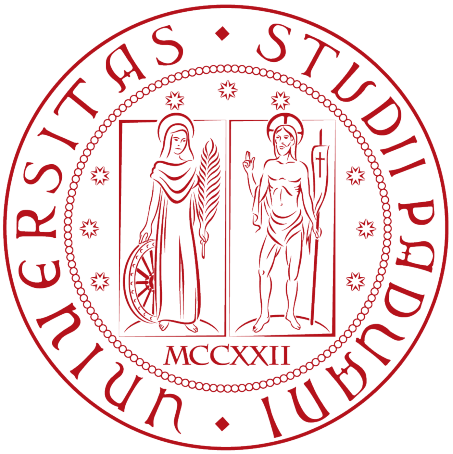
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



CAPTCHA e Accessibilità: Sviluppo di un Test di Turing accessibile a persone con disabilità visive.

Tesi di laurea

Relatore

Prof.ssa Ombretta Gaggi

Laureando

Edoardo Retis
Matricola 1100433

Anno Accademico 2022-2023

Nulla è impossibile. L'unico limite al nostro successo siamo noi stessi.

— Edoardo Retis

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage della durata di circa trecentoventi ore dal laureando Edoardo Retis presso l'azienda Zucchetti S.p.A. con sede a Padova (PD).
Lo scopo del progetto del tirocinio aziendale è sviluppare un CAPTCHA basato sulla visualizzazione di un orologio che possa essere fruibile per tutti i browser esistenti ed accessibile alle persone con disabilità visive. Il progetto è suddiviso in una prima fase di sviluppo del CAPTCHA, da presentare agli utenti quando accedono ad una pagina web, con al suo interno un algoritmo di "Proof of Work", necessario per scoraggiare gli utenti malevoli ad attaccare la pagina web protetta dal CAPTCHA con attacchi brute force; viene poi seguito dalla fase dello studio di possibili soluzioni per permettere al Test di Turing di essere risolto anche da persone con problemi alla vista, cercando così di assecondare le esigenze degli utenti che utilizzano gli screen reader senza però compromettere la sicurezza del Test oggetto della tesi.
Il documento è organizzato in quattro capitoli:

- **Introduzione**: in questo capitolo presento l'azienda dove ho svolto il tirocinio, il progetto di stage promosso dall'azienda ospitante, le motivazioni che hanno spinto lo sviluppo del progetto oggetto di questa tesi e gli obiettivi da superare;
- **Analisi dei requisiti**: in questo capitolo espongo l'analisi preliminare che ha portato allo sviluppo del prodotto oggetto di questa tesi;
- **Svolgimento dello stage**: in questo capitolo espongo il lavoro svolto durante le trecentoventi ore passate in azienda e il prodotto sviluppato durante lo stage;
- **Conclusioni**: presento un'analisi retrospettiva sugli obiettivi dello stage e una valutazione sui risultati conseguiti.

Nel documento utilizzerò le seguenti convenzioni tipografiche:

- `Font di codice software`: definizioni per codice software (variabili, metodi, classi);
- Termine in azzurro: termine a glossario, solo per la prima occorrenza per ogni capitolo. Ogni termine in azzurro sarà seguito dalla dicitura [g]che indica il termine "glossario";
- Numero naturale progessivo racchiuso da due parentesi quadre: riferimento bibliografico.

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine alla Prof.ssa Ombretta Gaggi, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero ringraziare con affetto mamma Patrizia e papà Diego per aver sempre creduto in me nonostante la mia poca costanza e nonostante più di una volta dimostrassi di non riuscire a portare a termine i compiti a me affidati.

Desidero ringraziare Anna L. per essermi sempre stata vicina in questi anni di studio e di duro lavoro e per avermi aiutato con la stesura di questa tesi.

Desidero ringraziare Igor P. per avermi insegnato che non si deve mai mollare e che gli obiettivi vanno sempre affrontati a testa alta.

Desidero ringraziare Jacopo A., Valentina C., Simone B., Michele C., Alberto M. per avermi dato la possibilità di svolgere lo stage e il progetto oggetto di questa tesi.

Desidero ringraziare inoltre i miei amici per tutti i bellissimi anni passati insieme e le mille avventure vissute.

Padova, Settembre 2023 Edoardo Retis

Indice

Elenco delle figure

Capitolo 1

Introduzione

Nel presente capitolo si introduce il percorso che ha contraddistinto il tirocinio in questione, offrendo un quadro completo e chiaro del contesto di riferimento; vengono presentati l'azienda presso la quale si è svolto lo stage, il piano di progetto relativo al tirocinio oggetto di questa tesi e le motivazioni che hanno portato alla stesura del suddetto piano di progetto. Infine vengono descritte le tecnologie utilizzate e imparate dal tirocinante durante lo stage.

1.1 L'azienda

La Zucchetti S.p.A. è un'azienda nata nel 1978 divenuta, oggigiorno, una delle migliori aziende italiane nel settore informatico, vantando un fatturato di 1,2 miliardi di euro (dato 2021) e ospitando nelle proprie sedi circa 8.000 (dato 2021)[1] dipendenti sparsi in sedi sparse in tutta Italia, tra cui 2.000 persone dedicate alla Ricerca e Sviluppo, oltre a sedi e Partner dislocati in tutto il mondo.
L'azienda fu fondata a Lodi (LO), in Italia, dal commercialista Domenico Zucchetti e inizialmente la neo azienda si occupava di distribuire ai vari commercialisti e professionisti nel settore economico un software capace di gestire le dichiarazioni dei redditi in modo automatizzato.
Attualmente, sotto la guida dei due figli di Domenico Zucchetti, Alessandro e Cristina Zucchetti, l'azienda si è posizionata in cima alle classifiche delle migliori aziende in tutti i settori dei servizi IT[g], offrendo soluzioni software, hardware e servizi per aziende, banche, assicurazioni, professionisti e associazioni di categoria, diventando presto una degli attori chiave nel mercato italiano delle soluzioni informatiche.
I principali prodotti offerti dalla Zucchetti S.p.A. sono software di:

- **ERP**, ovvero Enterprise Resource Planning (in italiano, pianificazione delle risorse aziendali), un tipo di software che le organizzazioni utilizzano per gestire le attività quotidiane di business, come ad esempio contabilità, procurement, project management, operazioni della supply chain, gestione del rischio e compliance;
- **Cloud computing**, un servizio che mette a disposizione ad utenti finali via internet delle risorse preesistenti, configurabili e disponibili in remoto sotto forma di architettura distribuita per l'archiviazione, l'elaborazione o la trasmissione dati;

- **Gestione del personale**, ovvero hardware e software applicativi finalizzati al supporto degli addetti HR (Human Resources, in italiano Risorse Umane) per la gestione del capitale umano all'interno di un ente (che sia esso pubblico o privato).

Per offrire la migliore qualità possibile, il personale tecnico di Zucchetti S.p.A. vanta molte certificazioni internazionali; quali lo standard UNI EN ISO 9001:2015[g]per la qualità dei processi interni, gli standard ISO/IEC 27001[g], 27017[g], 27018[g], 27701[g]e la certificazione BS 10012:2017[g]riguardante la sicurezza delle informazioni trasmesse tramite clouding e il rispetto delle norme che gestiscono il diritto alla privacy; la certificazione ISAE 3402[g]per garantire le best practice sulla fornitura di servizi e soluzioni IT. Inoltre, Zucchetti S.p.A. si avvale di best practice qualificati da AgID[g]per operare anche nell'ambito delle Pubbliche Amministrazioni.

1.2 I CAPTCHA

Nell'ambito informatico, un CAPTCHA[2] (dall'inglese, acronimo di Completely Automated Public Turing-test-to-tell Computers and Humans Apart, "test di Turing pubblico e completamente automatico per distinguere computer e umani") è un test completamente automatizzato, formato da una o più domande o sfide da superare che servono a determinare se un utente sia un umano e non un computer o, più precisamente, un bot[g].
I CAPTCHA vengono impiegati come barriera per evitare l'accesso dei bot a specifici servizi, quali forum, iscrizioni su siti web, inserimento di commenti e qualunque altro mezzo che potrebbe essere sfruttato per diffondere spam o perpetrare azioni di hacking, come gli attacchi brute force. Questo metodo di verifica è stato introdotto anche per contrastare lo spam prodotto da bot, imponendo all'autore di un messaggio email non riconosciuto dal destinatario di superare un test CAPTCHA prima che il messaggio possa essere consegnato.
I test CAPTCHA, per definizione sono totalmente automatici e solitamente non necessitano di interventi umani per la gestione o la manutenzione e offrono chiari benefici in termini di costi e affidabilità.
Gli algoritmi impiegati nella creazione dei test vengono frequentemente resi pubblici, sebbene in numerosi casi siano salvaguardati da brevetti. Questa politica di apertura mira a evidenziare che la robustezza del metodo non dipende dal segreto dell'algoritmo (che potrebbe essere decodificato tramite tecniche di reverse engineering[g]o metodi illeciti); al contrario, per "violare" l'algoritmo è necessario risolvere una sfida ritenuta "difficile" nel dominio dell'intelligenza artificiale.
L'uso di tecniche visive non è un requisito: qualunque problema di intelligenza artificiale di pari complessità, come il riconoscimento vocale, può fungere da base per un test di questo genere. Alcune implementazioni danno la possibilità all'utente di optare per un test basato su tecniche audio, sebbene questo approccio abbia avuto una crescita più lenta e non ne sia garantita la stessa efficacia del metodo visuale. Inoltre si possono adottare altri tipi di verifica che richiedano un'attività di comprensione testuale, come rispondere a una domanda o a un quiz logico, seguire precise istruzioni per creare una password, ecc; anche in questo caso le informazioni sulla resilienza di tali tecniche alle contromisure sono limitate.

1.3 Il progetto

1.3.1 Le motivazioni

L'idea del progetto di tirocinio è nata come una prosecuzione di un progetto didattico per il corso di Ingegneria del Software, tenuto dal Prof. Tullio Vardanega e promosso sempre dalla Zucchetti S.p.A., dove si chiedeva al gruppo di lavoro di cui il tirocinante faceva parte di sviluppare un CAPTCHA in grado di distinguere un utente umano da un bot e una applicazione web costituita da una pagina di login che presenti il suddetto CAPTCHA a protezione della suddetta login.
Come viene specificato dal capitolato del progetto[3], si chiedeva di sviluppare un'applicazione web che avesse un lato client implementato con HTML/CSS/JavaScript per visualizzare sul browser la form dell'applicazione e un sistema CAPTCHA che poteva essere una libreria Open Source, un servizio (obbligatoriamente gratuito) fruibile via web, un programma originale sviluppato dal gruppo di lavoro interagibile dall'utente e un lato server implementato con linguaggio Java o PHP che supporti il sistema CAPTCHA per evitare che il test sia facilmente eludibile. Dopo di chè, oltre ai prodotti sopra citati, si chiedeva di redigere un documento che analizzi la vulnerabilità del test e se il test è facilmente eludibile oppure se presenta caratteristiche che possano affrontare bene anche i bot più moderni in previsione alle future evoluzioni che avranno le IA del mondo odierno.

Inizialmente, il gruppo di lavoro ha fatto una ricerca per studiare quale CAPTCHA fosse idoneo a soddisfare le richieste del capitolato, facendo uno studio sui vari CAPTCHA attualmente in circolazione nel Web. Dopo un'attenta valutazione sui punti di forza e le vulnerabilità di ogni singolo CAPTCHA studiato, il gruppo di lavoro ha optato per sviluppare una propria libreria originale Open Source[g]basata sull'idea della visualizzazione di un orologio, richiedendo ad un utente che vuole accedere ad una sezione del sito web protetta dal CAPTCHA di visualizzarlo, opportunamente disturbato con linee e disegni, digitando in una form l'orario indicato dalle lancette dell'orologio. Un esempio del risultato del lavoro svolto dal gruppo di lavoro di Ingegneria del Software è mostrato nella figura 1.1
Durante le fasi finali di collaudo del progetto didattico, effettuato assieme al proponente aziendale della Zucchetti S.p.A., oltre a classificare il lavoro svolto come MVP (dall'inglese, acronimo di Minimum Viable Product, "minimo prodotto fattibile", acronimo che identifica una versione preliminare di un prodotto atteso, dotata di funzionalità minime e sufficienti per valutare la bontà della visione iniziale, consentire un adeguato uso esplorativo e permettere futuri avanzamenti del progetto/prodotto), il proponente aziendale è stato particolarmente entusiasta dell'idea originale e ben sviluppata, proponendo al gruppo di lavoro di inserire il CAPTCHA nel sistema aziendale per aggiornare la sicurezza degli applicativi prodotti dall'azienda.
Come si può vedere dall'immagine 1.2, il CAPTCHA a protezione delle login di accesso dei software Zucchetti è abbastanza obsoleto e facilmente eludibile dalle intelligenze artificiali attualmente in commercio, perché attualmente la maggior parte delle IA[g]riescono a superare i CAPTCHA testuali con un rate di successo molto alto[4] ed è necessario sostituire questo sistema con un test meno eludibile, utilizzando una nuova libreria Open Source per soddisfare le esigenze dei clienti che necessitano di ambienti chiusi e non collegati ad internet (a differenza di alcuni sitemi come, ad esempio, reCAPTCHA di Google o hCAPTCHA che necessitano di una connessione ad internet per poter funzionare).

