|  |  |
| --- | --- |
| Pictochat | |
|  | |
| Studente/i | Relatore |
| Carlo Pezzotti | Giacomo Poretti |
| Correlatore |
| Giacomo Poretti |
| Committente |
| Fondazione MindTime Sivia Valsangiacomo |
| Corso di laurea | Codice progetto |
| Ingegneria Informatica | - |
| Anno |  |
| 2024/2025 |  |

|  |  |
| --- | --- |
| Data |  |

03.12.2024

|  |  |
| --- | --- |
|  |  |

Dedica / Ringraziamenti (opzionale)

Un’eventuale dedica o eventuali ringraziamenti vanno inseriti subito dopo la copertina, su una pagina intera e in genere con uno stile differenziato rispetto al resto del testo, usando ad es. dei caratteri particolari come il **corsivo (*italic*)** e allineando il testo a destra.

**Attenzione**: di norma **non vanno ringraziate** le persone direttamente coinvolte nel progetto, quali i Docenti, i colleghi di progetto (per i progetti con più partecipanti), il rappresentante del Committente. Per queste persone un ringraziamento può essere espresso solo in casi particolari di impegno eccezionale che si vuole riconoscere.

Esempi:

*Questo libro è dedicato a mia moglie*

*Anna e a mia figlia Lucia.*

*Senza il loro sostegno, la loro comprensione*

*e il loro aiuto non lo avrei mai scritto.*

*Vorrei qui ringraziare la mia fidanzata Paola per il suo continuo sostegno*

*durante lo svolgimento del progetto.*

Indice generale

Contents

[Abstract 5](#_Toc184311254)

[Italiano 5](#_Toc184311255)

[English 6](#_Toc184311256)

[Capitolo 1 7](#_Toc184311257)

[Progetto assegnato 7](#_Toc184311258)

[Descrizione 7](#_Toc184311259)

[Compiti 7](#_Toc184311260)

[Obiettivi 8](#_Toc184311261)

[Tecnologie 8](#_Toc184311262)

[Capitolo 2 9](#_Toc184311263)

[Introduzione 9](#_Toc184311264)

[Capitolo 3 11](#_Toc184311265)

[Motivazione e Contesto 11](#_Toc184311266)

[Capitolo 4 13](#_Toc184311267)

[Obiettivo 13](#_Toc184311268)

[Obiettivi principali 13](#_Toc184311269)

[Componenti principali 14](#_Toc184311270)

[Obiettivo finale 14](#_Toc184311271)

[Capitolo 5 15](#_Toc184311272)

[Terminologia 15](#_Toc184311273)

[Capitolo 6 16](#_Toc184311274)

[Problema 16](#_Toc184311275)

[Barriere nella comunicazione a distanza 16](#_Toc184311276)

[Le principali sfide riscontrate 17](#_Toc184311277)

[La visione di Pictochat 17](#_Toc184311278)

[Capitolo 7 18](#_Toc184311279)

[Analisi dei requisiti 18](#_Toc184311280)

[Requisiti funzionali: 18](#_Toc184311281)

[Requisiti Non Funzionali 19](#_Toc184311282)

[Capitolo 8 20](#_Toc184311283)

[Approccio al Problema 20](#_Toc184311284)

[Linguaggi di Programmazione 20](#_Toc184311285)

[Strumenti Utilizzati 20](#_Toc184311286)

[Librerie usate 21](#_Toc184311287)

[Capitolo 10 22](#_Toc184311288)

[Architettura 22](#_Toc184311289)

[1. Panoramica 22](#_Toc184311290)

[2. Struttura dell'Applicazione 22](#_Toc184311291)

[2. Gestione delle Rotte 23](#_Toc184311292)

[1. Gestione dello Stato e del Contesto 23](#_Toc184311293)

[3. Protezione delle Rotte 23](#_Toc184311294)

[4. Gestione della UI 24](#_Toc184311295)

[5. Logica di Business e API 25](#_Toc184311296)

[6. Vantaggi dell'Architettura 25](#_Toc184311297)

# **Abstract**

## Italiano

Questo progetto presenta lo sviluppo di un'applicazione web e mobile (PWA) innovativa, progettata per agevolare la comunicazione di persone con difficoltà cognitive attraverso l'uso di pittogrammi. Basata sull'infrastruttura di Telegram, l'app consente agli utenti di interagire in modo semplice ed efficace, trasformando il testo in immagini e viceversa. L'applicazione è stata realizzata eliminando completamente il server per ridurre i costi di manutenzione, portando l'intera logica sul client e implementando un sistema di deployment automatico. Sono state sviluppate funzionalità avanzate, tra cui l'integrazione con l'API di ARASAAC per l'utilizzo di pittogrammi, la personalizzazione dei contenuti visuali e un'interfaccia ottimizzata per la creazione e la gestione di messaggi testuali e visivi.

Il progetto mira a essere uno strumento pratico per il miglioramento della qualità della vita, consentendo a ragazzi e adulti di comunicare in modo inclusivo. È stato strutturato con un approccio iterativo, consentendo test rapidi e il recepimento di feedback da parte della Fondazione MindTime, partner del progetto.

Le tecnologie utilizzate includono React con TypeScript per lo sviluppo dell'interfaccia e la sicurezza delegata all'infrastruttura di Telegram. Il lavoro futuro prevede l'integrazione di modelli di intelligenza artificiale per la predizione delle risposte e per facilitare la scrittura, completando una piattaforma già avanzata e robusta.

## English

This project presents the development of an innovative web and mobile application (PWA) designed to facilitate communication for people with cognitive difficulties through the use of pictograms. Based on Telegram's infrastructure, the app allows users to interact easily and effectively by turning text into images and vice versa. The app was built by completely eliminating the server to reduce maintenance costs, bringing all logic to the client and implementing an automatic deployment system. Advanced features were developed, including integration with ARASAAC's API for using pictograms, customization of visual content, and an optimized interface for creating and managing text and visual messages.

The project aims to be a practical tool for improving quality of life by enabling children and adults to communicate inclusively. It was structured with an iterative approach, allowing for rapid testing and the receipt of feedback from the MindTime Foundation, a partner in the project.

Technologies used include React with TypeScript for interface development and security delegated to Telegram's infrastructure. Future work includes integrating artificial intelligence models to predict responses and facilitate writing, complementing an already advanced and robust platform.

# Capitolo 1

## Progetto assegnato

Il progetto assegnato si propone di sviluppare un'applicazione di messaggistica innovativa che utilizzi pittogrammi per facilitare la comunicazione di persone con difficoltà cognitive. Basandosi sull'infrastruttura sicura di Telegram, il progetto mira a offrire un'interfaccia intuitiva e funzionalità personalizzabili per scrivere e leggere messaggi attraverso immagini, rendendo la comunicazione più accessibile.

