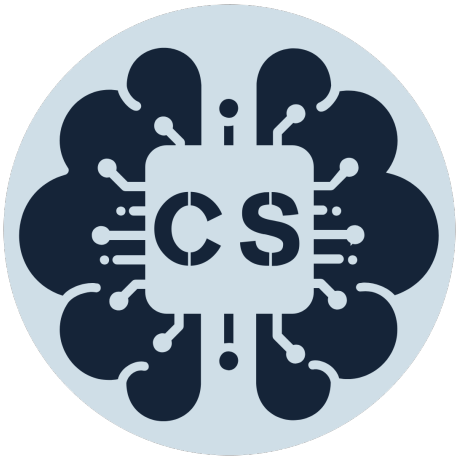
Specifica Tecnica

CyberSorcerers Team



Informazioni sul documento		
Destinatari:	Prof Tullio Vardanega	Prof Riccardo Cardin
G al pedice:	Consultare il Glossario	

Membri del team:
Sabrina Caniato
Giulia Dentone
Nicola Lazzarin
Giovanni Moretti
Andrea Rezzi
Samuele Vignotto

Registro dei Cambiamenti - Changelog$_G$

Versione	Data	Autore	Verificatore	Dettaglio
1.0.0	28/05/2024	Giulia Dentone	Samuele Vignotto	Verifica finale documento.
0.8.0	27/05/2024	Andrea Rezzi	Giovanni Moretti	Stesura della sezione 'Architettura Back-end'.
0.7.0	27/05/2024	Samuele Vignotto	Giulia Dentone	Stesura della sezione 'Architettura front-end Login e Password'.
0.6.0	27/05/2024	Sabrina Caniato	Giovanni Moretti	Stesura della sezione 'Architettura Front-End'.
0.5.0	26/05/2024	Sabrina Caniato	Giulia Dentone	Stesura della sezione 'Architettura Libreria'.
0.4.0	26/05/2024	Giovanni Moretti	Sabrina Caniato	Stesura della sezione 'Architettura Plugin'.
0.3.0	25/05/2024	Sabrina Caniato	Andrea Rezzi	Stesura della sezione 'Schema del Database'.
0.2.0	23/05/2024	Samuele Vignotto	Giovanni Moretti	Stesura della sezione 'Requisiti soddisfatti'.
0.1.1	23/05/2024	Giulia Dentone	Sabrina Caniato	Stesura della sezione 'Tecnologie'.
0.0.1	03/05/2024	Giulia Dentone	Samuele Vignotto	Definizione struttura del documento e scheletro delle sezioni. Scrittura introduzione ed obiettivi delle diverse sezioni.

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

1.1 Scopo del documento

Questo documento ha lo scopo di delineare e giustificare le decisioni architetturali prese durante le fasi di progettazione e sviluppo del prodotto. Sono presentati i diagrammi dei componenti React e dei pacchetti per illustrare le scelte dei pattern architetturali adottati per realizzare la struttura finale del prodotto. Inoltre, viene fornita una sezione dedicata ai requisiti soddisfatti dal team, offrendo così una panoramica completa dello stato di avanzamento del lavoro.

1.2 Scopo del prodotto

L'azienda proponente ha richiesto la creazione di una web app$_G$ che, tramite l'uso di IA$_G$ (in questo caso ChatGPT4 e Bedrock) è in grado di creare epic user stories$_G$ a partire dalle richieste del cliente e confrontarle con il codice sviluppato in modo da informare il cliente dello stato di avanzamento dello sviluppo del prodotto. Inoltre deve essere possibile, sia per il Project Manager$_G$, sia per il cliente rilasciare dei feedback (nel primo caso riguardanti l'adeguatezza delle stories, nel secondo caso riguardanti il prodotto finale) al fine di migliorare l'IA$_G$. È inoltre richiesta un' analisi comparativa tra le due IA$_G$ utilizzate e lo sviluppo di un plug-in$_G$ utile agli sviluppatori e al Project Manager$_G$.

1.3 Glossario

Alcuni termini presenti nel documento potrebbero essere ambigui, pertanto verranno inseriti nel Glossario v.1.0.0. La loro presenza all'interno di esso sarà indicata tramite una G maiuscola a pedice.

2 Riferimenti

2.1 Riferimenti normativi

- Capitolato **C7 - ChatGPT vs BedRock developer Analysis**

  https://github.com/CyberSorceres/CyberSorceresRepository
- Norme del way of working v 1.0.0
- Regolamento del progetto didattico

  https://www.math.unipd.it/ tullio/IS-1/2023/Dispense/PD2.pdf

2.2 Riferimenti informativi

- Slide del corso di Ingegneria del Software - Analisi dei requisiti

  https://www.math.unipd.it/ tullio/IS-1/2023/Dispense/T5.pdf
- Slide del corso di Ingegneria del Software - Progettazione e programmazione: Diagrammi delle classi

  https://www.math.unipd.it/ rcardin/swea/2023/Diagrammi%20delle%20Classi.pdf
- Slide del corso di Ingegneria del Software - Solid Programming

  https://www.math.unipd.it/ rcardin/swea/2021/SOLID%20Principles%20of%20Object-Oriented%20Design_4x4

2.3 Riferimenti tecnici

- Documentazione di React
  https://react.dev/
- Documentazione di Typescript
  https://www.typescriptlang.org/docs/
- Documentazione di MongoDB
  https://www.mongodb.com/docs/
- Documentazione di Amazon AWS
  https://docs.aws.amazon.com/it_it/
- Serverless Microservice Patterns
  https://medium.com/@jeremydaly/serverless-microservice-patterns-for-aws-6dadcd21bc02
- Aws Reference Architecture Diagrams
  https://aws.amazon.com/it/architecture/reference-architecture-diagrams
- React design patterns
  https://refine.dev/blog/react-design-patterns/

3 Tecnologie

In questa sezione è presente una panoramica generale delle tecnologie necessarie per la realizzazione del prodotto (in particolare del front-end, back-end,databse e plugIn), gli strumenti e le librerie utilizzate per lo sviluppo, il testing e la distribuzione.

3.1 Tecnologie per la codifica

Tecnologia	Descrizione	Versione
Linguaggi		
HTML	Linguaggio di markup per delineare la struttura delle pagine e definire i componenti dell'interfaccia.	5
CSS	Linguaggio per la gestione dello stile degli HTML	3
Typescript	Superset di JavaScript per utilizzare tipizzazione	5.0.x
Framework		
React	Libreria grafica per lo sviluppo front-end che permette di gestire le unità grafiche in maniera modulare	18.0.x
Servizi e strumenti		
Node.js	Ambiente di runtime open-source per l'esecuzione di codice JavaScript lato server tramite appositi script.	19.0.x
NPM	Gestore dell'installazione della gestione dei pacchetti utilizzati in TypeScript e nell'ambiente di esecuzione Node.js.	3
AWS Cognito	Servizio di gestione dell'autenticazione.	2023-16-02
AWS MongoDB	Servizio di database non relazionale gestito in modo scalabile.	2019-11-21
AWS Lambda	Servizio che consente di eseguire codice in maniera serverless, garantendo la scalabilità automatica durante l'esecuzione.	2023-03-16
AWS API Gateway	Servizio di gestione (creazione, pubblicazione e protezione) delle API.	2023-04-06
Git	Sistema di controllo del versionamento e della gestione del codice.	2.4.x