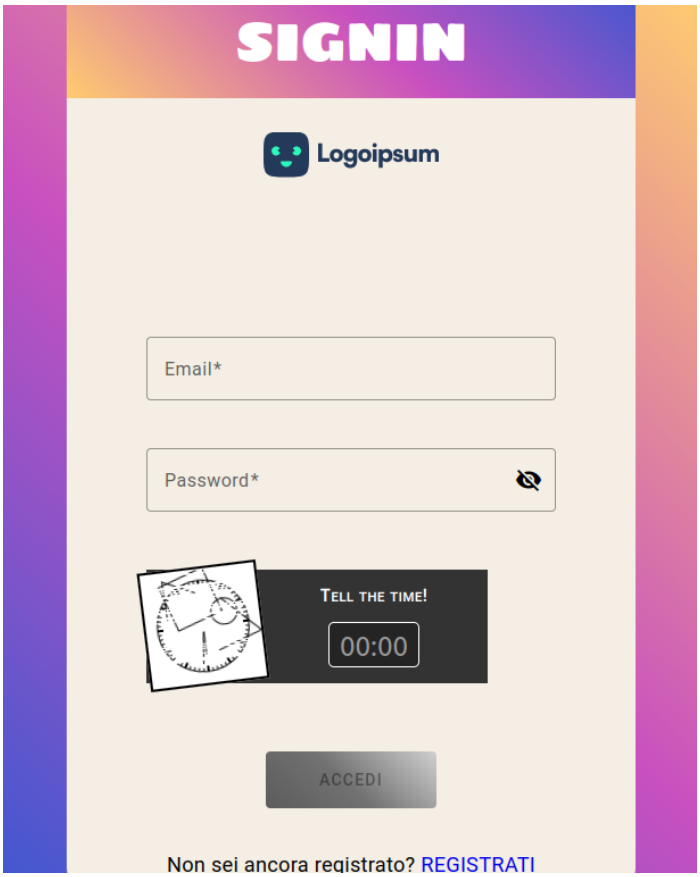


Figura 1.1: Login con CAPTCHA della WebApp sviluppata dal gruppo di lavoro di Ingegneria del Software.

Secondo quanto richiesto dall'azienda ospitante, un altro aspetto che il sistema da produrre deve tenere in considerazione è la sicurezza delle applicazioni protette dal CAPTCHA da utenti malevoli che utilizzano bot informatici per praticare attacchi brute force. Per attacchi brute force si intendono dei tentativi di decifrare una password o un nome utente oppure di trovare una pagina web nascosta o la chiave utilizzata per crittografare un messaggio utilizzando l'approccio "trial and error" con la speranza, alla fine, di indovinare. Si tratta di un vecchio metodo di attacco, ma è ancora efficace e molto usato dagli hacker. Per far fronte a questo problema, l'azienda ha richiesto allo studente di sviluppare un sistema di Proof of Work [5].
Un altro aspetto importante che l'azienda tiene in considerazione e che è oggetto di questa tesi è il problema dell'accessibilità[g]. In uno studio condotto nel 2021 dal Dott. Francesco Magarotto [6], si evince che la maggior parte dei CAPTCHA attualmente in commercio che richiedono di affrontare delle sfide, tra cui, per esempio, il riconoscimento del testo e la selezione di alcune immagini appartenenti ad un determinato contesto, discrimina molto le persone con disabilità visiva, rendendo le sezioni dei siti web protetti da questi sistemi poco fruibili da utenti con questo handicap. Per questo motivo, l'azienda Zucchetti S.p.A. ha chiesto al tirocinante che ha svolto lo stage in azienda di effettuare una ricerca per valutare se è fattibile rendere il sistema

CAPTCHA con la visualizzazione di un immagine di un orologio accessibile in modo che possa essere svolto e superato sia da utenti normodotati che da utenti con disabilità visive, cercando un buon compromesso tra sicurezza informatica e accessibilità.

1.3.2 Piano di progetto

Obiettivi

In base agli accordi stipulati con il tutor aziendale, per la conclusione del tirocinio aziendale è stato necessario soddisfare taluni obiettivi indicati nel seguente modo:

O[Tipologia dell'obiettivo][Numero intero progressivo]

Dove:

- **O**: sta per "Obiettivo";
- **Tipologia dell'obiettivo**: indica la tipologia di un determinato obiettivo, classificato in base al grado di importanza all'interno del progetto di stage;
- **Numero intero progressivo**: indica un numero progressivo per tipologia di obiettivo

La classificazione delle tipologie di obiettivi sono suddivisi nella seguente maniera:

- **N**, per gli obiettivi Necessari (Obbligatori);
- **D**, per gli obiettivi Desiderabili;
- **F**, per gli obiettivi Facoltativi (Opzionali).

Di seguito gli obiettivi accordati a livello aziendale:

- **ON1**: Sviluppo di un sistema CAPTCHA in HTML e JavaScript con algoritmo per contrastare gli attacchi *brute force*;
- **ON2**: Redazione documentazione dove viene spiegato il lavoro svolto dal tirocinante e i suoi risultati;
- **OD1**: Sviluppo di pratiche di accessibilità per rendere il CAPTCHA utilizzabile per utenti con disabilità visive;
- **OD2**: Utilizzo a livello avanzato del linguaggio di programmazione HTML e JavaScript;
- **OD3**: Utilizzo a livello avanzato del linguaggio di programmazione Java;
- **OF1**: Comprensione a livello avanzato dell'accessibilità dei siti web.

Pianificazione

In accordo con il piano di lavoro proposto da Zucchetti S.p.A., il tirocinio è durato 40 giorni (320 ore) con data di inizio il 19 giugno 2023 e data di fine il 25 agosto 2023. Il periodo di tirocinio è stato suddiviso in due blocchi come segue:

- **Blocco di sviluppo**: Periodo di lavoro dal 19 Giugno 2023 fino al 07 Luglio 2023 riservato per lo sviluppo del CAPTCHA in HTML, JavaScript e Java;

- **Blocco di ricerca**: Periodo di lavoro dal 17 Luglio 2023 fino al 25 Agosto 2023 riservato per lo sviluppo dell'accessibilità del CAPTCHA;
- **Blocco di documentazione**: Periodi di lavoro dal 10 Luglio 2023 fino al 14 Luglio 2023 e dal 21 Agosto 2023 fino al 25 Agosto 2023 riservati per la redazione della documentazione necessaria per spiegare il lavoro svolto dal tirocinante e i suoi risultati.

Come viene specificato nell'immagine 1.3, in base agli accordi presi con il tutor esterno, ogni blocco dello stage aziendale è stato suddiviso in opportune task da completare per raggiungere gli obiettivi fissati nel paragrafo precedente. Durante la fase di pianificazione del progetto di stage ad ogni task di ogni blocco è stata assegnata una determinata quantità di ore come stima del tempo di realizzazione delle suddette task per valutare l'andamento del progetto ed eventuali problemi che possono insorgere durante tutto lo svolgimento dello stage.
I blocchi dello stage aziendale sono stati pianificati nella seguente maniera:

- **Primo blocco**: In questo blocco viene sviluppato il sistema CAPTCHA da inserire negli applicativi Zucchetti e il Proof of Work necessario per scoraggiare gli attacchi brute force. Questo blocco è stato suddiviso nelle seguenti task:
  1. **Maschera minimale con CAPTCHA**: In questa task lo studente si dedicherà alla creazione di una maschera minimale in HTML e JavaScript che implementi un test CAPTCHA basato sulla visualizzazione di un orologio;
  2. **Metodo di Proof of Work**: In questa task lo studente si dedicherà alla creazione di un metodo di Proof of Work in JavaScript utilizzabile assieme al test CAPTCHA;
  3. **Inclusione della maschera con CAPTCHA**: In questa task lo studente si dedicherà all'inclusione della maschera con CAPTCHA sviluppata dallo studente nel framework aziendale.
- **Secondo blocco**: In questo blocco viene sviluppata una ricerca finalizzata a capire se è possibile rendere il sistema CAPTCHA accessibile agli utenti con disabiità visive. Questo blocco è stato suddiviso nelle seguenti task:
  1. **Bilanciamento tra nitidezza e tempo trascorso**: In questa task lo studente si dedicherà alla realizzazione di un algoritmo per ridurre i disturbatori dal test CAPTCHA in base al tempo trascorso per risolvere il test CAPTCHA;
  2. **Proposizione vocale dell'ora**: In questa task lo studente si dedicherà alla realizzazione di un metodo per permettere agli utenti non vedenti di distinguere l'orario da scrivere per il test CAPTCHA usando una proposizione vocale;
  3. **Ricerca accessibilità del sistema CAPTCHA**: In questa task lo studente si dedicherà alla ricerca per vericare se il CAPTCHA con visualizzazione dell'orologio e la sua controparte vocale possano essere accessibili per un utenza con disabilità visive.
- **Terzo blocco**: In questo blocco vengono svolti i test per valutare l'efficacia del codice prodotto e viene redatta la documentazione di tutto il progetto dello stage al fine di spiegare il lavoro svolto dal tirocinante e i risultati conseguiti a conclusione dello stage. I risultati conseguiti dal tirocinante saranno oggetto di questa tesi.

1.3.3 Tecnologie utlizzate

Di seguito presenterò le tecnologie con cui sono entrato in contatto durante il periodo di tirocinio.

Linguaggi

- **HTML (HyperText Markup Language, traduzione letterale: linguaggio a marcatori per ipertesti)**: linguaggio di markup nato per la formattazione e impaginazione di documenti ipertestuali. Oggi è utilizzato principalmente per il disaccoppiamento della struttura logica di una pagina web (definita appunto dal markup) e la sua rappresentazione, gestita tramite gli stili CSS;
- **Javascript**: linguaggio di programmazione multi paradigma orientato agli eventi, viene utilizzato sia nella programmazione lato client web che lato server per la creazione di rest API[g], applicazioni desktop e embedded, siti web e applicazioni web, di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso (mouse, tastiera, caricamento della pagina ecc...);
- **CSS (sigla di Cascading Style Sheets, in italiano fogli di stile a cascata)**: linguaggio usato per definire la formattazione di documenti HTML, XHTML e XML[g]come ad esempio i siti web e relative pagine web;
- **Java**: linguaggio di programmazione ad alto livello orientato agli oggetti e a tipizzazione statica, che si appoggia sull'omonima piattaforma software di esecuzione è specificamente progettato per essere il più possibile indipendente dalla piattaforma hardware di esecuzione (tramite compilazione in bytecode prima e interpretazione poi da parte di una JVM[g]);
- **JSON (JavaScript Object Notation)**: formato per lo scambio dati basato sul linguaggio di programmazione JavaScript ed è utilizzato in programmazione web come alternativa al formato XML. In questo caso è stato utilizzato per la programmazione in AJAX[g].

Ambiente di sviluppo

- **Visual Studio Code**: editor di codice sorgente sviluppato da Microsoft per Windows, Linux e macOS. Include il supporto per debugging, un controllo per Git integrato, syntax highlighting, IntelliSense, snippet e refactoring del codice; questo editor è stato utilizzato per scrivere il lato client del progetto;
- **Eclipse**: ambiente di sviluppo integrato (IDE[g]) per lo sviluppo del software. È utilizzato principalmente per lo sviluppo di Java, ma supporta anche altri linguaggi di programmazione come C++[g], Python[g]e PHP[g]. Eclipse fornisce una vasta gamma di strumenti e funzionalità per aiutare gli sviluppatori a scrivere, testare e mantenere il codice in modo più efficiente. È una multipiattaforma e funziona su Windows, Linux e MacOS. Questo editor è stato utilizzato per scrivere il lato server del progetto.

Software di supporto

- **Microsoft Edge**: browser web sviluppato da Microsoft ed è il browser disponibile di serie su Windows 11. Questo browser è stato utilizzato in fase di testing;
- **Mozilla Firefox**: browser web libero e multipiattaforma, mantenuto da Mozilla Foundation. Questo browser è stato utilizzato in fase di testing;
- **Google Chrome**: browser Web sviluppato da Google. Questo browser è stato utilizzato in fase di testing;
- **ChatGPT**: intelligenza artificiale sviluppata da OpenAI. Questa intelligenza artificiale è stata utilizzata in fase di studio sull'accessibilità;
- **Bing**: intelligenza artificiale sviluppata da Microsoft. Questa intelligenza artificiale è stata utilizzata in fase di studio sull'accessibilità;
- **Jwebserver**: tool disponibile con il Java Development Kit (JDK[g]) versione 20. Permette di sviluppare un ambiente server per scopi di ricerca o di testing. Questo tool è stato usato per sviluppare la parte server del progetto;
- **Microsoft Word**: applicazione per pacchetto office di Microsoft utilizzato per redigere la documentazione necessaria e richiesta dall'azienda.

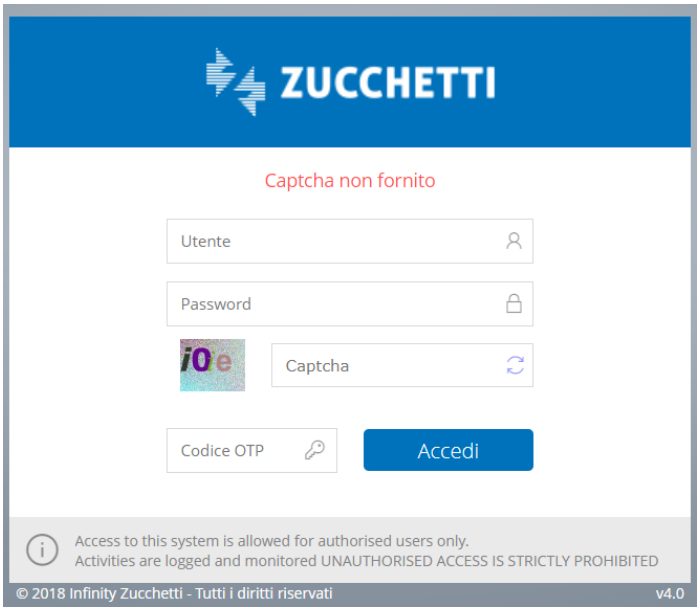


Figura 1.2: Esempio di login di un software Zucchetti.

Blocco	Task	Ore preventivate
Primo Blocco	Maschera minimale con CAPTCHA	40
	Metodo di Proof of Work	40
	Inclusione della maschera con CAPTCHA	40
Secondo blocco	Bilanciamento tra nitidezza e tempo trascorso	40
	Proposizione vocale dell'ora	40
	Ricerca accessibilità sistema CAPTCHA	40
Terzo blocco	Test e documentazione	80

Figura 1.3: Riassunto della pianificazione per il progetto di stage.

Capitolo 2

Analisi dei requisiti

Nel presente capitolo viene esposto il processo di analisi dei requisiti del progetto di stage, ovvero l'attività preliminare allo sviluppo del prodotto software il cui scopo è quello di definire le funzionalità che il prodotto deve offrire, cioè i requisiti che devono essere soddisfatti dal software sviluppato; verranno introdotti le notazioni e le convenzioni usate nel presente capitolo e successivamente verranno esposti i casi d'uso frutto della suddetta analisi.