### Descrizione

L'applicazione sviluppata è una Progressive Web App (PWA) progettata per l'interazione basata su pittogrammi. Essa consente agli utenti di:

* Comporre messaggi utilizzando una combinazione di testo e immagini.
* Visualizzare i messaggi sia in formato testuale che attraverso pittogrammi.
* Personalizzare i contenuti visivi tramite caricamenti di immagini e configurazioni specifiche.
* Usare un sistema avanzato di decodifica per tradurre automaticamente il testo in pittogrammi.

L'obiettivo principale è offrire una soluzione inclusiva per persone con difficoltà cognitive, garantendo facilità d'uso e sicurezza, con la logica completamente implementata lato client per minimizzare i costi infrastrutturali.

### Compiti

Per la realizzazione del progetto sono stati assegnati i seguenti compiti principali:

1. **Trasferire la logica sul client:** eliminazione del server per ridurre i costi di manutenzione.
2. **Integrazione con API ARASAAC:** consentire l'accesso a una libreria di pittogrammi per la traduzione del testo.
3. **Ottimizzazione delle pagine:**
   * Contatti e messaggi con design reattivo e accessibile.
   * Pagina di impostazioni e login per configurazioni personalizzate.
4. **Implementazione di nuove funzionalità:**
   * Creazione di frasi con pittogrammi.
   * Traduzione automatica del testo in pittogrammi.
   * Caricamento di pittogrammi personalizzati.
5. **Notifiche in-app:** per una migliore esperienza utente.
6. **Caching:** gestione avanzata delle immagini per migliorare le performance.
7. **Ai:** Studiare eventuali soluzioni con l’ausiglio dell’ai

### Obiettivi

Gli obiettivi del progetto includono:

1. **Testing rapido e iterativo:** Deployare una versione funzionante dell'app per consentire alla Fondazione MindTime di testare le funzionalità e fornire feedback.
2. **Zero costi di manutenzione:** Rimozione completa del server con logica basata esclusivamente sul client.
3. **Facilità d'uso e inclusività:** Implementare un'interfaccia utente intuitiva e personalizzabile per adattarsi alle esigenze di utenti con difficoltà cognitive.
4. **Predizione intelligente:** Integrare intelligenza artificiale per suggerire risposte e completamenti di frasi in futuro.

### Tecnologie

Il progetto utilizza un'ampia gamma di tecnologie, con particolare attenzione a quelle necessarie per un frontend performante e scalabile.

1. **Librerie principali:**
   * **React**: Framework base per lo sviluppo dell'interfaccia utente.
   * **TypeScript**: Per un codice tipizzato e robusto.
   * **React-Router-Dom**: Gestione delle route per una navigazione fluida.
   * **Ant Design**: Libreria di componenti per un design moderno e accessibile.
   * **Framer Motion**: Per animazioni fluide e interattive.
   * **React-Infinite-Scroll-Component**: Per caricare dinamicamente i contenuti dei messaggi e contatti.
2. **Funzionalità di comunicazione e gestione dati:**
   * **telegram**: Per interagire con l'API di Telegram e gestire messaggi e contatti.
   * **flatted**: Per serializzare e deserializzare strutture JSON complesse.
   * **Buffer e Stream**: Per ottimizzare la gestione dei dati lato client.
3. **Gestione dello sviluppo e deploy:**
   * **GH-Pages**: Per il deployment automatico su GitHub Pages.
   * **React-Scripts**: Per configurare e gestire il progetto con impostazioni predefinite.
4. **Altre librerie rilevanti:**
   * **React-Phone-Number-Input**: Per gestire input numerici, come il login o i contatti.
   * **TS-Loader**: Per integrare TypeScript nella pipeline di sviluppo.

# Capitolo 2

## Introduzione

Questa tesi si concentra sull’implementazione di un’applicazione di messaggistica innovativa progettata per facilitare la comunicazione di persone con difficoltà cognitive, utilizzando un approccio basato su pittogrammi. Il lavoro affronta le varie fasi dello sviluppo del progetto, dalla definizione dei requisiti iniziali fino alla valutazione dei risultati ottenuti, con un'attenzione particolare all'inclusività e alla sostenibilità del sistema.

Il percorso di sviluppo viene descritto attraverso una struttura articolata in capitoli, ciascuno dedicato a un aspetto specifico del progetto. Si parte dal capitolo "Motivazione e contesto", che esplora le ragioni per cui è stata scelta questa tematica, evidenziando il bisogno sociale e tecnologico di un’applicazione di questo tipo. Viene inoltre fornito uno sguardo al contesto in cui il progetto si colloca, con un’analisi delle esigenze degli utenti target.

Nel capitolo "Obiettivo" vengono definiti i traguardi che si intende raggiungere, enfatizzando l’importanza di creare una piattaforma accessibile, sostenibile e facile da utilizzare, capace di rispondere alle necessità di comunicazione di ragazzi e adulti con difficoltà cognitive.

Il capitolo "Terminologia" introduce i principali concetti e termini utilizzati, tra cui il significato di pittogrammi, la logica delle Progressive Web App (PWA) e la gestione della comunicazione tramite Telegram, per garantire una comprensione chiara e condivisa dei contenuti.

Segue il capitolo "Problema", che descrive le difficoltà affrontate dalle persone con difficoltà cognitive nella comunicazione tradizionale, e come queste possano essere mitigate attraverso un’applicazione costruita su misura. Si evidenziano, inoltre, le sfide tecniche affrontate nello sviluppo, come la traduzione automatica del testo in immagini e la gestione delle risorse lato client.

Nel capitolo "Analisi dei requisiti" si identificano le funzionalità chiave e gli obiettivi tecnici dell’applicazione, come la completa gestione client-side, la personalizzazione delle immagini e l’integrazione con API dedicate ai pittogrammi.

Il capitolo "Stato dell’arte" offre una panoramica delle tecnologie e delle soluzioni esistenti nel campo della comunicazione assistita, sottolineando l’unicità dell’approccio adottato per questo progetto, che si distingue per la combinazione di Telegram, React e il supporto ai pittogrammi.

La metodologia adottata viene approfondita nel capitolo "Approccio al problema", in cui si descrivono le fasi di sviluppo e i metodi iterativi che hanno permesso di rispondere alle esigenze degli utenti e di implementare nuove funzionalità in modo progressivo.

Il capitolo "Architettura" fornisce una descrizione dettagliata della struttura del sistema, delineando il flusso di dati, le interazioni tra le componenti e il ruolo centrale della logica client-side.

I dettagli pratici dello sviluppo vengono illustrati nel capitolo "Implementazione", che descrive le soluzioni adottate per superare i limiti tecnici e realizzare un’applicazione stabile e scalabile.

La verifica e la validazione del sistema sono trattate nel capitolo "Test", con una descrizione delle procedure adottate per garantire la qualità e l’affidabilità del prodotto.

