

UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

Corso di Laurea Triennale in Informatica



Definizione di un'interfaccia per la creazione e verifica dell'integrità di regole IFTTT

Relatori:

Prof. Vincenzo Deufemia

Dott. Bernardo Breve

Candidato:

Pasquale Esposito

Matr. 0512105774

ANNO ACCADEMICO 2020/2021

*Ai miei genitori e a mio
fratello*

Abstract

Grazie ai costi sempre più contenuti dei dispositivi IoT, molti utenti decidono di collegare i device posseduti tramite la definizione di regole *Event Condition Action* (ECA). Tali regole, che possono essere definite attraverso le numerose applicazioni web e mobile presenti in rete, possono però potenzialmente causare all'utente problemi di privacy o sicurezza causandogli danni fisici, personali o cybersecurity. IFTTT (acronimo di *If This Then That*) è uno dei servizi più utilizzati per creare queste regole grazie soprattutto alla sua facilità di utilizzo e alla vasta offerta di servizi che mette a disposizione agli utenti.

Il seguente lavoro di tesi è incentrato sull'implementazione di un'applicazione web che, utilizzando un modulo di Intelligenza Artificiale sviluppato dal dottor Cimino dal laboratorio di Data Science e Machine Learning, appartenente al dipartimento di Informatica dell'Università degli Studi di Salerno, permette di identificare se una regola creata tramite l'interfaccia utente, andando ad utilizzare gli stessi servizi che offre IFTTT, oppure una lista di regole caricate tramite un file celano problematiche basandosi sui campi da cui sono composte tali regole. Lo scopo dell'interfaccia è quello di andare a raccogliere i feedback degli utenti circa i risultati che fornisce il modello di classificazione.

Indice

1	Introduzione	1
2	Stato dell'arte	3
2.1	Metafore ed interfacce per la definizione di regole di comportamento in contesti IoT	3
2.1.1	EFESTO	5
2.1.2	Creazione di regole ECA dipendenti dal contesto di esecuzione	8
2.2	Modelli di AI per la privacy disclosure	10
2.3	Interfacce per il <i>crowdsourcing</i>	13
3	IFTTT	16
3.1	Descrizione della piattaforma	16
3.2	Storia di IFTTT	18
3.3	Funzionamento di IFTTT	20
3.4	Sicurezza	23

4	Classificatore di applet IFTTT basato su BERT	27
4.1	Descrizione del modello BERT	27
4.2	Il classificatore	29
4.2.1	Descrizione del <i>dataset</i> e delle categorie	30
4.2.2	Etichettatura	32
4.2.3	Classificazione sul <i>dataset</i> etichettato manualmente	33
4.2.4	Tecniche di apprendimento semi-supervisionato	35
5	Lavoro svolto	36
5.1	Struttura della piattaforma web	36
5.2	Firebase Realtime Database	39
5.3	Google Colab	41
5.4	React	43
5.4.1	Descrizione della sezione “Crea una regola”	44
5.4.2	Descrizione della sezione “Carica un file di regole”	49
6	Conclusioni	53

Elenco delle figure

2.1	Esempio di regola creata con E-Free con una catena di due eventi e 2 <i>actions</i>	7
2.2	Esempio di regola creata in modo dipendente al contesto di esecuzione. In questa immagine si può notare in alto come al momento sia stato scelto solamente l'evento mentre l'azione deve essere ancora settata.	10
2.3	Esempio di aggiunta di un'informazione sulla mappa di Waze. . . .	15
3.1	Creazione di una regola da dispositivo mobile.	19
3.2	Aggiunta di un <i>ingredient</i>	19
3.3	Funzionamento dell'ecosistema IFTTT.	21
3.4	Reticolo della segretezza.	23
3.5	Reticolo dell'integrità.	25
4.1	Esempio di applet presente nel <i>dataset</i>	31
5.1	Interfaccia con cui interagire per creare una regola.	37
5.2	Interfaccia con cui interagire per caricare un file di regole JSON. . .	37

5.3	Esempio di verifica di una regola creata nell'applicazione web. . . .	38
5.4	Esempio di verifica di un file caricato nell'applicazione web. . . .	38
5.5	Esempio di record salvato in Firebase.	40
5.6	Esempio di URL dopo la classificazione dell'applet	42
5.7	Invio di informazioni attraverso l'URL.	42
5.8	Sezione che mostra tutti i servizi messi a disposizione da IFTTT per la scelta dell' <i>action</i>	45
5.9	<i>Modal</i> utilizzato per la scelta dell' <i>action</i>	46
5.10	Scelta del <i>trigger</i>	47
5.11	<i>Modal</i> che viene mostrato al termine della creazione della regola. .	47
5.12	Interfaccia visibile dopo aver fatto il <i>sumbit</i> di una regola.	48
5.13	Risultati delle regole caricate attraverso un file JSON.	50
5.14	<i>Spinner</i> visualizzato durante il caricamento <i>realtime</i> delle applet caricate da un file.	50

Capitolo 1

Introduzione

L'*Internet of Things* (IoT) è una rete di oggetti fisici (o *smart objects*) quali device, strumenti, veicoli e altri oggetti dotati di circuiti elettronici, software, sensori e connessione alla rete che permettono a tali oggetti di comunicare tra loro e scambiarsi informazioni [7].

Questo paradigma è sicuramente applicabile in vari ambiti, a partire da quello medico (attualmente utilizzato nel monitoraggio e controllo dei pazienti, monitoraggio continuo del diabete, erogazione automatizzata di insulina [10]) oppure nelle *Smart Factory*, dove ci sono macchine che riescono ad organizzare la produzione automaticamente ed autonomamente, fino ad arrivare alla domotica, la quale migliora il livello di sicurezza e la qualità della vita delle persone all'interno delle proprie case.

Con l'avvento di questa nuova tecnologia all'interno delle case è nata anche la possibilità di permettere ai possessori di tali device di creare delle *routine*, ovvero delle azioni che vengono eseguite ogni volta che accade un preciso evento, come ad esempio "Ogni volta che il campanello suona, accendi la luce dell'uscio". Queste routine sono create basandosi sul modello ECA (*Event Condition Action*), un modello di definizione di una regola composto da un evento e dalla corrispondente azione. Le parti da cui è composta una regola ECA sono: un evento (*event*), la condizione (*condition*) che, se verificata dall'evento causa l'azione (*action*). Un

evento può essere anche chiamato *trigger*, un'azione *action*.

Molte applicazioni web e per smartphone sono nate per consentire la creazione di regole anche ad utenti con scarsa esperienza nel campo, come ad esempio IFTTT (acronimo di *If This Then That*), una delle piattaforme più utilizzate al mondo per la creazione di regole in ambito domotico grazie alla sua facilità d'uso ed alla sua interfaccia *user-friendly*. Regole del genere, però, molto spesso sono sottovalutate o nascondono potenziali pericoli (fisici, personali o cybersecurity) non del tutto visibili agli utenti che le definiscono.

Il seguente lavoro di tesi è incentrato sull'implementazione di un'interfaccia web utile per la creazione e verifica dell'integrità di regole IFTTT. In particolare, un utente può verificare e controllare se regole che ha creato (o che ha intenzione di creare) possono potenzialmente creare attacchi di tipo fisico, personale o cybersecurity.

Il documento di tesi è strutturato come segue: nel secondo capitolo è presentato lo stato dell'arte e le varie pubblicazioni consultate; il terzo capitolo spiega, in modo approfondito, cos'è IFTTT e il suo funzionamento; il quarto capitolo mostra il modello di Intelligenza Artificiale utilizzato per andare a classificare le regole; il quinto capitolo mostra il lavoro di tesi svolto, spiegando dettagliatamente il funzionamento della piattaforma web; nel sesto capitolo sono fornite le conclusioni del lavoro svolto con sviluppi futuri della piattaforma presentata.

Capitolo 2

Stato dell'arte

In questo capitolo si andrà a trattare rispettivamente dei vari paradigmi e interfacce utili per la creazione di routine che possono essere svolte da dispositivi IoT, dei problemi di privacy che l'*Artificial Intelligence* (Intelligenza Artificiale o AI) può causare andando ad analizzare i dati degli utenti su varie applicazioni che si utilizzano (e i metodi emergenti che aiutano la limitazione di questi problemi) e delle applicazioni *crowdsourcing* che, grazie all'aiuto di utenti qualsiasi fruitori di quest'ultime, permettono lo sviluppo dei programmi ricevendo feedback direttamente da utilizzatori qualsiasi.

2.1 Metafore ed interfacce per la definizione di regole di comportamento in contesti IoT

Con il progredire di oggetti smart *low-cost*, il mondo dell'IoT è diventato alla portata di tutti. Gli studi mostrati nei due successivi paragrafi si sono posti il problema di come permettere ad utenti che non hanno alcun tipo di background informatico di approcciarsi a queste tecnologie senza aver difficoltà nel creare ed impostare routine che vengono svolte attraverso regole ECA (*event-condition-action*). Molte piattaforme Web e applicazioni per smartphone sono nate per crea-

re queste regole, in modi più o meno semplici, utilizzando molteplici paradigmi e metafore per aiutare e facilitare gli *end-users* nella creazione di queste regole.

IFTTT¹, ad esempio, utilizza il *wizard paradigm*, ovvero aiuta l'utente con una procedura guidata che quindi non consente errori durante la creazione di nuove regole. Questo modo di aiutare appena citato ha fatto sì che questo sito Web diventasse uno dei più famosi per creare regole. Sicuramente la facilità d'uso di questo strumento è un punto di forza che tuttavia ha delle limitazioni, la prima su tutte la scarsa personalizzazione delle regole, non potendo essere molto complesse.

Un'altra piattaforma che permette la creazione di regole è Node-RED². Node-RED utilizza un altro paradigma per la creazione delle regole basato sul *wired programming*. Questo tipo di approccio permette la creazione di regole più espressive e complesse utilizzando i grafi come rappresentazione della regola. I grafi sono composti da nodi ed archi. Nel caso di Node-RED, i nodi sono i web services che l'utente ha intenzione di utilizzare mentre gli archi tra i nodi rappresentano le connessioni tra i vari web services. Inoltre Node-RED permette anche la registrazione di oggetti smart non predefiniti all'interno dell'applicazione invocando le loro interfacce RESTful, un passaggio chiaramente difficile da effettuare da utenti che non hanno un background informatico.

Grazie alla rappresentazione di regole attraverso i grafi, Node-RED permette anche la creazione di regole complesse, ovvero caratterizzate da più *trigger* e/o più *action*. Tuttavia, il tool appena presentato può essere utilizzato con facilità soltanto da utenti con già dimestichezza e *skills* nel campo della programmazione. Pertanto Node-RED è una piattaforma consigliata ad un numero di utenti molto ristretto poiché l'utilizzo di questa applicazione ha bisogno di conoscenze preliminari su cosa sia un grafo, in che modo si possa costruire una regola con un grafo o in che modo si possano aggiungere nuovi web services. Sicuramente la piattaforma IFTTT, sotto questi punti di vista, è molto più semplice da utilizzare visto che non ha bisogno di queste conoscenze per essere utilizzata.

Un paradigma molto più *user-friendly* è quello utilizzato da Zipato³, che permet-

¹<https://ifttt.com/home>

²<https://nodered.org/>

³<https://www.zipato.com/>

te cioè la creazione di regole utilizzando dei blocchi che possono essere uniti tra loro, rifacendosi cioè alla metafora del puzzle. Secondo uno studio effettuato da Rahmati et al. [13], tuttavia, nonostante Zipato abbia un paradigma semplice da comprendere, si rivolge ad un'audience differente da IFTTT. Infatti, IFTTT focalizza maggiormente l'utilizzo della piattaforma in contesti casalinghi fornendo definizioni di comportamenti per dispositivi IoT e applicazioni utilizzate quotidianamente (Facebook, Telegram, Instagram etc.); Zipato, al contrario, si concentra maggiormente su funzionalità di business, quali possono essere servizi di marketing. Questa distinzione è osservabile soprattutto nei *trigger* che entrambi i servizi mettono a disposizione: se mentre IFTTT rende disponibili *trigger* come ad esempio "foto, temperatura, rilevazioni", Zipato si concentra maggiormente su parole chiave legate al business.

Altro paradigma è quello *If-Do*, utilizzato da molte applicazioni per smartphone, quali *Atooma*⁴, *AutomateIt*⁵, *Tasker*⁶.

2.1.1 EFESTO

Vista la nascita di così tanti paradigmi e vari modi di composizione di regole, uno studio condotto da Desolda et al. [4] ha mostrato come, attraverso la realizzazione di tre diversi software basati su tre paradigmi differenti, quello più favorevole e con cui gli utenti si trovavano più a loro agio nella creazione di regole, più o meno complesse, fosse il paradigma *wizard*, seppur con qualche differenza. L'analisi effettuata è iniziata con uno studio dell'elicitazione al quale hanno partecipato 25 partecipanti al termine del quale sono emersi tre soluzioni al problema: EFESTO-Free, EFESTO-Wizard, EFESTO-Wired (chiamati per semplicemente, rispettivamente, E-Free, E-Wizard, E-Wired).

La particolarità di questi tre prototipi sta nel fatto che aggiungono maggior espressività e complessità alle regole, aggiungendo cioè anche il dove e quando l'evento

⁴<https://atooma.com/>

⁵<https://automateitapp.com/>

⁶<https://tasker.joaoapps.com/>

o l'azione devono essere eseguiti, oltre che ad aggiungere più eventi e/o più azioni connesse tra loro attraverso gli operatori logici *AND* e *OR*.

Andando più nel dettaglio,


- *E-Wizard* è un prototipo che permette agli utenti di creare regole in modo fortemente guidato. L'utente dapprima sceglie il web service che si usufruirà per monitorare l'evento, successivamente avviene la selezione del *trigger*. Dopo aver scelto il *trigger*, in modo del tutto analogo avverrà la scelta dell'azione da eseguire dopo che il *trigger* si è verificato. Al termine della creazione della regola si può cominciare a personalizzarla e renderla più complessa aggiungendo ulteriori eventi o azioni.
- *E-Free* (la soluzione scelta al termine dello studio) è un prototipo che funziona in modo del tutto analogo ad *E-Wizard* ma a differenza del precedente, non impone l'utente a creare prima una regola base per poi personalizzarla, bensì quest'ultimo ha la totale libertà di aggiungere gli eventi o le azioni a suo piacimento, ad esempio aggiungendo prima una catena di azioni che dovranno essere eseguite e poi gli eventi che dovranno verificarsi per eseguire le azioni (Figura 2.1).
- *E-Wired* è un prototipo basato sulla metafora del grafo: i nodi rappresentano i servizi che possono essere utilizzati per la creazione della regola, gli archi rappresentano le relazioni causa-effetto tra i servizi. Il prototipo ha due sezioni: la prima, sul lato sinistro, contiene tutti i servizi che l'utente può utilizzare, sul lato destro c'è il workspace dove l'utente può creare la regola.

Lo studio citato ha quindi portato alla conclusione che gli utenti, con o senza *skills*, preferiscono non essere forzati del tutto nel processo di creazione di una regola, preferendo il paradigma *wizard*, quindi un paradigma che accompagna l'utente nella creazione della regola attraverso *dialog-box* senza però costringerlo ad eseguire prima la scelta del *trigger* e poi quella dell'*action*.