Tabella 1: Tabella delle tecnologie per la codifica

3.2 Tecnologie per l'analisi del codice

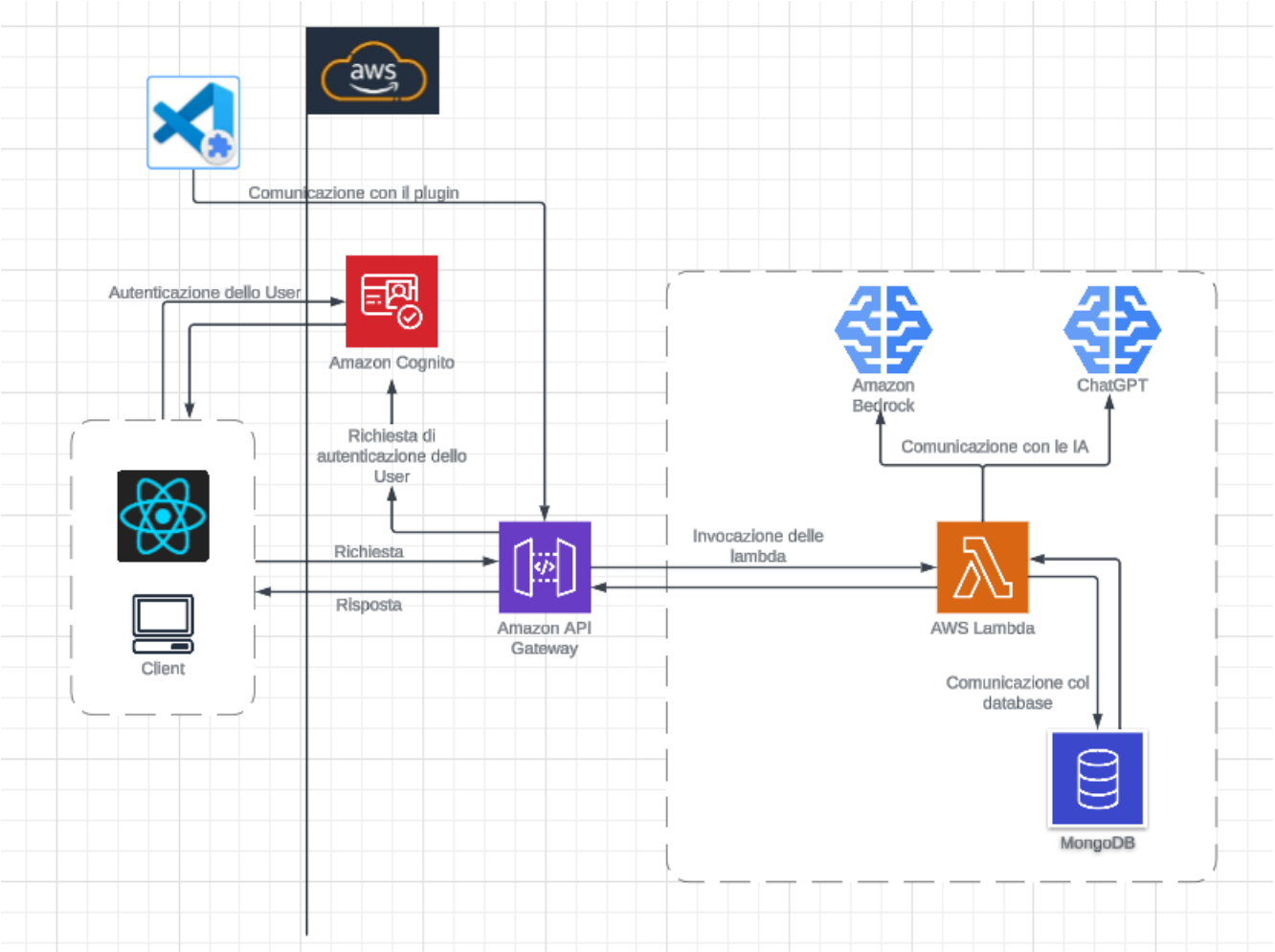
Tecnologia	Descrizione	Versione
Analisi statica		
Prettier	Strumento di formattazione del codice che mantiene lo stile di codifica coerente e leggibile.	3.0.x
ViTest	Framework di test per TypeScript che permette la creazione di mock e il testing del codice in modo asincrono.	1.6.0
Analisi dinamica		
React Testing Library	Libreria di test che consente di testare il comportamento dei componenti React da una prospettiva degli utenti finali.	14.0.x
Github Actions	Piattaforma di integrazione e distribuzione continua per automatizzare flussi di lavoro software come build, test e deployment direttamente su GitHub.	/
Postman	è un'applicazione software utilizzata per il testing delle API	10.16

4 Architettura

Nella fase di progettazione è stata scelta un'architettura a microservizi come la più conforme alle carateristiche di funzionamento e strutturali delle tecnologie AWS, soprattutto API Gateway. Inoltre è quella che consente nel nostro caso la comunicazione per consentire la comunicazione tra le diverse componenti, considerando anche la presenza di un plugIn. Dati i requisiti del nostro progetto, abbiamo deciso che non fosse adeguato adottare un'unica struttura architetturale per l'intera l'infrastruttura. Abbiamo suddiviso il sistema in tre parti principali:

- Front-end: la parte client dell'applicazione eseguibile localmente su qualsiasi browser.
- Back-end: utilizza le tecnologie AWS (listate nella sezione 3.1), con l'interazione tramite NodeJS lato client e la comunicazione con il plugIn.
- PlugIn: comunica con il back-end grazie ad una libreria sviluppata dal team.

La comunicazione tra le diverse parti avviene attraverso l'uso di API Gateway.



Schema dei componenti

4.1 Architettura front-end

L'architettura del frontend$_G$ della nostra applicazione è stata progettata per essere modulare e scalabile, utilizzando tecnologie moderne e seguendo i migliori pattern di design per garantire un'alta manutenibilità e facilità di sviluppo. Il frontend$_G$ è realizzato utilizzando React$_G$ e TypeScript$_G$, che offrono un ambiente robusto per la costruzione di interfacce utente interattive e reattive.

4.1.1 Pattern utilizzati

Per garantire un'architettura modulare e mantenibile, sono stati utilizzati diversi pattern di design tipici dello sviluppo con React$_G$:

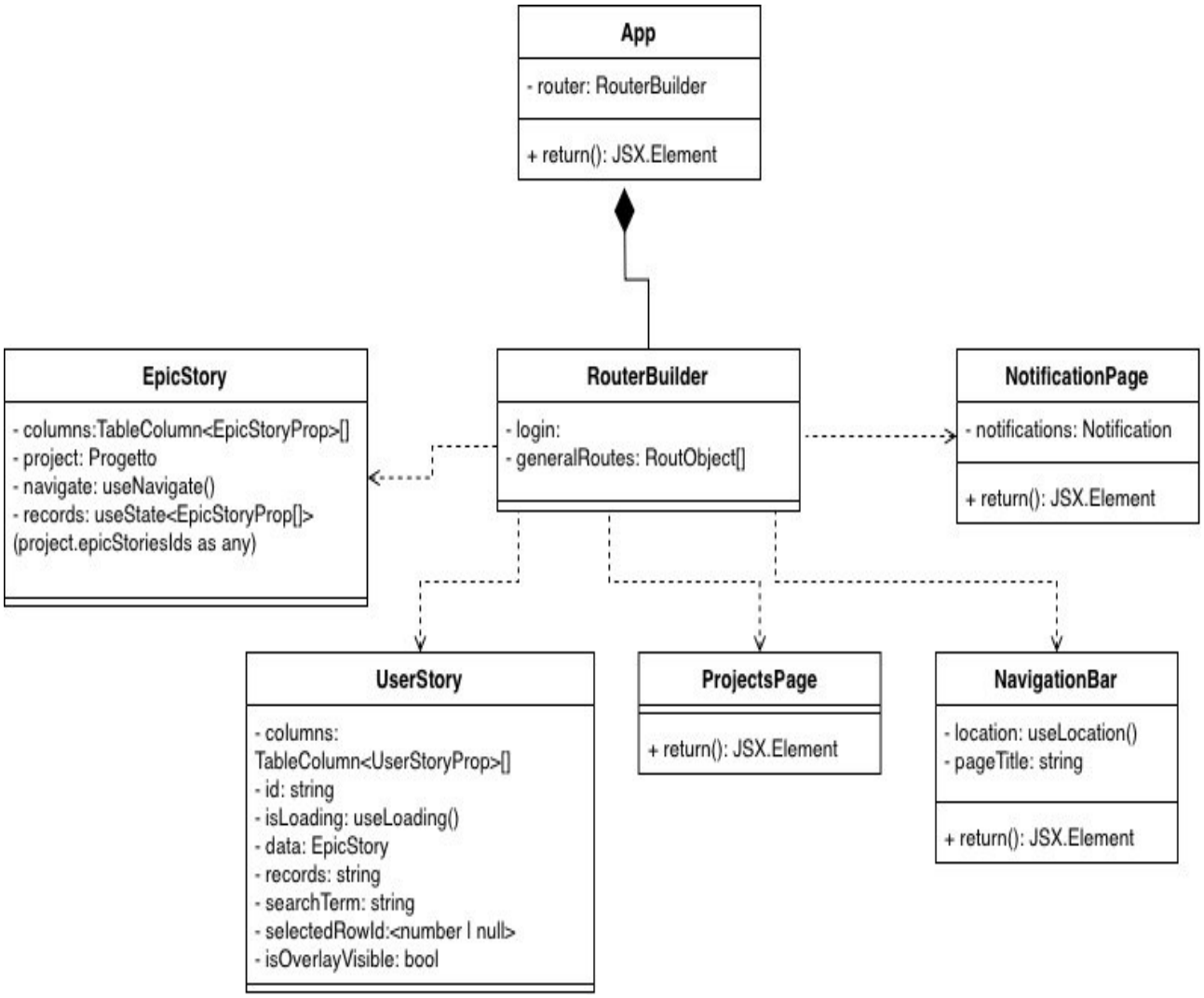
- **Controlled Components**: I valori di input dei componenti vengono gestiti direttamente e passati dai componenti "parent" ai componenti "children" sotto forma di props, garantendo che i dati siano costanti tra tutti i componenti.
- **Presentational and Container Components**: La logica di presentazione è separata dalla logica di business, realizzando componenti React$_G$ che si occupano di renderizzare una vista e utilizzandoli all'interno di un componente unico.

- **React Hooks**: Utilizzati per gestire lo stato dell'applicazione in modo efficiente e per aggiornare dinamicamente le informazioni. Sono stati utilizzati hooks$_G$ standard di React (useState, useEffect, useContext) e hooks$_G$ personalizzati in base alle esigenze delle singole viste e componenti (useLoaderData, useNavigate).
- **Provider**: Racchiude l'intera applicazione per fornire il componente di autenticazione a tutti i componenti, semplificando la gestione dell'autenticazione e delle autorizzazioni.
- **Conditional Rendering**: Consente di mostrare un contenuto diverso a seconda del ruolo dell'utente, rendendo l'interfaccia più dinamica e adattabile.
- **Compound Components**: Utilizzati per modularizzare le singole componenti, creando una gerarchia padre-figlio dove il componente padre contiene uno o più componenti figlio, specializzando la gestione dei dati e la personalizzazione dell'interfaccia utente in modo centralizzato.

4.1.2 Diagrammi delle classi

La sezione seguente descrive le singole pagine per i vari utenti utilizzando la sintassi UML$_G$ e una descrizione testuale appropriata. Questo serve a rendere ogni scelta progettuale del gruppo Cybersorceres chiara e facilmente interpretabile durante il periodo di sviluppo.

4.2 App



UML App

4.3 App

Descrizione

Il componente **App** costituisce il componente principale dell'applicazione. È responsabile per l'inizializzazione del contesto API e la gestione del routing.

Variabili di Stato

- **api**: un'istanza dell'oggetto API che gestisce le richieste verso il server.

Funzioni Principali

- **createAPI**: funzione per la creazione dell'oggetto API con il token memorizzato nel localStorage, se disponibile.

Rendering

Il componente **App** rende il contesto API all'interno di un `APIContext.Provider`, che è accessibile a tutti i componenti all'interno dell'applicazione. Inoltre, utilizza il componente `RouterProvider` per fornire il router creato tramite `createBrowserRouter`.

RouterBuilder

Descrizione

La funzione **RouterBuilder** è responsabile per la costruzione delle route dell'applicazione. Utilizza l'oggetto API fornito come argomento per determinare le route disponibili e configurare i loro componenti e loader.

Parametri

- **api**: un'istanza dell'oggetto API che gestisce le richieste verso il server.

Variabili Locali

- **login**: una variabile di stato che indica se l'utente è autenticato.
- **generalRoutes**: un array di oggetti che rappresentano le route generali dell'applicazione.