Il capitolo "Risultati" mostra i principali successi del progetto, evidenziando il feedback positivo ricevuto dai test preliminari e il valore aggiunto dell’app per la Fondazione MindTime.

Infine, nel capitolo "Conclusioni" si sintetizzano i risultati del lavoro, si discutono le implicazioni sociali e tecnologiche dell’applicazione e si suggeriscono possibili sviluppi futuri, come l’integrazione di sistemi di intelligenza artificiale per la predizione di risposte e frasi.

# Capitolo 3

## Motivazione e Contesto

In questo capitolo vengono illustrate le motivazioni alla base del progetto di tesi e il contesto in cui è stato sviluppato. Prima di analizzare il problema specifico affrontato, è fondamentale collocare il lavoro all'interno del quadro accademico e sociale che ha guidato lo sviluppo dell'applicazione.

Durante l’ultimo semestre del corso di laurea in Ingegneria Informatica presso l’istituto universitario, ogni studente è chiamato a realizzare un progetto di tesi, con l’obiettivo di mettere in pratica le competenze acquisite durante il percorso di studi. Questo lavoro rappresenta una sfida personale e tecnica, con l’opportunità di contribuire a una problematica reale e di proporre soluzioni innovative.

La scelta di sviluppare un’applicazione di messaggistica basata su pittogrammi è stata motivata da una combinazione di interessi personali e da un’esigenza concreta emersa nel contesto sociale. Il progetto è stato intrapreso in collaborazione con la **Fondazione MindTime**, un ente che si occupa di supportare individui con difficoltà cognitive e le loro famiglie. Questo contesto ha fornito una chiara direzione per la progettazione dell’applicazione, ponendo l’inclusività e la semplicità d’uso al centro dell’intero lavoro.

**Motivazioni personali e accademiche**

L’interesse per il progetto è nato dal desiderio di esplorare le applicazioni tecnologiche volte a migliorare la qualità della vita delle persone. Durante il percorso accademico, è emerso come la tecnologia possa essere uno strumento potente per abbattere barriere e creare opportunità per chi si trova in condizioni di svantaggio. Questo progetto rappresenta quindi non solo un esercizio tecnico, ma anche un contributo significativo al miglioramento della comunicazione per utenti con difficoltà cognitive.

Dal punto di vista accademico, la realizzazione di un’applicazione innovativa come questa ha permesso di mettere in pratica conoscenze acquisite in ambiti quali:

* **Sviluppo di applicazioni web e mobile**: sfruttando framework moderni come React.
* **Gestione della sicurezza**: integrando la piattaforma Telegram per garantire privacy e affidabilità.
* **Sviluppo front-end e design user-centered**: creando un’interfaccia intuitiva e accessibile.

**Contesto sociale e tecnologico**

La comunicazione è una componente essenziale della vita quotidiana. Tuttavia, per molte persone con difficoltà cognitive, gli strumenti di comunicazione convenzionali risultano poco accessibili o difficili da utilizzare. In questo contesto, le applicazioni che integrano soluzioni visive come i pittogrammi possono fare una grande differenza, offrendo un mezzo semplice e intuitivo per interagire con familiari, amici o educatori.

Dal punto di vista tecnologico, il progetto si colloca in un’area in cui convergono diverse tendenze innovative:

1. **Progressive Web Apps (PWA)**: garantendo accesso cross-platform senza necessità di installazione.
2. **Integrazione con API**: come quelle fornite da ARASAAC per l’utilizzo di pittogrammi di alta qualità.
3. **Eliminazione del server**: trasferendo tutta la logica al lato client per ridurre i costi di manutenzione.

Questi elementi rendono l’applicazione unica nel suo genere, rispondendo a una necessità specifica che non trova al momento valide alternative nel mercato.

**Collaborazione con la Fondazione MindTime**

Il progetto è stato sviluppato in collaborazione con la Fondazione MindTime, che ha svolto un ruolo cruciale nell’identificazione dei bisogni degli utenti finali. La fondazione ha fornito feedback durante le varie fasi dello sviluppo, contribuendo a definire le funzionalità essenziali e le priorità del progetto. Questa collaborazione ha garantito che il prodotto rispondesse concretamente alle esigenze degli utenti, valorizzando la componente sociale del lavoro.

In conclusione, il progetto nasce da una combinazione di motivazioni personali, accademiche e sociali, con l’obiettivo di creare un’applicazione che possa avere un impatto reale sulla vita delle persone. Lavorare in un contesto concreto come quello della Fondazione MindTime ha offerto una prospettiva pratica e mirata, rendendo il progetto significativo e altamente rilevante.

# Capitolo 4

## Obiettivo

L’obiettivo di questo lavoro di tesi è la progettazione e lo sviluppo di un’applicazione web e mobile (PWA) che consenta la comunicazione tramite pittogrammi. Questa soluzione è pensata principalmente per persone con difficoltà cognitive, offrendo loro uno strumento semplice ed efficace per interagire con i propri cari, educatori o amici.

L’applicazione deve integrare diverse funzionalità che permettano sia la scrittura che la lettura di messaggi attraverso pittogrammi, rendendo la comunicazione più accessibile e inclusiva. La progettazione segue i principi di usabilità, modularità e scalabilità, garantendo un’esperienza utente intuitiva e una facile manutenzione.

### Obiettivi principali

1. **Eliminazione del server**  
   La prima priorità è stata la rimozione del server, con la conseguente migrazione della logica al client. Questa scelta è stata dettata dalla necessità di ridurre i costi di manutenzione e di semplificare l’architettura del sistema, garantendo al contempo alte prestazioni.
2. **Scrittura e lettura di messaggi tramite pittogrammi**
   * Implementare un sistema che consenta agli utenti di comporre messaggi utilizzando esclusivamente pittogrammi.
   * Integrare un meccanismo per la conversione automatica del testo in pittogrammi, facilitando la lettura anche per chi non è in grado di comprendere il testo scritto.
3. **Personalizzazione dei pittogrammi**
   * Offrire la possibilità agli utenti, in particolare ai genitori, di caricare pittogrammi personalizzati e di associarli a specifici soggetti o oggetti, aumentando la flessibilità e l’adattabilità dell’applicazione.
4. **Predizione intelligente tramite intelligenza artificiale**
   * Sviluppare un sistema basato su AI per la predizione delle risposte alle domande ricevute.
   * Integrare un meccanismo per suggerire pittogrammi o completamenti automatici durante la composizione di frasi.
5. **Integrazione con piattaforme esistenti**
   * Utilizzare le API di Telegram per sfruttare la sua infrastruttura di sicurezza e autenticazione, riducendo i costi di sviluppo e manutenzione.
   * Integrare l’API di ARASAAC per l’accesso a un vasto repertorio di pittogrammi di alta qualità.
6. **Design inclusivo e intuitivo**
   * Progettare un’interfaccia utente che sia chiara e accessibile.
   * Fornire una modalità di visualizzazione combinata (testo + immagini) per facilitare la comprensione dei messaggi.

