SocialBridge

Object Design Document

SocialBridge

Riferimento	C18_ODD
Versione	0.8
Data	19/11/2024
Destinatario	Prof.ssa Filomena Ferrucci, Prof. Fabio Palomba
Presentato da	C18 team
Approvato da	

Team Members

Nome	*Cognome*	*Ruolo*	*Acronimo*	*Contatto*
Francesco	Peluso	PM	FP	f.peluso25@studenti.unisa.it
Luciano	Bercini	PM	LB	l.bercini@studenti.unisa.it
Mariarosaria	Rossi	PM	MR	m.rossi60@studenti.unisa.it
Fabio	Di Lallo	TM	FDL	f.dilallo3@studenti.unisa.it
Michele	Di Meo	TM	MDM	m.dimeo7@studenti.unisa.it
Luca	Fiore	TM	LF	l.fiore20@studenti.unisa.it
Andrea	Senatore	TM	AS	a.senatore158@studenti.unisa.it
Francesco	De Felice	TM	FDF	f.defelice4@studenti.unisa.it
Matteo	Nocerino	TM	MN	m.nocerino8@studenti.unisa.it
Alfio	Marra	TM	AM	a.marra39@studenti.unisa.it

Revision History

Data	*Versione*	*Descrizione*	*Autori*
19/11/2024	0.1	Prima stesura	FDL, MDM, LF, AS, FDF, MN, AM
19/11/2024	0.2	Scrittura paragrafi 1, 1.1, 1.2, 1.3, 1.4	FDL, MDM, LF, AS, FDF, MN, AM
20/11/2024	0.3	Scrittura sezione 2, Definizione package	FDL, MDM, LF, AS, FDF, MN, AM
22/11/2024	0.4	Scrittura sezione 3 Class Interfaces	FDL, MDM, LF, AS, FDF, MN, AM
24/11/2024	0.5	Scrittura sezione 4 Class Diagram Ristrutturato	FDL, MDM, LF, AS, FDF, MN, AM
26/11/2024	0.6	Definizione Design Pattern e Glossario	FDL, MDM, LF, AS, FDF, MN, AM
26/11/2024	0.7	Correzione Class Interface, Class Diagram Ristrutturato	FDL, MDM, LF, AS, FDF, MN, AM
26/11/2024	0.8	Revisione Finale	FDL, MDM, LF, AS, FDF, MN, AM

Sommario

1. Introduzione

SocialBridge è una piattaforma web inclusiva che facilita la partecipazione a eventi sociali per persone introverse o con disabilità. Questo documento descrive i principali obiettivi di design, linee guida, termini e riferimenti correlati alla fase di object design.

1.1: Object design goal

- **Inclusività e Accessibilità:** Garantire la creazione di eventi che rispettino le esigenze personali degli utenti.
- **Manutenibilità:** Progettare un sistema modulare e scalabile, con sottosistemi ben definiti.
- **Sicurezza:** Protezione dei dati sensibili tramite autenticazione avanzata e crittografia.
- **Usabilità Intuitiva:** Offrire un'interfaccia user-friendly per tutte le fasce di utenti.
- **Prestazioni Ottimizzate:** Ridurre i tempi di risposta garantendo la scalabilità del sistema.

1.1.1: Trade-Off

Sicurezza vs. Costi: a causa di tempi e costi limitati il sistema di sicurezza della piattaforma sarà basato solo su email e password. L'autenticazione verrà fatta tramite una pagina apposita e per ogni operazione che necessita la conoscenza dell'identità del cliente.

Persistenza vs. Scalabilità: la piattaforma utilizzerà un database NoSQL (MongoDB) per la memorizzazione dinamica dei dati relativi a utenti, eventi e interazioni. Questo approccio migliora la scalabilità e flessibilità, ma può comportare un lieve aumento della complessità nella gestione della consistenza dei dati.

Privacy vs. Personalizzazione: si garantirà la protezione dei dati sensibili di un utente, ma al contempo si cercherà di utilizzare i propri dati per offrire un'esperienza personalizzata all'interno della piattaforma.

1.2: Linee guida per la documentazione dell'interfaccia

- Framework Backend: Node.js, MongoDB.
- Framework Frontend: React.js, JavaScript.
- Standard: Seguire le linee guida WCAG per l'accessibilità.

1.3: Definizioni, acronimi e abbreviazioni

- **CSS:** Cascading Style Sheets
- **HTML:** HyperText Markup Language
- **JS:** JavaScript
- **ODD:** Object Design Document
- **SDD:** System Design Document
- **UML:** Unified Modeling Language

1.4: Riferimenti

Libro usato come riferimento: Object-Oriented Software Engineering (Using UML, Patterns, and Java) di Bernd Bruegge & Allen H.

2. Packages

Questa sezione descrive la suddivisione dei package del sistema SocialBridge, basata sull'architettura modulare di React per il frontend e Node.js con MongoDB per il backend. La struttura è progettata per garantire modularità, scalabilità e separazione delle responsabilità.
La suddivisione è definita di seguito:

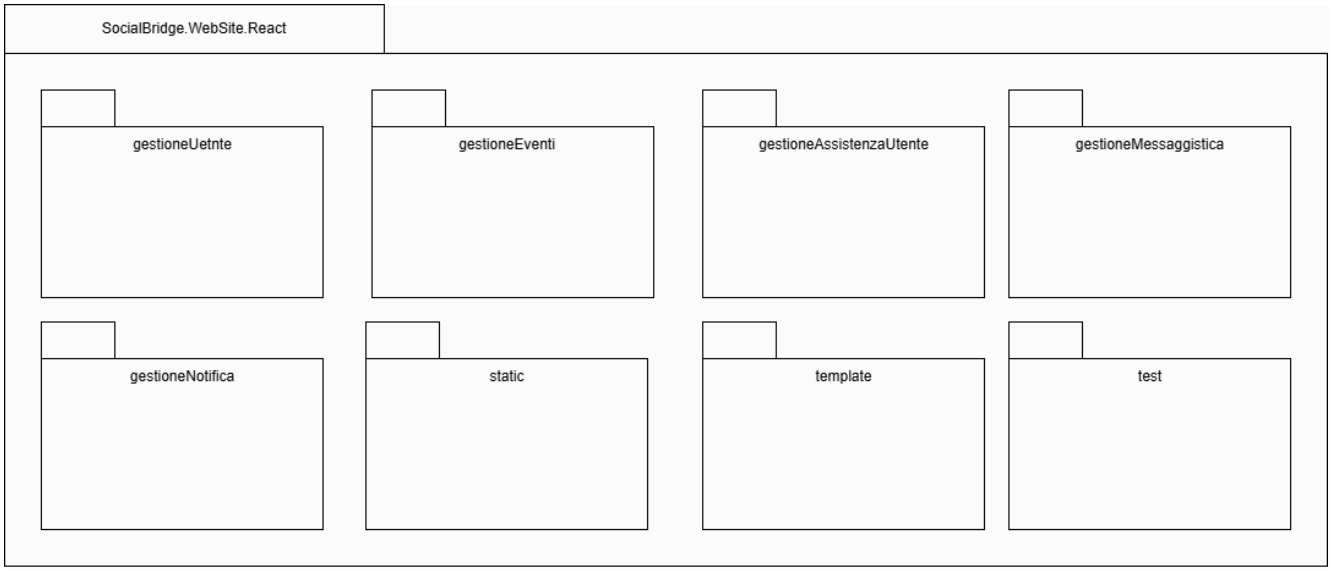
- **Front-end (React.js)**
  - src: Contiene i file sorgenti principali:
    - components: Componenti React per le funzionalità principali (es. utenti, eventi, navigazione).
    - pages: Le pagine dell'applicazione (es. Home, Eventi, Profilo).
    - assets: File statici (CSS, immagini).
    - services: Gestisce le chiamate API al backend.

- **Backend (Node.js + MongoDB)**
  - models: Modelli di dati per MongoDB (es. utenti, eventi, recensioni).
  - controllers: Logica per la gestione delle richieste API (es. autenticazione, gestione eventi).
  - routes: Rotte per le API REST.
  - middlewares: Middleware per funzioni come autenticazione (es. JWT).
  - utils: Funzioni di supporto (es. connessione al database).

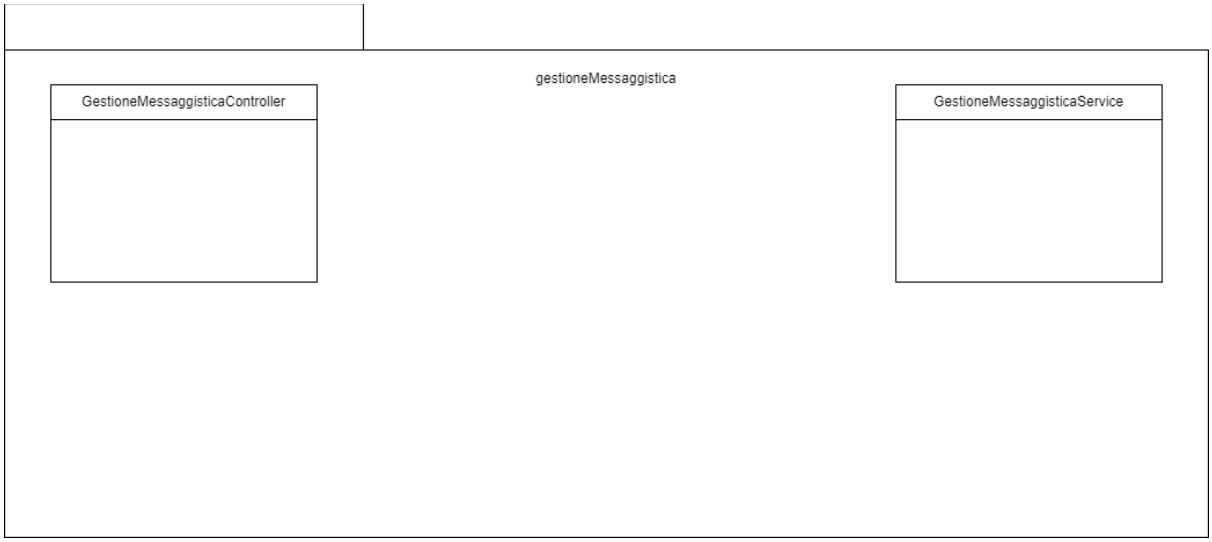
- **Test**
  - Test di frontend e backend per garantire la qualità del codice e la stabilità del sistema.

- **Configurazione**
  - .env: File per gestire variabili d'ambiente (es. URI MongoDB, chiavi API).
  - package.json: File per gestire dipendenze e script di avvio.