EFESTO-free Home My Rules New Rule cdesold@gmail.com

Creating Rule

Events

 JustAwake
from 07 a.m. to 09 a.m. edit - delete

or

 AlarmRinging
from 07 a.m. to 09 a.m. edit - delete

Add a new Event


+

 And


+

 Or

Actions

 TurnOnCoffeMachine edit - delete

and

 OpenRollUpShutter edit - delete

Add an Action

+

Why (Description of the recipe, as a reminder)

Save

Figura 2.1: Esempio di regola creata con E-Free con una catena di due eventi e 2 actions.

2.1.2 Creazione di regole ECA dipendenti dal contesto di esecuzione

Un altro studio effettuato da Ghiani et al. [6] è riuscito a definire un ambiente di creazione che può modellare regole ECA espressive e che integra un *middleware* capace di ricevere e processare gli eventi generati dai vari sensori, nascondendo così all'utente l'eterogeneità dei dispositivi IoT e permettendogli di creare regole non andando a selezionare direttamente il device IoT e la rispettiva azione da compiere ma limitandosi ai comportamenti che vorrebbe si effettuassero nel caso in cui determinati eventi accadessero. Inoltre, lo studio ha anche cercato di generalizzare questo ambiente in modo da renderlo applicabile in vari domini, come ad esempio per l'assistenza agli anziani, a casa, in un negozio.

Per permettere all'applicazione di cambiare il modo di operare a seconda delle situazioni, lo studio si è basato su 4 dimensioni principali: *User, Technology, Environment, Social Aspects*.

- La dimensione dello *User* include:
 - *Dati personali*: nome, cognome, età, preferenze personali (ad esempio a che ora l'utente preferisce pranzare, andare a dormire etc.);
 - *Stato mentale e fisico*: disabilità, malattie, emozioni;
 - *Posizione e attività*: la posizione può essere specificata con avanti, indietro, a fianco, l'attività è relativa ai comportamenti e agli obiettivi dell'utente, quali sedersi, muoversi, sdraiarsi;
 - *Connessioni sociali*: con ciò si riferisce ai rapporti che l'utente ha con altre persone, al tipo di relazione che c'è tra di esse.
- Con *Technology* s'intende le informazioni relative agli oggetti tecnologici disponibili, quali TV, smartphone, oggetti smart. Ogni oggetto ha attributi che specificano il nome dell'oggetto, il tipo, il proprietario, la posizione, la connettività;

- La dimensione *Environment* include informazioni quali: tipo (all'interno, all'esterno), nome, aspetto spaziale (taglia, forma, posizione), condizioni ambientali (temperatura, luce, rumore);
- La dimensione *Social Relationships* include:
 - *Type of network*: indica se la rete degli utenti è virtuale o fisica;
 - *Type of relationship*: la relazione che condividono i membri del network;
 - *Events*: gli eventi associati al network, con relativi attributi quali il tipo, il luogo, il tempo.

Dopo lo studio iniziale è stata implementata un'interfaccia Web che permette la creazione di regole trigger-action in maniera intuitiva (Figura 2.2). In modo analogo al risultato mostrato nella sezione precedente, anche in questo caso la creazione della regola può avvenire a partire dal *trigger* o dall'*action*. Inoltre, viene aggiunta la possibilità di andare a riutilizzare parti di regole già create in precedenza.

La regola che viene creata ha la seguente struttura:

IF/WHEN *trigger_expression* DO *action_expression*

Sia *trigger_expression* che *action_expression* indicano una lista di condizioni rispettivamente di trigger ed action connesse tra di loro attraverso gli operatori logici *AND* e *OR*. Inoltre viene aggiunta anche la possibilità di utilizzare l'operatore *NOT*.

La creazione di una regola avviene selezionando innanzitutto la dimensione nel quale si vuole eseguire la regola, quindi *User*, *Technology*, *Environment*, *Social Relationships*; successivamente si selezionano gli eventi o condizioni necessarie affinché le azioni vengano eseguite.

Esempi di regole che si possono creare con questa piattaforma sono:

- *When the user falls asleep, switch off the bedroom TV*

The Trigger Action Rule Editor Home Triggers Actions Private Rules Public Rules Simulator Settings Logout

Current Rule: New Rule*

WHEN user enters inside living room, **DO** [choose action(s)]

TRIGGERS

User
Environment
Technology
Social

Personal Data
Physical and Mental
Position and Activity
Social Connection

Position
Behaviour
Goal

Relative Position
AbsolutePosition

RelativePosition
becomes
equal to
inside
Environment
living room

Figura 2.2: Esempio di regola creata in modo dipendente al contesto di esecuzione. In questa immagine si può notare in alto come al momento sia stato scelto solamente l’evento mentre l’azione deve essere ancora settata.

- *When the user leaves home and the oven is on, send a reminder to turn it off*
- *The coloured light in the bedroom must be on between 6:00 and 7:00. It must emit yellow light between 6:00 and 6:45, and then white light till 7:00*

Gli studi appena presentati hanno permesso di capire quale paradigma gli utenti prediligessero per approcciarsi a tool simili, oltre che dare spunto sull’interfaccia da utilizzare nel progetto di tesi svolto.

2.2 Modelli di AI per la privacy disclosure

L’AI è sicuramente diventata popolare e conosciuta nel mondo nell’ultimo decennio grazie ad algoritmi di Intelligenza Artificiale utilizzati nelle piattaforme che ogni utente utilizza quotidianamente: Facebook, Instagram, Amazon, LinkedIn ne sono un esempio. Questa collezione di dati che gli utenti inseriscono (consapevolmente o inconsapevolmente attraverso un “like” o la pubblicazione di un post) fa sì

che ogni utente sia esposto a problemi di privacy e sicurezza e non sempre quest'ultimo ne è a conoscenza quando utilizza simili servizi.

Un esempio lampante di problemi di privacy è dato dall'enorme campagna elettorale del 2016 di Trump, durata anni, rivolta ai conservatori americani per comunicare con loro. La campagna elettorale svolta in modo subliminale fu condotta attraverso la società di consulenza Cambridge Analytica [1] che, attraverso il *data mining* e l'analisi dei dati, riusciva ad effettuare una comunicazione strategica attraverso Facebook. Come Brad Parscale, l'architetto che ha progettato questa campagna pubblicitaria, dice, "*The campaign is all about data collection*" [18].

Per sopperire a perdite di privacy che un utente può subire, come l'esempio appena menzionato, è possibile adoperare alcune soluzioni di AI che aiutano a limitare danni simili. Un articolo di Els [5] ha mostrato come la *differential privacy* e il *federating learning* possano aiutare a risolvere queste problematiche. In particolare, il principio alla base della *differential privacy* consiste nel non inviare informazioni che possano essere utilizzate per identificare le persone a cui sono associate questi dati a query effettuate su un *dataset*. La *differential privacy* è una definizione matematica di privacy che permette di calcolare con quanta probabilità, attraverso i dati condivisi, sia possibile risalire ai proprietari di tali dati. In realtà, il possessore dei dati deve impostare un *privacy budget* che definisce la massima quantità di dati personali che è possibile condividere affinché non si possa riuscire a risalire all'identità della persona a cui sono associati tali dati. Settare un giusto valore di *privacy budget* è alquanto importante per le tecniche di *differential privacy*. La *differential privacy* viene eseguita principalmente andando ad iniettare "rumore" in un *dataset* in modo da rendere le informazioni date in output minimamente modificate mentre la privacy dell'utente migliora.

Il campo della *differential privacy* è studiato e sperimentato soprattutto da ricercatori della Apple e Microsoft. Infatti, la Apple ha sviluppato tecniche molto sofisticate di *differential privacy* implementate attraverso l'*hashing* (ovvero la trasformazione di dati in un insieme di valori apparentemente casuali in un modo

matematicamente deterministico ma difficili da decifrare) e il sottocampionamento (ovvero l'elaborazione di una porzione di dati piuttosto che l'intero *dataset*); il primo sistema operativo Apple ad implementare questo tipo di protezione della privacy degli utenti è stato iOS 10 che randomizzava i dati degli utenti prima di inviarli ai server Apple e limitava la quantità di dati associati agli utenti che potevano essere collezionati.

Anche se la *differential privacy* è un passo avanti verso la protezione dei dati e della privacy degli utenti, essa ha comunque la limitazione di soffermarsi principalmente sul prevenire la reidentificazione dei dati ottenuti da un *dataset*.

Il *federated learning*, invece, è un processo sviluppato da Google che consiste nel permettere ad un modello centralizzato di *machine learning* di ricevere feedback dagli utenti senza però salvare i dati all'interno di un cloud. Il device dell'utente continua a salvare dati per migliorare il livello ma invece di inviare i dati "grezzi" calcola i cambiamenti che potrebbero essere effettuati al modello e invia dei piccoli *update* al modello che, attraverso una media effettuata con tutti gli aggiornamenti ottenuti dai dispositivi, si migliora.

Continuando, l'articolo dell'*Harvard Journal* [5] mostra come, nel 2016, i ricercatori del MIT hanno elaborato una piattaforma chiamata AI2 che permette di identificare cyberattacchi con una precisione dell'85% esaminando *log* e attività sospette. Il modello AI2 è in grado di processare dati in ingresso *realtime* e può generare e ridefinire nuovi modelli, in risposta ai feedback ottenuti, in poche ore. Il campo appena introdotto, ovvero quello di modelli di AI capaci di "ascoltare" e rilevare carenza di privacy, ha grandi margini di miglioramento e sviluppo, basti pensare, ad esempio, ad un modello di AI che potrebbe monitorare un *dataset* che venga utilizzato con l'unico scopo previsto senza andare a sfiorare il *budget privacy*, andando così a proteggere i dati personali degli utenti.

Altro utilizzo che se ne potrebbe fare è creare dei veri e propri algoritmi AI che funzionino come "guardiani", proteggendo così la privacy dell'utente e proteggere quest'ultimo da influenze esterne. Ad esempio, Uber ha ormai creato un algoritmo che crea continuamente bonus ed obiettivi da raggiungere ai propri *driver*, oltre che un algoritmo che efficacemente assegna continuamente corse

agli autisti, che influenza il partner a non fermarsi e continuare a lavorare. In questo caso, un modello di AI potrebbe “aiutare” il *driver* con degli avvisi o una chiusura forzata dell’applicazione permettendo così all’utente di combattere questi algoritmi insistenti.

Ricerche e letture effettuate in questo campo hanno fatto incentrare, quindi, il progetto di tesi sulla definizione di un’interfaccia che mostri la pericolosità a livello fisico, personale, cybersecurity di regole che un qualsiasi utente può creare sulla piattaforma IFTTT, andando ad utilizzare un modello di AI implementato da uno studente iscritto al Corso di Laurea Magistrale in Informatica dell’Università degli Studi di Salerno che funzioni similmente come “guardiano”, andando ad aiutare l’utente nel preservare la propria privacy e sicurezza valutando la potenziale pericolosità di una regola che quest’ultimo potrebbe creare.

2.3 Interfacce per il *crowdsourcing*

Il *crowdsourcing*, per definizione, è lo sviluppo collettivo di un progetto effettuato da un gruppo numeroso di persone, esterne all’azienda o comunità produttrice, che aiutano a completare task computazionalmente difficili da completare, visto il molto tempo e risorse che si impiegherebbero nel completarlo senza l’utilizzo di feedback dati dalle persone.

Uno dei più grandi problemi del *crowdsourcing* è, però, effettivamente la partecipazione di utenti; è importante per l’azienda o comunità produttrice cercare di esplorare e lavorare attraverso vari scenari e più persone partecipano al *crowdsourcing* maggiore è l’apporto che ogni singolo utente dà allo sviluppo del progetto.

Nel mondo dell’IoT, ci sono molti ambiti in cui utilizzare il *crowdsourcing*: un esempio può essere nella Salute Pubblica [16], dove con dispositivi connessi alla rete si possono inviare dati in tempo reale sul diabete o sullo stato di un paziente in generale agli ospedali, in modo da avere un quadro generale della salute del paziente ed inoltre permettere uno studio e analisi più accurata sullo stato di sa-

lute della popolazione.

Un altro esempio in cui si può utilizzare il *crowdsourcing* con l'IoT è nelle *smart cities*, dove ogni cittadino ha la possibilità di andare a riportare problemi che si possono riscontrare all'interno della città.

Esempi pratici di *crowdsourcing* possono essere:

- *Waze*⁷ [8]: questa applicazione fa affidamento sui dati degli utenti per rilevare informazioni sul traffico in tempo reale. Per fare ciò, Waze ha tre modi per collezionare dati:
 - Attivamente attraverso gli utenti: gli utenti possono notificare incidenti, autovelox mobili, lavori in corso (Figura 2.3);
 - Passivamente attraverso gli utenti: gli utenti inviano passivamente informazioni riguardanti il traffico anche soltanto tenendo aperta l'applicazione in background;
 - Volontari: un gruppo di 500.000 volontari modificano costantemente le informazioni disponibili riguardanti lo stato attuale delle strade;
- *BlaBlaCar*⁸: un altro strumento che permette ad un utente di condividere un viaggio che farà e mette a disposizione alcuni posti auto per chi debba dirigersi nella sua stessa destinazione o destinazioni intermedie che saranno coperte durante il viaggio;
- *Placemeter*: una start-up newyorkese che chiede agli utenti di attaccare con ventose i propri smartphone alle finestre ed inquadrare la strada sottostante così da ricevere (garantiscono in modo anonimo) dati sul traffico dei pendolari e tenere sempre aggiornati gli utenti sulle possibili code che potrebbero esserci sulle strade che percorrono.

Grazie agli studi effettuati sull'argomento appena presentato, il progetto di tesi è stato quindi indirizzato nella definizione di un'interfaccia web che, attraverso il

⁷<https://www.waze.com/it/>

⁸<https://www.blablacar.it/>

crowdsourcing, permetta di allenare e rendere sempre più accurati i valori forniti dal modulo di AI che sarà presentato nel capitolo 4.

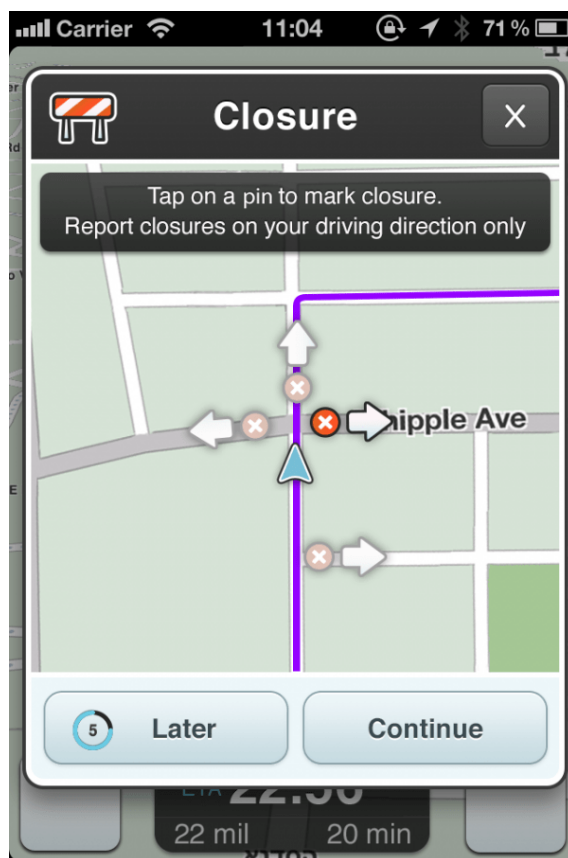


Figura 2.3: Esempio di aggiunta di un'informazione sulla mappa di Waze.