### Componenti principali

1. **Sistema per la gestione dei messaggi**
   * Pagine ottimizzate per la visualizzazione dei contatti e dei messaggi.
   * Meccanismi per rispondere a domande tramite pittogrammi e comporre frasi esclusivamente visive.
2. **Pagine principali dell’applicazione**
   * Pagina di login, per autenticarsi tramite Telegram.
   * Pagina di impostazioni, che consente la configurazione delle preferenze utente e la gestione dei pittogrammi personalizzati.
3. **Caching e ottimizzazione**
   * Implementare un sistema di caching per ridurre i tempi di caricamento delle immagini e ottimizzare le prestazioni dell’applicazione.
4. **Testing e feedback continuo**
   * Garantire una collaborazione costante con la **Fondazione MindTime** per testare l’applicazione e ricevere suggerimenti sui miglioramenti da apportare.

### Obiettivo finale

Il risultato atteso è il rilascio di una Progressive Web App funzionante che sia immediatamente utilizzabile dalla Fondazione MindTime per test interni. Questo garantirà una raccolta tempestiva di feedback, consentendo iterazioni rapide e miglioramenti basati su esperienze reali. L’obiettivo ultimo è creare uno strumento che renda la comunicazione più accessibile, favorendo l’inclusione sociale e abbattendo le barriere cognitive.

# Capitolo 5

## Terminologia

# Capitolo 6

## Problema

La comunicazione è un aspetto fondamentale della vita sociale e personale di ogni individuo, ma per molte persone con difficoltà cognitive rappresenta una sfida quotidiana. Questi utenti spesso faticano a esprimere i propri pensieri o a rispondere in modo adeguato a domande semplici. La complessità del processo comunicativo diventa ancora più evidente quando queste persone cercano di interagire con altri utenti affetti da problematiche simili.

Nella pratica della **Fondazione MindTime**, è stato osservato che alla domanda "Come stai?" molti utenti rispondono con un'emoji che può risultare ambigua o completamente scollegata dal contesto. Ad esempio, alla domanda "Cosa hai fatto ieri?", alcuni rispondono con un’emoji di una macchina di Formula 1, mentre altri evitano del tutto di rispondere, incapaci di trovare rapidamente un simbolo adatto alle loro necessità. Questa difficoltà non solo crea incomprensioni, ma rappresenta un significativo blocco nella comunicazione.

Un problema ancora più evidente emerge quando due persone con difficoltà comunicative tentano di dialogare tra loro. Attualmente, questo tipo di comunicazione è mediato dai genitori o dai tutori, seguendo uno schema inefficiente:

* Ragazzo → Genitore
* Genitore → Genitore
* Genitore → Ragazzo

Questo processo non solo rallenta lo scambio di informazioni, ma riduce significativamente l’autonomia degli utenti, limitando il loro senso di indipendenza e privacy. In molti casi, questo sistema di intermediazione porta anche a una distorsione o perdita parziale del messaggio originale, aggravando ulteriormente i problemi di comprensione.

## Barriere nella comunicazione a distanza

Ad oggi, non esiste una soluzione efficace che permetta a persone con difficoltà cognitive di comunicare correttamente e in modo autonomo attraverso piattaforme digitali. Le applicazioni esistenti, anche quelle che includono emoji o pittogrammi, non offrono strumenti sufficienti per costruire frasi complesse o per comprendere risposte contestualmente rilevanti. Questo limita drasticamente la possibilità di comunicare correttamente a distanza, un aspetto sempre più rilevante nel mondo digitale odierno.

## Le principali sfide riscontrate

1. **Ambiguità nelle risposte**  
   Gli utenti tendono a scegliere emoji non appropriate o scollegate dal contesto della domanda, causando fraintendimenti e compromettendo la qualità dello scambio comunicativo.
2. **Difficoltà di ricerca e selezione delle emoji**  
   Il tempo necessario per trovare un’emoji o un simbolo adatto spesso scoraggia gli utenti dal rispondere, interrompendo completamente la comunicazione.
3. **Comunicazione mediata**  
   La necessità di coinvolgere terze parti (genitori o tutori) nelle conversazioni riduce l’autonomia e l’efficacia della comunicazione, specialmente in contesti in cui la privacy è importante.
4. **Mancanza di strumenti dedicati**  
   Non esistono applicazioni progettate specificamente per risolvere questi problemi, lasciando un vuoto nel panorama tecnologico per la comunicazione assistita. Questo crea una barriera significativa per le persone con difficoltà cognitive, che non hanno strumenti per comunicare autonomamente, specialmente in situazioni a distanza.

## La visione di Pictochat

Pictochat mira a colmare questa lacuna offrendo una piattaforma innovativa che rende possibile la comunicazione diretta tra utenti con difficoltà cognitive. Grazie a funzionalità come la composizione automatica di risposte basate su pittogrammi e la personalizzazione di simboli specifici, Pictochat permette di superare le limitazioni delle piattaforme esistenti. La possibilità di integrare predizioni basate su intelligenza artificiale e suggerimenti contestuali offre agli utenti un supporto tangibile nel costruire risposte comprensibili e appropriate.

Inoltre, l’introduzione di strumenti specifici per la gestione e la personalizzazione dei pittogrammi consente agli utenti di creare un sistema di comunicazione realmente su misura, riducendo la frustrazione e migliorando la comprensione reciproca. Infine, l’eliminazione della necessità di mediatori (genitori o tutori) offre un’opportunità unica di autonomia e indipendenza.

In assenza di un’alternativa simile nel panorama delle applicazioni attualmente disponibili, Pictochat rappresenta una risposta concreta a un problema urgente e ancora irrisolto.