2.1 I casi d'uso

Lo scopo finale del prodotto sviluppato durante lo stage è quello di rimpiazzare l'attuale libreria CAPTCHA usato per le login e le form degli applicativi Zucchetti, ovvero la libreria CAPTCHA basata sulla visualizzazione di un testo distorto che attualmente (nel 2023) è un test obsoleto sia in termini di sicurezza informatica sia in termini di accessibilità; Questa è la ragione fondamentale per cui le diverse riunioni con il tutor aziendale (e unico stakeholder[g]) e l'indagine della soluzione per soddisfare lo stakeholder hanno portato alla compilazione di una lista di soli requisiti utente focalizzata unicamente sul CAPTCHA suggerito.
Per studiare i requisiti utente del prodotto e successivamente, pianificare lo sviluppo del codice sono stati creati dei casi d'uso. Un caso d'uso è un insieme di scenari (sequenze di azioni) che hanno in comune uno scopo finale (obiettivo) per un utente (attore). I Casi d'uso descrivono l'insieme di funzionalità del sistema come sono percepite dagli utenti, ponendo una visione esterna del sistema senza nessun dettaglio implementativo. A supporto dei casi d'uso sono stati creati dei diagrammi. I diagrammi dei casi d'uso sono diagrammi di tipo UML[g](dall'inglese, acronimo di Unified Modeling Language) dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso. Essendo il progetto finalizzato alla creazione di un semplice CAPTCHA per una login di un applicazione, le interazioni da parte dell'utilizzatore sono ridotte allo stretto necessario. Per questo motivo i diagrammi dei casi d'uso risultano semplici e in numero ridotto. Tutti i diagrammi dei casi d'uso sono stati creati con UML 2.0.

2.2 Notazione

Ogni caso d'uso rappresenta un insieme di scenari che hanno in comune uno scopo finale per un utente generico del sistema, chiamato attore. I casi d'uso possono essere descritti tramite dei diagrammi, possono estendersi in più sotto casi e possiedono una precondizione seguita da una postcondizione.
Ogni caso d'uso descritto in questa tesi è definito da un codice identificativo come segue:

UC[Numero caso d'uso padre](.[Numero caso d'uso figlio])*: [Nome del caso d'uso]

Dove:

- **UC**: sta per "Use Case";
- **Numero caso d'uso padre**: è il numero identificativo del caso d'uso associato al requisito principale e viene rappresentato da una stringa numerica che viene incrementata a ogni caso d'uso identificato;
- **Numero caso d'uso figlio**: è il numero identificativo del caso d'uso associato al sotto caso del requisito principale, viene rappresentato da una stringa numerica che viene incrementata a ogni caso d'uso identificato;
- **Nome del caso d'uso**: è un breve titolo descrittivo del caso d'uso.

Seguendo le best practices dell'Ingegneria del Software, ogni caso d'uso è descritto tramite la seguente struttura:

- **Attori**: indica gli attori principali e secondari del caso d'uso;
- **Diagramma UML**: indica il diagramma UML specifico del caso d'uso;
- **Descrizione**: viene riportata una breve descrizione del caso d'uso;
- **Precondizione**: specifica le condizioni che sono indicate come vere prima del verificarsi degli eventi del caso d'uso;
- **Postcondizione**: specifica le condizioni che sono indicate come vere dopo il verificarsi degli eventi del caso d'uso;
- **Scenario principale**: rappresenta il flusso degli eventi;
- **Estensioni**: specificano i casi d'uso che estendono questo caso d'uso (nella notazione dei casi d'uso, questa voce è facoltativa).

2.3 Attori

In seguito all'analisi degli obiettivi dello stage e agli incontri con il proponente gli attori dei casi d'uso individuati sono i seguenti:

- **Utente**: colui che interagisce col sistema; vista la natura del prodotto richiesto, l'utente è sempre inteso senza sessione attiva;

- **CAPTCHA lato client**: componente lato client del CAPTCHA; si occupa di visualizzare sul browser i dati ricevuti dal server e ricevere l'input dell'utente e far partire il sistema di Proof of Work;
- **CAPTCHA lato server**: componente lato server del CAPTCHA. Si occupa di generare i dati da mandare al client e di attestare se l'interazione con il sistema è fatta da un essere umano o da un bot.

La rappresentazione degli attori nei diagramma Use Case sono riportati nella figura 2.1.

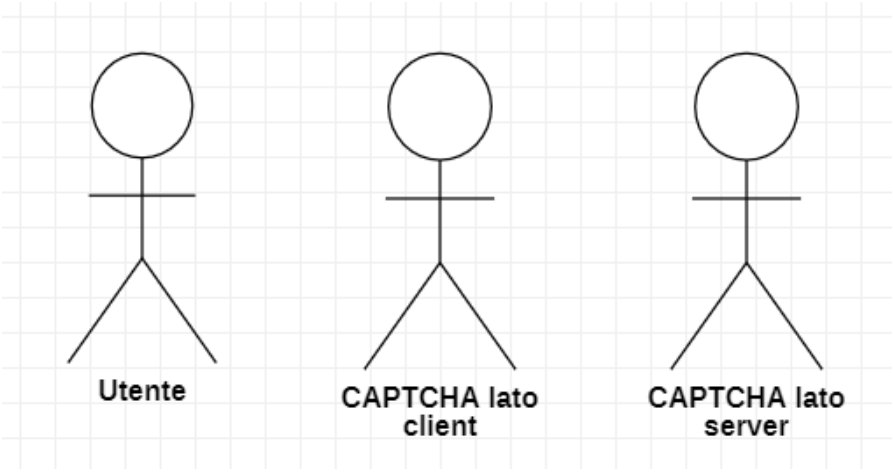


Figura 2.1: Rappresentazione degli attori nei casi d'uso.

2.4 Panoramica sui casi d'uso - CAPTCHA lato client

In questo paragrafo verranno esposti tutti i casi d'uso del sistema CAPTCHA lato client. Il diagramma UML generale è mostrato nella figura 2.2.

UC1: Risoluzione CAPTCHA

Attori: Utente, CAPTCHA Lato server.

Diagramma UML: Figura 2.3.

Descrizione: L'utente visualizza correttamente l'immagine creata dal sistema CAPTCHA, legge l'orario visualizzato nell'immagine a schermo e lo inserisce nell'apposito spazio.

Precondizioni: Il sistema contiene il modulo di test che riporta l'immagine distorta dell'orologio e la casella d'inserimento.

Postcondizioni: il sistema riconosce l'utente come un essere umano.

Scenario Principale:

1. L'utente legge l'orario dall'immagine distorta dell'orologio così come fornita dal sistema CAPTCHA [UC1.1];
2. L'utente inserisce l'orario letto nell'apposita casella [UC1.2];

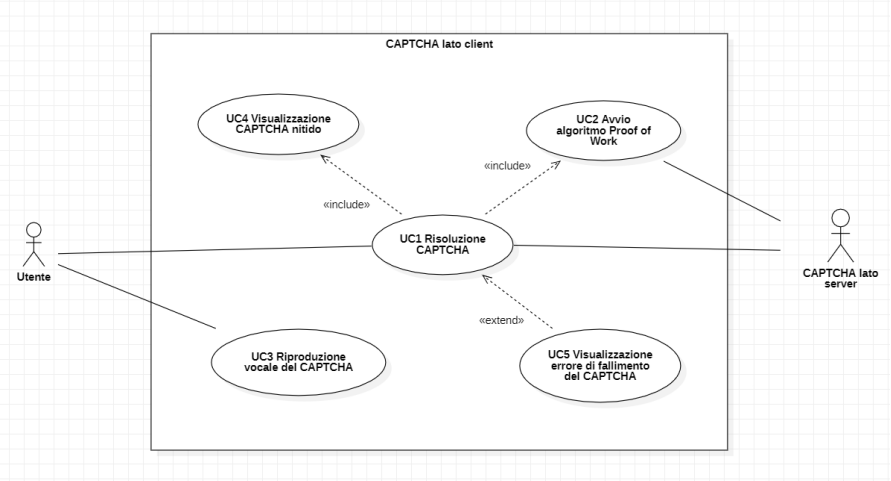


Figura 2.2: Casi d'uso primari del CAPTCHA lato client.

3. Il sistema riconosce che l'utente è un essere umano.

Estensioni:

- Nel caso in cui il valore inserito non corrisponda al valore mostrato:
  1. Viene notificato un errore di fallimento del test [UC5];
  2. L'azione non prosegue.

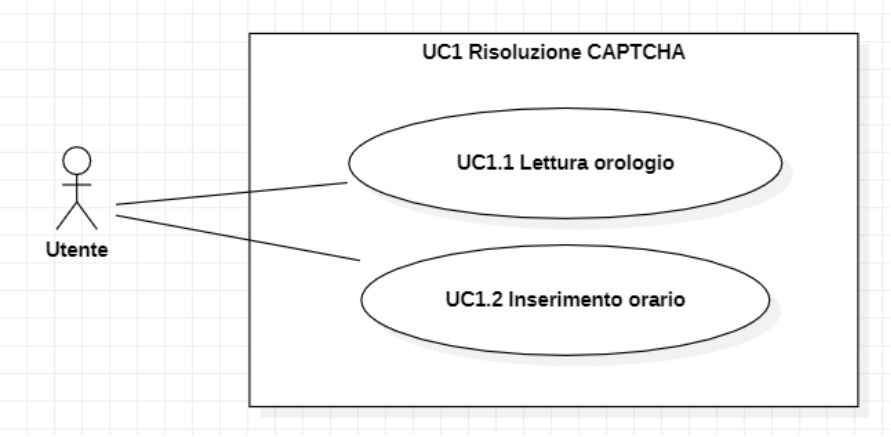


Figura 2.3: Use Case - UC1: Risoluzione CAPTCHA.

UC1.1: Lettura orologio

Attori: Utente.

Diagramma UML: Figura 2.4.

Descrizione: L'utente visualizza l'immagine distorta di un orologio analogico, che raffigura un orario ben preciso.

Precondizioni: Il sistema è funzionante e raggiungibile.

Postcondizioni: L'utente ha individuato le lancette dell'orologio con l'orario indicato.

Scenario Principale: L'utente visualizza l'orario indicato nell'immagine distorta dell'orologio analogico.

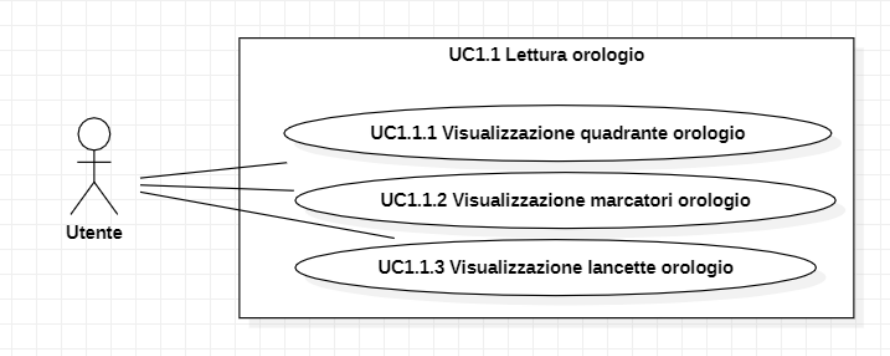


Figura 2.4: Use Case - UC1.1: Lettura orologio.

UC1.1.1: Visualizzazione quadrante orologio

Attori: Utente.

Diagramma UML: Figura 2.4.

Descrizione: Per essere in grado di riconoscere l'orario, l'utente deve poter individuare il quadrante dell'orologio.

Precondizioni: Il sistema è funzionante e raggiungibile.

Postcondizioni: L'utente riesce a individuare gli elementi fondamentali del quadrante dell'orologio, in modo da poter poi interpretare correttamente la posizione delle lancette.

Scenario Principale: L'utente individua nell'immagine il quadrante dell'orologio, il quale delimita la parte contenente il dato da leggere.

UC1.1.2: Visualizzazione marcatori orologio

Attori: Utente.

Diagramma UML: Figura 2.4.

Descrizione: Per essere in grado di riconoscere l'orario, l'utente deve poter individuare i marcatori dell'orologio.

Precondizioni: Il sistema è funzionante e raggiungibile, il quadrante deve essere visibile.

Postcondizioni: L'utente è in grado di individuare all'interno del quadrante i marcatori necessari per la lettura corretta dell'orario.

Scenario Principale: L'utente individua nell'immagine i marcatori di ore e minuti inseriti nel quadrante.

UC1.1.3: Visualizzazione lancette orologio

Attori: Utente.

Diagramma UML: Figura 2.4.

Descrizione: Per essere in grado di riconoscere l'orario, l'utente deve poter individuare le lancette dell'orologio.

Precondizioni: Il sistema è funzionante e raggiungibile, il quadrante e i marcatori devono essere visibili.

Postcondizioni: L'utente è in grado di individuare all'interno del quadrante la lancette che consentono la lettura dell'orario.

Scenario Principale: L'utente individua nell'immagine dell'orologio due tipi di lancette:

1. La lancetta più corta delle ore;
2. La lancetta più lunga dei minuti.

UC1.2: Inserimento orario

Attori: Utente.

Diagramma UML: Figura 2.5.

Descrizione: L'utente inserisce l'orario letto nell'immagine dell'orologio.

Precondizioni: Il sistema è raggiungibile e funzionante, l'utente ha visualizzato l'immagine dell'orologio.

Postcondizioni: L'utente ha inserito l'orario letto nell'immagine.

Scenario Principale: L'utente inserisce il valore dell'orario contenuto nel test appena visualizzato.

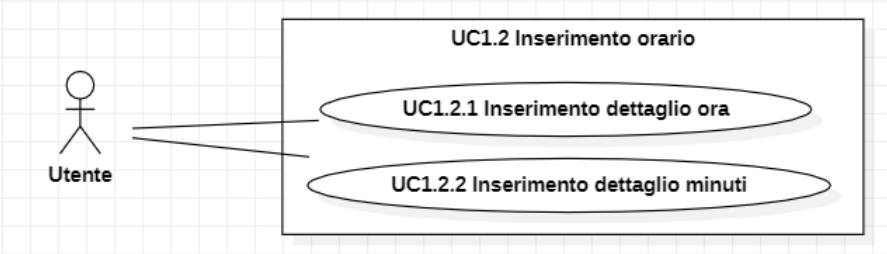


Figura 2.5: Use Case - UC1.2: Inserimento orario.