Capitolo 3

IFTTT

Nel seguente capitolo viene presentato il sito web IFTTT, applicazione da cui il progetto di tesi ha prelevato tutti i dati utilizzati successivamente per la creazione di regole attraverso l'interfaccia creata. Nei prossimi paragrafi, oltre alla presentazione, si mostra il funzionamento e la sicurezza della piattaforma.

3.1 Descrizione della piattaforma

IFTTT (acronimo di *If This Then That*) è un servizio web (gratuito o a pagamento) che permette di creare applet (o regole) che gestiscono oggetti IoT e non solo. Grazie a queste regole, un utente può settare delle azioni che verranno svolte una volta che un evento si verificherà. Un esempio pratico di ciò appena detto può essere:

If I post a new picture on Instagram then save the picture on Dropbox. (Figura 3.1)

Utilizzando l'esempio sopracitato, si introducono i termini utilizzati dalla piattaforma IFTTT per identificare le parti di cui è composta una regola:

- *Applet*: l'applet è una regola che l'utente può creare settando un *trigger* ed un'*action* (nell'esempio l'applet è l'intera proposizione);
- *Services*: i servizi sono i "blocchi" di base di IFTTT. Descrivono i dati di uno specifico servizio web oppure possono riguardare azioni controllate da APIs ed SMS. Ogni servizio ha degli specifici *trigger* e *action*. Nell'esempio, i *services* utilizzati sono *Instagram* e *Dropbox*;
- *Trigger*: Il *trigger* è l'evento che, se verificato, eseguirà l'azione decisa al momento della creazione della regola (nell'esempio il *trigger* è *I post a new picture on Instagram*);
- *Action*: L'*action* è l'azione che verrà eseguita nel caso in cui l'evento si verifica (nell'esempio precedente l'*action* è *save the picture on DropBox*);
- *Ingredients*: Gli *ingredients* sono delle variabili che possono essere aggiunte, all'atto della creazione dell'applet, sia al *trigger* che all'*action*; gli *ingredients* possono essere sia obbligatori, come nell'esempio della Figura 3.2, sia opzionali. Riprendendo l'esempio citato pocanzi, un *ingredient* obbligatorio da aggiungere all'*action save the picture on DropBox* è il *sourceURL* dell'immagine da salvare su DropBox mentre uno opzionale è il *file name* dell'immagine che andrà ad essere salvata.

Nella versione gratuita di IFTTT si possono creare al massimo 5 applet con la seguente struttura:

IF *trigger* THEN *action*

mentre nella versione PRO si potranno creare al massimo 20 applet e con la possibilità di creare delle catene di condizioni di *trigger* e *action*; in questo modo, quindi, se e solo se l'intera catena di condizioni di *triggers* sarà verificata allora potranno essere eseguite tutte le *actions* stabilite. La struttura di un'applet che può essere creata con la versione PRO è la seguente:

IF *trigger_conditions* THEN *action_conditions*.

3.2 Storia di IFTTT

«I'd like to humbly announce that the first beta invites for a project I'm incredibly excited about are out the door. The project is called ifttt, shorthand for "if this then that". With this blog I hope to begin fleshing out some of the initial inspirations that led to the inception of ifttt and provide you with a taste of how ifttt can help put the internet to work for you.» [9]

Il 14 dicembre 2010 Linden Tibbets, fondatore di IFTTT, pubblica sul blog `ifttt.com` un articolo dal nome *ifttt: the beginning* che afferma la messa in funzione della versione beta del sito IFTTT, sito che sarebbe diventato da lì a poco uno dei servizi più famosi per creare regole ECA.

Le prime applet sono create, per l'appunto, da Linden Tibbets e Jesse Tane, co-fondatore di IFTTT.

Il 7 settembre 2011, Tibbets annuncia che IFTTT non è più in versione beta ma disponibile a qualsiasi utente voglia usufruire dei suoi servizi.

Nel giugno 2012 la svolta: grazie all'integrazione con i dispositivi Belkin WeMo¹, IFTTT si interfaccia per la prima volta nel mondo fisico, entrando nel mondo dell'IoT e scoprendo le potenzialità che si possono avere da applet impiegando anche dispositivi utilizzabili dagli utenti in ambienti quali casa ed ufficio.

Il 10 luglio 2013 viene rilasciata l'applicazione IFTTT per iPhone mentre il 3 aprile 2014 viene rilasciata anche per iPad e iPod touch. I dispositivi Android dovranno aspettare il 24 aprile 2014 per ricevere la prima versione di IFTTT. Con l'ingresso di IFTTT nel mondo Android, vengono aggiunti servizi specifici supportati dal sistema operativo: Android SMS, Android Battery, Android Device, Android Phone Call, Android Photos.

Alla fine del 2014 IFTTT è stato valutato circa 140 milioni di dollari, rendendo

¹<https://www.belkin.com/>

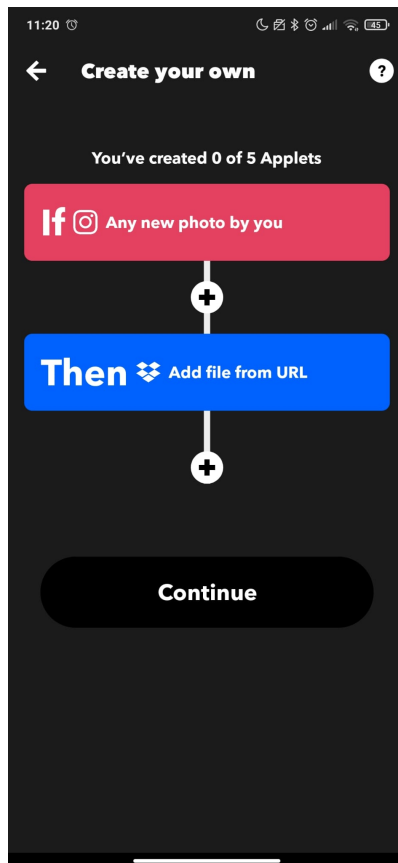


Figura 3.1: Creazione di una regola da dispositivo mobile.

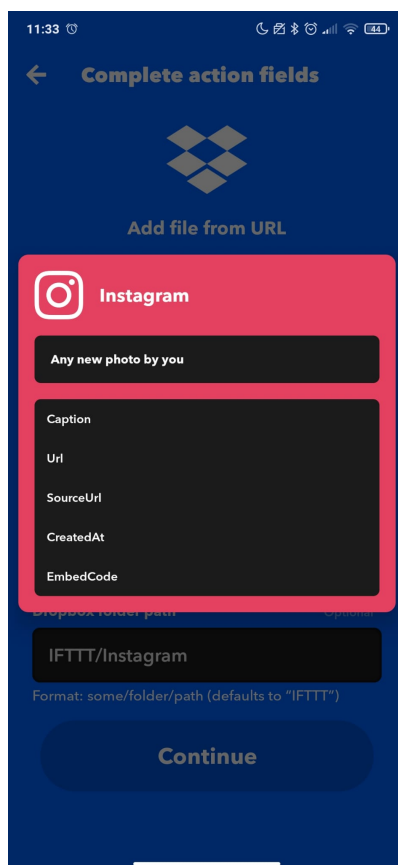


Figura 3.2: Aggiunta di un *ingredient*.

così la piattaforma una delle prime nel mondo nell'ambito IoT.

Nel 2015 IFTTT presenta tre nuovi servizi: *Do Button*, *Do Camera*, *Do Notes*:

- *Do Button* permette di avviare un'azione ogniqualvolta il bottone virtuale che si trova nell'applicazione per smartphone viene tappato;
- *Do Camera* permette di scattare una foto ed automaticamente, dopo averla scattata, caricarla su un servizio (quali Facebook, Dropbox...) prescelto dall'utente;
- *Do Notes*, in modo analogo a *Do Camera*, ogniqualvolta si crea una nota viene caricata su un servizio prescelto dall'utente.

Nel 2016 i tre servizi precedentemente presentati vengono incorporati in IFTTT.

Nel 2020 IFTTT diventa a pagamento, permettendo così agli utenti abbonati di creare fino ad un massimo di 20 applet e consentendo di selezionare più di un *trigger* ed *action* all'interno della stessa applet (nel servizio gratuito si possono creare fino a 5 applet e si possono usare solo un *trigger* e un *action* per regola).

Gli ultimi dati noti della piattaforma IFTTT, ottenuti all'inizio del 2017, mostrano che l'applicazione offre agli utenti più di 400 servizi che hanno portato alla creazione di all'incirca 320.000 applets create e pubblicate per essere utilizzate da altri fruitori. Nel 2017 ci sono stati all'incirca 20 milioni di utilizzi delle applet [11].

3.3 Funzionamento di IFTTT

Un utente di IFTTT, come già detto in precedenza, può creare da 1 a 5 applet se non si ha un abbonamento PRO, fino a 20 altrimenti. Inoltre, ogni utente che si avvicina ad IFTTT può andare a riutilizzare applet già create da altri utenti; infatti, ogni fruitore del servizio web, dopo aver creato un'applet, può decidere di condividerla e renderla disponibile per l'utilizzo anche ad altre persone. Uno studio [17] ha mostrato che, su più di 200.000 applet analizzate, esse sono state

riutilizzate da altri utenti almeno 12 milioni di volte.

Una regola resa pubblica da un utente è strutturata in questo modo:

- *ID*: identificatore numerico a 6 cifre di una ricetta, IFTTT assegna ad ogni ricetta un numero in modo sequenziale;
- *Trigger Channel*: il nome del servizio utilizzato per il *trigger*;
- *Trigger*: il *trigger* utilizzato;
- *Action Channel*: il nome del servizio utilizzato per l'*action*;
- *Action*: l'*action* utilizzata;
- *Author*: il nome dell'autore che ha creato l'applet;
- *Date*: la data in cui è stata resa pubblica l'applet;
- *Adoptions*: il numero di persone che hanno aggiunto l'applet alle proprie;
- *Title*: spazio dedicato all'utente per inserire il titolo della ricetta.
- *Description*: spazio dedicato all'utente per inserire la descrizione della ricetta.

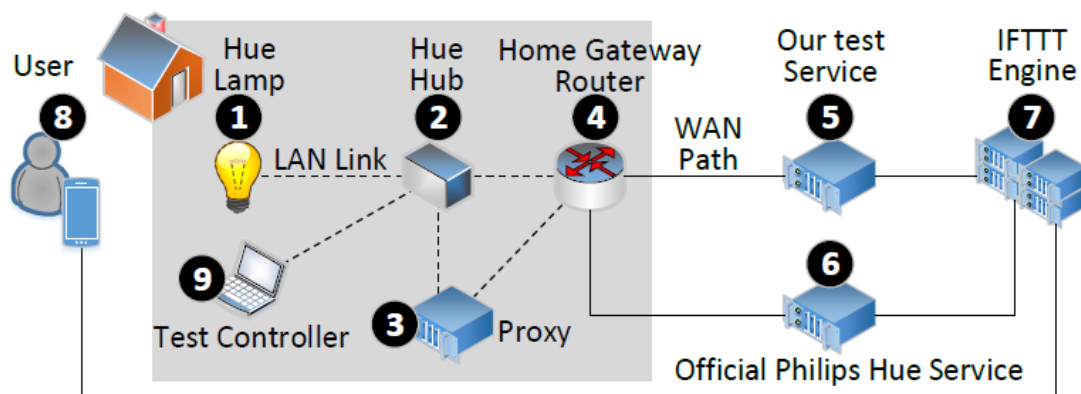


Figura 3.3: Funzionamento dell'ecosistema IFTTT.

Per quanto riguarda il funzionamento di IFTTT, multiple entità giocano un ruolo fondamentale nell'ecosistema; uno studio condotto da Mi et al. [11] chiarifica meglio, attraverso un esempio, come avviene la creazione di un'applet (Figura 3.3). I ricercatori, per capire al meglio le dinamiche e le comunicazioni che avvengono all'atto della creazione o esecuzione di un'applet, hanno pubblicato un proprio servizio (5) di test, quindi fornendo dei *trigger* ed *action*:

- Per pubblicare il server del *service* (5), il server di servizio deve mostrare un URL di base come `https://api.myservice.com` e altre opzioni come le configurazioni di autenticazione. Ogni *trigger* ed *action* ha un URL univoco sotto l'URL di base, (ad esempio `https://api.myservice.com/ifttt/actions/turn_on_light`). IFTTT genererà per il servizio una chiave che sarà inglobata nei successivi messaggi scambiati dal *service* (5) e IFTTT (7) per l'autenticazione.
- Per creare un applet, l'utente (8) si collega direttamente su IFTTT usando l'applicazione web o per smartphone e seleziona i servizi per *trigger* ed *action* (eventualmente anche gli *ingredients*). Molti servizi richiedono l'autenticazione che avviene utilizzando il framework OAuth2². L'utente sarà reindirizzato alla pagina di autenticazione e, al termine dell'autenticazione, sarà generato un token che sarà memorizzato da IFTTT (7) per rendere la futura esecuzione dell'applet completamente automatizzata.
- Nella fase di esecuzione dell'applet, IFTTT fa periodicamente il polling³ dei *services* (in questo esempio fa il polling del servizio (5)). La *polling query* viene inserita all'interno di un messaggio HTTP POST ed inviato all'URL del *trigger*. Il servizio del *trigger* determinerà se la condizione del *trigger* è soddisfatta dal polling attivo. Se il *trigger* si è verificato, il servizio del *trigger* notificherà IFTTT (7) che a sua volta contatterà l'URL dell'*action* e che eseguirà l'azione da intraprendere.

²<https://oauth.net/2/>

³una verifica ciclica di tutte le unità o periferiche associate ad IFTTT (7)

3.4 Sicurezza

Un punto sicuramente da affrontare quando si parla di internet e dati di un utente è la sicurezza.

Uno studio condotto da Surbatovich et al. [15] ha analizzato all'incirca 20.000 applet pubbliche (quindi che possono essere riutilizzate da altri utenti) e scoperto che quasi il 50% di esse ha problemi legati alla violazione di segretezza⁴ o integrità⁵.

Lo studio condotto si è svolto andando ad etichettare ogni *trigger* ed *action* con una o più etichette che descrivessero il livello di pericolosità di ognuno di esse. In particolare, uno studio preliminare prima della fase di etichettatura ha permesso di creare due reticoli, uno per le violazioni di integrità e l'altro per le violazioni di segretezza.