# Capitolo 7

### Analisi dei requisiti

### Requisiti funzionali:

1. **Ricerca avanzata di pittogrammi**  
   L’applicazione deve consentire agli utenti di effettuare ricerche rapide e mirate tra i pittogrammi disponibili. La ricerca deve supportare:
   * Filtraggio per categorie tematiche (es. Luoghi, Alimentazione, …).
   * Inserimento di parole chiave per trovare pittogrammi specifici.  
     Questo requisito è essenziale per ridurre il tempo necessario a comporre un messaggio e semplificare l’esperienza utente.
2. **Personalizzazione delle frasi predefinite**  
   Gli utenti devono poter creare e modificare preset di frasi standard per adattarle alle loro preferenze personali o esigenze specifiche. Questo strumento consente di velocizzare le comunicazioni frequenti e garantire un maggiore livello di personalizzazione.
3. **Interazione con contatti**  
   L’applicazione deve permettere agli utenti di comunicare direttamente con i loro contatti inviando:
   * Messaggi testuali.
   * Combinazioni di testo e pittogrammi.  
     Questo requisito risponde alla necessità di favorire una comunicazione diretta, superando le barriere attualmente esistenti.
4. **Gestione dei contatti e delle relazioni**  
   Gli utenti devono avere la possibilità di:
   * Aggiungere nuovi contatti.
   * Modificare i dettagli dei contatti esistenti.
   * Organizzare i contatti in base alla frequenza o al tipo di relazione.  
     Questo requisito mira a rendere l’applicazione più funzionale e utile nelle interazioni quotidiane.
5. **Feedback immediato e guida all’uso**  
   Per garantire un uso agevole, l’app deve fornire feedback visivi e uditivi immediati sulle azioni dell’utente, accompagnati da:
   * Suggerimenti contestuali.
   * Tutorial integrati per aiutare nuovi utenti a comprendere le funzionalità principali.
6. **Integrazione con intelligenza artificiale (IA)**  
   L’applicazione deve integrare un sistema di IA in grado di:
   * Suggerire automaticamente pittogrammi o frasi basandosi sul contesto.
   * Convertire immagini in testo e viceversa.  
     Questa funzionalità è fondamentale per migliorare l’autonomia degli utenti e accelerare la composizione di messaggi.

### Requisiti Non Funzionali

1. **Accessibilità**  
   L’applicazione deve essere progettata tenendo conto delle esigenze di utenti con disabilità cognitive o visive. Tra le misure previste:
   * Interfaccia utente intuitiva e semplice da navigare.
   * Opzioni di configurazione per adattare colori, dimensioni dei caratteri e feedback visivi/sonori.
2. **Riservatezza e sicurezza dei dati**  
   L’applicazione deve garantire:
   * Crittografia dei dati per proteggere le informazioni personali.
   * Conformità alle normative sulla privacy, come il GDPR.
   * Meccanismi di autenticazione sicuri per evitare accessi non autorizzati.
3. **Performance e tempi di risposta**  
   Per offrire un’esperienza utente fluida, l’app deve:
   * Garantire tempi di caricamento rapidi.
   * Essere reattiva anche su dispositivi con risorse limitate.  
     Questo è particolarmente importante per utenti con poca pazienza o attenzione limitata.
4. **Compatibilità multi-piattaforma**  
   L’applicazione deve essere accessibile su diverse piattaforme, tra cui:
   * Dispositivi mobili (iOS e Android).
   * Browser web per desktop.  
     Questo requisito aumenta la flessibilità e l’accessibilità per utenti in diverse situazioni.
5. **Adattabilità del layout**  
   Il design dell’app deve essere responsivo, adattandosi automaticamente a:
   * Dimensioni e risoluzioni dello schermo.
   * Orientamenti orizzontale e verticale.  
     Questo garantisce una buona esperienza utente su tutti i dispositivi.

# Capitolo 8

## Approccio al Problema

Durante lo sviluppo del progetto, ho adottato un approccio iterativo e orientato ai feedback, senza suddividere il lavoro in sprint o milestone prestabiliti. La principale fonte di guida e revisione è stata rappresentata dai feedback del mio committente. Questo metodo mi ha permesso di adattare rapidamente il progetto alle sue esigenze e aspettative, concentrandomi su miglioramenti incrementali e modifiche specifiche.

La gestione del repository è avvenuta tramite GitHub, dove ho utilizzato il versionamento del codice per tracciare le modifiche e mantenere una cronologia chiara dello sviluppo. Le pull request e i commit ben documentati hanno contribuito a garantire trasparenza e facilità di revisione durante tutte le fasi del progetto.

### Linguaggi di Programmazione

Il progetto è stato sviluppato utilizzando i seguenti linguaggi di programmazione e framework:

* **TypeScript**: Linguaggio principale utilizzato per garantire tipizzazione statica e una maggiore robustezza del codice. Questo ha ridotto gli errori durante lo sviluppo e migliorato la manutenibilità del progetto.
* **React**: Framework JavaScript scelto per lo sviluppo dell'interfaccia web. La sua modularità e la capacità di creare componenti riutilizzabili hanno reso possibile uno sviluppo rapido e ben organizzato.

### Strumenti Utilizzati

Per supportare lo sviluppo del progetto, sono stati impiegati i seguenti strumenti:

* **NPM (Node Package Manager)**: Gestore dei pacchetti utilizzato per installare e gestire le dipendenze del progetto. Grazie a NPM, è stato possibile integrare rapidamente librerie utili e mantenerle aggiornate.
* **Postman**: Strumento essenziale per il test delle API. È stato utilizzato per verificare la corretta comunicazione tra il frontend e i servizi esterni, consentendo un debug efficace delle interazioni.

### Librerie usate

**antd e @ant-design/icons**

* **Descrizione**: Ant Design è una libreria di componenti React predefiniti che segue un design system professionale. Include un'ampia varietà di componenti per la creazione di interfacce utente intuitive e moderne.
* **Utilizzo**: L'ho usata per creare interfacce utente uniformi e accattivanti. Gli **@ant-design/icons** sono stati utilizzati per arricchire l'aspetto grafico con icone vettoriali.

**react-infinite-scroll-component**

* **Descrizione**: Fornisce un componente React che implementa lo scrolling infinito.
* **Utilizzo**: L'ho utilizzata per caricare dinamicamente i messaggi nella chat, migliorando l'usabilità dell'applicazione e per caricare dinamicamente i contatti.

**react-phone-number-input**

* **Descrizione**: Una libreria per la gestione e la validazione di numeri di telefono.
* **Utilizzo**: Utilizzata per implementare un input affidabile e user-friendly per i numeri di telefono, fondamentale per le funzionalità di registrazione e autenticazione.

**react-router-dom**

* **Descrizione**: Una libreria per gestire il routing delle applicazioni React.
* **Utilizzo**: Implementata per navigare tra le diverse pagine dell'applicazione, mantenendo una struttura chiara e organizzata.

**telegram**

* **Descrizione**: Libreria per interagire con le API di Telegram.
* **Utilizzo**: Utilizzata per interragire con telegram nel suo complesso. Ovvero permette di mandare messaggi, leggere I messaggi presenti, tramite websocket permette di ottenere le notifiche. Insomma tutto ciò che telegram permette di fare questa lib lo mette a disposizione

**typescript**

* **Descrizione**: Linguaggio di programmazione che estende JavaScript aggiungendo il supporto per i tipi statici.
* **Utilizzo**: Fondamentale per lo sviluppo del progetto, ha garantito una maggiore affidabilità del codice.

**gh-pages**

* **Descrizione**: Una libreria per distribuire applicazioni React su GitHub Pages.
* **Utilizzo**: Usata per pubblicare la mia applicazione in modo semplice e rapido. Nella mia repo github in modo da rimuvore i costi di infrastruttura.

**webpack-cli**

* **Descrizione**: Strumento per configurare e gestire Webpack, il bundler utilizzato nel progetto.