UC1.2.1: Inserimento dettaglio ora

Attori: Utente.

Diagramma UML: Figura 2.5.

Descrizione: L'utente inserisce il numero corrispondente all'ora letta nell'immagine dell'orologio.

Precondizioni: Il sistema è raggiungibile e funzionante, l'utente ha visualizzato l'immagine dell'orologio.

Postcondizioni: L'utente ha inserito il valore numerico corrispondente esclusivamente all'ora letta nell'immagine.

Scenario Principale: L'utente inserisce il valore dell'ora contenuto nel test appena visualizzato.

UC1.2.2: Inserimento dettaglio ora

Attori: Utente.

Diagramma UML: Figura 2.5.

Descrizione: L'utente inserisce il numero corrispondente ai minuti letti nell'immagine dell'orologio.

Precondizioni: Il sistema è raggiungibile e funzionante, l'utente ha visualizzato l'immagine dell'orologio.

Postcondizioni: L'utente ha inserito il valore numerico corrispondente esclusivamente ai minutti letti nell'immagine.

Scenario Principale: L'utente inserisce il valore dei minuti contenuto nel test appena visualizzato.

UC2: Avvio algoritmo Proof of Work

Attori: Utente, CAPTCHA lato server.

Diagramma UML: Figura 2.2.

Descrizione: Il sistema CAPTCHA avvia in automatico l'algoritmo di Proof of Work, l'algoritmo genera l'hash corretto e invia il risultato al CAPTCHA lato server.

Precondizioni: Il CAPTCHA è correttamente istanziato all'interno del browser.

Postcondizioni: L'hash correttamente generato è inviato al CAPTCHA lato server.

Scenario Principale:

1. Il sistema CAPTCHA lato client avvia l'algoritmo di Proof of Work;
2. L'algoritmo genera l'hash corretto;
3. L'hash corretto viene comunicato al sistema CAPTCHA lato server.

UC3: Riproduzione vocale del CAPTCHA

Attori: Utente.

Diagramma UML: Figura 2.2.

Descrizione: L'utente ascolta l'orario da indovinare tramite un test sonoro.

Precondizioni: Il CAPTCHA è correttamente istanziato all'interno del browser.

Postcondizioni: L'orario da indovinare viene ascoltato tramite un audio vocale.

Scenario Principale: L'utente ascolta l'orario da indovinare.

UC4: Visualizzazione CAPTCHA nitido

Attori: Utente.

Diagramma UML: Figura 2.2.

Descrizione: L'utente visualizza l'immagine dell'orario da indovinare con un immagine sempre più nitida e con meno disturbatori man mano che il tempo passa.

Precondizioni: Il CAPTCHA è correttamente istanziato all'interno del browser.

Postcondizioni: L'immagine dell'orario da indovinare è completamente nitida e priva di disturbatori.

Scenario Principale:

1. L'utente visualizza l'immagine dell'orario da indovinare;
2. Man mano che il tempo passa vengono tolti sempre più disturbatori;
3. L'immagine dell'orario da indovinare è completamente nitida e priva di disturbatori.

UC5: Visualizzazione errore di fallimento del CAPTCHA

Attori: Utente.

Diagramma UML: Figura 2.2.

Descrizione: L'utente non inserisce un valore accettabile e viene notificato dell'accadimento.

Precondizioni: L'utente ha inserito la sua interpretazione del test.

Postcondizioni: L'utente è informato del risultato del test.

Scenario Principale: L'utente visualizza un messaggio di errore che lo informa di non aver superato il test e di doverlo ripetere per proseguire.

2.5 Panoramica sui casi d'uso - CAPTCHA lato server

In questo paragrafo verranno esposti tutti i casi d'uso del sistema CAPTCHA lato client. Il diagramma UML generale è mostrato nella figura 2.6.

UC6: Generazione dati del test

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.7.

Descrizione: Il CAPTCHA lato client richiede al CAPTCHA lato server di generare i dati per il test.

Precondizioni: Il CAPTCHA lato client contiene un ancora nella quale ritiene necessario un CAPTCHA.

Postcondizioni: Il CAPTCHA lato client contiene il test ben visibile e ben formato

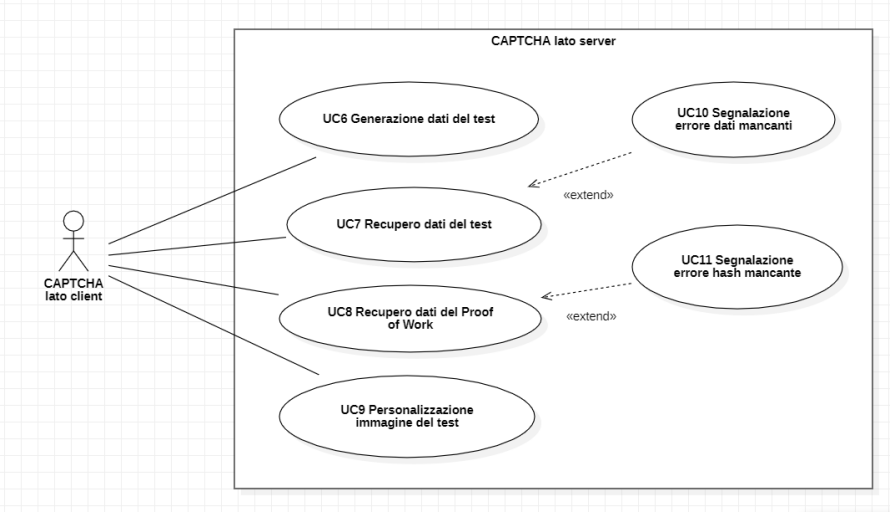


Figura 2.6: Casi d'uso primari del CAPTCHA lato server.

nel punto indicato dall'ancora.

Scenario Principale:

1. Il CAPTCHA lato client richiede la generazione dell'immagine distorta dell'orologio [UC6.1];
2. Il CAPTCHA lato client richiede la generazione dell'orario dell'orologio [UC6.2];
3. Il CAPTCHA lato client richiede la generazione dei dati per l'algortmo di Proof of Work [UC6.3];
4. Il CAPTCHA lato client richiede la generazione del testo per la riproduzione vocale dell'orario dell'orologio [UC6.4];
5. Il CAPTCHA lato server genera i dati necessari per il test.

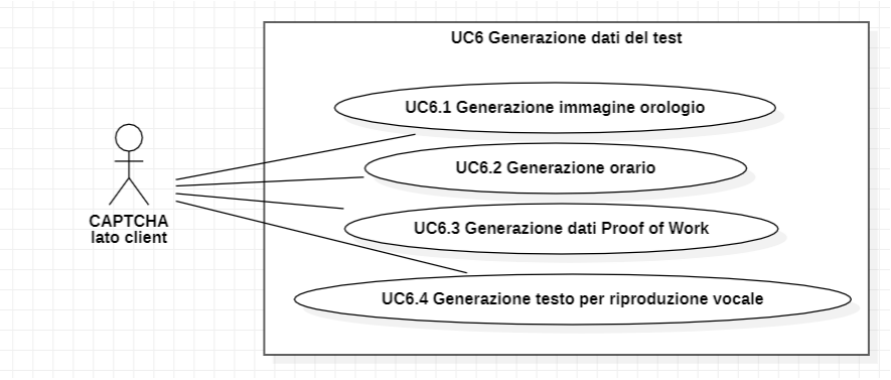


Figura 2.7: Use Case - UC6: Generazione dati del test.

UC6.1: Generazione immagine orologio

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.8.

Descrizione: Il CAPTCHA lato client richiede al CAPTCHA lato server di generare l'immagine distorta dell'orologio oggetto del test.

Precondizioni: Il CAPTCHA lato client contiene un ancora nella quale ritiene necessario l'inserimento dell'immagine dell'orologio da utilizzare come test.

Postcondizioni: Il CAPTCHA lato client contiene l'immagine dell'orologio ben visibile e ben formata nel punto indicato dall'ancora.

Scenario Principale:

1. Il CAPTCHA lato cliente richiede la generazione dell'immagine dell'orologio da utilizzare come test;
2. Il CAPTCHA lato server genera e costruisce l'immagine, definendo:
   - L'orario che la figura dovrà rappresentare,
   - Il quadrante dell'orologio [UC6.1.1],
   - I marcatori dell'orologio [UC6.1.2],
   - Le lancette dell'orologio [UC6.1.3],
   - Gli elementi di disturbo all'interno dell'orologio [UC6.1.4].

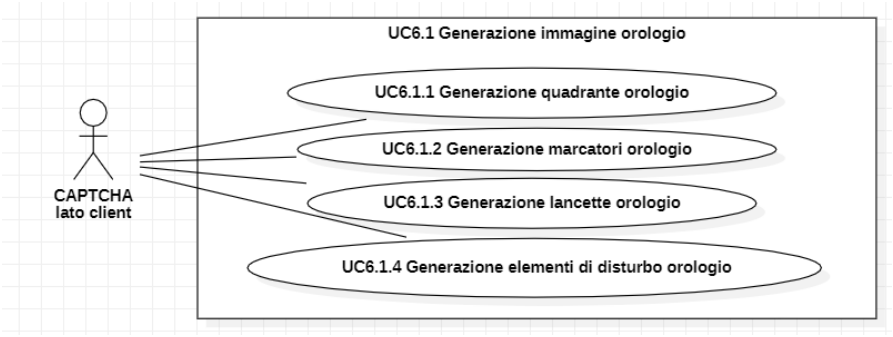


Figura 2.8: Use Case - UC6.1: Generazione immagine orologio.

UC6.1.1: Generazione quadrante orologio

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.8.

Descrizione: Il CAPTCHA lato client richiede al CAPTCHA lato server di generare l'immagine distorta dell'orologio oggetto del test, a partire dal quadrante dell'orologio.

Precondizioni: Il CAPTCHA lato client contiene un ancora nella quale ritiene necessario l'inserimento dell'immagine dell'orologio da utilizzare come test.

Postcondizioni: Il quadrante dell'orologio è stato correttamente definito ed è disponibile per essere utilizzato come elemento dell'immagine in fase di istanziazione del CAPTCHA.

Scenario Principale:

1. Il CAPTCHA lato client ha richiesto l'inserimento dell'immagine dell'orologio da utilizzare come test;
2. Il CAPTCHA lato server definisce il quadrante dell'orologio che avrà la forma finale di un cerchio.

UC6.1.2: Generazione marcatori orologio

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.9.

Descrizione: L'immagine dell'orologio richiesta dal CAPTCHA lato client dovrà necessariamente essere costituita da marcatori.

Precondizioni: Il CAPTCHA lato client contiene un ancora nella quale ritiene necessario l'inserimento dell'immagine dell'orologio da utilizzare come test.

Postcondizioni: All'interno del quadrante sono stati istanziati i marcatori dell'orologio.

Scenario Principale: All'interno del quadrante vengono allocati i marcatori delle ore e dei minuti, in modo tale da rendere possibile la lettura finale dell'orario indicato dalle lancette.

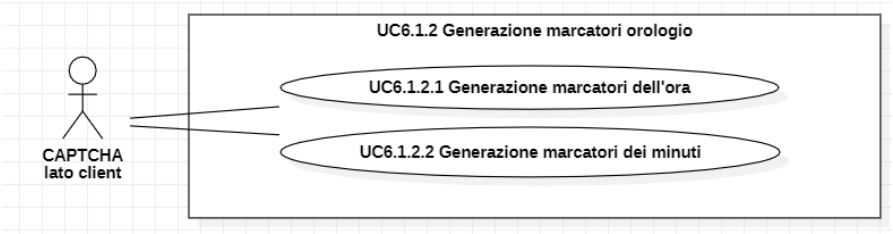


Figura 2.9: Use Case - UC6.1.2: Generazione marcatori orologio.

UC6.1.2.1: Generazione marcatori dell'ora

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.9.

Descrizione: L'immagine dell'orologio richiesta dal CAPTCHA lato client dovrà necessariamente essere costituita da marcatori dell'orario, i quali dovranno apparire più marcati rispetto ai marcatori dei minuti.

Precondizioni: Il CAPTCHA lato client contiene un ancora nella quale ritiene necessario l'inserimento dell'immagine dell'orologio da utilizzare come test.

Postcondizioni: Le ore sono ben raffigurate all'interno del quadrante.

Scenario Principale: All'interno del quadrante vengono allocati i marcatori delle ore, in modo tale da rendere possibile la lettura finale dell'orario preciso indicata dalla lancetta dell'ora.

UC6.1.2.1: Generazione marcatori dei minuti

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.9.

Descrizione: L'immagine dell'orologio richiesta dal CAPTCHA lato client dovrà necessariamente essere costituita da marcatori dei minuti, i quali dovranno apparire meno marcati rispetto ai marcatori delle orei.

Precondizioni: Il CAPTCHA lato client contiene un ancora nella quale ritiene necessario l'inserimento dell'immagine dell'orologio da utilizzare come test.

Postcondizioni: I minuti sono ben raffigurate all'interno del quadrante.

Scenario Principale: All'interno del quadrante vengono allocati i marcatori delle ore, in modo tale da rendere possibile la lettura finale dell'orario preciso indicata dalla lancetta dell'ora.

UC6.1.3: Generazione lancette orologio

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.10.

Descrizione: L'immagine dell'orologio richiesta dal CAPTCHA lato client dovrà necessariamente contenere le lancette delle ore e dei minuti.

Precondizioni: Il CAPTCHA lato client contiene un ancora nella quale ritiene necessario l'inserimento dell'immagine dell'orologio da utilizzare come test.

Postcondizioni: All'interno del quadrante sono individuabili le lancette delle ore e dei minuti.

Scenario Principale:

1. Il CAPTCHA lato client ha richiesto l'inserimento dell'immagine dell'orologio da utilizzare come test;
2. Il CAPTCHA lato server definisce l'orario che deve essere rappresentato;
3. Attraverso l'orario precedentemente definito, le lancette vengono generate e opportunamente posizionate all'interno del quadrante.

UC6.1.3.1: Generazione lancette dell'ora

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.10.