Il reticolo della segretezza, visibile in Figura 3.4, è composto da tre livelli:

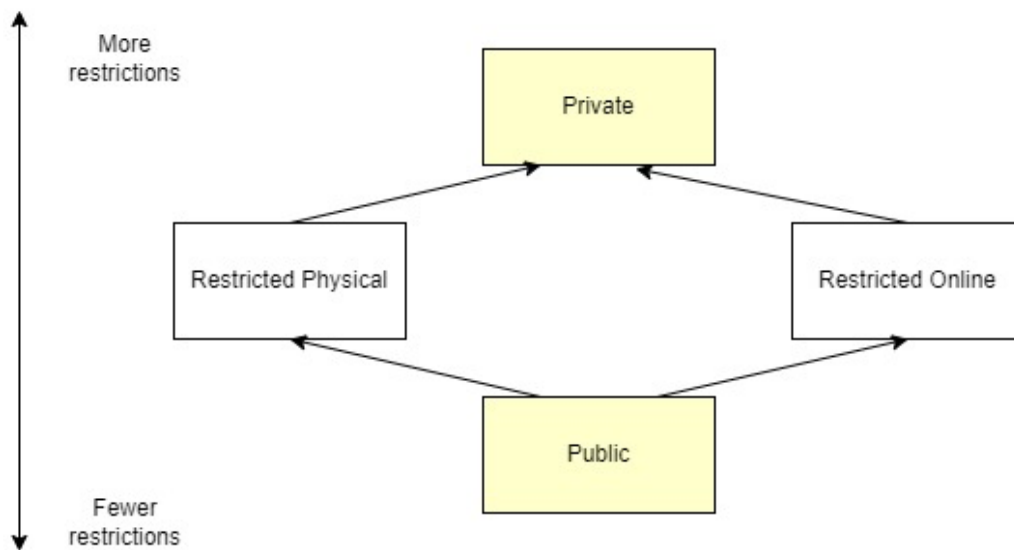


Figura 3.4: Reticolo della segretezza.

⁴violazione che si verifica nel caso in cui informazioni che dovrebbero essere conosciute solo da un gruppo ristretto di persone viene reso in realtà disponibile ad un gruppo più ampio, diventando potenzialmente pericoloso

⁵violazione che si verifica nel caso in cui informazioni provenienti da fonti meno attendibili influenzano informazioni provenienti da fonti più attendibili, danneggiandole potenzialmente.

- Il primo livello, quello più alto, contiene l’etichetta Private che viene utilizzata per etichettare azioni che solamente l’utente che ha creato la ricetta conosce, ad esempio l’attività Fitbit;
- Il secondo livello, quello intermedio, contiene due etichette: la prima, Restricted Physical, viene utilizzata per etichettare eventi che si svolgono in ambienti chiusi o comunque visibili esternamente, come ad esempio azioni che possono essere effettuate in casa o in ufficio quali lo squillo di un telefono; la seconda, Restricted Online, viene invece utilizzata per eventi che si svolgono online con un pubblico limitato, come ad esempio un post su Facebook che può essere visualizzato solo da un gruppo ristretto di utenti selezionati in precedenza;
- Il terzo livello, quello più basso, contiene l’etichetta Public che viene utilizzata per etichettare azioni che potranno essere viste e notate pubblicamente.

Il reticolo dell’integrità, visibile in Figura 3.5, è composto invece da cinque livelli:

- Il primo livello, quello più alto, contiene l’etichetta Untrusted che viene utilizzata per etichettare eventi potenzialmente generabili da chiunque;
- Il secondo livello, subito dopo Untrusted, è quello che contiene Untrusted Group, ovvero eventi che possono essere generati da gruppi sconosciuti di persone (ad esempio un gruppo di persone può decidere di attivare un *listener* che invia una notifica nel caso in cui un post pubblicato su un social media diventa popolare);
- Il terzo livello contiene due etichette, Restricted Physical e Restricted Online che hanno lo stesso significato delle etichette presentate in precedenza per il reticolo della segretezza;
- Il quarto livello contiene l’etichetta Trusted Other che descrive fonti non controllabili dall’utente ma difficili da manipolare, quali possono essere l’ora, la data, eventi atmosferici;

- Il quinto livello, quello più basso, contiene l'etichetta Trusted, utilizzata per etichettare eventi che solo l'utente può generare.

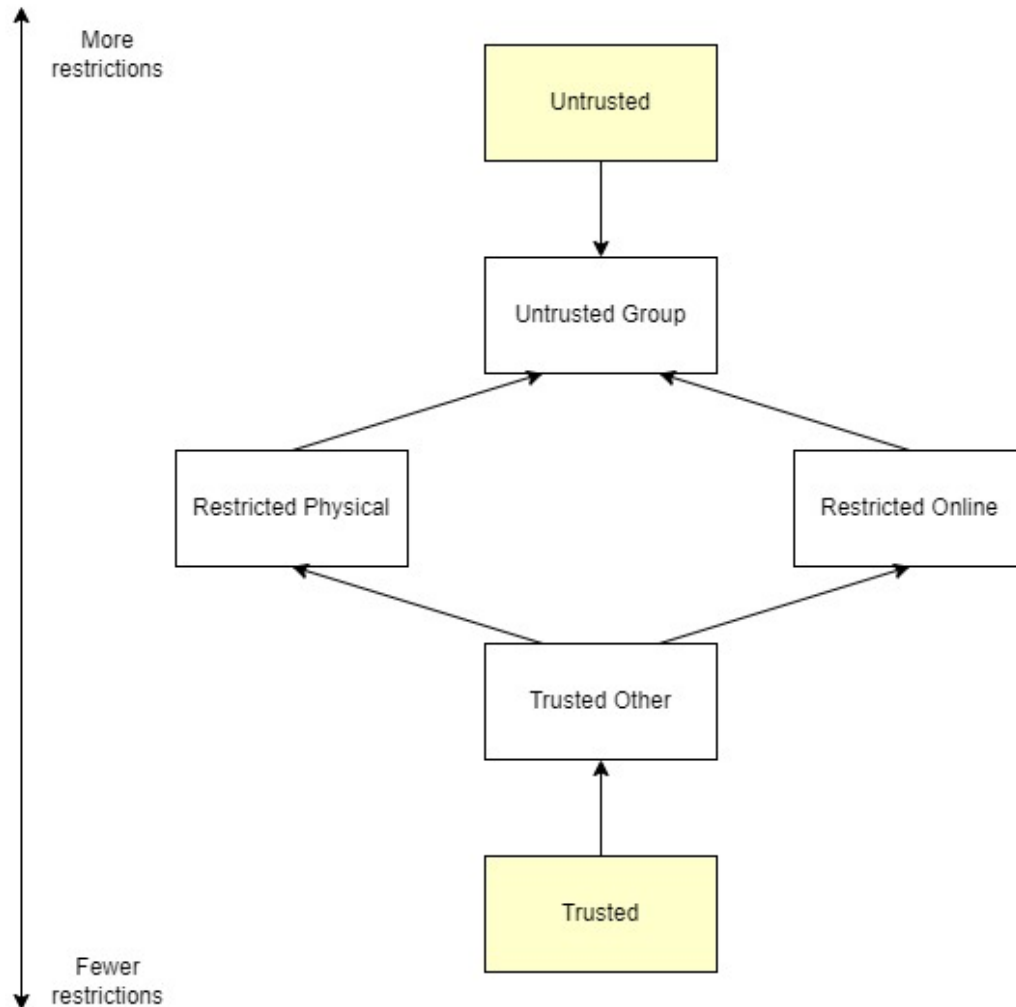


Figura 3.5: Reticolo dell'integrità.

Ogni *trigger* ed *action* considerato nello studio, quindi, è stato etichettato con almeno una di queste etichette. Di tutte le applet che sono state analizzate, quasi la metà sono risultate potenzialmente pericolose. Per definire se un'applet è pericolosa o meno, la regola deve partire da un'etichetta considerata sicura e terminare in un'azione che è stata etichettata con un'etichetta considerata pericolosa. Tuttavia lo studio appena presentato è stato successivamente ripreso da Cobb et al. [3] e ha mostrato che in realtà i risultati ottenuti non consideravano il modo in cui gli utenti interagiscono con il servizio, scoprendo così che in realtà regole che

hanno violazioni di integrità e segretezza sono in numero nettamente inferiore. Tale tesi è stata dimostrata andando ad analizzare 732 applet create da 28 utenti e ponendo domande di un sondaggio a quest'ultimi, scoprendo così che in realtà le regole che possono causare rischi di segretezza o integrità sono in numero di gran lunga inferiore allo studio precedentemente esposto.

Capitolo 4

Classificatore di applet IFTTT basato su BERT

In questo capitolo viene introdotto e mostrato il modello di *Machine Learning*, implementato dal dottor Cimino dell'Università degli Studi di Salerno, utilizzato per classificare le applet IFTTT sulla base di eventuali rischi connessi alla sicurezza ed alla privacy [2].

4.1 Descrizione del modello BERT

BERT (*Bidirectional Encoder Representations from Transformers*) è un algoritmo di Google basato su *Transformers*, una tecnologia che consente di comprendere al meglio il *Natural Language Processing*¹ (NLP).

In particolare, il *Transformer* è un modello basato su una rete neurale artificiale, una rete che simula il comportamento della rete neurale umana, che utilizza un meccanismo di codifica bidirezionale, andando a leggere cioè il contenuto di una frase (nel caso del progetto di tesi la frase è la stringa che sarà poi processata dal

¹Il NLP è una parte della *Computer Science* che si occupa dell'interazione tra computer e linguaggio umano.

classificatore) partendo dall'inizio e dalla fine, contemporaneamente; i precedenti modelli, invece, associavano un significato specifico ad ogni parola letta, andando così a non permettere di definire un contesto ben preciso ad una frase data in input.

I vantaggi che si ottengono nell'utilizzare BERT sono molteplici. Infatti, esso permette uno sviluppo più rapido, in quanto i pesi del modello pre-addestrato codificano già molte informazioni sulla lingua, andando a ridurre il tempo di addestramento e bisogna solo regolare i layer inferiori mentre si utilizza il loro output per la classificazione.

Altro vantaggio è quello di andare ad utilizzare un modello pre-addestrato, che permette così di poter essere utilizzato in scenari in cui sono a disposizione pochi dati (o dati non omogenei) su cui viene data la possibilità di fare *fine-tuning*. Difatti, siccome BERT è composto da un consistente numero di parametri, l'addestramento da zero su un piccolo set di dati condurrebbe ad un *overfitting*, ovvero che in fase di *training* il modello dia risultati ottimali e risultati poco ottimali in fase di testing (ciò è causato dal fatto che il modello, adattandosi ai dati di training non è poi in grado di generalizzare su nuovi dati). Per questo motivo è bene utilizzare un modello BERT pre-addestrato che ha già eseguito un *training* su un enorme set di dati, andando poi ad addestrarlo ulteriormente su un altro *dataset*. Un altro passo in avanti, rispetto alle precedenti tecnologie, è che BERT introduce due strategie di apprendimento per predire le parole che precedono/seguono la parola analizzata: il *Masked Language Modeling (Masked LM)* e il *Next Sentence Prediction (NSP)*:

- con il *Masked LM*, prima di inserire la sequenza di parole in BERT, il 15% delle parole vengono sostituite da un token, chiamato *[MASK]*, ed il modello tenta di prevedere la parola originaria che occupava il posto del token in base al contesto e significato delle parole non oscurate che si trovano all'interno della sequenza. La *loss function* utilizzata da BERT prende in considerazione solo le previsioni sui valori mascherati e ignora le parole non mascherate; così facendo, il modello converge più lentamente rispetto ai

modelli direzionali guadagnandoci, però, una maggior consapevolezza sul contesto delle parole;

- il *NSP*, invece, prevede che BERT riceva in input coppie di frasi e debba predire se la seconda frase è il proseguo della prima all'interno del documento. Il 50% dell'input dato all'algoritmo è effettivamente composto da coppie di frasi in cui la seconda è il proseguo della prima mentre l'altro 50% no.

Quando si addestra BERT, vengono utilizzate entrambe le strategie contemporaneamente con l'obiettivo di ridurre al minimo la *loss function*² combinata delle due strategie.

4.2 Il classificatore

Il classificatore basato su BERT è stato sviluppato andando a congelare l'intera architettura del modello pre-esistente ed andando ad aggiungere nuovi *layer* alla rete neurale ed addestrare il nuovo modello, andando ad aggiornare, durante la fase di addestramento, solamente questi ultimi *layer*.

Infatti, per utilizzare BERT bisogna decidere in che modo approcciarsi ad esso. Esistono 3 approcci: il primo, già descritto, consiste nell'andare a congelare l'intera architettura pre-esistente; il secondo consiste nell'andare ad addestrare il modello già pre-addestrato; il terzo è un ibrido dei due approcci precedenti, cioè si congelano alcuni *layer* e se ne addestrano altri.

I *layer* sono dei livelli contenenti neuroni della rete neurale artificiale. Esistono tre tipi di *layer*:

- *Input layer*: livello progettato per ricevere le informazioni in ingresso provenienti dall'esterno per imparare come riconoscere tali informazioni;

²La funzione obiettivo.

- *Hidden layer*: livello che connette il livello di ingresso con quello in uscita e che aiuta la rete neurale a comprendere relazioni complesse analizzate dai dati;
- *Output layer*: livello finale che mostra il risultato di quanto il programma è riuscito ad analizzare ed apprendere.

Il lavoro svolto per l'implementazione del modello si è suddiviso in due fasi: nella prima fase è stato utilizzato un sottoinsieme di applet, etichettate manualmente, prese dal *dataset* (che sarà presentato nel prossimo paragrafo) per addestrare il modello, nella seconda fase sono state adottate tecniche di apprendimento semi-supervisionato per ottenere un'etichettatura semi-automatica del *dataset*, in modo da classificare regole potenzialmente pericolose.

4.2.1 Descrizione del *dataset* e delle categorie

Durante la prima fase, come già introdotto, è stato effettuato un lavoro preliminare per addestrare il modello. Il dataset utilizzato per lo svolgimento del lavoro di apprendimento è stato creato da un gruppo di ricercatori dell'Indiana University Bloomington [11] andando ad utilizzare un processo di *crawling*³ sul sito web di IFTTT. Gli autori hanno conservato la struttura dell'applet utilizzata da IFTTT e tutti i suoi attributi sono visibili in Figura 4.1. Il dataset preso in questione è stato esposto in formato JSON.

Per il lavoro di tesi sono stati tuttavia presi in considerazione solo i seguenti attributi:

- *title*: titolo della regola inserita dall'utente;

³Un tipo di bot che acquisisce una copia di documenti presenti in una pagina web creando una raccolta di dati presenti in questi ultimi.

```
{
  "isDoRecipe": false,
  "title": "Log Particle Data to Google Drive Spreadsheet",
  "desc": "Log Particle Data to Google Drive Spreadsheet",
  "triggerChannelTitle": "Particle",
  "triggerChannelId": "1978418576",
  "triggerChannelUrl": "https://ifttt.com//particle",
  "triggerTitle": "New event published",
  "triggerDesc": null,
  "triggerId": -1,
  "actionChannelTitle": "Google Drive",
  "actionChannelId": "55",
  "actionChannelUrl": "https://ifttt.com//google_drive",
  "actionTitle": "Add row to spreadsheet",
  "actionDesc": null,
  "actionId": -1,
  "favoritesCount": -1,
  "addCount": "5",
  "creatorName": "payments8",
  "creatorUrl": "https://ifttt.com//maker/payments8",
  "url": "https://ifttt.com/applets/482958p-log-particle-data-to-google-drive-spreadsheet"}

```

Figura 4.1: Esempio di applet presente nel *dataset*.

- desc: descrizione della regola inserita dall'utente;
- triggerTitle: stringa che rappresenta il nome del *trigger*;
- triggerChannelTitle: stringa che rappresenta il nome del servizio scelto dall'utente come *trigger* per la regola;
- actionTitle: stringa che rappresenta il nome dell'*action*;
- actionChannelTitle: stringa che rappresenta il nome del servizio scelto dall'utente come *action* per la regola.