# Capitolo 10

## Architettura

In questo capitolo viene illustrata l’architettura del sistema, partendo da una panoramica generale della sua struttura e di come i vari componenti interagiscano tra di loro, per poi approfondire la struttura del codice di ciascun componente

### Panoramica

L'applicazione si basa su un'architettura **React** con un'organizzazione modulare che sfrutta **React Router**, **Context API**, e una struttura di componenti ben definita. Questo design è ideale per applicazioni scalabili e interattive, garantendo separazione delle responsabilità, flessibilità, e riutilizzabilità del codice. Di seguito esploreremo i principali aspetti architetturali dell'applicazione:

### Struttura dell'Applicazione

La struttura del progetto è organizzata in modo gerarchico e modulare per supportare la manutenibilità e l'espansione futura:

* + **Root Files**:
    - **index.tsx**: Punto di ingresso dell'applicazione, responsabile del rendering del componente principale (MyApp) tramite il provider di sessione, il tema (ConfigProvider) e il router.
    - **MyApp.tsx**: Il contenitore principale che gestisce la logica di autenticazione, le redirezioni, e il layout di base.
  + **Cartelle principali**:
    - **components**: Contiene componenti UI specifici e riutilizzabili come ChatBubble, DialogItem, e CustomFooter.
    - **context**: Implementa SessionContext, che gestisce lo stato globale relativo alla sessione dell'utente.
    - **data**: Include file JSON per dati statici e risorse come temi e frasi predefinite.
    - **lib**: Librerie di supporto come TgApi, Database, e Controller, che gestiscono la logica di business e l'integrazione con API esterne.
    - **routes**: Contiene le definizioni delle rotte e i componenti per ciascuna pagina, come Contacts, Login, e Profile.
    - **styles**: Fogli di stile CSS organizzati per singolo componente.

### Gestione delle Rotte

**Router Hash-Based**

L'app utilizza un router basato su **hash** (createHashRouter), configurato in AppRoutes.tsx. Questo approccio è ottimale per ambienti in cui il server non gestisce routing lato backend, ad esempio applicazioni distribuite su CDN o piattaforme statiche.

* **Rotte principali**: Ogni percorso è associato a un componente specifico:
  + /welcomePage: Pagina di benvenuto.
  + /login: Schermata di login.
  + /contacts: Elenco dei contatti.
  + /chat/:chatId: Chat specifica con un utente o un gruppo.
* **Rotte protette**: Sono implementate con il componente ProtectedRoute, che verifica l'autenticazione dell'utente prima di consentire l'accesso.

### Gestione dello Stato e del Contesto

L'applicazione utilizza **Context API** per la gestione dello stato globale, in particolare per la sessione dell'utente, tramite SessionContext. Questo approccio consente di evitare l'uso di strumenti più complessi come Redux in contesti dove la gestione dello stato è semplice e centralizzata.

**Esempio: SessionProvider**

* + In index.tsx, il provider della sessione è il wrapper più esterno, che consente a tutti i componenti figli di accedere al contesto della sessione. Funziona in tandem con hook personalizzati come useIsAuthenticated per verificare lo stato di login dell'utente.

### Protezione delle Rotte

La sicurezza è implementata tramite:

* **ProtectedRoute**: Verifica l'autenticazione prima di permettere l'accesso alle rotte sensibili.
* **Redirect**: In MyApp.tsx, un effetto useEffect reindirizza gli utenti non autenticati verso /login o /settings.

### Gestione della UI

La UI è progettata con un focus sulla modularità:

* **Componenti riutilizzabili**: La cartella components contiene elementi generici come ChatBubble e DialogItem, che possono essere riutilizzati in diversi contesti.
* **Stili modulari**: Gli stili sono divisi per componente, riducendo il rischio di conflitti e semplificando le modifiche future.
* **Theme Customization**: Il tema dell'app è centralizzato in ConfigProvider, consentendo un controllo coerente sull'aspetto visivo.

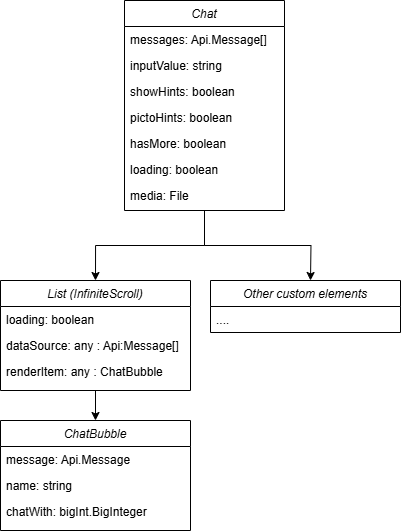


Figura 1 Parentela oggetti

In questo esempio, il componente della chat è organizzato gerarchicamente per garantire modularità e manutenibilità:

* **Container**: Componente principale che racchiude l'intera chat.
  + **InfiniteScroll**: Gestisce lo scorrimento infinito per caricare dinamicamente i messaggi.
    - **List**: Componente responsabile del rendering della lista di messaggi.
      * **ChatBubble**: Componente individuale che rappresenta ogni messaggio.
      * Altri componenti legati ai messaggi.
  + Altri elementi personalizzati, come pulsanti o intestazioni.

Questa struttura modulare consente di aggiornare facilmente la funzionalità di un singolo componente senza influire sugli altri. Ad esempio, se in futuro sarà necessario modificare il comportamento o lo stile di ChatBubble, sarà sufficiente intervenire sul suo codice interno, mantenendo intatta l'integrità del sistema.

### Logica di Business e API

La logica di business è centralizzata nella cartella **lib**, che comprende i seguenti moduli:

* **Controller**: Gestisce le interazioni principali, incluse le notifiche in tempo reale attraverso tgApi.
* **TgApi**: Si occupa della comunicazione con un'API esterna per ricevere aggiornamenti e inviare messaggi.
* **Database e Store**: Implementano la persistenza dei dati lato client, probabilmente utilizzando localStorage o IndexedDB.
* **Utils**: Contiene funzioni di supporto generali.

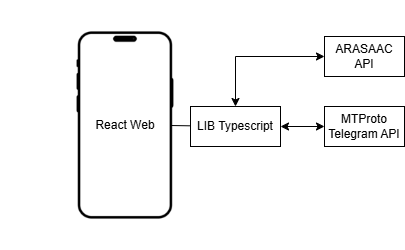


Figura 2 Architettura REST

**Esempio di gestione degli aggiornamenti:**

* + In MyApp.tsx, l'effetto useEffect utilizza Controller.tgApi.handleUpdates per ascoltare aggiornamenti in tempo reale, gestendoli con un updateManager associato alla pagina attiva (getActivePage()).