Descrizione: L'immagine dell'orologio richiesta dal CAPTCHA lato client dovrà necessariamente contenere le lancetta relativa alle ore.

Precondizioni: Il CAPTCHA lato client contiene un ancora nella quale ritiene necessario l'inserimento dell'immagine dell'orologio da utilizzare come test e il CAPTCHA

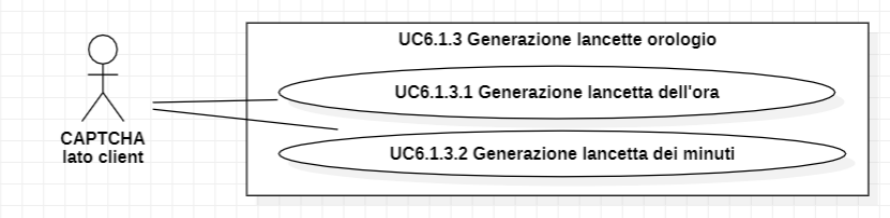


Figura 2.10: Use Case - UC6.1.3: Generazione lancette orologio.

lato server ha calcolato l'orario da rappresentare.

Postcondizioni: All'interno del quadrante è individuabile la lancetta delle ore.

Scenario Principale:

1. Il CAPTCHA lato client ha richiesto l'inserimento dell'immagine dell'orologio da utilizzare come test;
2. Il CAPTCHA lato server definisce l'orario che deve essere rappresentato;
3. Attraverso l'orario precedentemente definito la lancetta dell'ora viene generata e opportunamente posizionata all'interno del quadrante.

UC6.1.3.2: Generazione lancette dei minuti

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.10.

Descrizione: L'immagine dell'orologio richiesta dal CAPTCHA lato client dovrà necessariamente contenere le lancetta relativa ai minuti.

Precondizioni: Il CAPTCHA lato client contiene un ancora nella quale ritiene necessario l'inserimento dell'immagine dell'orologio da utilizzare come test e il CAPTCHA lato server ha calcolato l'orario da rappresentare.

Postcondizioni: All'interno del quadrante è individuabile la lancetta dei minuti.

Scenario Principale:

1. Il CAPTCHA lato client ha richiesto l'inserimento dell'immagine dell'orologio da utilizzare come test;
2. Il CAPTCHA lato server definisce l'orario che deve essere rappresentato;
3. Attraverso l'orario precedentemente definito la lancetta dei minuti viene generata e opportunamente posizionata all'interno del quadrante.

UC6.1.4: Generazione elementi di disturbo orologio

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.8.

Descrizione: L'immagine dell'orologio richiesta dal CAPTCHA lato client dovrà

necessariamente contenere elementi di disturbo.

Precondizioni: L'immagine dell'orologio con i suoi elementi base, quadrante, marcatori e lancette, è stata definita.

Postcondizioni: All'interno dell'immagine dell'orologio sono stati inseriti elementi di disturbo.

Scenario Principale: All'interno dell'orologio vengono inseriti elementi di disturbo, definiti come segue:

- Forme geometriche diverse: quadrati, cerchi, triangoli e linee;
- Linee con diverse dimensioni: piccole, medie o grandi;
- Linee orizzontali nere che simulano una distorsione dell'immagine.

Tutti gli elementi definiti devono essere allocati all'interno del quadrante dell'orologio.

UC6.2: Generazione orario

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.7.

Descrizione: Il CAPTCHA lato client richiede al CAPTCHA lato server di generare l'orario per il test del CAPTCHA.

Precondizioni: Il CAPTCHA lato client contiene un CAPTCHA appena inizializzato nella quale ritiene necessario la comunicazione di un orario per popolare il test.

Postcondizioni: Il CAPTCHA lato client contiene un CAPTCHA con un test ben visibile e ben formato con il test contenente l'orario richiesto.

Scenario Principale:

1. Il CAPTCHA lato client richiede la generazione di un orario;
2. Il CAPTCHA lato server genera e comunica l'orario per il test.

UC6.3: Generazione dati Proof of Work

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.7.

Descrizione: Il CAPTCHA lato client richiede al CAPTCHA lato server di generare i dati per inizializzare l'algoritmo di Proof of Work.

Precondizioni: Il CAPTCHA lato client contiene un CAPTCHA appena inizializzato nella quale ritiene necessario la comunicazione dei dati per l'algoritmo di Proof of Work.

Postcondizioni: Il CAPTCHA lato client contiene un CAPTCHA con un test ben visibile e ben formato contenente l'algoritmo di Proof of Work inizializzato e funzionante.

Scenario Principale:

1. Il CAPTCHA lato client richiede la generazione del dati per l'algoritmo di Proof of Work;
2. Il CAPTCHA lato server genera e comunica i dati al CAPTCHA lato client.

UC6.4: Generazione testo per riproduzione vocale

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.7.

Descrizione: Il CAPTCHA lato client richiede al CAPTCHA lato server di generare il testo per consentire la riproduzione vocale del test CAPTCHA.

Precondizioni: Il CAPTCHA lato client contiene bottone che consente la riproduzione vocale del test CAPTCHA e ritiene necessario la comunicazione di un testo da riprodurre.

Postcondizioni: Il CAPTCHA lato client contiene un CAPTCHA con un test ben visibile e ben formato con associato un bottone per la riproduzione vocale del test CAPTCHA inizializzato e funzionante.

Scenario Principale:

1. Il CAPTCHA lato client richiede la generazione del testo;
2. Il CAPTCHA lato server genera e comunica il testo al CAPTCHA lato client.

UC7: Recupero dati del test

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.6.

Descrizione: Il CAPTCHA lato client recupera il risultato del test, il quale identifica l'utente come umano o come bot.

Precondizioni: Il test è stato effettuato, per cui il CAPTCHA lato client fornisce al CAPTCHA lato server il valore orario ricevuto in fase di istanziazione e l'input orario da valutare.

Postcondizioni: Il CAPTCHA lato server ha elaborato l'informazione ricevuta e il CAPTCHA lato client è a conoscenza del risultato del test.

Scenario Principale:

1. Il CAPTCHA lato client richiede il risultato del test fornendo al CAPTCHA lato server il valore numerico corrispondente alla soluzione indicata dall'utente;
2. Il CAPTCHA lato server confronta i dati ricevuti;
3. Il CAPTCHA lato server comunica il risultato del test.

Estensioni:

- Nel caso in cui non venga fornito alcun valore dal CAPTCHA lato client o i dati forniti non corrispondono a quelli richiesti:

1. Viene segnalato l'errore nei dati forniti [UC10];
2. L'azione non prosegue.

UC8: Recupero dati del Proof of Work

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.6.

Descrizione: Il CAPTCHA lato client recupera il risultato dell'algoritmo, il quale identifica l'utente come utente non malevolo o malevolo.

Precondizioni: L'algoritmo è stato effettuato, per cui il CAPTCHA lato client fornisce al CAPTCHA lato server il valore hash calcolato.

Postcondizioni: Il CAPTCHA lato server ha elaborato l'informazione ricevuta e il CAPTCHA lato client è a conoscenza del risultato del controllo.

Scenario Principale:

1. Il CAPTCHA lato client richiede il risultato dell'algoritmo fornendo al CAPTCHA lato server il valore corrispondente all'hash generato dall'algoritmo;
2. Il CAPTCHA lato server confronta i dati ricevuti;
3. Il CAPTCHA lato server comunica il risultato del test.

Estensioni:

- Nel caso in cui non venga fornito alcun valore dal CAPTCHA lato client o i dati forniti non corrispondono a quelli richiesti:
  1. Viene segnalato l'errore nei dati forniti [UC11];
  2. L'azione non prosegue.

UC9: Personalizzazione immagine del test

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.11.

Descrizione: Il CAPTCHA lato server crea l'immagine del test sulla base delle richieste avanzate dal CAPTCHA lato client.

Precondizioni: Il CAPTCHA lato server è funzionante.

Postcondizioni: L'immagine di test viene adattata alle richieste dal CAPTCHA lato client.

Scenario Principale:

1. Il CAPTCHA lato client richiede la personalizzazione dell'immagine di test nei seguenti modi possibili:
   (a) Definendo il numero di elementi di disturbo che l'immagine dovrà contenere [UC9.1];

(b) Definendo il livello di distorsione da applicare all'immagine finale [UC9.2].

2. Il servizio applica le richieste nella creazione dell'immagine del test.

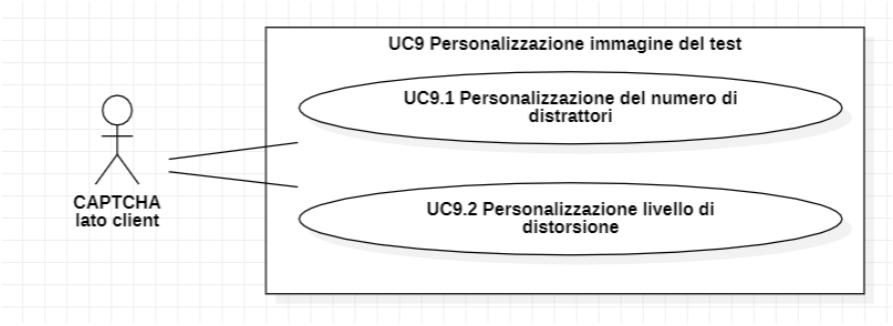


Figura 2.11: Use Case - UC9: Personalizzazione immagine del test.

UC9.1: Personalizzazione del numero di distrattori

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.11.

Descrizione: Il CAPTCHA lato server definisce gli elementi di disturbo all'interno dell'immagine dell'orologio precedentemente generata sulla base del numero fornito dal CAPTCHA lato client.

Precondizioni: L'immagine dell'orologio è stata generata.

Postcondizioni: L'immagine dell'orologio presenta il numero di elementi di disturbo richiesto dal CAPTCHA lato client.

Scenario Principale:

1. Il CAPTCHA lato client fornisce al CAPTCHA lato server il numero di elementi di disturbo da inserire all'interno dell'orologio;
2. Il CAPTCHA lato server aggiunge il numero richiesto di elementi di disturbo all'interno dell'immagine di test.

UC9.2: Personalizzazione livello di distorsione

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.11.

Descrizione: Il CAPTCHA lato client può decidere il livello di distorsione da applicare all'immagine del test, a seconda del grado di sicurezza desiderato.

Precondizioni: L'immagine dell'orologio è stata definita.

Postcondizioni: L'immagine dell'orologio ha subito le modifiche richieste dal CAPTCHA lato client.

Scenario Principale:

1. Il CAPTCHA lato client richiede al CAPTCHA lato server di alterare l'immagine dell'orologio tramite selezionando un livello di distorsione da applicare:
   - Livello 0: Nessuna distorsione da applicare, l'immagine rimane ben chiara e definita. Questo livello di distorsione serve solo in fase di testing e non è adatto all'uso pubblico in quanto rende il CAPTCHA facilmente bypassabile e poco usabile;
   - Livello 1: Lieve distorsione da applicare, i tratti che definiscono l'immagine risultano più frammentati e meno netti, il compito risulta ancora semplice da portare a termine;
   - Livello 2: Moderata distorsione da applicare, i tratti che definiscono l'immagine risultano ancora più frammentati e il compito richiede più attenzione.
2. Il CAPTCHA lato server applica il livello di distorsione come richiesto.

UC10: Segnalazione errore dati mancanti

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.11.

Descrizione: Il CAPTCHA lato client ha richiesto al CAPTCHA lato server l'invio del risultato del test, ma riceve un messaggio di errore.

Precondizioni: Il CAPTCHA lato client ha richiesto il risultato del test al CAPTCHA lato server ed è in attesa di ricevere risposta.

Postcondizioni: Il CAPTCHA lato client riceve la segnalazione di errore nei dati forniti.

Scenario Principale:

1. Il CAPTCHA lato client ha inviato la richiesta al CAPTCHA lato server per ricevere i risultati del test;
2. Il CAPTCHA lato server ritorna al Il CAPTCHA lato client un messaggio di errore relativo ai dati mancanti nell'invio della richiesta.

UC11: Segnalazione errore hash mancante

Attori: CAPTCHA lato client.

Diagramma UML: Figura 2.11.

Descrizione: Il CAPTCHA lato client ha richiesto al CAPTCHA lato server l'invio del risultato del controllo sull'hash generato dal Proof of Work, ma riceve un messaggio di errore.

Precondizioni: Il CAPTCHA lato client ha richiesto il risultato del controllo sull'hash generato dal Proof of Work al CAPTCHA lato server ed è in attesa di ricevere risposta.

Postcondizioni: Il CAPTCHA lato client riceve la segnalazione di errore nei dati forniti.

Scenario Principale:

1. Il CAPTCHA lato client ha inviato la richiesta al CAPTCHA lato server per ricevere i risultati del test;
2. Il CAPTCHA lato server ritorna al Il CAPTCHA lato client un messaggio di errore relativo ai dati mancanti nell'invio della richiesta.

Capitolo 3

Svolgimento dello stage

Nel presente capitolo viene esposto il lavoro svolto dal tirocinante durante il periodo di stage aziendale, ovvero viene mostrato lo svolgimento del lavoro in tutte le sue fasi; verrà spiegato, soffermandomi sui punti salienti, in che modo è avvenuto lo sviluppo del prodotto richiesto allo studente, la ricerca che ho fatto sull'accessibilità e i risultati ottenuti.

3.1 Modello incrementale

Da come viene mostrato dal Piano di progetto esposto nel cap. 1, le task e che lo studente doveva rispettare erano organizzate in modo tale da sviluppare prima le funzionalità fondamentali e successivamente sviluppare le feature accessorie utilizzando l'approccio del metodo incrementale delle best practice dell'Ingegneria del Software per consolidare le parti importanti da sviluppare per poi avere una base solida da cui partire per incrementi che possano produrre valore ad ogni passo. In base a questo approccio, durante le prime settimane dello stage le forze dello stagista si sono concentrate sullo sviluppo di un prototipo iniziale del sistema CAPTCHA e successivamente il prototipo è stato aggiornato con un insieme crescente di funzionalità utili sia per la sicurezza del CAPTCHA sia per la ricerca sull'accessibilità del CAPTCHA stesso.