Il *dataset* introdotto ha poi successivamente subito un processo di *data cleaning*⁴, andando ad eliminare record poco rappresentativi o con titolo e descrizione non in lingua inglese. Al termine di questo processo, è stato possibile avere un *dataset* di 116.825 regole, uniforme dal punto di vista linguistico e privo di elementi poco rappresentativi.

La fase successiva è stata quella di etichettatura delle regole, considerando il lavoro in [15]. In particolare, in questo studio le applet sono state catalogate in quattro classi:

⁴Processo capace di garantire in modo alquanto affidabile la correttezza di una collezione di dati.

- **Innocua:** categoria che comprende regole apparentemente innocue, un esempio è la regola “Se inserisco un tag specifico quando pubblico un post su Facebook, salva l’immagine in una directory specifica su Amazon Cloud” (classe 0);
- **Personale:** categoria che comprende regole che causano la diffusione di dati sensibili, un esempio è la regola “Se scatto una nuova foto, pubblicala su Facebook” che potrebbe condividere fotografie potenzialmente private rendendole pubbliche (classe 1);
- **Fisico:** categoria che comprende regole che arrecano danno alla salute fisica, alla proprietà o beni, un esempio è la regola “Se l’ultimo membro della famiglia esce di casa, spegni le luci” che potrebbe aiutare un ladro nel compito di derubare la casa “avvertendolo” della presenza di nessun componente della famiglia all’interno dell’abitazione (classe 2);
- **Cybersecurity:** categoria che comprende regole che causano l’interruzione di un servizio online o la distribuzione di un malware⁵, un esempio è la regola “Se nella posta in arrivo è presente un’e-mail con un allegato, carica il file su OneDrive” che potrebbe salvare un allegato dannoso su un account sincronizzato su vari dispositivi, aumentando la probabilità di apertura del file (classe 3).

4.2.2 Etichettatura

Poiché il *dataset* utilizzato era sprovvisto di etichette per poter addestrare il modello, durante la prima fase sono state aggiunte manualmente etichette ad un sottoinsieme di 1000 regole selezionate casualmente dal *dataset*. Al termine del lavoro, svolto da tre studenti iscritti al Corso di Laurea Magistrale in Informatica presso l’Università degli Studi di Salerno, ci si è accorti che però le regole appartenenti alle classi 2 e 3 erano decisamente poche, per questo si è ricorsi all’utilizzo

⁵Un programma che mette a rischio un sistema informatico.

di SentenceBERT per aumentare il numero di regole appartenenti a queste classi. SentenceBERT è un insieme di tecniche utilizzate nel NLP dove le frasi sono rappresentate sotto forma di vettori. Il modello è stato innanzitutto applicato sulle applet già etichettate appartenenti alla classe 2 (ma successivamente il lavoro è stato svolto anche sulle applet della classe 3) così da ricavarne i *sentence embedding*, ovvero i vettori rappresentanti le applet processate. Dopo questa fase, al modello sono state date in input altre regole ed, attraverso un calcolo di *cosine similarity*⁶ si è riusciti ad inserire all'interno di un file Excel, in ordine decrescente in base al valore ottenuto dalla *cosine similarity*, tutte le applet analizzate. Dopo questa fase il lavoro di etichettatura manuale è continuato, seppur su un sottoinsieme molto ristretto di applet e comunque filtrate e quindi potenzialmente appartenenti alla classe 2, permettendo così di aumentare i record delle classi 2 e 3.

4.2.3 Classificazione sul *dataset* etichettato manualmente

Al termine della fase di etichettatura, BERT è stato addestrato sulle applet etichettate manualmente. In particolare, la fase di addestramento è stata ripetuta considerando tre diverse combinazioni di *feature*:

- title e desc;
- triggerTitle, actionTitle, actionChannelTitle e triggerChannelTitle;
- tutte le *feature*.

Per quanto riguarda la prima combinazione, innanzitutto si è dovuto utilizzare il *tokenizer* fornito da BERT, ovvero uno strumento che permette di dividere la frase data in input in token appartenenti al dizionario del modello (nel caso in cui un token non appartenga al modello, BERT gestisce in modo automatico questa

⁶Una tecnica euristica per calcolare la similitudine tra due vettori.

manca). Per essere utilizzato con BERT, il *dataset* necessita dapprima, però, una trasformazione in un formato comprensibile al modello. Per questo motivo, vengono eseguite automaticamente queste operazioni:

- Vengono aggiunti token speciali *[CLS]* e *[SEP]* all’inizio e alla fine di ogni frase;
- Tutte le frasi vengono uniformate ad un’unica lunghezza costante (50, in questo caso);
- Si differenziano i token reali da quelli di *padding* attraverso l’utilizzo della “maschera di attenzione”.

Il token *[PAD]* (ovvero il token utilizzato per il padding) è un token che viene aggiunto a frasi che non raggiungono la lunghezza massima prefissata (in questo caso 50). Ad esempio, se una frase sarà formata da 34 token, saranno aggiunti altri 15 token *[PAD]* per raggiungere la lunghezza di 50 token (l’ultimo token è quello che indica la fine della frase, *[SEP]*).

La “maschera di attenzione”, invece, è una stringa composta da 0 e 1 che permette al modello il riconoscimento dei token appartenenti alla frase e quelli di *[PAD]*. Riprendendo l’esempio di prima, in quel caso la “maschera di attenzione” sarà formata da trentaquattro 1 seguiti da quindici 0 ed un 1, quello finale, per il token *[SEP]*. Al termine del cambio di formato del *dataset*, quest’ultimo è stato partizionato in *training set* e *test set* ed è stato configurato il modello BERT andando ad utilizzare l’approccio già citato in precedenza, ovvero congelando i layer preesistenti ed andando ad addestrare solamente il layer aggiunto. È stato quindi aggiunto un ulteriore layer all’architettura, chiamato *Linear Layer*, che, dopo aver settato tutti i parametri per il *training* e di convalida, è stato addestrato sul task preso in considerazione. Queste fasi sopracitate sono state ripetute anche per le due successive combinazioni, andando semplicemente a cambiare la lunghezza massima delle frasi, settando a 16 la lunghezza delle stringhe formate da *triggerTitle*, *actionTitle*, *actionChannelTitle* e *triggerChannelTitle* e a 70 per le stringhe che contengono tutte le *feature*.

4.2.4 Tecniche di apprendimento semi-supervisionato

Dopo aver lavorato su un sottoinsieme del *dataset* etichettato manualmente, sono state applicate tecniche di apprendimento semi-supervisionato, già introdotto in precedenza, al fine di andare ad etichettare automaticamente regole senza etichetta ampliando il numero di applet etichettate a disposizione nella fase di *training*. Comincia, quindi, la seconda fase.

Per l'apprendimento semi-supervisionato sono state adottate tre strategie. Ognuna di esse è stata eseguita sulle tre diverse combinazioni di *feature* presentate in precedenza.

Al termine dell'esecuzione delle tre tecniche di apprendimento sono stati generati nove nuovi *dataset*, poiché ogni strategia è stata applicata per tre volte su ogni combinazione.

Per addestrare nuovamente il modello si è deciso di andare ad utilizzare un "voto a maggioranza", cioè, fissando una particolare combinazione di *feature*, sono state prese in considerazione solo le applet per la quale almeno due algoritmi su tre hanno fornito la stessa etichetta. In questo modo alcune regole non etichettate non sono state considerate ma tale approccio ha portato ad una maggior affidabilità nella classificazione delle regole valutate. Al termine di questa operazione sono stati ottenuti tre *dataset* globali per poter addestrare nuovamente BERT. Ogni *dataset* ottenuto è stato successivamente partizionato in *training set* e *test set*, andando a riaddestrare il modello sulle stesse *feature* illustrate in precedenza.

Dai risultati ottenuti al termine della classificazione sul *test set* è stato notato che, grazie all'incremento delle istanze all'interno del *training set*, la qualità delle predizioni del modello sono aumentate. In particolare, i risultati migliori sono stati ottenuti nella previsione di applet appartenenti alla classe 0 mentre valori di precisione relativamente più bassi sono stati ottenuti per le classi 1, 2 e 3. Tuttavia, siccome l'obiettivo è quello di segnalare dei *warning* agli utenti, è molto più rilevante il fatto che il modello riesca a classificare in modo corretto se un'applet non appartenga alla classe 0 piuttosto che identificare precisamente il tipo di danno che potrebbe essere inflitto all'utente.

Capitolo 5

Lavoro svolto

In questo capitolo sarà presentata e mostrata l'interfaccia web implementata che potrà essere resa disponibile agli utenti. La piattaforma permette al fruitore di creare una regola o creare un file di regole, mostrandogli in output una classificazione dell'applet (o delle applet, nel caso del file) in base alle classi "innocua", "attacco personale", "attacco fisico", "attacco cybersecurity" presentate nel Capitolo 4.

5.1 Struttura della piattaforma web

La piattaforma web è formata da due sezioni:

- **Crea una regola:** in questa sezione l'utente può creare una regola per verificarne la sicurezza. Dopo averla creata e fatto il *submit*, all'utente viene mostrato un "blocco" contenente le informazioni circa la percentuale di appartenenza ad ogni classe presentata nel capitolo precedente (innocua, attacco personale, attacco fisico, attacco cybersecurity);
- **Carica un file di regole:** in questa sezione l'utente può fare l'upload di un file JSON contenente una lista di applet e man mano gli verranno mostrati

“blocchi”, ognuno contenente una applet, con le percentuali di appartenenza della regola per ogni classe già presentata.

In particolare, l’interfaccia della sezione “Crea una regola”, appena la piattaforma web è caricata, è visibile in Figura 5.1, l’interfaccia per l’upload di un file JSON contenente regole IFTTT in Figura 5.2.

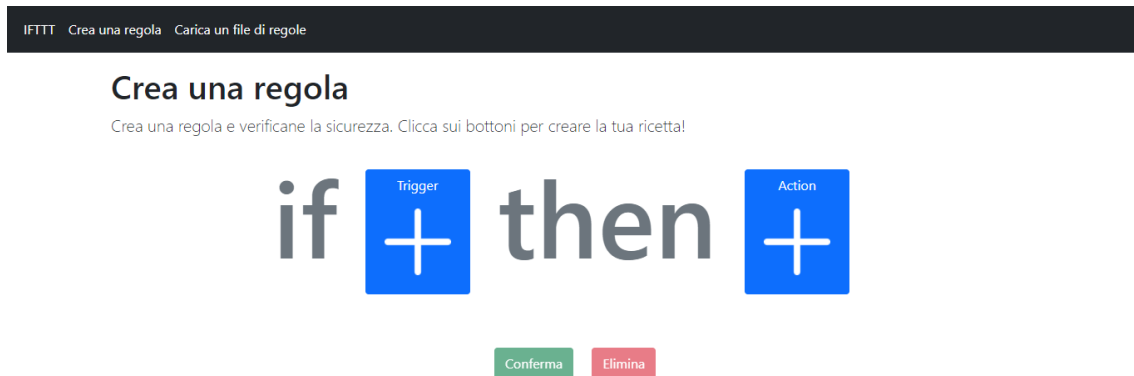


Figura 5.1: Interfaccia con cui interagire per creare una regola.

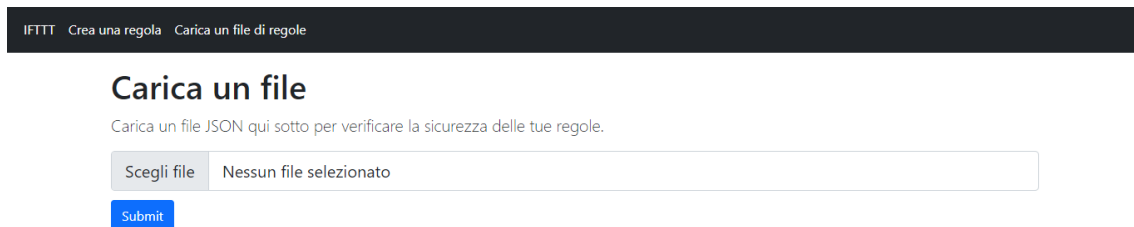


Figura 5.2: Interfaccia con cui interagire per caricare un file di regole JSON.

La piattaforma web funziona nel seguente modo: dopo aver aperto l’interfaccia nel proprio browser, l’utente può o decidere di creare una regola sfruttando tutti i *services*, *trigger* ed *action* presenti in IFTTT, e, una volta aggiunti il titolo e descrizione alla nuova regola creata ed aver premuto il bottone “Submit”, l’utente vedrà sotto la pagina mostrata all’inizio dell’interazione un “blocco” contenente le informazioni già citate in precedenza (un esempio del “blocco” mostrato è vi-

sibile in Figura 5.3) oppure caricare un file JSON e, una volta premuto il tasto “Submit”, tutte le sue regole saranno classificate e visualizzate come è mostrato in Figura 5.4.

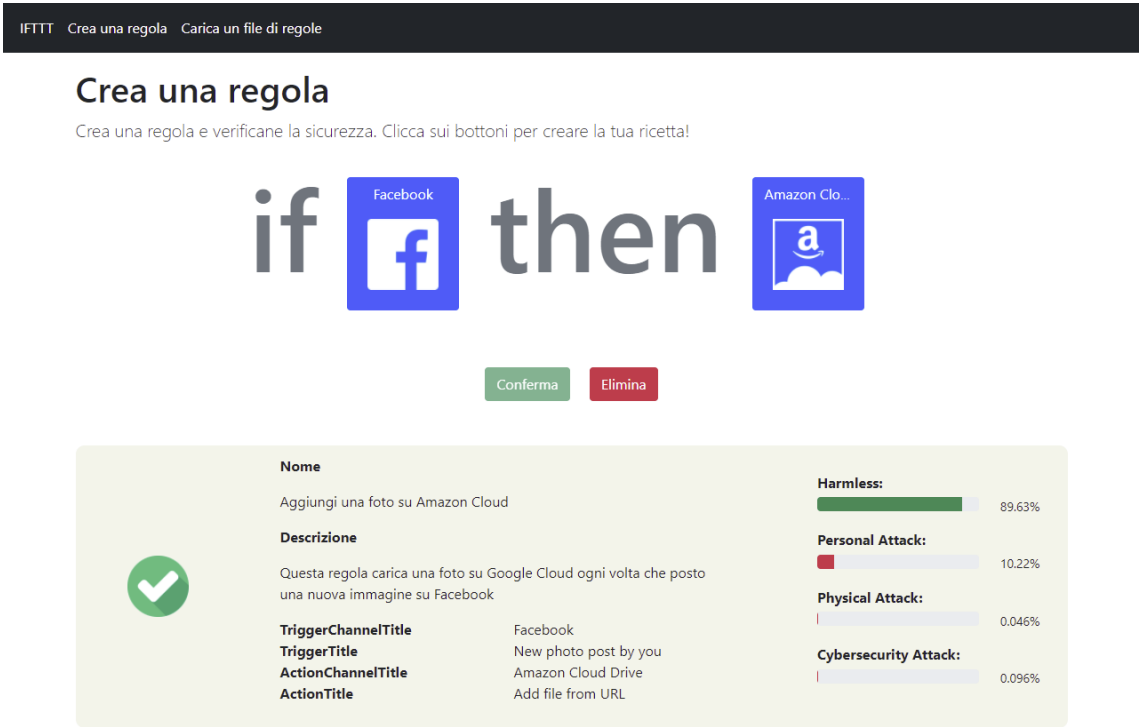


Figura 5.3: Esempio di verifica di una regola creata nell'applicazione web.