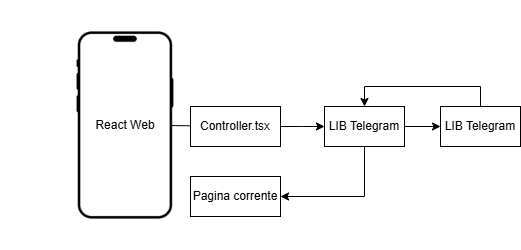


Figura 3 Archietettura websocket

### Vantaggi dell'Architettura

* **Scalabilità**: La separazione in moduli come lib e routes rende semplice aggiungere nuove funzionalità.
* **Manutenibilità**: La chiara divisione delle responsabilità facilita la comprensione e l'aggiornamento del codice.
* **Interattività**: L'uso di Context API e hook React garantisce un'interfaccia reattiva e personalizzata.

## Implementazione

### Backend

1. **Telegram MTProto 2.0**

MTProto (Mobile Telegram Protocol) è il protocollo di comunicazione utilizzato da Telegram per garantire la sicurezza, l'affidabilità e la velocità nella trasmissione di messaggi tra client e server. Esso offre due tipi principali di cifratura:

1. **Crittografia server-client:** utilizzata per le chat normali (cloud chats), in cui il server gestisce i dati.
2. **Crittografia end-to-end:** utilizzata per le chat segrete (secret chats).

**Struttura del Messaggio**

Un messaggio in MTProto 2.0 è composto da tre principali sezioni:

1. **Header esterno (non cifrato):** include auth\_key\_id e msg\_key, che servono a identificare il messaggio e la chiave di autorizzazione.
2. **Header interno (cifrato):** contiene informazioni come server\_salt, session\_id, message\_id, msg\_seqno e la lunghezza del messaggio.
3. **Payload cifrato:** il corpo del messaggio vero e proprio, cifrato utilizzando AES-256 in modalità IGE.

**Crittografia**

MTProto 2.0 utilizza un'architettura crittografica basata su:

* **SHA-256:** per generare hash e chiavi.
* **AES-256 IGE:** per cifrare i messaggi.

Il messaggio chiaro è pre-processato con il server\_salt, session\_id e altri metadati. Questi dati, insieme a una porzione di auth\_key, vengono usati per calcolare msg\_key. Successivamente, da msg\_key e auth\_key vengono derivati aes\_key e aes\_iv, che sono usati per cifrare il messaggio.

**Passaggi Principali per Inviare un Messaggio**

1. Creazione della chiave di autorizzazione (**auth\_key**).
2. Calcolo dei metadati e dell'hash (**msg\_key**).
3. Cifratura del messaggio con AES-256 IGE.
4. Aggiunta dell'header esterno per costruire il payload.
5. Invio del payload al server tramite TCP o HTTP.

**A diagram of a computer network

Description automatically generated**

Figura 4 MTProto

**Libreria telegram (*GramJS*)**

Per semplificare l'integrazione con il protocollo **MTProto** e accedere rapidamente alle funzionalità di Telegram, ho scelto di utilizzare una libreria preesistente: [telegram](https://www.npmjs.com/package/telegram), un modulo NPM che fornisce un'interfaccia semplice per interagire con le API di Telegram.

**Perché la Libreria telegram**

La libreria telegram è basata su **MTProto 2.0** e consente di eseguire operazioni come l'invio di messaggi, la gestione delle chat, il download di file, e altre funzionalità offerte dalle API di Telegram. Offre un'astrazione sopra il protocollo crittografico, semplificando l'autenticazione, la creazione delle sessioni e l'invio dei messaggi.

**Configurazione e Installazione**

Per utilizzare la libreria, è necessario installarla nel progetto Node.js:

bash

***npm install telegram***

**Configurazione di Base**

La libreria richiede due parametri fondamentali:

1. **API ID** e **API Hash**: ottenuti registrando un'applicazione sul Portale per Sviluppatori Telegram [App configuration (telegram.org)](https://my.telegram.org/apps)

**File di sessione**: per salvare lo stato dell'autenticazione e gestire sessioni persistenti.

Ho usato la libreria telegram, includendo le chiavi API nel client per semplicità e costi ridotti. Questa soluzione, seppur sufficiente per sviluppo iniziale, espone le chiavi a rischi di abuso. In un sistema ottimale, le chiavi sarebbero gestite su un server per garantire maggiore sicurezza e protezione.

1. **ARASAAC**

Nel mio progetto, ho deciso di implementare a mano le chiamate alle API REST di ARASAAC, data la loro semplicità e l'ampia documentazione disponibile. ARASAAC offre un sistema RESTful ben strutturato per accedere a dati relativi ai pittogrammi e ai relativi metadati, come parole chiave e traduzioni. Di seguito descrivo le principali funzionalità delle API utilizzate.

**Endpoint Principali**

1. **Keywords (Parole Chiave)** Questo endpoint restituisce tutta la lista di parole associate ai pittogrammi.
   * **GET** /keywords/{language}  
     *Parametri*: language indica la lingua dei termini restituiti.
2. **Pittogrammi** Gli endpoint relativi ai pittogrammi permettono di recuperare informazioni sui pittogrammi disponibili nel database ARASAAC. Esempi di utilizzo:
   * **GET** /pictograms/all/{language}  
     Restituisce tutti i pittogrammi disponibili in una determinata lingua.
   * **GET** /pictograms/{idPictogram}  
     Recupera informazioni dettagliate su un pittogramma specifico, identificato da idPictogram.
   * **GET** /pictograms/{language}/{idPictogram}  
     Simile al precedente, ma con dati contestualizzati alla lingua specificata.
   * **GET** /pictograms/{idPictogram}/languages/{languages}  
     Fornisce le traduzioni di un pittogramma in più lingue.
3. **Ricerca** Gli endpoint di ricerca consentono di trovare pittogrammi in base a termini specifici o sinonimi:
   * **GET** /pictograms/{language}/search/{searchText}  
     Cerca pittogrammi basandosi sul termine searchText nella lingua indicata.
   * **GET** /pictograms/{language}/bestsearch/{searchText}  
     Effettua una ricerca avanzata per individuare il miglior corrispondente di searchText.
4. **WordNet e Sinonimi** Per un'integrazione semantica più avanzata, è possibile cercare pittogrammi basati su WordNet.
   * **GET** /pictograms/{language}/wordnet/{wordnet}/id/{synset}  
     Cerca pittogrammi utilizzando identificatori semantici di WordNet

**Utilizzo nel progetto:**

Durante lo sviluppo del progetto, l'utilizzo delle API di ARASAAC era inizialmente molto frequente. Ogni parola di un messaggio inviato dall'utente generava una richiesta HTTP verso il sito di ARASAAC per recuperare i pittogrammi corrispondenti. Questa strategia, benché funzionale, presentava significative inefficienze in termini di numero di richieste e prestazioni complessive del sistema.

**Dimensione delle Richieste**

* Per comprendere l'entità del problema, consideriamo una tipica conversazione:
* Una chat media comprende circa 20 messaggi.
* Ogni messaggio contiene in media 10 parole.
* Ogni parola genera una richiesta API verso ARASAAC per cercare i pittogrammi corrispondenti.