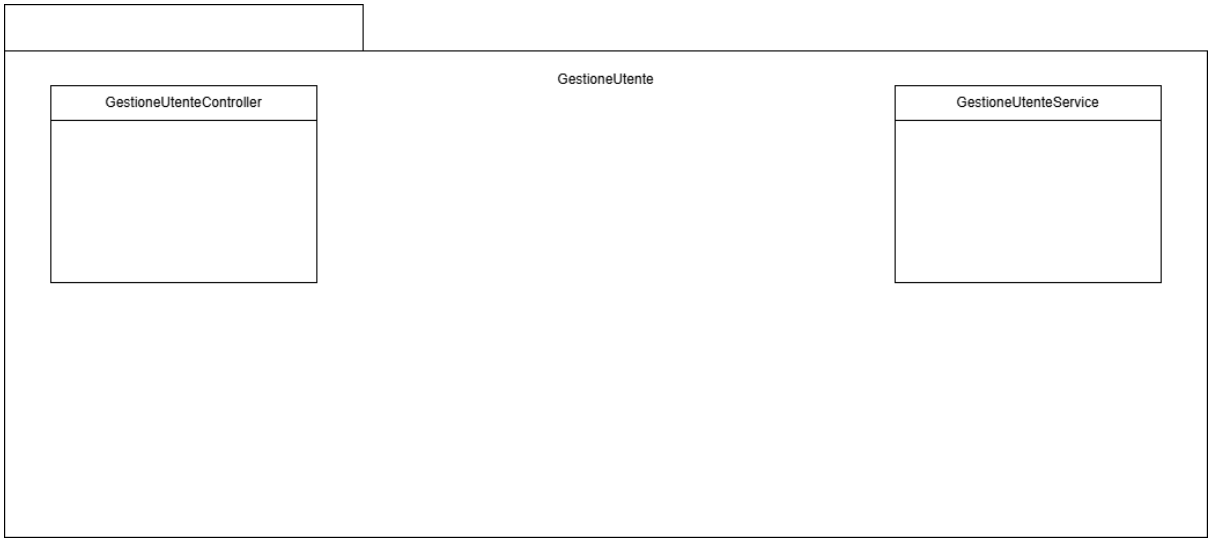
Package SocialBridge



Package GestioneMessaggistica

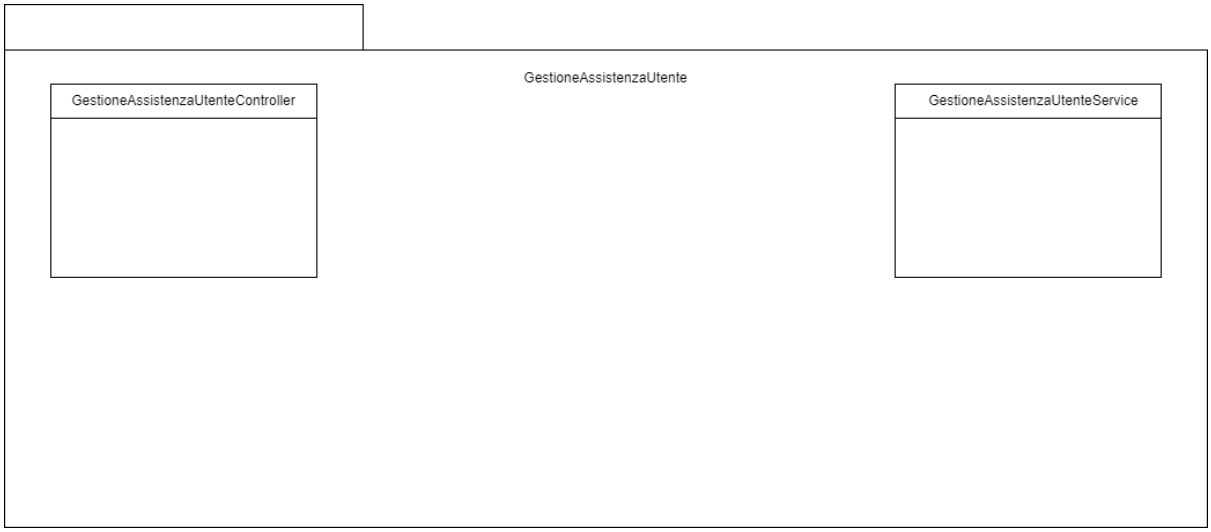


Package GestioneUtente

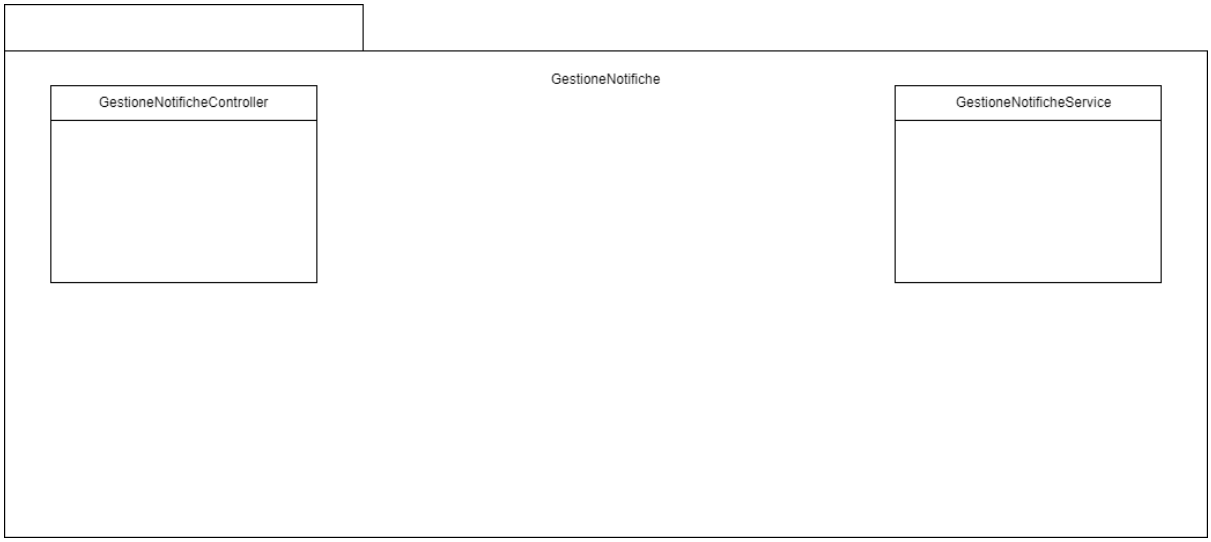


Package GestioneEventi

Package GestioneAssistenzaUtente



Package GestioneNotifiche



3. Class Interfaces

Class Interface: Utente

Nome Classe	GestioneUtenteService
Descrizione	Gestisce la registrazione, il login, logout, la cancellazione, la ricerca e la modifica dei dati personali di un utente
Metodi	+**registerUser**(email:str, password:str, name:str): void +**searchByName**(name:str, cognome:str): User +**loginUser**(email:str, password:str): void +**updateProfile**(email:str, password:str, nome:str, cognome:str, ruolo:str, interessi:str): User +**deleteUser**(userID:int): void +**getUserById**(userID:int): User
Invarianti di classe	/