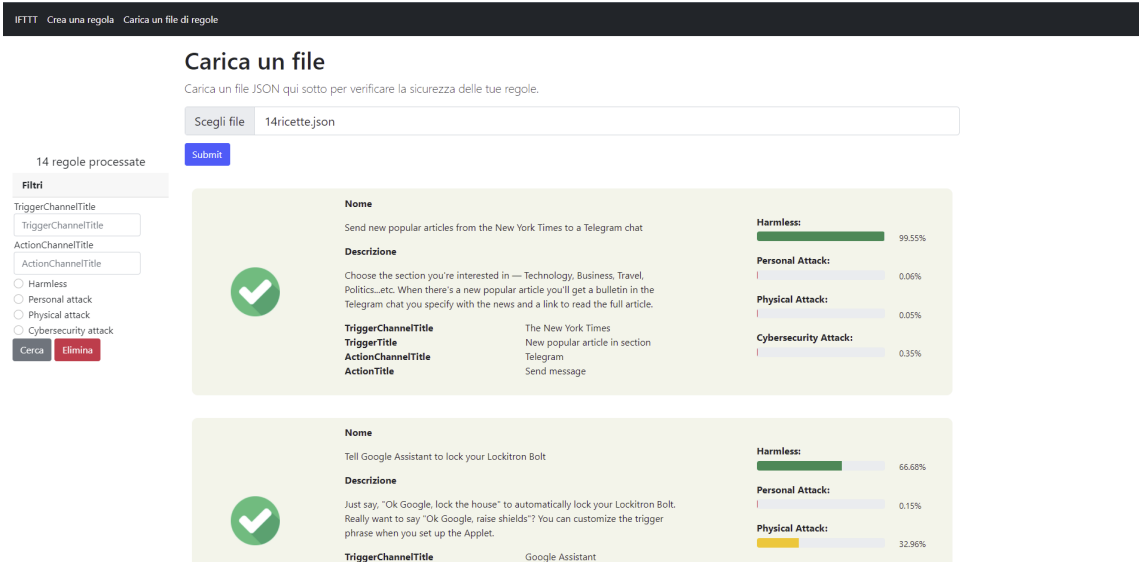


Figura 5.4: Esempio di verifica di un file caricato nell'applicazione web.

L'applicazione web funziona in questo modo:

- Nel caso in cui l'utente crei una regola, per classificare l'applet il client invia al server (che si trova su Google Colab) una richiesta HTTP POST e, una volta classificata, i dati verranno restituiti all'interno dell'URL della pagina;
- Nel caso in cui, invece, l'utente carichi un file di regole per classificarle, il file caricato verrà inviato al server attraverso una richiesta HTTP POST. Tuttavia, in questo caso, il server andrà ad inserire le regole classificate, sotto forma di record, all'interno di un *database realtime* su Firebase. L'interfaccia, così, andrà a leggere i record inseriti nel database man mano che il server li andrà a scrivere in quest'ultimo.

Nei prossimi paragrafi saranno introdotte tutte le tecnologie utilizzate per andare a sviluppare l'interfaccia web appena descritta.

5.2 Firebase Realtime Database

Firebase¹ è una piattaforma *serveless*, acquistata da Google nel 2014, utilizzata per lo sviluppo di applicazioni web e mobile. Essa, attraverso l'utilizzo dell'infrastruttura di Google ed il suo cloud, fornisce una suite di strumenti per fornire funzionalità come analisi, database (Firestore Database e Realtime Database, usando strutture NoSQL), messaggistica e segnalazione di arresti anormali per la gestione di applicazioni web, Android e iOS.

Nel caso dell'applicazione web implementata si è utilizzato il servizio di Realtime Database fornito da Firebase con lo scopo di rendere l'applicazione *realtime*, ovvero ogni volta che un'applet viene classificata essa è immediatamente disponibile all'interno del database e disponibile all'utilizzo da parte dell'interfaccia web. Infatti, grazie all'aiuto di React (che sarà introdotto successivamente), solamente la porzione di interfaccia colpita dal cambiamento sarà aggiornata.

¹<https://firebase.google.com/>

Il Realtime Database di Firebase è un database NoSQL che utilizza il formato JSON per il salvataggio dei record. Concretamente, il database non è altro che un file JSON con all'interno degli oggetti JSON. Per il progetto di tesi svolto, i record all'interno del database sono stati inseriti considerando tutti gli stessi attributi visibili in Figura 5.5.



Figura 5.5: Esempio di record salvato in Firebase.

In particolare:

- actionName: è il nome del *service* che offre l'*action* selezionato dall'utente;
- actionTitle: è l'azione eseguita quando il *trigger* si verifica;
- cl0: è la percentuale di appartenenza dell'applet alla classe 0 (innocua);

- cl1: è la percentuale di appartenenza dell'applet alla classe 1 (attacco personale);
- cl2: è la percentuale di appartenenza dell'applet alla classe 2 (attacco fisico);
- cl3: è la percentuale di appartenenza dell'applet alla classe 3 (attacco cybersecurity);
- desc: è la descrizione data dall'utente all'applet;
- title: è il titolo dato dall'utente all'applet;
- triggerName: è il nome del *service* che offre il *trigger* selezionato dall'utente;
- triggerTitle: è l'evento che fa scattare l'esecuzione dell'applet.

I dati inseriti all'interno del database hanno tutti la stessa *root*, ovvero *tiroci-default-rtdb*, e la chiave primaria di ogni singolo record inserito è generato automaticamente da Firebase. Oltre al salvataggio dei dati, il database offerto da Firebase è stato utilizzato anche per eseguire query in modo da filtrare i record in base a vari filtri che l'utente può settare nell'interfaccia web; in particolare le query create sono sugli attributi *actionTitle*, *triggerTitle*, *cl0*, *cl1*, *cl2*, *cl3*.

5.3 Google Colab

Google Colab è una piattaforma online, di proprietà di Google, che fornisce un servizio di hosting.

Il server utilizzato per processare le *request* HTTP POST inviate dal client è stato scritto in Python utilizzando, per l'appunto, il servizio offerto da Google. Il server risponde a due tipi di richieste: *predict* e *predict-all*. Nel metodo *predict* il server classifica la regola inviata dal client e restituisce tutti i dati attraverso l'URL, come è visibile nella Figura 5.7.

L'URL conterrà i seguenti parametri:

http://localhost:
 3000/?recipe=1&cl0=6.326691806316376&cl1=93.45681667327881&cl2=
 0.05407226271927357&cl3=0.16241678968071938&ist=Facebook. You
 are tagged in a photo. Flickr. Upload public photo
 from URL. Ogni volta che sono taggato in una foto su
 Facebook, caricala su Flickr. Questa applet carica le
 foto in cui sono taggato su Facebook su Flickr

Figura 5.6: Esempio di URL dopo la classificazione dell'applet

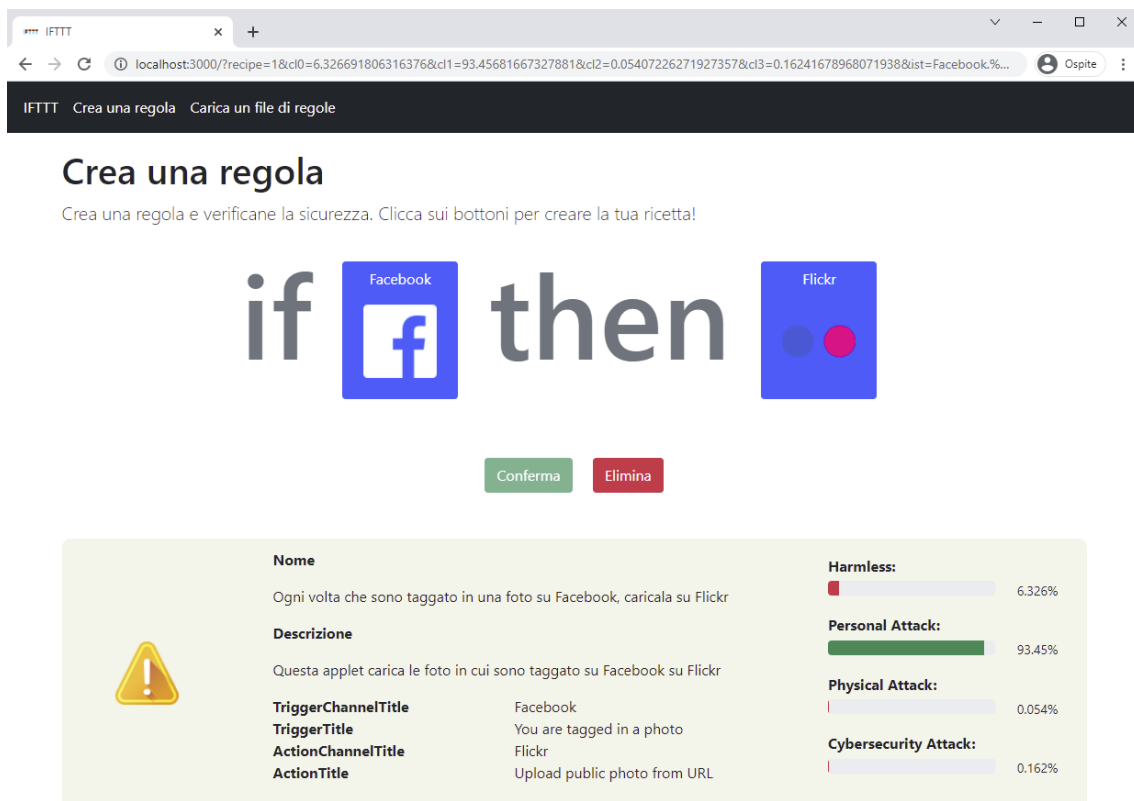


Figura 5.7: Invio di informazioni attraverso l'URL.

- recipe: è un numero che identifica la classe in cui è stata classificata l'applet (nella Figura 5.6, ad esempio, la classe è 1);
- cl0: è la percentuale con la quale l'applet può essere considerata innocua;
- cl1: è la percentuale con la quale l'applet può essere potenzialmente consi-

derata causa di attacchi personali;

- cl2: è la percentuale con la quale l'applet può essere potenzialmente considerata causa di attacchi fisici;
- cl3: è la percentuale con la quale l'applet può essere potenzialmente considerata causa di attacchi cybersecurity;
- ist: è la stringa passata in input al server.

Per quanto riguarda il metodo *predict-all*, invece, il client invia al server l'intero file JSON caricato dall'utente e, una volta ricevuto, il server fa il *parse*² del file, andando a creare per ogni applet la stringa che viene processata dal classificatore tramite la chiamata al metodo *predict*. Successivamente ogni applet classificata viene caricata su Firebase.

5.4 React

React³ [14] è una libreria JavaScript open source⁴ per la creazione di interfacce utente, sviluppato da Meta⁵ e da una comunità di singoli sviluppatori e aziende. React è nato principalmente per lo sviluppo di applicazioni web *single-page* (applicazioni a pagina singola, ad esempio le stesse Facebook e Instagram) ma anche applicazioni per smartphone con la successiva distribuzione di React native, altra libreria implementata da Meta, che tramuta i componenti React in componenti nativi iOS e Android. React si occupa solamente della presentazione, e quindi il render, della pagina; per questo motivo è stato utilizzato il realtime database di Firebase per la gestione dei dati.

È stato preferito React ad altre tecnologie per la creazione di UI per la facilità ed

²Processo che consiste nell'analizzare un flusso di dati in input.

³<https://it.reactjs.org/>

⁴Un software si definisce open source se i detentori dei diritti permettono la modifica, lo studio e la redistribuzione del codice sorgente di quest'ultimo.

⁵Impresa statunitense che controlla servizi quali Facebook, Instagram, WhatsApp.

intuitività nell'implementazione dell'interfaccia. Questo framework aggiorna la pagina ogni volta che il suo stato cambia, andando però a modificare solamente la porzione di interfaccia che viene intaccata dalla modifica della variabile.

La sintassi utilizzata da React è JSX, ovvero JavaScript XML, anche se si può programmare andando ad utilizzare semplice codice JavaScript.

Per eseguire il codice React è stato necessario installare Node.js⁶ [12], un runtime system⁷ utilizzato per l'esecuzione di codice JavaScript.

Per il design dell'interfaccia è stato utilizzato Bootstrap⁸, un framework implementato da alcuni sviluppatori Twitter con lo scopo di unificare tutti i vari componenti utilizzati all'interno di interfacce web. Si è deciso di utilizzare Bootstrap per la facilità d'uso del framework e perché supporta il responsive web design⁹.

5.4.1 Descrizione della sezione “Crea una regola”

La pagina principale dell'interfaccia web implementata mostra la sezione “Crea una regola”, già mostrata in Figura 5.1. In particolare, in quella sezione ci sono due grandi pulsanti blu per la scelta del *trigger* e l'*action*. Se si clicca su uno dei due pulsanti è possibile, così, scegliere quale *service* selezionare (Figura 5.8) per poi visionare i *trigger* o *action* che quest'ultimo offre (Figura 5.9). L'ordine della scelta del *trigger* o *action* non è importante, quindi l'utente può scegliere di selezionare prima l'*action* e poi il *trigger*, andando così ad utilizzare un *wizard paradigm* che comunque non forza l'utente in un unico *flow* di interazione.

Una volta scelti entrambi (Figura 5.10), sarà mostrata all'utente un *modal*¹⁰ do-

⁶<https://nodejs.org/>

⁷Software adibito all'esecuzione di un programma.

⁸<https://getbootstrap.com/>

⁹tecnica utilizzata per permettere alla piattaforma web di adattarsi graficamente su schermi di varie dimensioni

¹⁰Una finestra che può mostrare un'informazione, un messaggio di errore etc.

Crea una regola

Crea una regola e verificane la sicurezza. Clicca sui bottoni per creare la tua ricetta!



Conferma Elimina

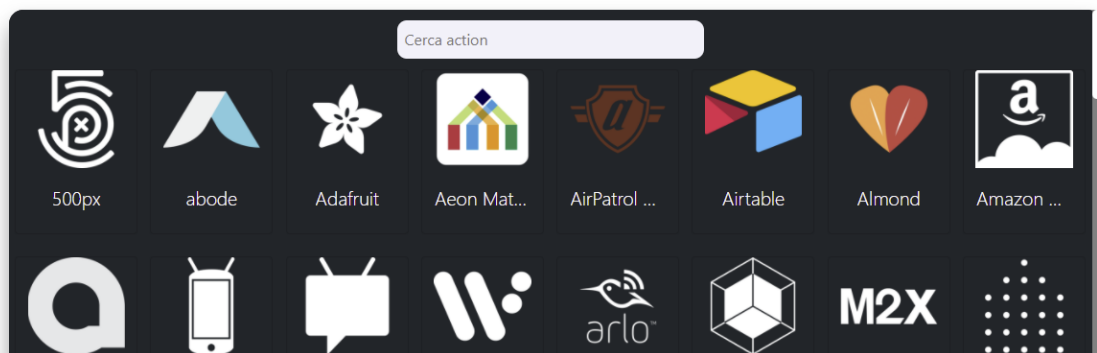


Figura 5.8: Sezione che mostra tutti i servizi messi a disposizione da IFTTT per la scelta dell'*action*.

ve sono mostrati i *service*, il *trigger* ed *action* scelti e che chiede all'utente di inserire un titolo ed una descrizione alla regola appena creata (Figura 5.11). Da notare che in Figura 5.10 il pulsante "Elimina", in rosso, è diventato "clickable", ovvero è possibile cliccarlo. Questo perché, nel caso in cui l'utente abbia sbagliato a scegliere un *trigger* o *action* può decidere di cancellare la regola e ricomporla daccapo. In modo analogo, anche il pulsante "Conferma", in verde, diventa "clickable" nel caso in cui l'utente decida di rivedere nuovamente la regola appena creata prima di compilare il form e fare il *sumbit* della regola per classificarla.