3.2 Proof of Concept

Nelle prime fasi del progetto dello stage, lo studente si è dedicato nello sviluppo un Proof of Concept del prodotto da sviluppare, ovvero un prototipo eseguibile e funzionante per testare la fattibilità e la praticabilità della soluzione proposta. Il Proof of Concept serviva allo studente per eseguire un esplorazione tecnologica delle varie tecnologie necessarie per lo sviluppo del progetto di stage e serviva all'azienda per valutare la "bontà" dell'architettura scelta e della soluzione proposta.
Il Proof of Work sviluppato in questo stage consisteva in una maschera minimale scritta totalmente in HTML e Javscript ed interamente sviluppata client-side per agevolare lo sviluppo dei successivi incrementi e il testing del codice. In accordo con il tutor aziendale, per il test automatico da implementare si è deciso di optare per un test che impone ad un utente di visualizzare il quadrante di un orologio analogico e di scrivere in una form su browser l'orario indicato dal quadrante. Dato che esistono già IA che possono leggere e riconoscere le lancette di un orologio analogico (vedi articolo

[7]), l'immagine del quadrante deve essere opportunamente distorta con opportuni disturbatori che possono essere disegni da apportare sul quadrante oppure pezzi di immagine del quadrante tolti dall'immagine.
La prima cosa che è stata sviluppata dallo studente è l'oggetto `ClockCAPTCHAView`. L'oggetto `ClockCAPTCHAView` rappresenta un elemento HTML utile per mostrare a schermo i dati generati dalla libreria, raccogliere l'input dell'utente e visualizzare eventuali errori.
L'oggetto `ClockCAPTCHAView` si preoccupa di costruire l'elemento HTML utilizzando il proprio costruttore ed invocando un metodo chiamato `moduleBuild()` che si occupa di creare i vari tag HTML di cui è composto il test e di inserire per ogni tag dello stile CSS per formare la View del CAPTCHA. Inoltre, dopo aver costruito il CAPTCHA, l'oggetto `ClockCAPTCHAView` va ad inserire gli event listener per raccogliere l'input dell'utente e per mostrare a schermo i messaggi per comunicare all'utente se l'esito del CAPTCHA è andato bene oppure se è andato storto qualcosa.
Dopo aver costruito la parte che costruisce la struttura del CAPTCHA, lo studente ha scritto l'algoritmo che genera randomicamente un orario, attraverso il metodo `generateData()`, e l'algoritmo che genera un immagine di un orologio a partire da un determinato orario passatogli come parametro, utilizzando il metodo `generateClockImage()`.
Il metodo `generateClockImage()` non solo crea il disegno del quadrante dell'orologio, ma si preoccupa anche di disegnare, sempre in maniera randomica per non aiutare eventuali utenti malevoli, i disturbatori da mettere sopra il quadrante.
I disturbatori sono di tre tipi:

- Linee colorate
- Forme geometriche
- Strisce di immagine cancellate

Il numero di linee colorate, di forme e di strisce cancellate può essere facilmente modificato perché tali numeri sono passati come parametri quando il metodo viene invocato e sta allo sviluppatore che utilizza la libreria scegliere il grado di difficoltà da imporre nel test.
L'algoritmo di generazione dell'immagine viene invocato direttamente all'interno del metodo che genera l'orario, perché i dati glieli passa il metodo direttamente come parametri. Dopo i calcoli fatti, i due algoritmi ritornano tre valori:

- `image` che rappresenta l'immagine dell'orologio in formato Base64;
- `time` che rappresenta l'orario generato randomicamente partendo da un range 0-12;
- `pmtime` che rappresenta l'orario convertito al pomeriggio in un range 12-24.

`time` e `pmtime` servono per fare in modo di evitare di confondere l'utente mentre svolge il test, perché visualizzando un quadrante analogico non si riesce a capire se indica le ore al mattino o al pomeriggio e quindi si è deciso di rendere accettabili entrambe le soluzioni.
Costruito l'oggetto `ClockCAPTCHAView` e i due algoritmi per generare il test, lo studente è passato a scrivere il codice per costruire la maschera richiesta dall'azienda. Per prima cosa, è stato creato un file HTML dove all'interno è stato aggiunto un tag `<div>` utilizzato come ancora a quello che sarà il CAPTCHA costruito tramite Javascript.

Dopodiché, come mostrato nella figura 3.1, è stato creato uno script in Javascript che ha il compito di fare tre cose:

- Istanziare l'oggetto `ClockCAPTCHAView` e ancorare l'oggetto al tag `<div>` attraverso il metodo `inject()`;
- Generare una data e un immagine randomica richiamando gli appositi algoritmi;
- Inserire i dati forniti dagli algoritmi nel CAPTCHA invocando il metodo `fill()`.

```
import { generateData } from "./imageGenerator.js";
import * as cc_view from "./clockCAPTCHAView.js";

//Creazione CAPTCHA
var captchaModule = new cc_view.ClockCAPTCHAView();
captchaModule.inject(document.getElementById("clock-captcha"));

//Generazione dei dati da inserire nel test
//(l'algoritmo di generazione dell'immagine è invocato internamente da generateData())
var cc_data = generateData();

//inserimento dell'orario dell'immagine all'interno del CAPTCHA
captchaModule.fill(cc_data.image, cc_data.time, cc_data.pmtime);
```

Figura 3.1: Script scritto in JavaScript per l'inizializzazione del CAPTCHA nel browser.

Con la figura 3.2 verrà mostrato un esempio del CAPTCHA sviluppato dal tirocinante.

Figura 3.2: Esempio del test CAPTCHA basato sulla visualizzazione di un orologio.

3.3 Metodo di Proof of Work

Dopo aver appurato che il Proof of Concept sviluppato fornisca una solida base di partenza per il progetto, lo studente si è dedicato allo sviluppo dell'algoritmo di Proof of Work necessario per contrastare gli attacchi brute force.
Uno dei requisiti dell'azienda e soprattutto uno degli scopi del CAPTCHA implementato nei suoi prodotti è quello di bloccare (o scoraggiare) gli attacchi brute force che possono insorgere da utenti malevoli.
Per far fronte a questo problema, l'azienda ha richiesto allo studente di sviluppare un algoritmo di Proof of Work, ovvero un semplice algoritmo che fa lavorare in background la CPU del client, togliendogli per pochi secondi risorse da utilizzare (quali memoria in RAM, Mhz di calcolo, ecc.).
Apparentemente, se viene eseguito solo una volta l'algoritmo non genera problemi,

ma se una macchina prova a far partire l'algoritmo molte volte e in maniera ripetuta l'algoritmo comincia a mettere in difficoltà la CPU.
Per questo motivo, il Proof of Work è pensato per scoraggiare gli utenti malevoli ad attuare attacchi brute force.
Per sviluppare questo algoritmo si è deciso di utilizzare le funzioni crittografiche di hash. Tali funzioni sono una classe speciale delle funzioni di hash, che dispone di alcune proprietà che la rendono adatta all'uso in crittografia. Si tratta di un algoritmo matematico che mappa dei dati di lunghezza arbitraria (messaggio) in una stringa binaria di dimensione fissa chiamata valore di hash, ma spesso viene indicata anche con il termine inglese message digest (o semplicemente digest). Tale funzione di hash è progettata per essere unidirezionale (one-way), ovvero una funzione difficile da invertire. Il livello di crittografia scelto per il Proof of Work è il SHA-256. Nella figura 3.3 viene mostrato l'algoritmo per produrre l'hash per il Proof of Work.

```
export async function digestMessage(message) {
  const msgUint8 = new TextEncoder().encode(message); // encode as (utf-8) Uint8Array
  const hashBuffer = await crypto.subtle.digest("SHA-256", msgUint8); // hash the message
  const hashArray = Array.from(new Uint8Array(hashBuffer)); // convert buffer to byte array
  const hashHex = hashArray
    .map((b) => b.toString(16).padStart(2, "0"))
    .join(""); // convert bytes to hex string
  return hashHex;
}
```

Figura 3.3: Algoritmo del metodo digestMessage() per produrre l'hash in SHA-256.

Quando il CAPTCHA viene inizializzato, Il metodo `proofOfWork()`, come mostrato nella figura 3.4, viene invocato con una stringa arbitraria scelta dallo sviluppatore e passata come parametro. All'interno del metodo viene fatto partire un ciclo `while()` che ad ogni ciclo esegue tre compiti:

- Inserisce in coda alla stringa un contatore numerico incrementale;
- Prende la stringa con il contatore aggiornato ed invoca il metodo `digestMessage()` per convertire la stringa arbitraria in un hash;
- Controlla se l'hash fornito dal metodo `digestMessage()` produce una sottostringa desiderata dallo sviluppatore per interrompere il ciclo `while()`, determinando così che Proof of Work ha fatto il suo lavoro.

Questo ciclo viene ripetuto fino a che non viene prodotto l'hash desiderato e soltanto quando l'hash desiderato viene trovato che si può inserire i dati nella form del CAPTCHA per verificare se il test è superato oppure no. In generale, il ciclo `while()` dura all'incirca dall'1 ai 5 secondi.

3.4 Codifica finale sistema CAPTCHA

Avendo a disposizione la base client-side e il Proof of Work, in questa fase lo studente si è prodigato nell'adattamento della maschera scritta interamente client-side in un sistema che funzioni con una comunicazione client-server. Per motivi di codifica e testing, lo studente ha lavorato al prodotto CAPTCHA scrivendo codice tutto client-side, ma in

```
async proofOfWork(textToValidate) {
  // Setta la variabile _isRunning per indicare che il Proof Of Work è in esecuzione.
  this._isRunning = true;

  // Setting di variabili per l'algoritmo
  const test = "0000";
  let hashSubString = "";
  let index = -1;
  let textWithNumbers = textToValidate + index;

  // L'algoritmo di hashing viene invocato finchè non viene trovata la sottostringa corretta
  while (test != hashSubString) {
    textWithNumbers = textToValidate + ++index;
    hashSubString = await digestMessage(textWithNumbers).then((digestHex) =>
      digestHex.substring(0, test.length)
    );
  }

  return true;
}
```

Figura 3.4: Metodo di Proof of Work per scoraggiare gli attacchi brute force.

un contesto reale, avere tutto il codice (e di conseguenza, tutto il comportamento) del CAPTCHA rende il sistema molto eludibile perché un utente con intenzioni malevoli può facilmente manipolare i dati e il codice sul browser.
Per questo motivo, per rendere effettivamente utilizzabile il sistema CAPTCHA a livello aziendale è stato richiesto allo studente di sviluppare una parte backend[g]del sistema CAPTCHA, costituita dalla parte di generazione dei dati da fornire al test automatico, dalla parte di verifica degli input dell'utente e del Proof of Work da stanziare in un server a parte ed inoltre modificare la libreria che abbia al suo interno un'architettura che permetta una comunicazione client-server.
Infine, per permettere una facile compatibilità con i prodotti Zucchetti, è stato chiesto al tirocinante di sviluppare la parte backend con il linguaggio Java.
Per generare i dati da inserire nel test automatico, il server backend si serve della classe `ClockCAPTCHAJava` che ha al suo interno il metodo `generateData()` che si occupa sempre generare randomicamente, attraverso l'uso della classe `Random` fornito dal package `java.util.Random` di Java, l'orario da comprendere per il test di Turing, ma, a differenza del Proof of Concept, si serve di un oggetto di tipo `ClockImageGenerator` per generare l'immagine e quest'ultimo contiene un oggetto di tipo `ClockImageGenerationStrategy` che implementa l'algoritmo di generazione dell'immagine dal quale deriva al momento della consegna la strategia che utilizza gli oggetti di tipo `BufferedImage`.
Per la gestione dell'algoritmo di generazione, per creare codice mantenibile e facilmente modificabile per futuri aggiornamenti, sono stati usati i seguenti design pattern:

- **Dependency injection pattern**, utilizzato nel metodo `generateData()` per rimuovere la responsabilità di creazione dell'immagine dalla classe che la contiene. L'utilizzo del pattern favorisce la modularità del codice, facilitando la fase di test e migliorando la manutenibilità del prodotto;
- **Strategy pattern**, utilizzato sempre nel metodo `generateData()` per differenziare l'algoritmo di generazione dell'immagine dalla classe dedicata, rappresentata dall'oggetto `ClockImageGenerator`. In questo modo si facilita estremamente

l'introduzione di nuove tecnologie e relative metodologie al prodotto, rendendo più facile e gestibile anche la fase di test del codice;

- **Decorator pattern**, utilizzato da `ClockImageGenerator` per gestire le funzionalità di disturbo del sistema CAPTCHA. Il pattern utilizzato consente di concatenare diverse tipologie di disturbo, consentendo la creazione di immagini con grado e tipo di disturbo personalizzabile. Inoltre, il pattern aiuta la manutenibilità del codice, permettendo di effetturare modifiche mirate sull'algoritmo di generazione dell'immagine.

Per quanto riguarda il Proof of Work, la classe `ClockCAPTCHAJava`, oltre a generare randomicamente i dati del test, si occupa anche di generare randomicamente il testo del messaggio da dare in pasto all'algoritmo di Proof of Work e, assieme hai dati di inizializzazione del CAPTCHA, viene passato al client tramite file JSON. Il client, utilizzando uno script in AJAX, entrapola i dati dal file JSON e li passa all'algoritmo di Proof of Work per generare la soluzione da rispedire al server. A questo punto il server utilizza un handler per ricevere i dati dal client contenente l'hash calcolato dal client e lo confronta con l'hash calcoltato internamente per verificare la bontà della soluzione verificando che l'utente sia affidabile.

3.5 Studi sull'accessibilità

Ultimato lo sviluppo del sistema CAPTCHA basato sulla visualizzazione dell'orologio analogico, lo studente ha dedicato il restante tempo a disposizione dello stage (circa 3 settimane) per fare degli studi sull'accessibilità e per provare a sviluppare delle soluzioni che potesssero rendere il sistema CAPTCHA sviluppato dallo studente accessibile alla maggior parte degli utenti che interagiscono con il sistema CAPTCHA. I due studi fatti dallo studente riguardano:

- Il bilanciamento tra nitidezza del test CAPTCHA e il tempo trascorso;
- Alternative audio o sonore non vocali al test CAPTCHA per utenti con disabilità visiva.