Nome Metodo	registerUser(email:str, password:str, name:str): void
Descrizione	Registra un nuovo utente.
Pre-condizione	**context:** GestioneUtenteService::registerUser(email, password, name) **pre:** email != none AND email != “” AND password != none AND password != “” AND name != none AND name!= “”
Post-condizione	**context:** GestioneUtenteService::registerUser(email, password, name) **post:** GestioneUtenteDAO.save(email, password, name)

Nome Metodo	searchByName(name:str, cognome:str): User
Descrizione	Trova utenti tramite nome e cognome
Pre-condizione	**context:** GestioneUtenteService::searchByName(name, cognome) **pre:** name!= none AND name!= “” AND cognome!= none AND cognome!= “”
Postcondizione	**context:** GestioneUtenteService::searchByName(name, cognome) **post:** return User se esistente.

Nome Metodo	**loginUser(email:str, password:str): void**
Descrizione	Autentica un utente.
Pre-condizione	**context:** GestioneUtenteService::registerUser(email, password, name) **pre:** email != none AND email != “” AND password != none AND password != “”
Post-condizione	**context:** GestioneUtenteService::registerUser(email, password, name) **post:** login effettuato correttamente OR utente non è registrato

Nome Metodo	**updateProfile(email:str, password:str, nome:str, cognome:str, ruolo:str, interessi:str): User**
Descrizione	Aggiorna i dati del profilo utente.
Pre-condizione	**context:** GestioneUtenteService::updateProfile(email, password, nome, cognome, ruolo, interessi) **pre:** email != none AND email != “” AND password != none AND password != “” AND name != none AND name!= “” AND cognome != none AND cognome != “” AND ruolo != none AND ruolo != “” AND interessi != none AND interessi != “”
Post-condizione	**context:** GestioneUtenteService::updateProfile(email, password, nome, cognome, ruolo, interessi) **post:** return User con le modifiche effettuate

Nome Metodo	**deleteUser(userID:int): void**
Descrizione	Elimina un account utente.
Pre-condizione	**context:** GestioneUtenteService::deleteUser(userID) **pre:** userID != none
Post-condizione	**context:** GestioneUtenteService::deleteUser(userID) **post:** GestioneUtenteDAO.delete(userID, email, password, nome, cognome, ruolo, interessi)

Nome Metodo	**getUserById(userID:int): User**

Descrizione	Recupera i dati di un utente tramite ID.
Pre-condizione	**context:** GestioneUtenteService::getUserbyID(userID) **pre:** userID != none
Post-condizione	**context:** GestioneUtenteService::getUserbyID(userID) **post:** return User OR none se non esiste

Class Interface: Evento

Nome Classe	**GestioneEventiService**
Descrizione	Gestisce la creazione modifica, iscrizione agli eventi
Metodi	+**createEvent**(titolo:str, descrizione:str, data:date, ora:datetime, luogo:str, accessibilità: str, partecipantiMAX:int, pieno: bool, labels: List[str]): void +**updateEvent**(titolo:str, descrizione:str, data:date, luogo:str, ora:datetime, accessibilità: str, partecipantiMAX: int, pieno: bool, labels: List[str]): Event +**deleteEvent**(eventID: int): void +**getAllEvents**(): void +**removeParticipant**(eventID: int, userID: int): void +**registerToEvent**(eventID: int, userID: int, pieno: bool): boolean +**searchEvents**(titolo:str, descrizione:str, data:date, ora:datetime, luogo:str, accessibilità: str, labels: List[str]): List[Event] +**getEventParticipants**(eventID: int): List[User] +**getEventDetails**(eventID: int): Event
Invarianti di classe	**/**

Nome Metodo	**createEvento(titolo:str, descrizione:str, data:date, ora:datetime, luogo:str, accessibilità: str, partecipantiMAX: int, pieno: bool, labels: List[str]): void**
Descrizione	Crea un nuovo evento con i dati forniti
Pre-condizione	**context:** GestioneEventiService::createEvent(titolo, descrizione, data,ora, luogo, accessibilità, partecipantiMAX, pieno, labels)

	pre: titolo != “” AND descrizione != “” AND data != none AND luogo != “” AND accessibilità != “” AND ora !=”” AND pieno !=“”
Post-condizione	**context:** GestioneEventiService::createEvent(titolo, descrizione, data, ora, luogo, accessibilità, partecipantiMAX, pieno, labels) **post:** GestioneEventoDAO.save(titolo, descrizione, data, luogo, accessibilità, partecipantiMAX, pieno, labels)

Nome Metodo	**updateEvent(titolo:str, descrizione:str, data:date, ora:datetime, luogo:str, accessibilità: str, partecipantiMAX: int, pieno: bool, labels: List[str]): Event**
Descrizione	Modifica dettagli di un evento esistente
Pre-condizione	**context:** GestioneEventiService::updateEvent(titolo, descrizione, data, ora, luogo, accessibilità, partecipantiMAX, pieno, labels) **pre:** titolo != “” AND descrizione != “” AND data != none AND luogo != “” AND accessibilità != “” AND ora != “” AND pieno != “”
Post-condizione	**context:** GestioneEventiService::updateEvent(titolo, descrizione, data, luogo, accessibilità, partecipantiMAX, pieno, labels) **post:** return Event con le modifiche effettuate