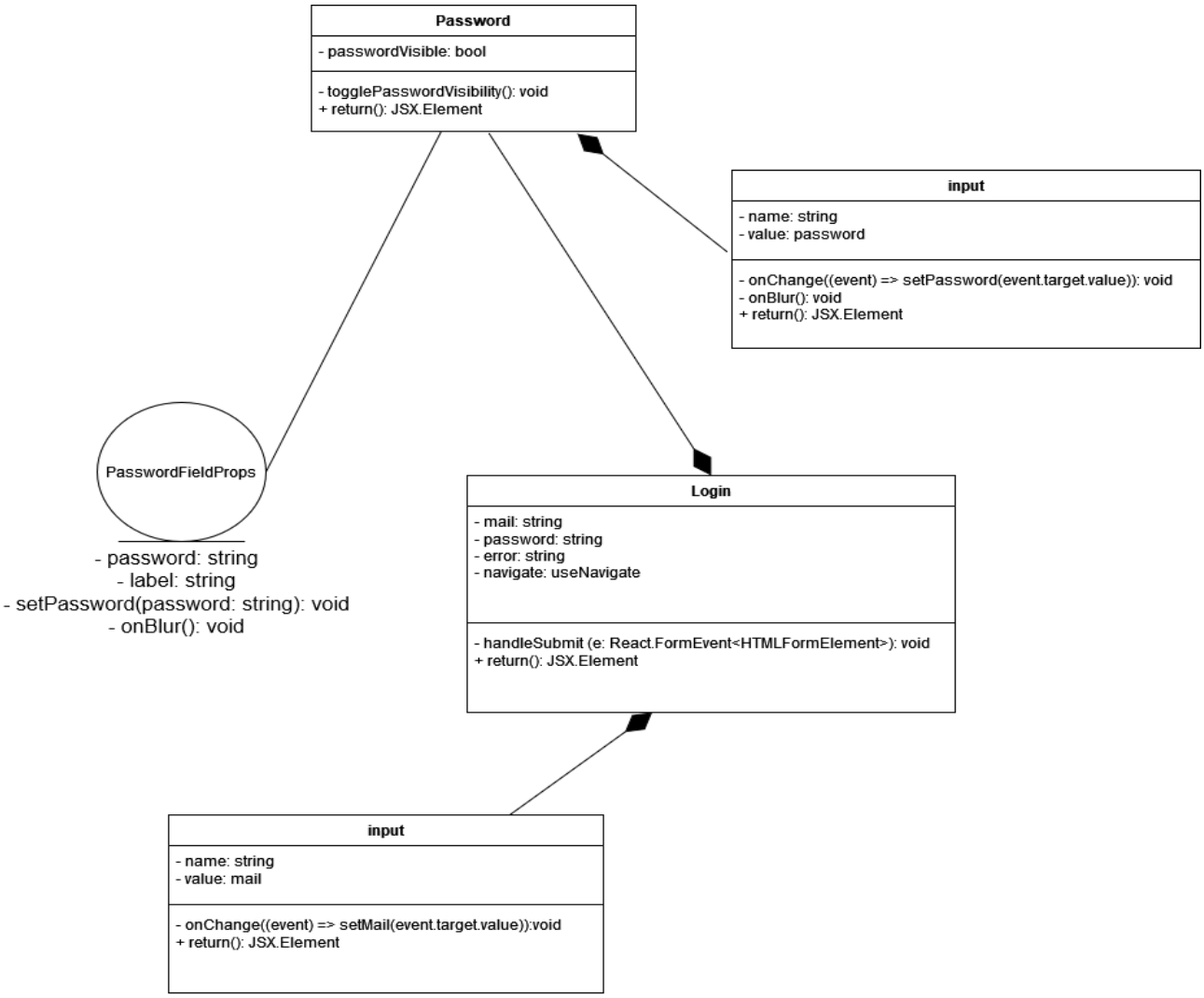
Funzioni Principali

- **setLogin**: funzione per impostare lo stato di autenticazione dell'utente.

Rendering

La funzione **RouterBuilder** restituisce un array di route che definiscono la struttura dell'applicazione. Queste route includono le route generali e i loro componenti corrispondenti. Le route vengono renderizzate all'interno di un componente `Suspense`, che fornisce un fallback durante il caricamento asincrono delle route.

4.4 Login



UML Login

Il componente `Login` è una funzione che prende una proprietà `onLogin` come parametro. Questo componente gestisce lo stato dell'email, della password e degli errori di login.

Stato e Contesto

- `mail`: Memorizza l'email inserita dall'utente.
- `password`: Memorizza la password inserita dall'utente.
- `error`: Memorizza eventuali messaggi di errore.

Funzione handleSubmit

La funzione `handleSubmit` è chiamata quando l'utente invia il modulo di login:

1. Previene il comportamento predefinito del modulo.
2. Esegue un tentativo di login con l'API usando le credenziali fornite.

3. Se il login ha successo (`LoginState.LOGGED_IN`):
    - Chiama la funzione `onLogin`.
    - Memorizza il token nel `localStorage`.
    - Naviga alla pagina principale.
4. Se l'utente deve registrarsi (`LoginState.MUST_SIGN_UP`):
    - Naviga alla pagina di cambio password con l'email dell'utente.
5. In caso di fallimento, imposta un messaggio di errore.

Funzione handleBlur
La funzione `handleBlur` azzera il messaggio di errore quando l'utente rimuove il focus dall'input.

Rendering
Il componente restituisce il markup JSX della pagina di login:

- Un'intestazione "Login".
- Un modulo con:
    - Un campo per l'email.
    - Un componente `Password` per l'input della password, con una funzione `onBlur` per gestire il focus.
    - Un messaggio di errore (se presente).
    - Un pulsante di submit.
- Un link per reimpostare la password in caso di password dimenticata.

Interfaccia PasswordFieldProps
Definisce le proprietà che il componente `Password` si aspetta di ricevere:

- **password**: La stringa della password attuale.
- **setPassword**: Una funzione per aggiornare la password.
- **label**: Un'etichetta per il campo password.
- **onBlur**: Una funzione chiamata quando l'input perde il focus.

Componente Password
Il componente `Password` è una funzione di tipo `React.FC<PasswordFieldProps>` che utilizza le proprietà definite dall'interfaccia `PasswordFieldProps`.

Stato

- **passwordVisible**: Uno stato booleano che determina se la password è visibile o meno. Inizialmente è impostato su `false`.

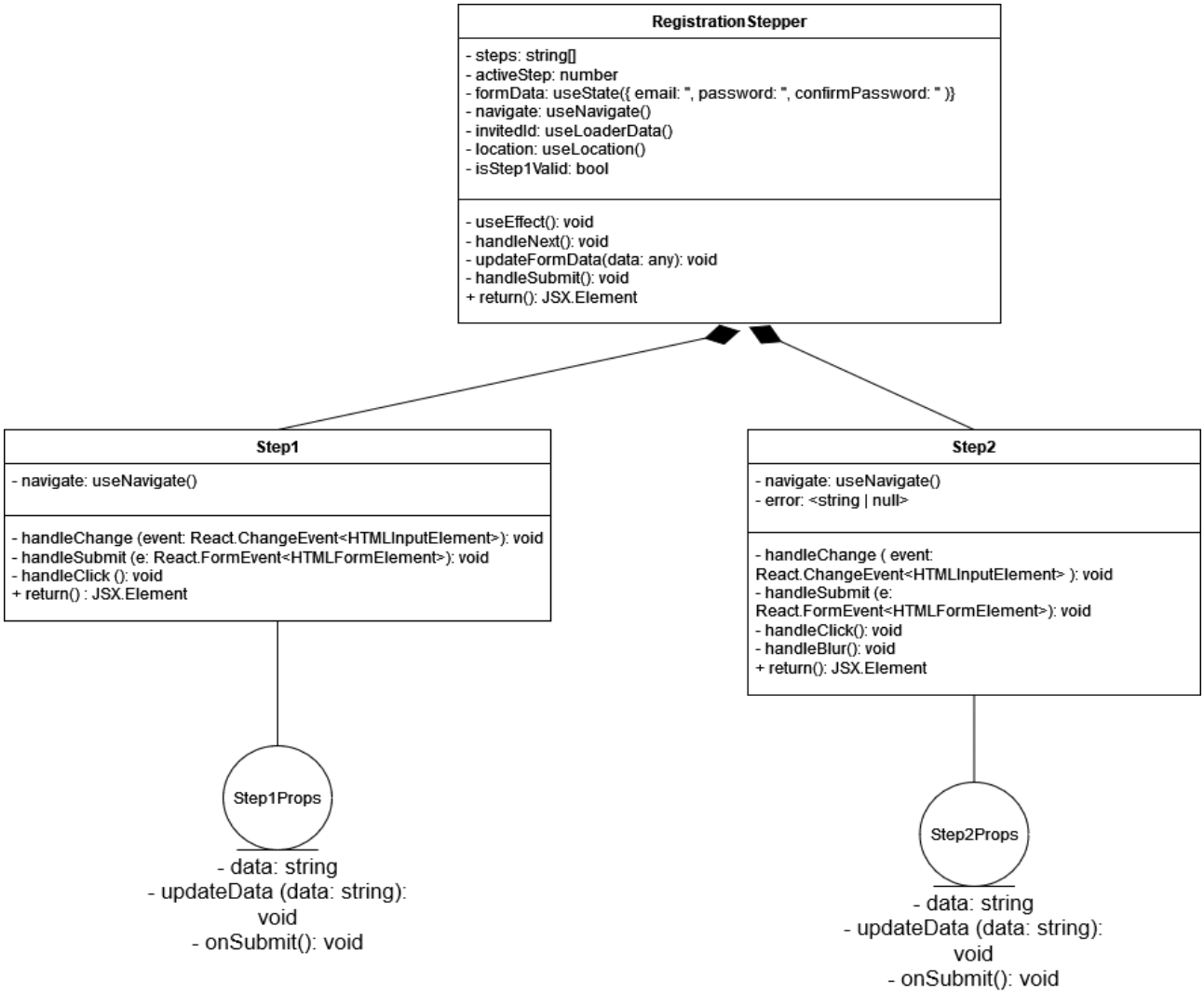
Funzioni

- **togglePasswordVisibility**: Questa funzione inverte lo stato di `passwordVisible` quando viene chiamata, alternando la visibilità della password.