3.5.1 Bilanciamento tra nitidezza e tempo trascorso

Per rendere davvero efficace un CAPTCHA ad immagini in termini di sicurezza bisogna apporre sopra l'immagine da visualizzare opportuni disturbatori per cercare di confondere le IA che tentano di leggere l'immagine oppure modificare l'immagine per renderla difficile da riconoscere. E' naturale pensare che questo approccio comporta il fatto che anche un utente umano può avere difficoltà a leggere l'immagine, causando una discriminazione involontaria dell'utente umano. Per sopperire a questo problema si è pensato di rendere il test CAPTCHA più nitido man mano che il tempo passa diminuendo il numero di disturbatori in base al tempo trascorso per superare il test.
Ad un primo momento può sembrare che questo sistema crei una falla nella sicurezza, ma, in base alle esigenze aziendali, il sistema CAPTCHA richiesto allo studente è stato concepito per bloccare gli attacchi brute force. Infatti, il grosso del lavoro per il controllo degli utenti malevoli lo fa il Proof of Work e l'immagine del test CAPTCHA comincerà a diventare più nitida soltanto dopo che il Proof of Work avrà completato il suo lavoro.
Per implementare questa feature è stato aggiunto un timer all'interno del backend

scritto in Java che si attiva non appena riceve la risposta del Proof of work dal client e ha validato che il Proof of Work è corretto. Seguendo degli intervalli prefissati il backend invia al client delle immagini dell'orologio con sempre meno disturbatori ad intervalli regolari sfruttando file in formato JSON. Il client non deve far altro che visualizzare l'immagine che riceve ogni volta dal backend.
Questa feature è molto buona per permettere allo sviluppatore che installa il CAPTCHA nell'applicazione da proteggere di inserire molti disturbatori sull'immagine senza svantaggiare l'utente umano, ma presenta delle complicanze da non sottovalutare:

- E' necessaria una connessione costante con il server per permettere la ricezione delle immagini più nitide. Un'interruzione della connessione tra client e server potrebbe compromettere la feature;
- Questa feature non porta nulla di nuovo sulla sfera dell'accessibilità e gli utenti con disabilità visiva vengono comunque discriminati dal CAPTCHA.

3.5.2 Il CAPTCHA vocale

Per cercare di rendere fruibile il sistema CAPTCHA ad utenti con disabilità visiva si è deciso di inserire un alternativa vocale al test basato sulla visualizzazione dell'immagine dell'orologio, ovvero si è inserito all'interno del sistema lato client un API che fornisce un Text-to-speech[g]che riceve dal sistema lato server il testo della soluzione del CAPTCHA e sintetizza un audio da far ascoltare all'utente per permettergli di inserire la risposta corretta.
Ovviamente per evitare che bot malevoli possano scrivere automaticamente la soluzione nella form del sistema, l'audio andrebbe distorto con suoni di sottofondo o white noise, ma da un articolo del w3.org [8] si evince che distorcere l'audio può creare problemi anche agli utenti con disabilità visive. Per evitare di dover distorcere l'audio del Text-to-speech, invece di fargli dire la soluzione vera e propria, si è optato per passargli un testo modificato in base alla soluzione che deve essere inserita nel test. Per esempio, se la soluzione da inserire è "11:50", invece di far dire al Text-to-speech "sono le undici e cinquanta minuto", possiamo fargli dire "sono le dodici meno dieci".
Per vedere se effettivamente questa soluzione è efficace sono stati condotti dei test sull'intelligenza artificiale ChatGPT di OperAI e sull'intelligenza artificiale Bing di Microsoft per capire se effettivamente le intelligenze artificiale non riescono a comprendere le soluzioni specificate nei modi precedentemente descritti. Su ogni intelligenza artificiale sono stati condotti 70 test, effettuati manualmente dallo studente e registrati su un file excel, registrando che gli test passava oppure no.
Dai risultati si evince che i 2/3 dei test vengono sbagliati dalle intelligenze artificiali, dimostrando che gli audio per come sono stati pensati per il CAPTCHA vocale è efficace contro le intelligenze artificiali che possono attaccare il test CAPTCHA, ma questi risultati sono validi soltanto con determinati presupposti:

- Nell'audio da produrre non si devono fare riferimento ai minuti da togliere, cioè bisogna sempre scrivere, per esempio, "Le 12 meno 5" anziché "Le 12 meno 5 minuti" (ovviamente nella form di immissione del risultato inseriremo un messaggio che dice che il formato della soluzione deve essere "hh:mm", dove "hh" sta per le ore e "mm" sta per i minuti);
- L'intelligenza artificiale sbaglia soltanto scegliendo un opportuno range di minuti da togliere o da aggiungere all'orario da indovinare (nella maggior parte dei test

che sono stati fatti l'intelligenza artificiale sbaglia solo quando si cerca di togliere o aggiungere numeri dall'1 al 10).

Nonostante i risultati trovati siano comunque buoni, il CAPTCHA vocale sviluppato rimane comunque inaccessibile per utenti con difficoltà cognitiva, dato che tali soggetti, pur riuscendo visivamente a leggere il quadrante, necessitano di un ulteriore aiuto per fare in modo che la loro mente (limitata da un deficit) riesca ad interpretare il suggerimento audio, rendendo perciò questa alternativa non fruibile per questa categoria di utenti. Inoltre bisogna tener conto che questa soluzione rimane valida e funziona solo con la lingua italiana. Anche i test fatti con ChatGPT e Bing sono stati fatti utilizzando frasi con la lingua italiana. Per utilizzare questo tipo di CAPTCHA con diverse lingue bisogna trovare soluzioni ad hoc pensate per una precisa lingua.

3.6 Verifica

Durante tutto il periodo di stage lo studente ha effettuato delle attività di verifica per accertarsi che durante le attività di codifica non siano stati introdotti errori nel codice. Le attività di verifica effettuate dal tirocinante si possono suddividere in due tipi:

- Attività di analisi statica;
- Attività di analisi dinamica.

3.6.1 Analisi statica

Questa attività utilizza tecniche che non richiedono l'esecuzione del software sviluppato dallo studente e si basa sulla di rilettura del codice scritto in fase di codifica utilizzando un approccio "walkthrough[g]". Durante il mio stage, l'analisi statica è avvenuta tramite rilettura del codice al raggiungimento di ogni traguardo significativo di sviluppo.

3.6.2 Analisi dinamica

Questa attività utilizza che richiedono l'esecuzione del software sviluppato dallo studente. Spesso si avvale di test progettati per essere ripetibili ed utilizzabili ogni volta che viene effettuata un modifica sul software o aggiunta una nuova feature o un nuovo metodo o classe. I test che ho effettuato durante il processo di codifica possono essere così catalogati:

- **Test di Unità**, ovvero i test di verifica sulle singole unità del prodotto sviluppato dallo studente;
- **Test di Integrazione**, ovvero i test di verifica sulle parti di software del prodotto per verificare che interagiscono bene tra di loro;
- **Test di Sistema**, ovvero la verifica del comportamento dinamico del sistema completo rispetto ai requisiti richiesti per il software.

3.7 Documentazione

Tra gli obiettivi assegnati allo studente i questo stage, è stato richiesto allo studente di redigere una documentazione sul lavoro svolto. Lo stage aziendale era organizzato

in modo tale che dopo aver superato una fase significativa del progetto, essa doveva essere documentata in un documento scritto dallo studente.
Durante tutto l'arco temporale dello stage aziendale, lo studente ha redatto i seguenti documenti:

- **Specifica tecnica**, dove lo studente spiega il funzionamento del sistema CAPTCHA e la composizione di tutte le sue parti;
- **Diario di Bordo**, dove lo studente spiega il lavoro che ha svolto durante il periodo di stage settimana per settimana;
- **Documentazione di test e risultati**, dove lo studente spiega i testi che ha svolto sul sistema CAPTCHA e sull'accessibilità riportando i risultati che ha trovato.

Tali documenti sono stati scritti utilizzando il Microsoft Word che ha permesso una veloce stesura di testi da parte dello studente e permette ai lettori di usufruire facilmente dei contenuti scritti nei documenti.

Capitolo 4

Conclusioni

Nel presente capitolo enuncio un'analisi a posteriori su tutta l'attività di stage evidenziando gli obiettivi che ho raggiunto, l'esperienza che ho maturato e le considerazioni sui risultati ottenuti.

4.1 Raggiungimento degli obiettivi

Il prodotto da sviluppare nello stage descritta da questa tesi è quello di sviluppare un sistema CAPTCHA che blocca e/o scoraggia gli attacchi brute force, costruendo dei test che siano accessbibili per utenti disabili. Dal punto di vista della sicurezza contro gli attacchi brute force, gli obiettivi posti dallo stage sono stati superati, producendo un prodotto in linea con le aspettative. Per quanto riguarda l'accessibilità possiamo dire che gli obiettivi non sono stati superati, in quanto non è possile dire con certezza che il sistema CAPTCHA prodotto sia accessibile a tutti gli utenti disabili.
Di seguito verranno elencati gli obiettivi che lo studente doveva raggiungere in questo stage:

- **ON1**: Sviluppo di un sistema CAPTCHA in HTML e JavaScript con algoritmo per contrastare gli attacchi *brute force*;
- **ON2**: Redazione documentazione dove viene spiegato il lavoro svolto dal tirocinante e i suoi risultati;
- **OD1**: Sviluppo di pratiche di accessibilità per rendere il CAPTCHA utilizzabile per utenti con disabilità visive;
- **OD2**: Utilizzo a livello avanzato del linguaggio di programmazione HTML e JavaScript;
- **OD3**: Utilizzo a livello avanzato del linguaggio di programmazione Java;
- **OF1**: Comprensione a livello avanzato dell'accessibilità dei siti web.

Di seguito verranno elencati gli obiettivi che lo studente che lo studente ha superato oppure no:

- **ON1** - Superato;

- **ON2** - Superato;
- **OD1** - Non superato;
- **OD2** - Superato;
- **OD3** - Superato;
- **OF1** - Superato;

Lo studente è riuscito a soddisfare tutti gli obiettivi obbligatori richiesti dall'azienda e a superare quasi tutti gli obiettivi desiderabili e facoltativi, riuscendo a sviluppare un sistema CAPTCHA funzionante per gli applicativi aziendali e riuscendo a comprendere a livello avanzato i linguaggi di programmazione e le tecnologie utilizzate durante lo stage. L'unico obiettivo non raggiunto è lo sviluppo delle pratiche di accessibilità per rendere il CAPTCHA utilizzabile per utenti con disabilità visive perché, nonostante le ricerche svolte e le alternative sviluppate, il CAPTCHA sviluppppato dallo studente continua a discriminare alcuni utenti umani, renendolo non fruibile a tutti i tipi di utenti.

4.2 Valutazione personale

Secondo una mia valutazione personale, ritengo che questo stage sia stato utile ad entrambi le parti.
Si vede che l'azienda Zucchetti S.p.A. è molto interessata al problema sull'accessibilità ed è interessante vedere che l'azienda investa tempo per cercare soluzioni per rendere i propri prodotti fruibili a tutti quanti i ti pi di utente. Nonostante il CAPTCHA sviluppato dallo studente non abbia portato i risultati sperati, rimane comunque utilizzabile dall'azienda che lo sfrutterà per prossimi sviluppi futuri nel campo dell'accessibilità.
Per quanto riguarda la mia persona, sono soddisfatto del lavoro svolto, soprattutto per la possibilità offerta dall'azienda di sfrutttare lo stage non solo per sviluppare prodotti utili a loro ma anche per effetuare ricerche in ambito dell'accessibilità. Inoltre, considero pienamente superati gli obiettivi personali che mi ero prefissato all'inizio di questo stage che si differenziano in:

- **Obiettivi professionali**: Durante lo stage ho potuto conoscere più approfonditamente i linguaggi di programmazione per lo sviluppo web quali HTML e Javascript, il linguaggio di programmazione orientato agli oggetti Java e vari ambienti di sviluppo (come Eclipse e VS Code). Ho potuto mettere in pratica le nozioni imparate durante il corso di laurea in Informatica, quali la programmazione orientata agli oggetti e le best practices dell'Ingegneria del Software;
- **Obiettivi personali**: Durante lo stage mi sono confrontato con persone che si occupano di sviluppo, imparando da loro come lavorare in questo ambiente. Ho potuto avere esperienze più tangibili in un contesto aziendale vero e proprio, accrescendo il mio bagaglio di esperienze. Infine, ho potuto migliorare le mie hard skills e soft skills.

4.3 Considerazioni sui risultati ottenuti

Il CAPTCHA basato sulla visualizzazione di un orologio e la sua alternativa vocale sono soluzioni efficaci per differenziare l'umano dal bot di un potenziale attaccante, però

bisogna tenere conto che un hacker per bypassare il CAPTCHA potrebbe servirsi delle CAPTCHA farm ed utilizzare della manodopera a basso costo fornita da lavoratori del terzo mondo per entrare illegalmente nell'applicazione protetta dal sistema CAPTCHA, perché è comunque un umano che prova a risolvere il test, quindi il sistema di protezione soffre a causa di questo problema.
Inoltre bisogna anche tener conto che le soluzioni fornite dallo studente sono soluzioni efficaci contro le tecnologie attualmente in uso nel mondo. Le intelligenze artificiali sono sempre in costante evoluzione e sicuramente in un futuro prossimo possono essere sviluppate delle IA ad hoc che possono superare i prodotti sviluppati dallo studente. Basti pensare che esiste già una IA capace di leggere le lancette di un orologio [7] all'interno di un immagine. Occorre quindi fare un lavoro continuo di aggiornamento dei livelli di sicurezza per stare sempre al passo con le nuove evoluzioni delle IA.
Per quanto riguarda l'accessibilità, a causa delle intelligenze artificali che riescono sempre di più a superare i test CAPTCHA e la discriminazione che i test portano nei confronti degli utenti con disabilità la comunità mondiale si stà sempre più chiedendo se sia ancora utile o meno utilizzare i CAPTCHA. Di fatto esistono già musei o pubbliche amministrazione che non utilizzano più questo tipo di protezione.
In ogni caso e la Zucchetti S.p.A. vuole ancora utilizzare i CAPTCHA come sistema di sicurezza contro i bot, è necessario poter presentare oltre al test di base una serie di alternative tutte pensate per sopperire ad tutte le disabilità che un utente può avere (che sia disabilità visive, cognitive, uditive, ecc.) oppure utilizzare dei CAPTCHA meno invasivi, cioè che non richiedano ad un utente di risolvere test o puzzle.