Nome Metodo	**deleteEvent(eventID: int): void**
Descrizione	Elimina un evento creato
Pre-condizione	**context:** GestioneEventiService::deleteEvent(eventID) **pre:** eventID != none AND eventID != “”
Post-condizione	**context:** GestioneEventiService::deleteEvent(eventID) **post:** GestioneEventoDAO.delete(eventID, titolo, descrizione, data, luogo, accessibilità, partecipantiMAX, pieno, labels)

Nome Metodo	**registerToEvent(eventID: int, userID; int, pieno: bool): boolean**

Descrizione	Registra un utente ad un evento
Pre-condizione	**context:** GestioneEventiService::registerToEvent(eventID, userID, pieno) **pre:** eventID != none AND eventID != “” AND userID != none AND userID != “” AND pieno == false
Post-condizione	**context:** GestioneEventiService::deleteEvent(eventID) **post:** return True se la registrazione è avvenuta con successo OR False se i posti sono terminati

Nome Metodo	**searchEvents(titolo:str, descrizione:str, data:date, ora:datetime, luogo:str, accessibilità: str, labels: List[str]): List[Event]**
Descrizione	Cerca eventi che corrispondono ai parametri specificati.
Pre-condizione	**context:** GestioneEventiService::updateEvent(titolo, descrizione, data, ora, luogo, accessibilità, labels) **pre:** titolo != “” AND descrizione != “” AND data != none AND luogo != “” AND accessibilità != “”
Post-condizione	**context:** GestioneEventiService::updateEvent(titolo, descrizione, data, luogo, accessibilità, labels) **post:** return List di Event che rispettano i criteri di ricerca; la lista è vuota se non ci sono corrispondenze.

Nome Metodo	**getEventDetails(eventID: int): Event**
Descrizione	Recupera i dettagli di un evento
Pre-condizione	**context:** GestioneEventiService::getEventDetails(eventID) **pre:** eventID != none AND eventID != “”
Post-condizione	**context:** GestioneEventiService::getEventDetails(eventID) **post:** return Event OR none se non esiste.

Nome Metodo	**getEventParticipants(eventID: int): List[User]**

Descrizione	Recupera i partecipanti di un evento
Pre-condizione	**context:** GestioneEventiService::getEventParticipants(eventID) **pre:** eventID != none AND eventID != “”
Post-condizione	**context:** GestioneEventiService::getEventParticipants(eventID) **post:** return una lista di User iscritti all’evento.

Nome Metodo	**getAllEvents() : List[Event]**
Descrizione	Recupera tutti gli eventi
Pre-condizione	**context:** GestioneEventiService::getAllEvents() **pre:** esistono eventi
Post-condizione	**context:** GestioneEventiService::getAllEvents() **post:** return una lista di Event.

Nome Metodo	**removeParticipant(eventID: int, userID: int): void**
Descrizione	Rimuove la registrazione di un utente ad un evento
Pre-condizione	**context:** GestioneEventiService::removeParticipant(eventID, userID) **pre:** eventID != none AND eventID != “” AND userID != none AND userID != “”
Post-condizione	**context:** GestioneEventiService::removeParticipant(eventID, userID) **post:** registrazione annullata con successo

Class Interface: Messaggistica

Nome Classe	**GestioneMessaggisticaService**
Descrizione	Gestisce l'invio, la ricezione e l'archiviazione dei messaggi

	tra gli utenti.
Metodi	+**sendMessage**(senderID: int, receiverID: int, text: str): void +**getMessagesBetweenUsers**(user1ID: int, user2ID: int): List[Messages] **+notifyMessage**(receiverID:int, messageID:int): void **+sendMultimediaMessage**(senderID: int, receiverID: int, media: File): void +**deleteMessage**(messageID: int): void +**markMessageAsRead**(messageID: int, stato: str): void
Invarianti di classe	**/**

Nome Metodo	**sendMessage(senderID: int, receiverID: int, text: text): void**
Descrizione	Invia un messaggio da un utente a un altro.
Pre-condizione	**context:** GestioneMessaggisticaService::sendMessage(senderID, receiverID, text) **pre:** senderID != none AND receiverID != none AND text!= “”
Post-condizione	**context:** GestioneMessaggisticaService::sendMessage(senderID, receiverID, text) **post:** GestioneMessaggioDAO.save(senderID, receiverID, text)

Nome Metodo	**getMessagesBetweenUsers(user1ID: int, user2ID: int): List[Messages]**
Descrizione	Recupera la cronologia dei messaggi tra due utenti.
Pre-condizione	**context:** GestioneMessaggisticaService::getMessaggesBetweenUsers(user1ID, user2ID) **pre:** senderID != none AND receiverID != none AND text!= “”
Post-condizione	**context:** GestioneMessaggisticaService::getMessaggesBetweenUsers(user1ID, user2ID) **post:** return List di Messages tra due persone

Nome Metodo	notifyMessage(receiverID:int, messageID:int): void
Descrizione	Notifica un utente della ricezione di un messaggio.
Pre-condizione	**context:** GestioneMessaggisticaService::notifyMessage(receiverID, messageID) **pre:** AND receiverID != none AND messageID != none
Post-condizione	**context:** GestioneMessaggisticaService::notifyMessage(receiverID, messageID) **post:** notifica inviata al destinatario.