Rendering
Il componente restituisce il markup JSX del campo password:

- Una `label` per il campo password, con il testo fornito dalla proprietà `label`.
- Un `div` contenitore per l'input della password e l'icona:
  - Un `input` con le seguenti proprietà:
    - **type**: Determina il tipo di input, che può essere `text` o `password` a seconda del valore di `passwordVisible`.
    - **id** e **name**: Identificatori per il campo.
    - **className**: Classe CSS per lo stile.
    - **value**: Il valore della password, gestito dallo stato esterno.
    - **onChange**: Una funzione chiamata quando il valore dell'input cambia, che aggiorna lo stato della password.
    - **onBlur**: Una funzione chiamata quando l'input perde il focus, fornita dalle proprietà del componente.
    - **required**: Specifica che il campo è obbligatorio.
  - Un'icona `FontAwesomeIcon` che cambia tra `faEye` e `faEyeSlash` a seconda dello stato di `passwordVisible`. Quando l'icona viene cliccata, chiama `togglePasswordVisibility` per alternare la visibilità della password.

Cambio password



UML Cambio password

Il componente `RegistrationStepper` è una funzione di tipo `React.FC` che gestisce il processo di registrazione a più fasi. **Stato**

- `activeStep`: Memorizza il passo attivo corrente dello stepper.
- `formData`: Memorizza i dati del modulo di registrazione, inclusi `email`, `password` e `confirmPassword`.
- `isStep1Valid`: Memorizza lo stato di validità del primo passo.

Navigazione e Dati

- `navigate`: Hook per navigare tra le pagine.
- `inviteId`: Dati caricati utilizzando `useLoaderData`.
- `location`: Hook per ottenere la posizione corrente.

Effetti
`useEffect` viene utilizzato per validare i dati del primo passo:

- Se `email` non è vuoto, `isStep1Valid` viene impostato su `true`.
- Altrimenti, `isStep1Valid` viene impostato su `false`.

Funzioni

- **handleNext**: Avanza al passo successivo dello stepper se il passo attivo è inferiore alla lunghezza dei passi.
- **updateFormData**: Aggiorna i dati del modulo di registrazione.
- **handleSubmit**: Esegue la registrazione dell'utente, il login e l'accettazione dell'invito.

Rendering Il componente restituisce il markup JSX dello stepper di registrazione:

- Un `Box` contenente:
  - Un'intestazione "Registrazione".
  - Un componente `Stepper` con passi definiti.
  - Un componente condizionale che rende `Step1` o `Step2` a seconda del passo attivo.
  - Un `Box` per contenere i pulsanti di navigazione (non utilizzato in questo esempio).

Componente Step1
Il componente `Step1` rappresenta il primo passo del processo di cambio password. **Interfaccia Step1Props** Definisce le proprietà che il componente `Step1` si aspetta di ricevere:

- `data`: Un oggetto contenente l'email.
- `updateData`: Una funzione per aggiornare i dati del modulo.
- `onSubmit`: Una funzione chiamata quando il modulo viene inviato.

Funzioni

- **handleChange**: Aggiorna l'email nei dati del modulo quando cambia il valore dell'input.
- **handleSubmit**: Previene il comportamento predefinito del modulo e chiama la funzione `onSubmit`.
- **handleClick**: Naviga alla pagina di login.

Rendering
Il componente restituisce il markup JSX del primo passo della modifica password:

- Un `div` contenitore con una classe `container`.
- Un `form` con:
  - Una `label` per l'email.
  - Un `input` per inserire l'email.
  - Un pulsante per inviare il modulo (Successivo).
  - Un pulsante per navigare indietro (Indietro).

Componente Step2
Il componente `Step2` rappresenta il secondo passo del processo di cambio password. **Interfaccia Step2Props** Definisce le proprietà che il componente `Step2` si aspetta di ricevere:

- `data`: Un oggetto contenente `password` e `confirmPassword`.
- `updateData`: Una funzione per aggiornare i dati del modulo.
- `onSubmit`: Una funzione chiamata quando il modulo viene inviato.

Funzioni

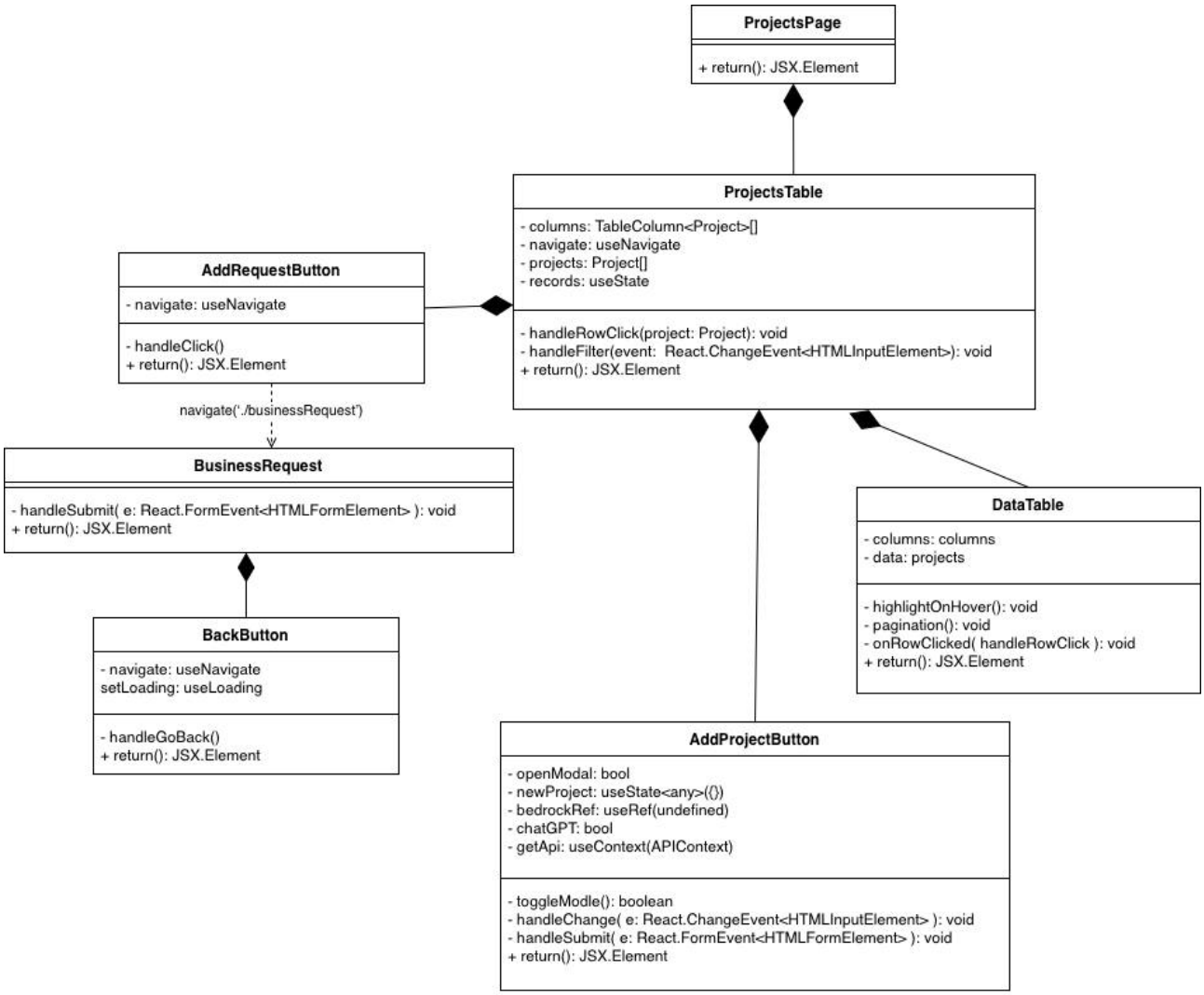
- **handleChange**: Aggiorna `password` e `confirmPassword` nei dati del modulo quando cambia il valore dell'input.
- **handleSubmit**: Previene il comportamento predefinito del modulo e controlla se `password` e `confirmPassword` corrispondono. Se non corrispondono, imposta un messaggio di errore. Altrimenti, chiama la funzione `onSubmit`.
- **handleClick**: Naviga al primo passo della registrazione.
- **handleBlur**: Azzera il messaggio di errore quando l'input perde il focus.