Una volta fatto il "sumbit", quindi, viene inviato al server, attraverso una *request* HTTP POST, una stringa formata nel seguente modo:

triggerService. triggerName. actionService. actionName. title. desc,

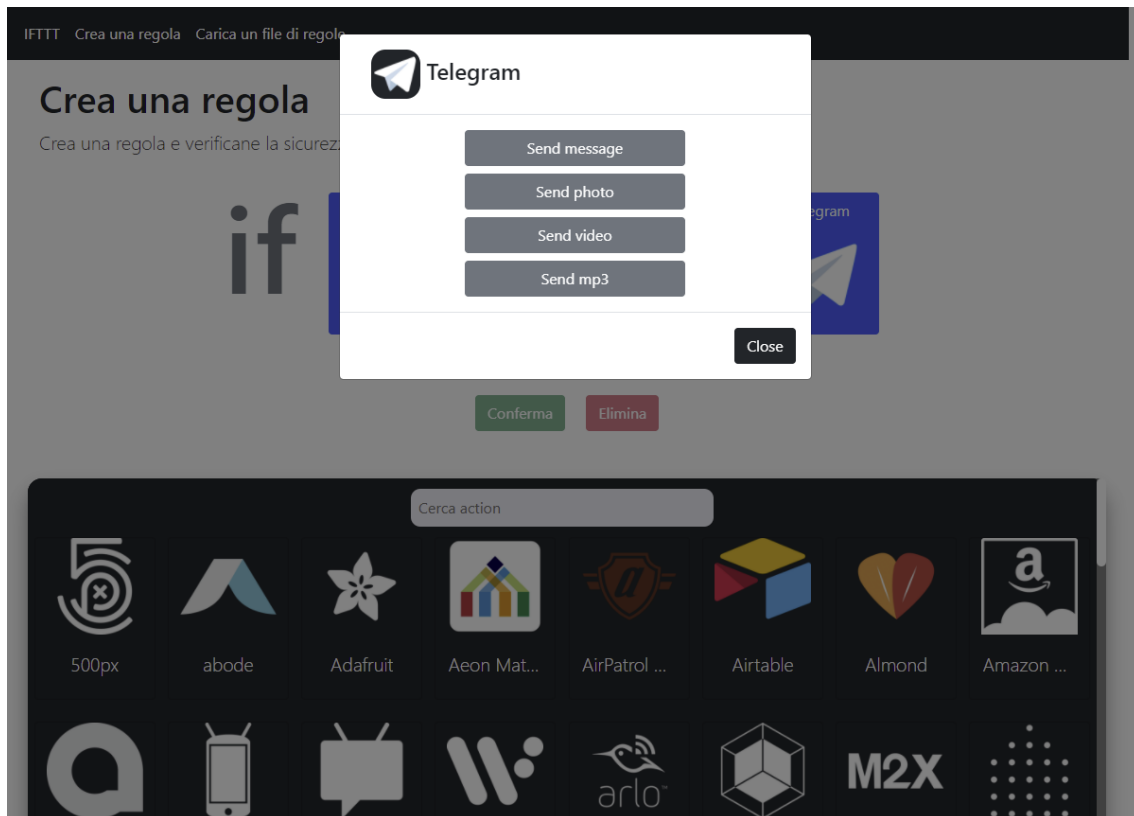


Figura 5.9: *Modal* utilizzato per la scelta dell'*action*.

dove:

- *triggerService* è il servizio utilizzato per la scelta del *trigger*;
- *triggerName* è il nome del *trigger* scelto;
- *actionService* è il servizio utilizzato per la scelta dell'*action*;
- *actionName* è il nome dell'*action* scelto;
- *title* è il titolo dato alla regola dall'utente;
- *desc* è la descrizione data alla regola dall'utente.

Una volta che il server riceve la *request*, essa viene processata ed al termine della classificazione la pagina viene reindirizzata ad un URL del tutto simile a quello mostrato nel Paragrafo 5.3, che permette così alla pagina di visualizzare un

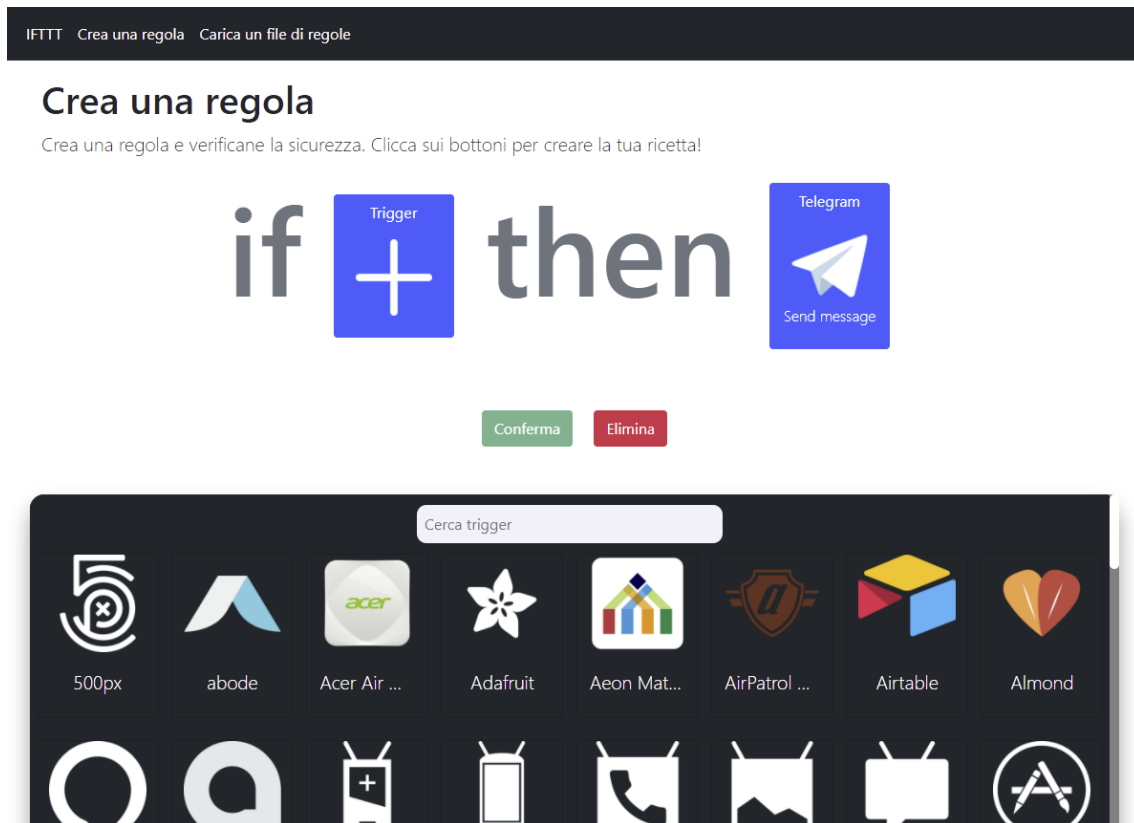


Figura 5.10: Scelta del *trigger*.

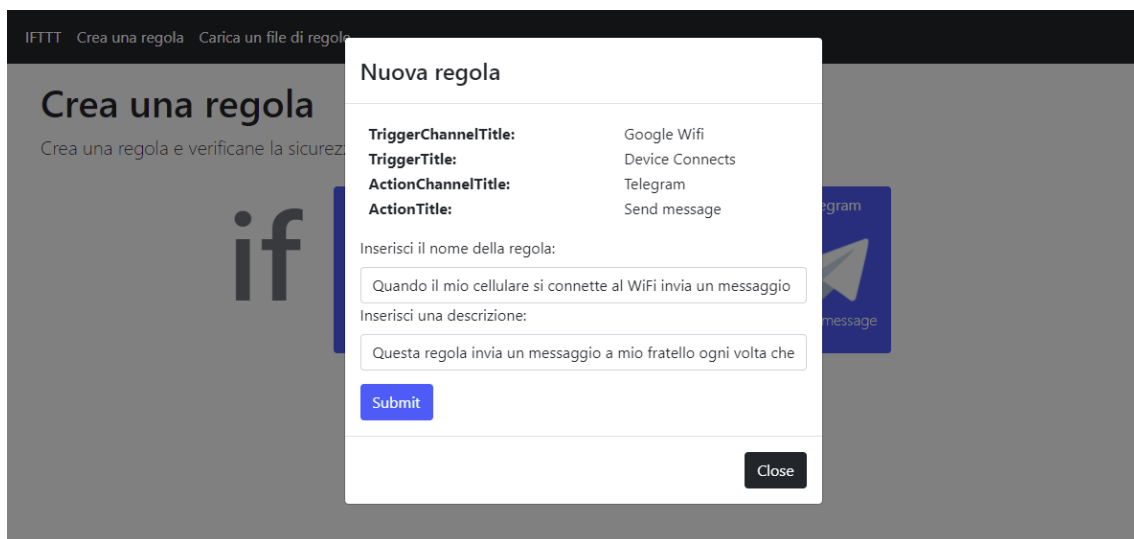


Figura 5.11: *Modal* che viene mostrato al termine della creazione della regola.



“blocco”, visibile in Figura 5.7. In particolare, il “blocco” visibile dopo l’esecuzione dell’esempio finora mostrato è presente in Figura 5.11. Il “blocco” visualizzato è suddiviso in tre sezioni:

- La prima sezione può mostrare due immagini: il segno di spunta (visibile in Figura 5.12) oppure un segnale di pericolo. Il segno di spunta viene visualizzato quando la regola è classificata come appartenente alla classe 0 (innocua), il segnale di pericolo nel caso in cui la regola è classificata nella tre restanti classi;
- La seconda sezione contiene i dati della regola, quindi il titolo e descrizione dati dall'utente poco prima del *submit*, i servizi, *trigger* ed *action* selezionati;
- La terza sezione contiene le percentuali con la quale la regola è stata classificata nelle quattro classi. In particolare, le *progress bar* possono assumere 3 colori: rosso (quando la percentuale è minore del 30%), verde (se la percentuale è maggiore del 50%), giallo (se la percentuale è compresa tra il 30% e il 50%).


IFTTT
Crea una regola
Carica un file di regole

Crea una regola

Crea una regola e verificane la sicurezza. Clicca sui bottoni per creare la tua ricetta!

if

then


Conferma
Elimina



Nome

Quando il mio cellulare si connette al WiFi invia un messaggio a mio fratello

Descrizione

Questa regola invia un messaggio a mio fratello ogni volta che torno a casa

TriggerChannelTitle	Google Wifi
TriggerTitle	Device Connects
ActionChannelTitle	Telegram
ActionTitle	Send message

Harmless:

99.75%

Personal Attack:

0.060%

Physical Attack:

0.062%

Cybersecurity Attack:

0.120%

Figura 5.12: Interfaccia visibile dopo aver fatto il *sumbit* di una regola.

5.4.2 Descrizione della sezione “Carica un file di regole”

La sezione "Carica un file", già mostrata in Figura 5.2, permette quindi all'utente di caricare un file JSON contenente tutte le regole di cui vuole verificare la pericolosità.

Il file che deve essere caricato, oltre ad essere in formato JSON, deve contenere oggetti JSON che contengono almeno i seguenti sei attributi (se le regole contengono altri attributi questi ultimi non saranno utilizzati per la classificazione della regola):

- *triggerChannelTitle*: è il servizio del *trigger*;
- *triggerTitle*: è il nome del *trigger*;
- *actionChannelTitle*: è il servizio dell'*action*;
- *actionTitle*: è il nome dell'*action*;
- *title*: è il titolo della regola;
- *desc*: è la descrizione della regola.

Per caricare il file basta cliccare sul *form* visibile in Figura 5.2 e cliccare sul pulsante *Submit*.

La *request* inviata al server, quindi, in questo caso conterrà l'intero file caricato dall'utente. Quando il server comincerà a caricare le applet classificate su Fibrebase, l'applicazione web sarà notificata del fatto che un nuovo record è stato aggiunto e otterrà l'elemento, permettendo così di caricarlo all'interno dell'interfaccia (Figura 5.13). Finché tutte le applet che si trovano all'interno del file non saranno caricate, sarà mostrata al termine dell'interfaccia uno *spinner* visibile in Figura 5.14. Inoltre, lo stesso spinner sarà visualizzato nel caso in cui il file da caricare è di grandi dimensioni.

Come è possibile notare nelle Figure 5.13 e 5.14, sul lato sinistro dell'interfaccia c'è un riquadro che mostra un contatore di tutte le applet fino a quel momento

classificate e mostrate all'interno dell'interfaccia e una sezione dedicata ai filtri.

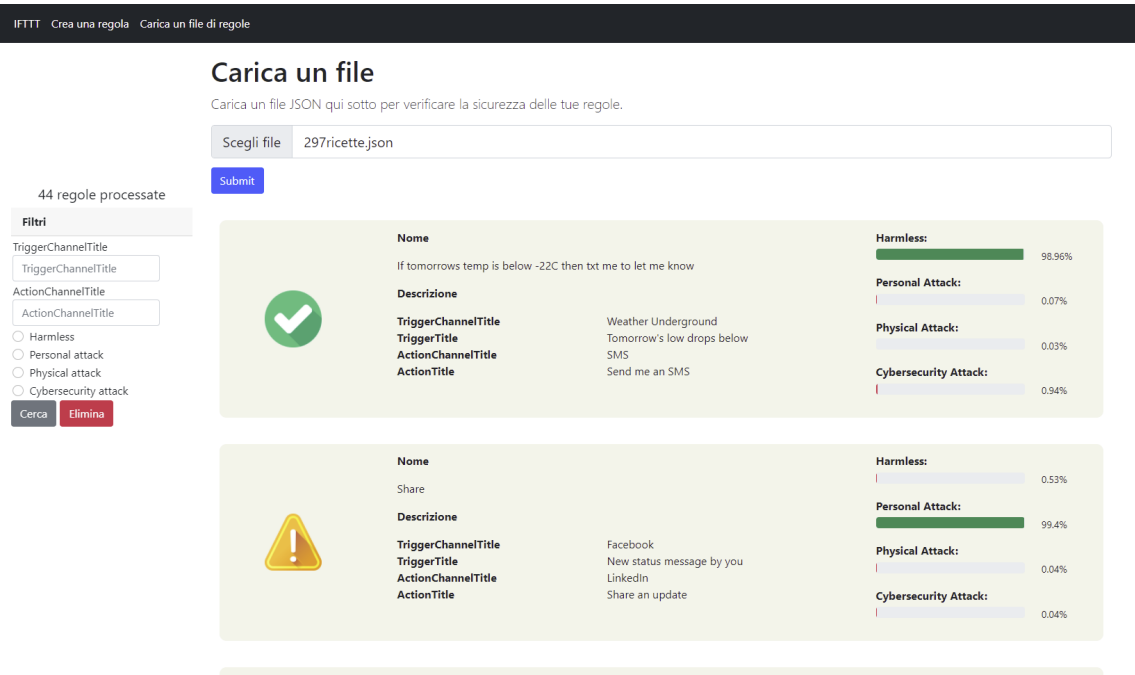


Figura 5.13: Risultati delle regole caricate attraverso un file JSON.