Nome Metodo	sendMultimediaMessage(senderID: int, receiverID: int, media: File): void
Descrizione	Invia un messaggio con file multimediale.
Pre-condizione	**context:** GestioneMessaggisticaService::sendMultimediaMessage(senderID, receiverID, media) **pre:** senderID != none AND receiverID != none AND media != none
Post-condizione	**context:** GestioneMessaggisticaService::sendMultimediaMessage(senderID, receiverID, media) **post:** GestioneMessaggioDAO.save(senderID, receiverID, media)

Nome Metodo	deleteMessage(messageID: int): void
Descrizione	Elimina un messaggio specifico.
Pre-condizione	**context:** GestioneMessaggisticaService::deleteMessage(messageID) **pre:** messageID != none AND messageID != “”
Post-condizione	**context:** GestioneMessaggisticaService::deleteMessage(messageID) **post:** GestioneMessaggioDAO.delete(messaggeID, text)

Nome Metodo	markMessageAsRead(messageID: int, stato: str): void
Descrizione	Segna un messaggio come letto.
Pre-condizione	**context:** GestioneMessaggisticaService::markMessageAsRead(messageID, stato) **pre:** messageID != none AND messageID != “” AND stato != “letto”
Post-condizione	**context:** GestioneMessaggisticaService::deleteMessage(messageID: int, stato: str) **post:** stato del Message aggiornato

Nome classe	GestioneAssistenzaUtenteService
Descrizione	Gestisce le richieste di supporto degli utenti.
Metodi	**+submitSupportRequest**(userID: int, issueDetails: str, attachments: List[File]): void +**getSupportRequests**(userID: int): List +**resolveSupportRequest**(requestID, status:str): Request **+deleteSupportRequest**(requestID: int): void
Invarianti di classe	**/**

Class Interface: AssistenzaUtente

Nome Metodo	submitSupportRequest(userID: int, issueDetails: str, attachments: List[File]): void
Descrizione	Invia una richiesta di supporto al team di assistenza, con allegati opzionali.
Pre-condizione	**context:** GestioneAssisstenzUtenteService::submitSupportRequest(userID, issueDetails, attachments) **pre:** userID != none AND userID != “” AND issueDetails != “”
Post-condizione	**context:** GestioneAssisstenzUtenteService::submitSupportRequest(userID,

	issueDetails, attachments) **post:** GestioneAssistenzaUtenteDAO.save(userID, issueDetails, attachments)

Nome Metodo	**getSupportRequests(userID: int): List**
Descrizione	Recupera tutte le richieste di supporto inviate da un utente.
Pre-condizione	**context:** GestioneAssisstenzUtenteService::getSupportRequest(userID) **pre:** userID != none AND userID != “”
Post-condizione	**context:** GestioneAssisstenzUtenteService::getSupportRequest(userID) **post:** return List di richieste di assistenza inviate da un utente

Nome Metodo	**resolveSupportRequest(requestID: int, status: str): Request**
Descrizione	Risolve una richiesta di supporto.
Pre-condizione	**context:** GestioneAssisstenzUtenteService::resolveSupportRequest(requestID: int, risolto: bool) **pre:** requestID != none AND requestID != “” AND status == “in progress”
Post-condizione	**context:** GestioneAssisstenzUtenteService::resolveSupportRequest(requestID: int, status: str) **post:** return Request con lo stato aggiornato

Nome Metodo	**deleteSupportRequest(requestID: int): void**
Descrizione	Elimina una richiesta di supporto.
Pre-condizione	**context:** GestioneAssisstenzUtenteService::deleteSupportRequest(requestID) **pre:** requestID != none AND requestID != “”
Post-condizione	**context:**

	GestioneAssistenzaUtenteService::deleteSupportRequest(requestID) **post:** GestioneAssistenzaUtenteDAO.delete(requestID, issueDetails)

Nome classe	**GestioneNotificheService**
Descrizione	Gestisce tutte le notifiche degli utenti.
Metodi	**+sendNotification**(userID:int, message:str): void **+getNotifications**(userID:int): List[Notification] **+markNotificationAsRead**(notificationID:int, letto:boolean): void **+deleteNotification**(notificationID:int): void
Invarianti di classe	**/**

Class Interface: Notifiche

Nome Metodo	**sendNotification(userID:int, messaggio:str): void**
Descrizione	Invia una notifica a un utente.
Pre-condizione	**context:** GestioneAssisstenzUtenteService::sendNotification(userID, messaggio) **pre:** userID != none AND userID != “” AND messaggio != “”
Post-condizione	**context:** GestioneAssisstenzUtenteService::sendNotification(userID, messaggio) **post:** GestioneNotificheDAO.save(userID, messaggio)