Rendering

Il componente restituisce il markup JSX del secondo passo della modifica password:

- Un `div` contenitore.
- Un `form` con:
  - Due componenti `Password` per inserire `password` e `confirmPassword`.
  - Un messaggio di errore (se presente).
  - Un pulsante per inviare il modulo (Invia).
  - Un pulsante per navigare indietro (Indietro).

4.5 Table



UML Progetto

Il componente `ProjectsTable` è un componente React che visualizza una tabella di progetti utilizzando il pacchetto `react-data-table-component`. Il componente permette all'utente di filtrare i progetti tramite un campo di ricerca e di navigare ai dettagli di un progetto selezionato. Il componente Table corrisponde alla parte principale della web app. Qua si possono vedere tutti i progetti, in più solo il Project Manager può creare nuovi progetti e solo il cliente potrà proporre i requisiti di business.

Stato Interno

- `records`: Stato che memorizza i progetti filtrati visualizzati nella tabella.

Contesto

- `projects`: Dati dei progetti caricati tramite `useLoaderData`.
- `isLoading`: Stato del caricamento gestito dal contesto `LoadingContext`.

Funzione `handleFilter`

La funzione `handleFilter` aggiorna lo stato `records` in base al testo inserito dall'utente nel campo di ricerca.

Effetti

- `useEffect`: Gestisce lo stato di caricamento in base allo stato della navigazione.

Rendering

Il componente restituisce il markup JSX della tabella dei progetti:

- `AddProjectButton`: Pulsante per aggiungere un nuovo progetto.
- `AddRequestButton`: Pulsante per aggiungere una nuova richiesta.
- `input`: Campo di ricerca per filtrare i progetti.
- `ClipLoader`: Spinner di caricamento visualizzato durante l'operazione di caricamento.
- `DataTable`: Componente della tabella che visualizza i progetti.

Colonne della Tabella

- `Titolo`: Nome del progetto.
- `Cliente`: Nome del cliente associato al progetto.
- `Data di Inizio`: Data di inizio del progetto.
- `Progress`: Stato di avanzamento del progetto visualizzato come barra di progresso.

Gestione del Click sulla Riga

Quando una riga della tabella viene cliccata, l'utente viene navigato alla pagina dei dettagli del progetto.

AddProjectButton

Il componente `AddProjectButton` è un componente React che permette all'utente di creare un nuovo progetto tramite un modal. Utilizza i componenti di `mdb-react-ui-kit` per la costruzione del modal e dei suoi elementi interattivi.

Stato Interno

- `openModal`: Booleano che gestisce l'apertura e la chiusura del modal.
- `newProject`: Oggetto che contiene i dati del nuovo progetto da creare.
- `bedrockRef`: Riferimento al componente radio per determinare l'AI scelta.
- `chatGPT`: Booleano che gestisce lo stato della scelta dell'AI.

Funzione `toggleModal`

La funzione `toggleModal` gestisce l'apertura e la chiusura del modal.

Funzione `handleChange`

La funzione `handleChange` aggiorna lo stato `newProject` in base ai dati inseriti dall'utente nei campi di input.

Funzione `handleSubmit`

La funzione `handleSubmit` gestisce l'invio del form e l'aggiunta del nuovo progetto tramite l'API.

Rendering

Il componente restituisce il markup JSX del pulsante e del modal per la creazione del nuovo progetto:

- `button`: Pulsante per aprire il modal.
- `MDBModal`: Modal che contiene il form per la creazione del progetto.
- `MDBModalDialog`: Dialog del modal centrato.
- `MDBModalContent`: Contenuto del modal.
- `MDBModalHeader`: Header del modal con il titolo e il pulsante per chiudere.
- `MDBModalBody`: Corpo del modal che contiene il form.
- `MDBModalFooter`: Footer del modal con i pulsanti per inviare o annullare.
- `MDBInput`, `MDBRadio`: Componenti di input e radio per il form.

Componenti del Form

- `input` per il titolo del progetto.
- `textarea` per la descrizione del progetto.
- `MDBRadio` per la selezione del tipo di AI (ChatGPT o Bedrock).

AddRequestButton-BusinessRequest

Il componente `AddRequestButton` fornisce un pulsante per navigare alla pagina di richiesta business, mentre `BusinessRequest` fornisce un form per inviare i requisiti di business ad un'API.

Descrizione

Il componente `AddRequestButton` è un semplice pulsante che, quando cliccato, reindirizza l'utente alla pagina `/businessRequest` utilizzando il hook `useNavigate` di React Router.

Proprietà

Il componente `AddRequestButton` non accetta proprietà.

Stato Interno

Il componente `AddRequestButton` non mantiene stato interno.

Funzione `handleClick`

Questa funzione viene chiamata al click del pulsante e naviga l'utente alla pagina `/businessRequest`.

`BusinessRequest`

Descrizione

Il componente `BusinessRequest` fornisce un form che permette all'utente di inserire e inviare requisiti di business tramite un'API. Utilizza il hook `useRef` per ottenere il valore del campo textarea e inviarlo tramite la funzione `handleSubmit`.

Proprietà

Il componente `BusinessRequest` non accetta proprietà.

Stato Interno

Il componente `BusinessRequest` non mantiene stato interno.

Funzione `handleSubmit`

Questa funzione viene chiamata al submit del form, previene il comportamento predefinito del submit, ottiene il valore della textarea tramite il riferimento `input`, e invia questo valore all'API tramite la funzione `api.sendBusinessRequirementsToAI`.