Questo significa che:

*20 messaggi x 10 parole/messaggio = 200 richieste per ogni chat*

Considerando molteplici conversazioni in un breve arco di tempo, il numero totale di richieste diventa rapidamente ingestibile, causando:

1. **Rallentamenti**: Ogni richiesta richiede un round trip verso il server.
2. **Overhead per ARASAAC**: Un utilizzo intensivo può sovraccaricare il servizio.
3. **Limiti di banda e latenza**: Le prestazioni della chat peggiorano, specialmente in ambienti con connettività limitata.

**Soluzione: Creazione di un File JSON Localizzato**

* Per mitigare questi problemi, ho deciso di adottare un approccio alternativo basato sull'uso di un file JSON statico. Questo file contiene tutte le informazioni necessarie per visualizzare i pittogrammi, eliminando la necessità di effettuare richieste API in tempo reale.

**Generazione del File JSON**

* Utilizzando le keyword fornite dall'endpoint */keywords/{language}* di ARASAAC, ho recuperato in anticipo l'intero set di pittogrammi. Per ogni parola chiave, ho memorizzato:
* La parola chiave stessa.
* Il link al pittogramma corrispondente.

**Struttura:**

Il *JSON* contiene tutte e 15k pittogrammi con \_*id*, *word*, *categoria*, *url* (che punta all’immagine)*, data di creazione*.

Il risultato finale è un file di *8 MB*, molto più manipolabile in quanto ogni accesso al file può avvenire in modo sincrono e quindi migliora la compatibilità con *React*

1. **Integrazione con Ai e Lemmatizzazione**

Durante lo sviluppo del progetto, diverse limitazioni tecniche e di risorse hanno ostacolato l’implementazione di soluzioni AI avanzate e di strumenti di lemmatizzazione. Tuttavia, sono stati esplorati diversi approcci per integrare tali funzionalità nel sistema, con particolare attenzione alla possibilità di gestirle sia lato frontend che attraverso servizi backend personalizzati.

**Tentativi di Integrazione AI**

1. **NLP per la Categorizzazione delle Domande**  
   L’obiettivo era categorizzare ogni domanda in una categoria predefinita, utilizzando tecniche di Natural Language Processing (NLP) come il *zero-shot learning*. Tuttavia, i modelli pre-addestrati disponibili, come GPT-2, non si sono rivelati sufficientemente affidabili per l’italiano, soprattutto nelle versioni gratuite. In più a molte domande si potrebbe rispodnere in modi molto diversi.
2. **Predizione del Testo con GPT-2**  
   Si è provato ad adottare GPT-2 per migliorare la comprensione delle frasi, ma la mancanza di modelli ottimizzati per l’italiano ha limitato i risultati. Inoltre, il costo per ospitare un modello GPT-2 su piattaforme come Hugging Face è risultato proibitivo, rendendo questa soluzione impraticabile per il progetto.  
   GPT-2 è stato utilizzato anche per cercare di predirre quale possibili risposte si potrebbe ottenere da una domanda prese da un array di risposte.
3. **Text-to-Speech**  
   Per il text-to-speech, la funzione nativa del browser (Google) offre una soluzione gratuita ma meccanica. Le alternative di qualità superiore, fornite da servizi online, sono disponibili solo a pagamento.

**Proposta di Soluzione: Utilizzo di un Server Domestico**

Un server domestico appena ricevuto rappresenta una potenziale soluzione per gestire in locale modelli AI e servizi di lemmatizzazione. Con questa configurazione, potrei:

* Ospitare un modello Python personalizzato per migliorare la categorizzazione delle domande.
* Implementare algoritmi di lemmatizzazione ottimizzati per l’italiano, garantendo prestazioni rapide e affidabili.
* Eliminare la dipendenza da servizi esterni costosi o poco efficienti.

**Approccio con Servizi Esterni**

Come alternativa al server domestico, ho sperimentato l’utilizzo di **Render.com**, una piattaforma che permette di ospitare servizi personalizzati come API REST. Il processo prevedeva:

1. Scrivere il codice necessario.
2. Pubblicarlo su GitHub.
3. Dockerizzare il servizio e farlo ospitare da Render.com.
4. **Render.com** fornisce un URL API per accedere al servizio, ma presenta problemi di performance: alcune richieste richiedono diversi minuti per restituire un risultato, poiché i servizi gratuiti non hanno priorità elevata.

Questo servizio era stato pensato per:

* Implementare l’AI necessaria per la categorizzazione delle domande.
* Fornire un motore di lemmatizzazione per gestire correttamente le parole nelle frasi.

**Lemmatizzazione: Soluzioni Esplorate**

La lemmatizzazione è cruciale per garantire il corretto funzionamento del sistema ARASAAC. Parole come "avessimo" o "chiamato" non vengono riconosciute se non riportate alla loro forma infinitiva.

**Approccio iniziale**

L’utilizzo di servizi AI per la lemmatizzazione non si è rivelato praticabile a causa delle limitazioni sopra descritte.

**Soluzione Alternativa: Dump dei Verbi Italiani**

Ho deciso di scaricare un elenco completo di verbi italiani dal sito [Italian Verbs](https://www.italian-verbs.com/verbi-italiani/verbi-italiani-top.php?pg=1). Questo sito, non protetto da Cloudflare, ha permesso di ottenere i dati in modo semplice. I verbi sono stati organizzati in un file JSON contenente tutte le coniugazioni per ciascun tempo e persona.

Con questo approccio, è possibile ricondurre rapidamente ogni verbo coniugato alla sua forma infinitiva tramite una semplice ricerca nel file JSON, senza necessità di utilizzare modelli AI costosi o complessi.

**Implementazione Frontend e Backend**

Questa soluzione può essere gestita completamente lato frontend, tramite:

1. **Accesso Locale al File JSON**
   * Il file JSON viene caricato nel frontend, consentendo una rapida mappatura delle parole senza richieste esterne.
2. **Ottimizzazione del Parsing**
   * Un algoritmo frontend esegue il parsing delle frasi, sostituendo i verbi con la loro forma infinitiva utilizzando i dati JSON.
3. Oppure, per maggiore scalabilità, lato backend:

**Vantaggi di un’Implementazione Personalizzata**

1. **Autonomia**: Eliminando la dipendenza da servizi esterni costosi o poco performanti.
2. **Efficienza**: Le richieste vengono elaborate rapidamente grazie all’accesso diretto ai dati locali o al server dedicato.
3. **Flessibilità**: È possibile personalizzare i modelli AI o i dati JSON per migliorare ulteriormente le prestazioni.

### Frontend

Spiegare ogni lib che ho scritto, prendere esempio di pagina e spiegarlo, tipo chat. Scrivere rigardo i problemi avuti e i tentativi.