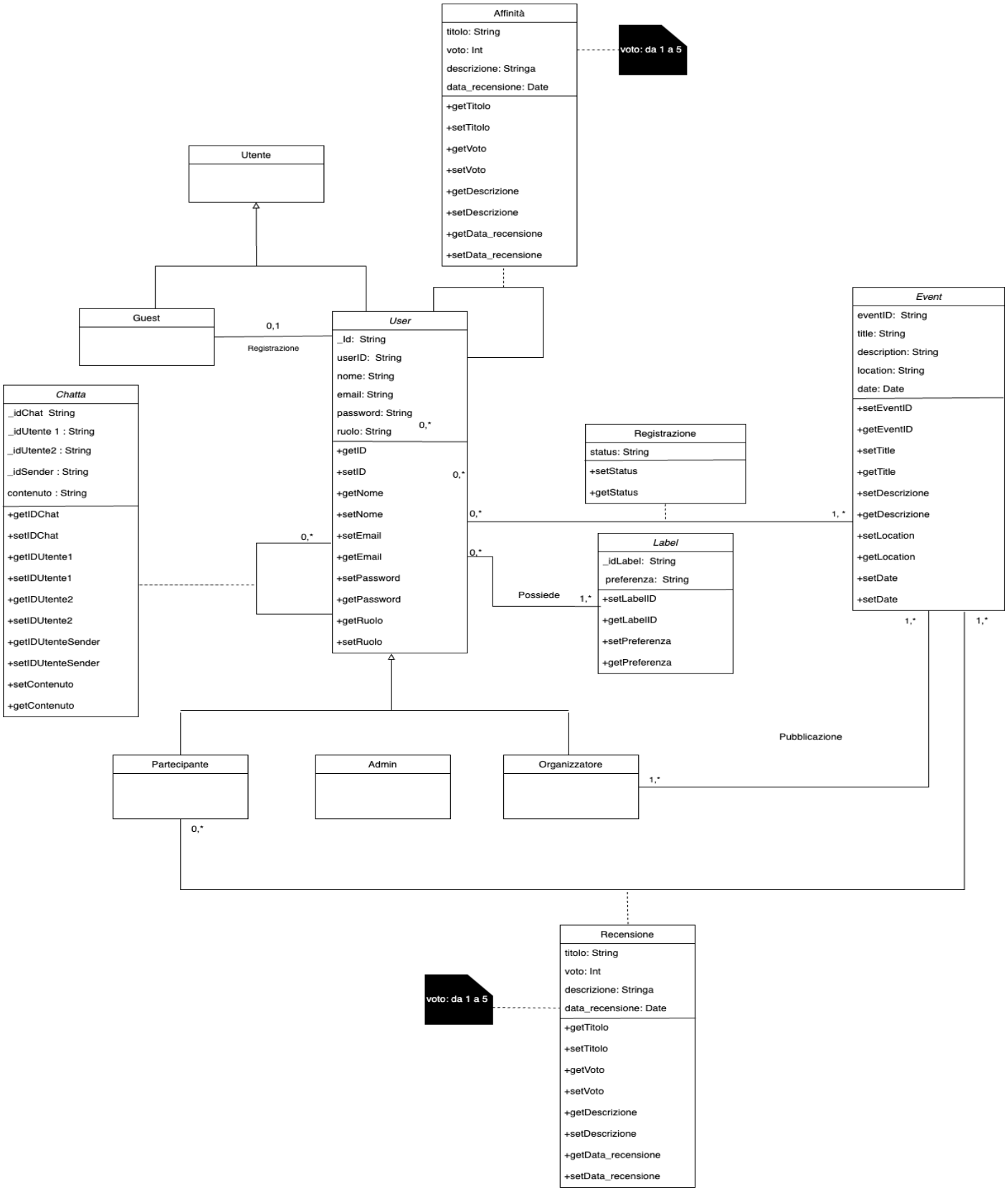
Nome Metodo	getNotifications(userID:int): List[Notification]
Descrizione	Recupera tutte le notifiche di un utente.
Pre-condizione	**context:** GestioneNotificheService::getNotifications(userID) **pre:** userID != none AND userID != “”
Post-condizione	**context:** GestioneNotificheService::getNotifications(userID) **post:** return List di Notification dell’utente

Nome Metodo	markNotificationAsRead(notificationID: int, letto:boolean): void
Descrizione	Segna una notifica come letta.
Pre-condizione	**context:** GestioneMessaggisticaService::markNotificationAsRead(notificationID, value) **pre:** notificationID != none AND notificationID != “” AND letto!= “true”
Post-condizione	**context:** GestioneMessaggisticaService::markNotificationAsRead(notificationID, letto) **post:** valore della Notification aggiornato

Nome Metodo	deleteNotification(notificationID:int): void
Descrizione	Elimina una notifica
Pre-condizione	**context:** GestioneNotificheService::deletetNotification(notificationID) **pre:** notificationID != none AND notificationID != “”
Post-condizione	**context:** GestioneNotificheService::deletetNotification(notificationID)

	post: notifica eliminata con successo

4. Class Diagram Ristrutturato



5. Design Patterns

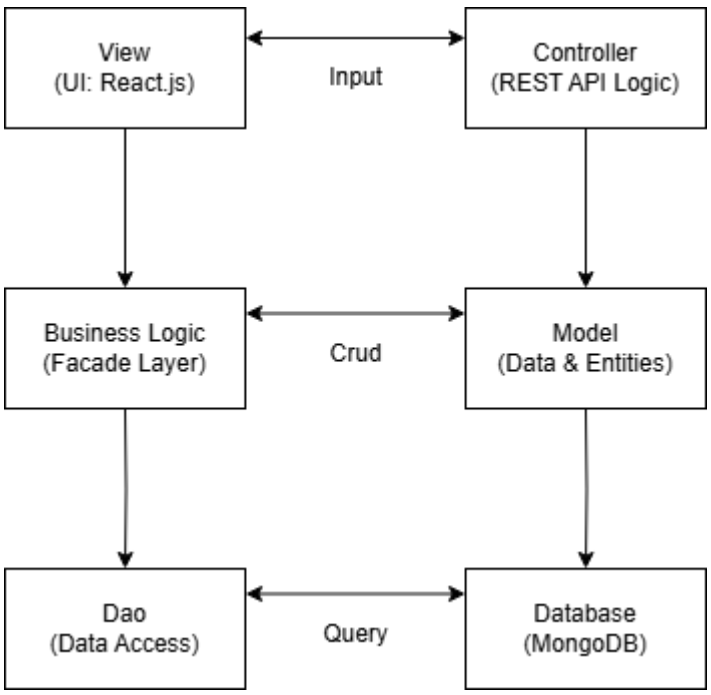
Introduzione ai Design Patterns

Nel progetto SocialBridge, per garantire una gestione efficiente, manutenibile e scalabile delle funzionalità, sono stati utilizzati due design patterns principali: Facade e Model-View-Controller (MVC).