Rendering dei Componenti

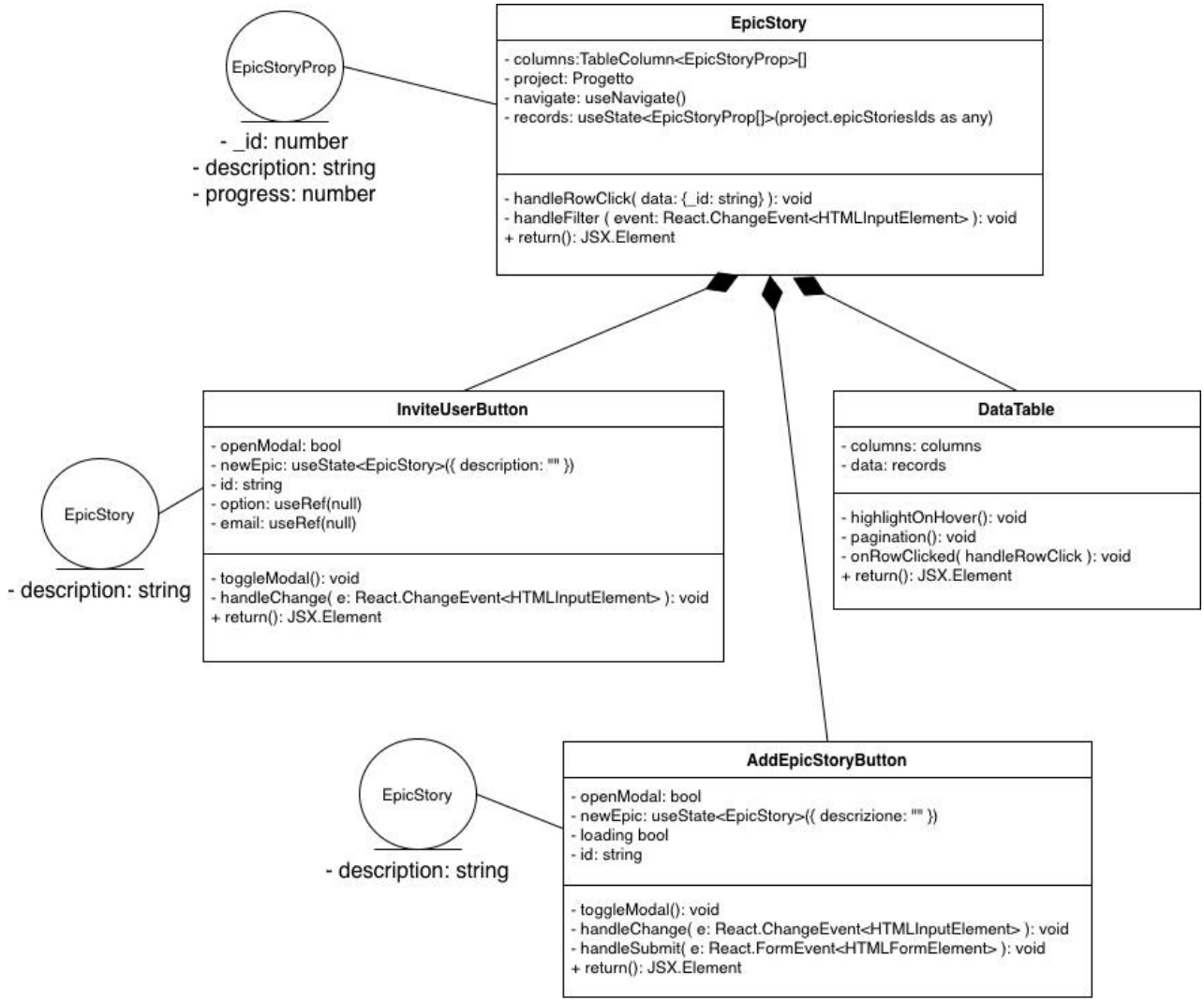
Componente `AddRequestButton`

Il componente `AddRequestButton` restituisce il markup JSX di un pulsante che, quando cliccato, reindirizza l'utente alla pagina `/businessRequest`.

Componente `BusinessRequest`

Il componente `BusinessRequest` restituisce il markup JSX di un form con una textarea per inserire i requisiti di business e un pulsante per inviare il form. Comprende anche un header per il titolo "Requisiti di Business".

4.6 EpicStory



UML Epic Story

Il componente `EpicStory` visualizza le epic story di un progetto, mentre il componente `EpicDetails` visualizza i dettagli di una specifica epic story.

Descrizione

Il componente `EpicStory` visualizza le epic story di un progetto. Consente all'utente di filtrare le epic story per nome e di cliccare su una epic story per visualizzare i dettagli.

Stato Interno

Il componente `EpicStory` mantiene uno stato interno per gestire le epic story visualizzate.

Funzione `handleRowClick`

Questa funzione viene chiamata quando l'utente clicca su una riga della tabella delle epic story. Naviga l'utente alla pagina dei dettagli della epic story corrispondente.

Funzione `handleFilter`

Questa funzione viene chiamata quando l'utente digita nel campo di ricerca. Filtra le epic story in base al nome.

Descrizione

Il componente `EpicDetails` visualizza i dettagli di una specifica epic story. In questo caso, visualizza le user story associate alla epic story selezionata.

Stato Interno

Il componente `EpicDetails` mantiene uno stato interno per gestire i dettagli della epic story e le user story associate.

Funzioni Principali

Il componente `EpicDetails` non contiene funzioni principali oltre all'effetto di caricamento dei dati.

InviteUserButton

Il componente **InviteUserButton** è responsabile della visualizzazione di un pulsante che apre un modale per invitare un utente a un progetto.

Stato Interno

- **openModal**: booleano che indica se il modale è aperto o chiuso.
- **newEpic**: oggetto che rappresenta la nuova epic story inserita dall'project manager.

Funzioni Principali

- **toggleModal**: funzione che cambia lo stato di *openModal*, aprendo o chiudendo il modale.
- **handleChange**: funzione che gestisce il cambiamento di valore degli input dell'utente.
- **handleSubmit**: funzione che gestisce l'invio del form per l'invito dell'utente al progetto.

Rendering

Il componente **InviteUserButton** rende un pulsante che, quando cliccato, apre un modale. Il modale contiene un campo di input per l'email dell'utente da invitare e un menu a tendina per selezionare il ruolo dell'utente (cliente o sviluppatore). In fondo al modale, ci sono pulsanti per confermare o annullare l'operazione di invito.

AddEpicStoryButton

Descrizione

Il componente **AddEpicStoryButton** è responsabile della visualizzazione di un pulsante che apre un modale per consentire all'project manager di aggiungere una nuova epic story.

Stato Interno

- **openModal**: booleano che indica se il modale è aperto o chiuso.
- **newEpic**: oggetto che rappresenta la nuova epic story inserita dall'utente.
- **loading**: booleano che indica se è in corso un'operazione di caricamento.

