti Stateless Sono componenti React che non presentano stati al loro interno . 39

UML in ingegneria del software *UML, Unified Modeling Language* (ing. linguaggio di modellazione unificato) è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. L'*UML* svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. 57

WCAG sono le linee guida per l'accessibilità dei contenuti Web. Seguirle permette di rendere accessibili i contenuti di un sito Web ad un ampio numero di persone con disabilità, tra le quali cecità, ipovisione, sordità e limitazioni motorie. In generale seguire le WCAG permette di rendere i contenuti offerti più usabili dagli utenti in generale. 41

Bibliografia

Siti web consultati

[1] *Browser supportati da ECMAscript 6*. URL: https://www.w3schools.com/js/js_es6.asp#:~:text=Browser%20Support%20for%20ES6%20(2015) (cit. a p. 4).

[2] *Classifica di popolarità dei linguaggi di programmazione di PYPL*. URL: https://pypl.github.io/PYPL.html (cit. a p. 3).

[3] *componentDidMount(): documentazione*. URL: https://it.reactjs.org/docs/react-component.html#componentdidmount (cit. a p. 47).

[4] *Continous Delivery: definizione*. URL: https://en.wikipedia.org/wiki/Continuous_delivery (cit. a p. 59).

[5] *Continous Integration: definizione*. URL: https://it.wikipedia.org/wiki/Integrazione_continua (cit. a p. 59).

[6] *database MySQL: definizione*. URL: https://it.wikipedia.org/wiki/MySQL (cit. a p. 59).

[7] *database relazionale: definizione*. URL: https://aws.amazon.com/it/relational-database/ (cit. a p. 59).

[8] *Differenze tra sviluppo funzionale e sviluppo a class-component*. URL: https://djoech.medium.com/functional-vs-class-components-in-react-231e3fbd7108 (cit. a p. 39).

[9] *DOM: definizione*. URL: https://www.geekandjob.com/wiki/dom (cit. a p. 59).

[10] *Git Feature Branch Workflow: come funziona*. URL: https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow#:~:text=their%20branching%20models.-,How%20it%20works,-The%20Feature%20Branch (cit. a p. 8).

[11] *Git: definizione*. URL: https://git-scm.com/ (cit. a p. 8).

[12] *GitHub: definizione*. URL: https://docs.github.com/en/get-started/using-git/about-git (cit. a p. 8).

[13] *Google Maps Geocoding API*. URL: https://developers.google.com/maps/documentation/geocoding (cit. a p. 31).

[14] *IDE (Integrated development environment): definizione*. URL: https://en.wikipedia.org/wiki/Integrated_development_environment (cit. a p. 59).

[15] *leaflet map accessibility*. URL: https://leafletjs.com/examples/accessibility/ (cit. a p. 48).

[16] *Manifesto Agile*. URL: http://agilemanifesto.org/iso/it/.

[17] *npm: definizione*. URL: https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/ (cit. a p. 7).

[18] *Overpass API*. URL: https://wiki.openstreetmap.org/wiki/Overpass_API (cit. a p. 31).

[19] *Prettier: Opinionated Code Formatter*. URL: https://prettier.io/ (cit. a p. 51).

[20] *query-overpass package*. URL: https://www.npmjs.com/package/query-overpass (cit. a p. 31).

[21] *react-bootstrap accessibility*. URL: https://getbootstrap.com/docs/4.0/getting-started/accessibility/ (cit. a p. 41).

[22] *react-bootstrap package*. URL: https://www.npmjs.com/package/react-bootstrap (cit. a p. 40).

[23] *react-geocode package*. URL: https://www.npmjs.com/package/react-geocode (cit. a p. 31).

[24] *react-leaflet package*. URL: https://react-leaflet.js.org/docs/start-introduction/ (cit. a p. 48).

[25] *Redux: definizione*. URL: https://segunsubair.medium.com/redux-react-state-management-23be854799d2 (cit. a p. 60).

[26] *RiskApp: clienti e partner commerciali*. URL: https://riskapp.it/#:~:text=Customers%20%26%20Partners (cit. a p. 1).

[27] *Scrum: definizione*. URL: https://it.wikipedia.org/wiki/Scrum_(informatica) (cit. a p. 60).

[28] *Software as a Service: definizione*. URL: https://it.wikipedia.org/wiki/Software_as_a_service (cit. a p. 60).

[29] *Sviluppo funzionale e sviluppo a class-component*. URL: https://reactjs.org/docs/components-and-props.html (cit. alle pp. 39, 40).

[30] *W3C Markup Validation Service*. URL: https://validator.w3.org/ (cit. a p. 52).

[31] *WebAIM: Contrast Checker*. URL: https://webaim.org/resources/contrastchecker/ (cit. a p. 52).