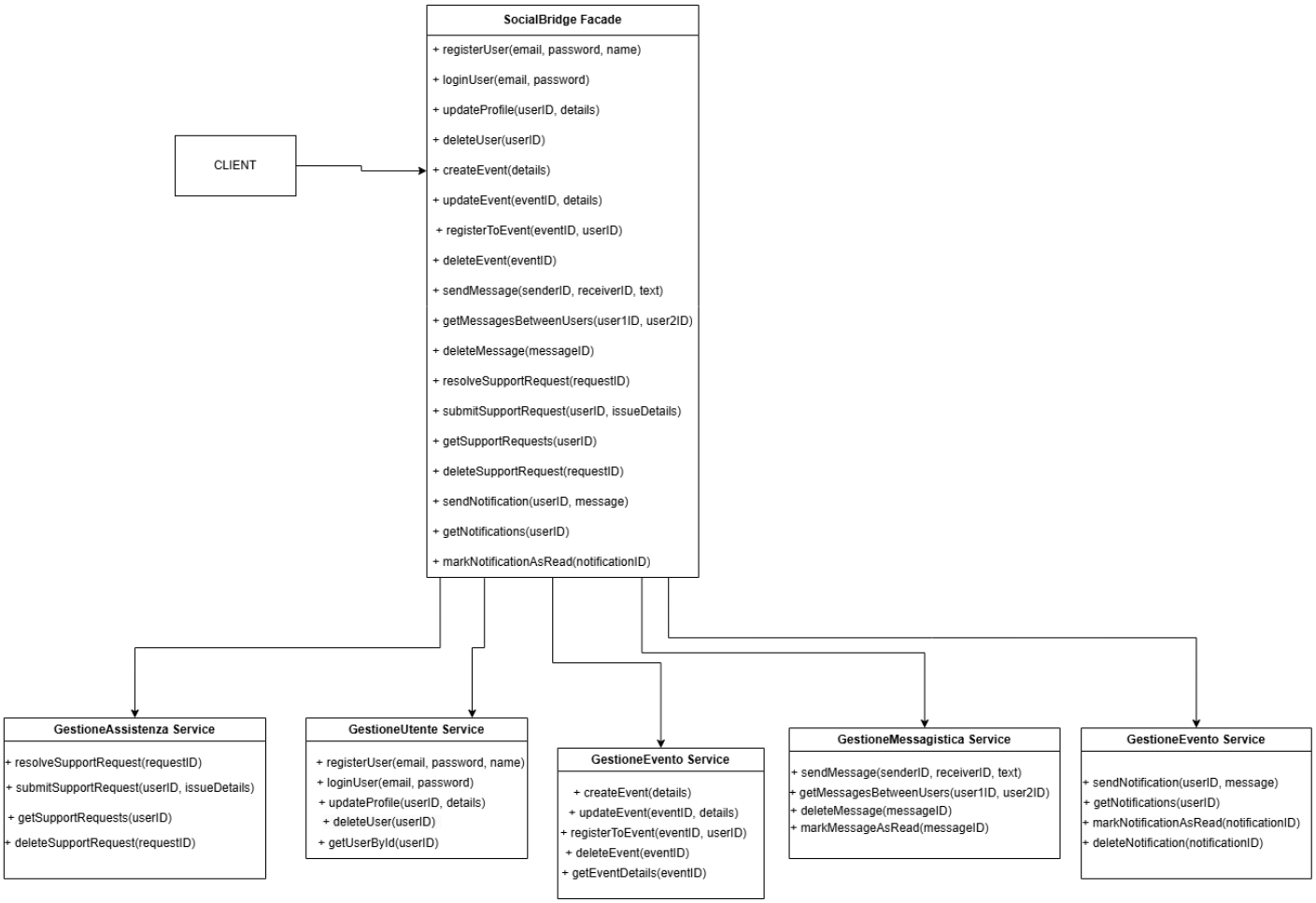
- **Façade**: Il pattern Facade viene utilizzato per fornire un'interfaccia unificata e semplificata per l'accesso a sottosistemi complessi, come la gestione degli utenti, degli eventi, dei messaggi e delle notifiche. Questo pattern aiuta a separare la logica di business dalle operazioni interne di ciascun sottosistema, migliorando la manutenibilità e la comprensione del sistema da parte degli sviluppatori.
- **Model-View-Controller (MVC)**: Il pattern MVC viene adottato per separare la logica di presentazione dalla logica di business e gestione dei dati. In un'applicazione web come SocialBridge, il modello MVC garantisce che la View (interfaccia utente) sia separata dal Model (gestione dei dati) e dal Controller (che gestisce il flusso di dati tra la View e il Model). Questa separazione migliora la modularità e rende il sistema facilmente estendibile.

Questi due pattern, pur svolgendo funzioni distinte, lavorano insieme per migliorare l'architettura complessiva del sistema, garantendo che il flusso di dati e la gestione delle interfacce siano ben separati, mantenendo una chiara separazione delle preoccupazioni e una gestione efficiente della logica di business.

MVC Design Patterns:



Facade Patterns:



6. Glossario

Di seguito un'utile raccolta di termini chiave e relative definizioni per l'aiuto della comprensione di concetti specifici. Una guida rapida e chiara attraverso il linguaggio tecnico, fornendo una panoramica esaustiva e accessibile a tutti. Ha lo scopo di fornire chiarezza ed una migliore comprensione dei termini essenziali che sono stati utilizzati all'interno del ODD.

Package: Raccolta di risorse o file correlati organizzati in una directory. Contiene codice sorgente, librerie, dati, configurazioni o altri elementi necessari allo sviluppo, distribuzione o esecuzione del sistema.

Design Patterns: Modelli architetturali o strutturali che forniscono soluzioni standardizzate per problemi comuni di progettazione del software, migliorando l'organizzazione e la manutenibilità del codice.

MVC (Model-View-Controller): Un design pattern utilizzato per separare la logica di presentazione (View), la logica di gestione dei dati (Model) e la logica di controllo (Controller), garantendo modularità e scalabilità.

Façade: Un design pattern che fornisce un'interfaccia semplificata per l'accesso a funzionalità complesse di un sottosistema, migliorandone l'uso e la manutenibilità.

DAO (Data Access Object): Un pattern architetturale che separa la logica di accesso ai dati dalla logica di business, migliorando la manutenibilità e la sicurezza.

React.js: Framework JavaScript utilizzato per sviluppare interfacce utente dinamiche e modulari. Node.js: Runtime JavaScript per lo sviluppo di applicazioni lato server, noto per la sua efficienza e scalabilità.

MongoDB: Database NoSQL orientato ai documenti, ideale per la memorizzazione dinamica di dati non strutturati.

WCAG (Web Content Accessibility Guidelines): Linee guida per garantire l'accessibilità dei contenuti web a persone con disabilità.