Appendice A

Glossario

Accessibilità L'accessibilità è la caratteristica di un dispositivo, di un servizio, di una risorsa o di un ambiente d'essere fruibile con facilità da una qualsiasi tipologia d'utente. Il concetto di design accessibile e la pratica dello sviluppo accessibile garantiscono sia l'"accesso diretto" (cioè senza assistenza) sia l'"accesso indiretto", cioè la compatibilità con le tecnologie assistive di una persona (per esempio, gli screen reader del computer).

AgID L'Agenzia per l'Italia Digitale è l'agenzia tecnica della Presidenza del Consiglio che ha il compito di garantire la realizzazione degli obiettivi dell'Agenda digitale italiana e contribuire alla diffusione dell'utilizzo delle tecnologie dell'informazione e della comunicazione, favorendo l'innovazione e la crescita economica.

AJAX In informatica AJAX, acronimo di Asynchronous JavaScript and XML, è un insieme di tecniche e metodologie di sviluppo software per la realizzazione di applicazioni web interattive (Rich Internet Application), basandosi su uno scambio di dati in background fra web browser e server, consentendo così l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente. AJAX è asincrona nel senso che i dati extra sono richiesti al server e caricati in background senza interferire con il comportamento della pagina esistente. Normalmente le funzioni richiamate sono scritte con il linguaggio JavaScript. Tuttavia, e a dispetto del nome, l'uso di JavaScript e di XML non è obbligatorio, come non è detto che le richieste di caricamento debbano essere necessariamente asincrone.

API In un programma informatico, con application programming interface (API), in italiano "interfaccia di programmazione dell'applicazione", si indica un insieme di procedure (in genere raggruppate per strumenti specifici) atte a risolvere uno specifico problema di comunicazione tra diversi computer o tra diversi software o tra diversi componenti di software; spesso tale termine designa le librerie software di un linguaggio di programmazione, sebbene più propriamente le API sono il metodo con cui le librerie vengono usate per sopperire ad uno specifico problema di scambio di informazioni.

Backend Il termine inglese back end (in sigla BE) (anche scritto e backend o back-end) in informatica denota la parte che permette l'effettivo funzionamento di un programma e l'effettivo funzionamento delle interazioni tra utente e programma. Questa parte si differenzia dalla parte front end (in sigla FE) (anche scritto

frontend o front-end) che denota la parte visibile all'utente di un programma e con cui egli può interagire (tipicamente un'interfaccia utente). Il front end, nella sua accezione più generale, è responsabile dell'acquisizione dei dati di ingresso e della loro elaborazione con modalità conformi a specifiche predefinite e invarianti, tali da renderli utilizzabili dal back end. Il collegamento del front end al back end è un caso particolare di interfaccia.

Bot Il bot (abbreviazione di robot) in terminologia informatica in generale è un programma che accede alla rete attraverso lo stesso tipo di canali utilizzati dagli utenti (per esempio che accede alle pagine Web, invia messaggi in una chat, si muove nei videogiochi, e così via). Programmi di questo tipo sono diffusi in relazione a molti diversi servizi in rete, con scopi vari, ma in genere legati all'automazione di compiti che sarebbero troppo gravosi o complessi per gli utenti.

BS 10012:2017 Lo standard BS 10012:2017, elaborato e promosso dall'Organizzazione britannica di standardizzazione (British Standards Institution), è uno standard che assicura alle aziende che richiedono il certificato l'adozione di un sistema affidabile per la gestione dei dati personali trattati, in conformità agli obblighi stabiliti dal Reg. UE 2016/679 (GDPR). E' uno strumento per garantire alla Direzione Aziendale (Titolare del trattamento) e a terzi (privati, enti di controllo, ecc.) l'adozione di policy ed istruzioni per il corretto trattamento dei dati.

C++ In informatica, C++ è un linguaggio di programmazione general purpose sviluppato in origine da Bjarne Stroustrup nei Bell Labs nel 1983 come evoluzione del linguaggio C inserendo la programmazione orientata agli oggetti; col tempo ha avuto notevoli evoluzioni, come l'introduzione dell'astrazione rispetto al tipo.

IA L'intelligenza artificiale (in sigla IA) è una disciplina che studia se e in che modo si possano realizzare sistemi informatici intelligenti in grado di simulare la capacità e il comportamento del pensiero umano.

IDE Un ambiente di sviluppo integrato (in inglese integrated development environment ovvero IDE), in informatica, è un software che, in fase di programmazione, supporta i programmatori nello sviluppo e debugging del codice sorgente di un programma. Spesso l'IDE aiuta lo sviluppatore segnalando errori di sintassi del codice direttamente in fase di scrittura, oltre a tutta una serie di strumenti e funzionalità di supporto alla fase stessa di sviluppo e debugging.

IT La tecnologia dell'informazione, in acronimo TI (in inglese information technology, in acronimo IT), è l'insieme dei metodi e delle tecnologie che vengono utilizzate in ambito pubblico, privato o aziendale per l'archiviazione, la trasmissione e l'elaborazione di dati e informazioni attraverso l'uso di reti (reti aziendali, internet ecc.), elaboratori (PC, server, mainframe ecc.) e attrezzature di telecomunicazione (datacenter, router, smartphone, tablet, GPS ecc.). In generale, hardware, software, e comunicazione digitale (ICT) sono i tre settori su cui vengono sviluppate le tecnologie IT che oggi sono impiegate in modo diffuso nei contesti sociali, commerciali ed economici di tutto il mondo.

ISAE 3402 L'International Standard on Assurance Engagements 3402 (ISAE 3402) è uno standard di garanzia internazionale che descrive impegni di Service Organization Control (SOC), che garantisce ai clienti di un organizzazione che i servizi offerti dalla suddetta organizzazione hanno adeguati controlli interni.

ISO/IEC 27001 Lo standard ISO/IEC 27001 è l'unica norma internazionale soggetta a verifica e certificabile che definisce i requisiti per un SGSI (Sistema di Gestione della Sicurezza delle Informazioni) ed è progettata per garantire la selezione di controlli di sicurezza adeguati e proporzionati. In questo modo è possibile proteggere le informazioni e dare fiducia agli stakeholder, in particolare ai propri clienti.

ISO/IEC 27017 Lo standard ISO/IEC 27017 rientra tra gli standard della serie ISO/IEC 27001 e definisce controlli avanzati sia per fornitori, sia per i clienti di servizi cloud. Chiarisce ruoli e responsabilità dei diversi attori in ambito cloud con l'obiettivo di garantire che i dati conservati in cloud computing siano sicuri e protetti.

ISO/IEC 27018 Lo standard ISO/IEC 27018 - Codice di condotta per la protezione delle PII (Personally Identifiable information) nei servizi di public cloud per i cloud provider- è una linea guida per i fornitori di servizi cloud pubblici che vogliono migliorare la gestione dei dati personali. L'obiettivo di questo standard è quello di fornire una modalità strutturata, basata sul privacy by design, per far fronte alle principali questioni giuridiche, sia di natura legale che contrattuale, legate alla gestione dei dati personali in infrastrutture informatiche distribuite seguendo il modello del cloud pubblico.

ISO/IEC 27701 Lo standard ISO/IEC 27701 rientra tra gli standard della serie ISO/IEC 27001 e definisce controlli avanzati per la gestione della privacy e delle PII (Personally Identifiable Information). Chiarisce come "migliorare" (adattare ed estendere) un sistema ISO / IEC 27001 nel contesto sia dei rischi per la sicurezza delle informazioni sia dei rischi connessi al trattamento delle PII, compresi i rischi per i principi delle PII.

JDK In informatica, il JDK (acronimo di Java Development Kit) è l'insieme degli strumenti per sviluppare programmi da parte dei programmatori Java. È un prodotto della Oracle Corporation, e fin dall'introduzione di Java è sempre stato l'ambiente di sviluppo più utilizzato dai programmatori Java soprattutto per applicazioni desktop. Per applicazioni più complesse (es. applicazioni web) oggi sempre più spesso si utilizzano per lo sviluppo ed esecuzione programmi IDE a cui è possibile agganciare la JRE.

JVM In informatica, la macchina virtuale Java (detta anche Java Virtual Machine o JVM) è il componente software della piattaforma Java che esegue i programmi tradotti in bytecode dopo la prima fase di compilazione in bytecode (tra i linguaggi di programmazione che possono essere tradotti in bytecode troviamo Java, Groovy, Clojure, Scala ed Eta).

Open Source Con Open Source (in italiano sorgente aperto), in informatica, si indica un software distribuito sotto i termini di una licenza open source, che ne concede lo studio, l'utilizzo, la modifica e la redistribuzione.

PHP Il PHP (acronimo ricorsivo di "PHP: Hypertext Preprocessor", preprocessore di ipertesti; originariamente acronimo di "Personal Home Page") è un linguaggio di scripting interpretato, originariamente concepito per la programmazione di pagine web dinamiche. L'interprete PHP è un software libero distribuito sotto la PHP License. Attualmente è principalmente utilizzato per sviluppare applicazioni

web lato server, ma può essere usato anche per scrivere script a riga di comando o applicazioni stand-alone con interfaccia grafica.

Python Python è un linguaggio di programmazione ad alto livello, orientato a oggetti, adatto, tra gli altri usi, a sviluppare applicazioni distribuite, scripting, computazione numerica e system testing.

Reverse engineering Con reverse engineering (in italiano "ingegneria inversa", "ingegnerizzazione inversa") si indica un anglicismo che indica quell'insieme di analisi delle funzioni, degli impieghi, della collocazione, dell'aspetto progettuale, geometrico e materiale di un manufatto o di un oggetto che è stato rinvenuto (ad esempio un reperto, un dispositivo, componente elettrico, un meccanismo, software). Il fine può essere quello di produrre un altro oggetto che abbia un funzionamento analogo o migliore, o più adatto al contesto in cui ci si trova (fitting); un altro fine può essere, quello di tentare di realizzare un secondo oggetto in grado di interfacciarsi con l'originale.

Stakeholder In economia lo stakeholder (inglese, lett. "portatore di bastone") o "portatore di interesse" è genericamente qualsiasi soggetto o gruppo coinvolto in una qualsiasi iniziativa economica, una società o altro progetto, e in generale con interessi legati all'esecuzione o dall'andamento dell'iniziativa stessa. Fanno dunque parte di tale insieme clienti, fornitori, finanziatori (es. banche e azionisti, o shareholder), collaboratori, dipendenti, ma anche gruppi di interesse locali o esterni, come i residenti di aree limitrofe a un'azienda e le istituzioni statali relative all'amministrazione locale.

Text-to-speech La sintesi vocale (in inglese speech synthesis) è la tecnica per la riproduzione artificiale della voce umana. Un sistema usato per questo scopo è detto sintetizzatore vocale e può essere realizzato tramite software o via hardware. I sistemi di sintesi vocale sono noti anche come sistemi text-to-speech (TTS) (in italiano, "testo a voce") per la loro possibilità di convertire il testo in parlato. Esistono inoltre sistemi in grado di convertire simboli fonetici in parlato. Il processo inverso è chiamato riconoscimento vocale.

UML In ingegneria del software, l'UML (dall'inglese, acronimo di Unified Modeling Language) 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.

UNI EN ISO 9001:2015 Lo standard UNI EN ISO 9001:2015 è una norma riconosciuta a livello internazionale per la creazione, implementazione e gestione di un Sistema di Gestione della Qualità per qualsiasi azienda. È pensata per essere utilizzata da organizzazioni di qualunque dimensione o settore, nonché da qualsiasi azienda.

Walkthrough In ingegneria del software, il walkthrough (letteralmente procedura dettagliata, dall'inglese walk through, "attraversare") è una tecnica di analisi statica che viene utilizzata per rilevare la presenza di difetti attraverso lettura critica ad ampio spettro di un prodotto in esame. Viene fatto un esame privo di assunzioni o presupposti del codice e dei documenti del prodotto in esame. L'esame del codice viene fatto percorrendolo simulendone possibili esecuzioni

e l'esame dei documenti viene fatto studiandone ogni parte come farebbe un compilatore.

XML In informatica, l'XML (sigla di eXtensible Markup Language, lett. "linguaggio di marcatura estendibile") è un metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo. Il nome indica che si tratta di un linguaggio estendibile, in quanto permette di creare tag personalizzati e costituisce il tentativo di produrre una versione semplificata dello Standard Generalized Markup Language (SGML), che consente di definire nuovi linguaggi di markup.

Appendice B

Bibliografia

[1] *Da studio di commercialisti ad azienda software da 1 miliardo di fatturato: la parabola di successo della famiglia Zucchetti.* Giugno 2021. url: https://forbes.it/2021/06/15/da-studio-di-commercialisti-ad-azienda-software-da-1-miliardo-di-fatturato-la-parabola-di-successo-della-famiglia-zucchetti/

[2] *Definizione di CAPTCHA*. Maggio 2023. url: https://it.wikipedia.org/wiki/CAPTCHA

[3] *CAPTCHA: Umano o Sovrumano - Capitolato d'appalto per il progetto didattico di Ingegneria del Software (Anno accademico 2022/2023).* Ottobre 2022. url: https://www.math.unipd.it/ tullio/IS-1/2022/Progetto/C1.pdf

[4] *A generative vision model that trains with high data efficiency and breaks text-based CAPTCHAs.* Ottobre 2017. url: https://www.science.org/doi/10.1126/science.aag2612

[5] *Definizione di Proof of Work.* Agosto 2023. url: https://it.wikipedia.org/wiki/Proof-of-work

[6] *L'evoluzione di Google reCaptcha nel miglioramento dell'inclusività delle persone con disabilità visiva.* Settembre 2021. url: https://thesis.unipd.it/handle/20.500.12608/4133

[7] *It's About Time: Analog Clock Reading in the Wild.* Novembre 2021. url: https://arxiv.org/abs/2111.09162

[8] *Inaccessibility of CAPTCHA: Alternatives to Visual Turing Tests on the Web.* Dicembre 2021. url: https://www.w3.org/TR/turingtest/