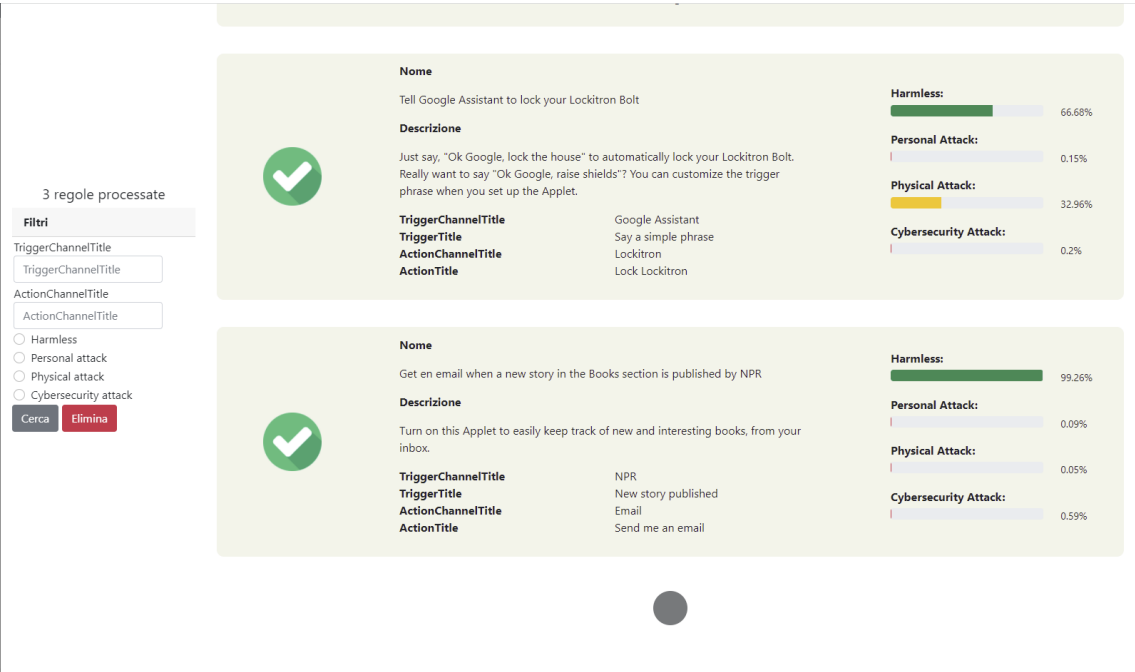


Figura 5.14: *Spinner* visualizzato durante il caricamento *realtime* delle applet caricate da un file.

I filtri servono per filtrare le applet classificate mostrate nell'interfaccia e possono essere utilizzati seguendo le seguenti combinazioni:

- Solo il nome del *triggerChannelTitle*, andando a mostrare solo le applet che hanno il *triggerChannelTitle* che cominciano per la stringa inserita in input (ad esempio, se si inserisce "Googl" saranno visibili solamente le applet che hanno come *triggerChannelTitle* "Google Assistant", "Google Calendar", "Google Contacts", "Google WiFi");
- Solo il nome dell'*actionChannelTitle*, che ha un comportamento uguale al precedente applicato all'*actionChannelTitle* delle applet;
- Solo una delle quattro classi di classificazione, andando a mostrare solo le applet che hanno una percentuale della classe selezionata maggiore del 50% (ad esempio, se si sceglie la *Harmless* saranno mostrate solo le applet classificate che sono considerate innocue);
- Il nome del *triggerChannelTitle* e dell'*actionChannelTitle*, andando a mostrare solo le applet che hanno il *triggerChannelTitle* ed *actionChannelTitle* che cominciano per le stringhe rispettivamente inserite negli input adibiti (ad esempio, se si inserisce in *triggerChannelTitle* "RSS" e in *actionChannelTitle* "Tesc", saranno visualizzate solamente le applet che hanno "RSS Feed" come *triggerChannelTitle* e Tesco come *actionChannelTitle*);
- Il nome del *triggerChannelTitle* e una delle quattro classi, andando a mostrare solamente le applet che hanno il *triggerChannelTitle* che cominciano per la stringa inserita in input e appartenenti alla classe selezionata (ad esempio, se si inserisce "RSS" in *triggerChannelTitle* e si seleziona "Harmless", saranno visualizzate solamente le applet che hanno "RSS Feed" come *triggerChannelTitle* e sono considerate innocue);
- Il nome dell'*actionChannelTitle* e una delle quattro classi, andando a mostrare solamente le applet che hanno l'*actionChannelTitle* che cominciano per la stringa inserita in input e appartenenti alla classe selezionata (ad

esempio, se si inserisce “Tesc” in *actionChannelTitle* e si seleziona “Harmless”, saranno visualizzate solamente le applet che hanno “Tesco” come *actionChannelTitle* e sono considerate innocue);

- Il nome del *triggerChannelTitle*, dell’*actionChannelTitle* una delle quattro classi, andando a mostrare solo le applet che hanno il *triggerChannelTitle* ed *actionChannelTitle* che cominciano per le stringhe rispettivamente inserite negli input adibiti e che appartengono alla classe selezionata (ad esempio, se si inserisce in *triggerChannelTitle* “RSS”, in *actionChannelTitle* “Tesc” e si seleziona “Harmless”, saranno visualizzate solamente le applet che hanno “RSS Feed” come *triggerChannelTitle*, Tesco come *actionChannelTitle* e che sono considerate innocue).

Se, dopo aver filtrato, l’utente vuole rivedere nuovamente la lista di tutte le applet può essere cliccato il bottone “Elimina” che si trova nel riquadro dei filtri e tutti i filtri saranno azzerati.

Capitolo 6

Conclusioni

L'IoT è ormai diventato un concetto ben noto a tutti, vista la trasversale applicazione di questo paradigma in molti ambiti. La comparsa dell'IoT ha dato una spinta all'innovazione e alla diffusione degli oggetti *smart*, capaci di comunicare e scambiarsi informazioni tra loro attraverso la definizione di regole, permettendo un miglioramento della qualità della vita. Tuttavia la definizione di tali regole possono potenzialmente causare agli utenti che le creano attacchi fisici, personali o cybersecurity.

Il lavoro di tesi presentato ha riguardato l'implementazione di un'interfaccia web utile alla creazione, definizione e verifica di regole IFTTT valutandone la pericolosità, tramite l'uso di un modello di AI in grado di classificare le regole in quattro classi: innocua, attacco fisico, attacco personale, attacco cybersecurity. La piattaforma implementata può essere utilizzata come piattaforma di *crowdsourcing*, permettendo così di ottenere valori sempre più accurati ottenuti dal classificatore.

Il progetto presentato può essere un valido punto di partenza per eventuali miglioramenti e sviluppi futuri, come ad esempio la possibilità di, dopo aver creato e verificato la solidità di una regola IFTTT, crearla in modo del tutto automatico nel proprio account IFTTT. Altri sviluppi futuri possono essere la possibilità di creare e verificare regole con più *trigger* ed *action* e la possibilità di scegliere

re il paradigma con cui creare le regole (*wizard paradigm, If-Do paradigm, wired paradigm*).

Bibliografia

- [1] *Cambridge Analytica*. it. Page Version ID: 121853814. Lug. 2021. URL: https://it.wikipedia.org/w/index.php?title=Cambridge_Analytica&oldid=121853814 (visitato il 02/12/2021).
- [2] Gaetano Cimino. «Identificazione di violazioni di sicurezza e privacy in regole IFTTT tramite tecniche di Natural Language Processing». In: *Tesi di Laurea Magistrale in Informatica, Dipartimento di Informatica, Università degli Studi di Salerno* (2021).
- [3] Camille Cobb et al. «How Risky Are Real Users' IFTTT Applets?». In: *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. USENIX Association, ago. 2020, pp. 505–529. ISBN: 978-1-939133-16-8. URL: <https://www.usenix.org/conference/soups2020/presentation/cobb>.
- [4] Giuseppe Desolda, Carmelo Ardito e Maristella Matera. «Empowering End Users to Customize their Smart Environments: Model, Composition Paradigms, and Domain-Specific Tools». In: *ACM Transactions on Computer-Human Interaction* 24.2 (apr. 2017), 12:1–12:52. ISSN: 1073-0516. DOI: 10.1145/3057859. URL: <https://doi.org/10.1145/3057859> (visitato il 19/11/2021).
- [5] Andrea Scripa Els. «Artificial Intelligence as a Digital Privacy Protector». In: *Harvard Journal of Law & Technology (Harvard JOLT)* 31 (2017), p. 217. URL: <https://heinonline.org/HOL/Page?handle=hein.journals/hjlt31&id=223&div=&collection=>.

- [6] Giuseppe Ghiani et al. «Personalization of Context-Dependent Applications Through Trigger-Action Rules». In: *ACM Transactions on Computer-Human Interaction* 24.2 (apr. 2017), 14:1–14:33. ISSN: 1073-0516. DOI: 10 . 1145/3057861. URL: <https://doi.org/10.1145/3057861> (visitato il 19/11/2021).
- [7] Pradyumna Gokhale, Omkar Bhat e Sagar Bhat. «Introduction to IOT». In: 5.1 (gen. 2018), pp. 41–44. ISSN: 2393-8021. URL: https://www.researchgate.net/profile/Omkar-Bhat/publication/330114646_Introduction_to_IOT/links/5c2e31cf299bf12be3ab21eb/Introduction-to-IOT.pdf.
- [8] *How crowdsourcing is changing the Waze we drive*. en-US. URL: <https://digital.hbs.edu/platform-rctom/submission/how-crowdsourcing-is-changing-the-waze-we-drive/> (visitato il 24/11/2021).
- [9] IFTTT. *IFTTT the beginning*. URL: <https://ifttt.com/explore/ifttt-the-beginning> (visitato il 24/11/2021).
- [10] *IoT e la medicina: i passi avanti che sono stati fatti, le soluzioni future*. it-IT. Ago. 2021. URL: <https://edalab.it/iot-medicina/> (visitato il 03/12/2021).
- [11] Xianghang Mi et al. «An empirical characterization of IFTTT: ecosystem, usage, and performance». In: *Proceedings of the 2017 Internet Measurement Conference*. IMC '17. London, United Kingdom: Association for Computing Machinery, nov. 2017, pp. 398–404. ISBN: 9781450351188. DOI: 10 . 1145 / 3131365 . 3131369. URL: <https://doi.org/10.1145/3131365.3131369> (visitato il 26/11/2021).
- [12] *Node.js*. it. Page Version ID: 121510037. Giu. 2021. URL: <https://it.wikipedia.org/w/index.php?title=Node.js&oldid=121510037> (visitato il 28/11/2021).
- [13] Amir Rahmati et al. «IFTTT vs. Zapier: A Comparative Study of Trigger-Action Programming Frameworks». In: *arXiv:1709.02788 [cs]* (set. 2017). ar-

- Xiv: 1709.02788. URL: <http://arxiv.org/abs/1709.02788> (visitato il 03/12/2021).
- [14] *React (web framework)*. it. Page Version ID: 123994419. Nov. 2021. URL: [https://it.wikipedia.org/w/index.php?title=React_\(web_framework\)&oldid=123994419](https://it.wikipedia.org/w/index.php?title=React_(web_framework)&oldid=123994419) (visitato il 28/11/2021).
- [15] Milijana Surbatovich et al. «Some Recipes Can Do More Than Spoil Your Appetite: Analyzing the Security and Privacy Risks of IFTTT Recipes». en. In: *Proceedings of the 26th International Conference on World Wide Web*. Perth Australia: International World Wide Web Conferences Steering Committee, apr. 2017, pp. 1501–1510. ISBN: 9781450349130. DOI: 10.1145/3038912.3052709. URL: <https://dl.acm.org/doi/10.1145/3038912.3052709> (visitato il 29/11/2021).
- [16] Ramine Tinati, Aastha Madaan e Wendy Hall. «The Role of Crowdsourcing in the Emerging Internet-Of-Things». In: *Proceedings of the 26th International Conference on World Wide Web Companion*. WWW '17 Companion. Perth, Australia: International World Wide Web Conferences Steering Committee, apr. 2017, pp. 1669–1672. ISBN: 9781450349147. DOI: 10.1145/3041021.3051693. URL: <https://doi.org/10.1145/3041021.3051693> (visitato il 24/11/2021).
- [17] Blase Ur et al. «Trigger-Action Programming in the Wild: An Analysis of 200,000 IFTTT Recipes». In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. CHI '16. San Jose, California, USA: Association for Computing Machinery, mag. 2016, pp. 3227–3231. ISBN: 9781450333627. DOI: 10.1145/2858036.2858556. URL: <https://doi.org/10.1145/2858036.2858556> (visitato il 26/11/2021).
- [18] Julia Carrie Wong. «One year inside Trump’s monumental Facebook campaign». en-GB. In: *The Guardian* (gen. 2020). ISSN: 0261-3077. URL: <https://www.theguardian.com/us-news/2020/jan/28/donald-trump-facebook-ad-campaign-2020-election> (visitato il 24/11/2021).

Ringraziamenti

Al termine di questo percorso appena concluso è doveroso fare dei ringraziamenti a tutte le persone che mi sono state vicine durante questi anni.

In primis voglio ringraziare mia mamma e mio papà per essermi stati sempre vicini e avermi supportato nei momenti per me più critici, facendomi anche a volte un po' arrabbiare per l'insistenza con la quale mi chiedevano "Come va lo studio?", "Che giorno hai l'esame?", ma so che dopotutto lo facevano per farmi capire che erano lì nel caso avessi avuto bisogno di sfogarmi e di parlare con qualcuno. Questo traguardo lo dedico a voi.

Ringrazio mio fratello Antonio che, nonostante non lo abbia mai dimostrato esplicitamente, so che se avessi avuto necessità di svagarmi o semplicemente parlare con qualcuno lui era disponibile e pronto ad ascoltarmi.

Ringrazio i miei compagni di avventura Saverio, Nicola e Roberto per i bei momenti di spensieratezza passati insieme per distrarci e staccare la spina, per l'aiuto con lo studio e l'incoraggiamento che mi hanno dato prima di ogni esame.

Ringrazio zio Gaetano e zia Franca per avermi dato la possibilità di studiare in pace e tranquillità durante i mesi estivi, dove la concentrazione per lo studio calava.

Ringrazio tutti i miei amici, dal primo all'ultimo, perché nel bene o nel male sono sempre stati pronti a dare una parola di conforto nei momenti in cui ne avevo più bisogno, a darmi una pacca sulla spalla ogni volta che pensavo di non farcela. In particolare ringrazio Antonio, che in questi anni non si è mai arreso nel chiedermi "Stasera esci? *Ja, accussi stacc' nu poc''*", a volte convincendomi e, nonostante abbia passato un brutto periodo, ha sempre avuto il tempo e il modo di farmi sentire la

sua presenza, auguro a tutti di avere un amico come lui, e ringrazio Valerio, amico di mille avventure, di tanti viaggi che abbiamo fatto e che spero continueremo a fare.

Ringrazio, infine, il professor Deufemia e il dottor Breve per essere sempre stati disponibili e pronti ad aiutarmi ad ogni difficoltà riscontrata nel progetto di tesi svolto.