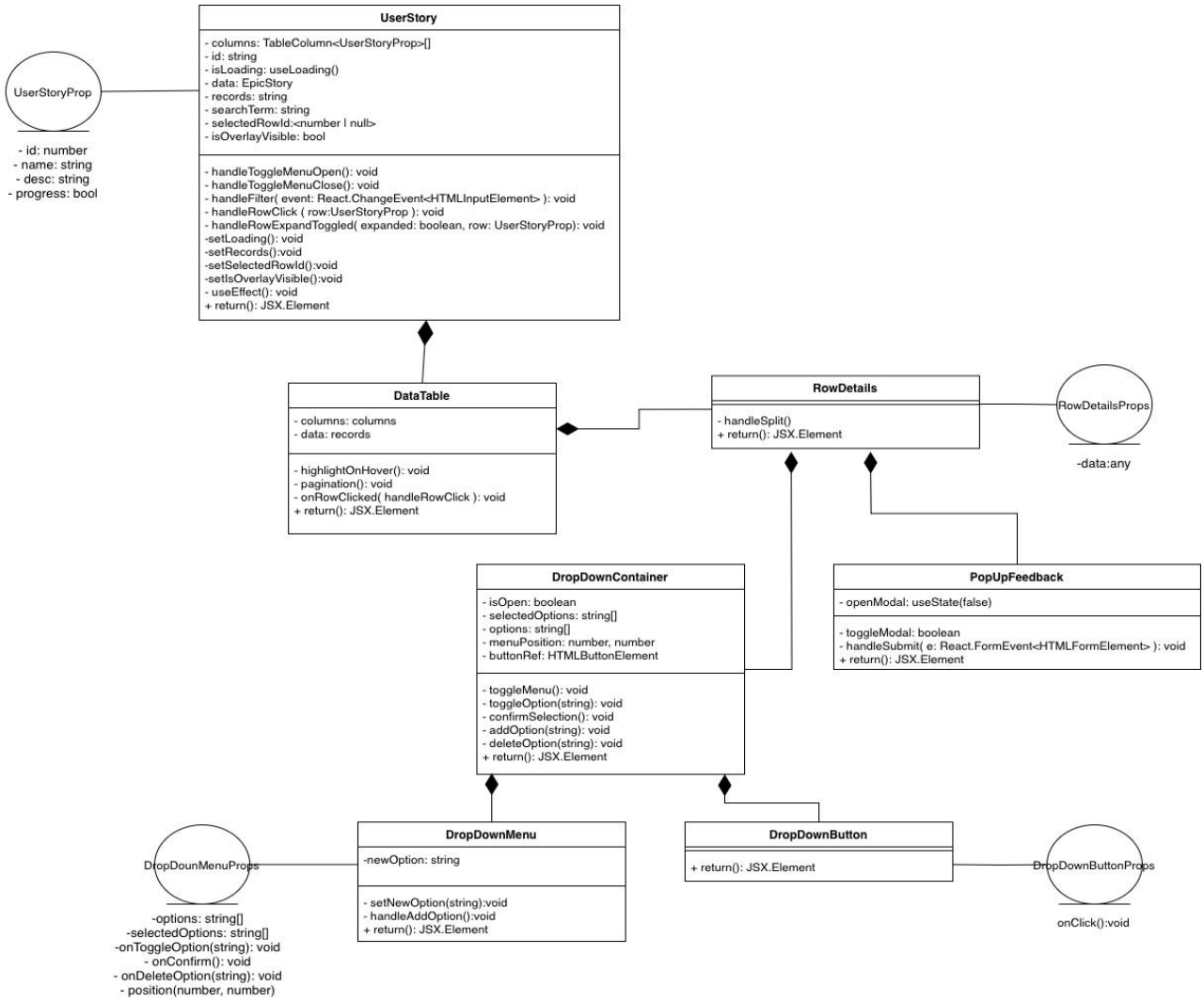
Funzioni Principali

- **toggleModal**: funzione che cambia lo stato di *openModal*, aprendo o chiudendo il modale.
- **handleChange**: funzione che gestisce il cambiamento di valore dell'input della descrizione della nuova epic story.
- **handleSubmit**: funzione che gestisce l'invio del form per l'aggiunta della nuova epic story al progetto.

Rendering

Il componente **AddEpicStoryButton** rende un pulsante che, quando cliccato, apre un modale. Il modale contiene un campo di input per la descrizione della nuova epic story. In fondo al modale, ci sono pulsanti per confermare o annullare l'aggiunta della epic story.

4.7 UserStory



UML User Story

Il componente **UserStory** è responsabile della visualizzazione delle storie degli utenti all'interno di una tabella, con possibilità di ricerca, selezione e visualizzazione dei dettagli di ogni user story.

Stato Interno

- **records**: array di oggetti che rappresentano le storie degli utenti.
- **searchTerm**: stringa che rappresenta il termine di ricerca inserito dall'utente.
- **selectedRowId**: identificatore della riga selezionata.
- **isOverlayVisible**: booleano che indica se l'overlay per il menu è visibile o meno.

Funzioni Principali

- **handleFilter**: funzione per filtrare le storie degli utenti in base al termine di ricerca inserito dall'utente.

- **handleRowClick**: funzione per gestire il click su una riga della tabella, selezionandola o deselezionandola.
- **handleRowExpandToggled**: funzione per gestire l'espansione o la chiusura di una riga della tabella.
- **handleToggleMenuOpen, handleToggleMenuClose**: funzioni per aprire e chiudere l'overlay del menu.

Rendering

Il componente **UserStory** rende una tabella contenente le storie degli utenti. Ogni riga della tabella mostra il tag, il titolo e lo stato di completamento della user story. È presente un campo di ricerca per filtrare le storie degli utenti. È possibile selezionare una riga della tabella per visualizzare i dettagli della user story.

RowDetails

Il componente **RowDetails** è responsabile della visualizzazione dei dettagli di una riga della tabella delle storie degli utenti.

Proprietà

- **data**: oggetto che contiene i dati della riga della tabella.

Funzioni Principali

- **handleSplit**: funzione per gestire la divisione di una user story in due user story, mandando la richiesta all'intelligenza artificiale.

Rendering

Il componente **RowDetails** rende i dettagli di una riga della tabella delle storie degli utenti, inclusa la descrizione della user story e i componenti *DropdownContainer* e *PopupFeedback*. Inoltre, include un pulsante per eseguire l'azione di divisione della riga.

PopUpFeedback

Il componente PopupFeedback è stato implementato utilizzando React ed è composto dai seguenti elementi:

Stato

Il componente utilizza uno stato locale per gestire l'apertura e la chiusura della finestra modale.

Funzionalità

Il componente fornisce le seguenti funzionalità:

- Mostra un pulsante "Feedback".
- Quando il pulsante "Feedback" viene cliccato, viene visualizzata una finestra modale.
- La finestra modale contiene un modulo per inserire il feedback.
- Gli utenti possono inserire il loro feedback nel modulo.

- Gli utenti possono inviare il feedback all'intelligenza artificiale tramite il pulsante "Invia".
- Gli utenti possono annullare l'operazione e chiudere la finestra modale tramite il pulsante "Annulla".

DropDownContainer

Il componente `DropdownContainer` gestisce il menu a discesa per l'assegnazione degli sviluppatori alla user story e le azioni associate ai pulsanti. Questo componente offre un'interfaccia intuitiva e interattiva per la selezione delle opzioni tramite un menu a discesa.

Funzionalità Principali

1. **Gestione dello Stato**:
   - Utilizza lo stato per gestire l'apertura e la chiusura del menu a discesa (`isOpen`).
   - Mantiene lo stato delle opzioni selezionate (`selectedOptions`).
   - Memorizza la lista delle opzioni disponibili (`options`).
   - Conserva la posizione del menu a discesa rispetto al pulsante associato (`menuPosition`).
2. **Interazione Utente**:
   - Fornisce un pulsante per aprire e chiudere il menu a discesa.
   - Gestisce l'apertura e la chiusura del menu a discesa quando il pulsante viene cliccato.
   - Passa la posizione calcolata del menu a discesa al componente `DropdownMenu` tramite le proprietà.
3. **Gestione delle Opzioni**:
   - Consente all'utente di selezionare o deselezionare le opzioni nel menu a discesa, ovvero gli sviluppatori.
   - Permette all'utente di confermare le selezioni fatte.
   - Fornisce funzionalità per aggiungere nuovi sviluppatori alla lista e per eliminarne.
4. **Utilizzo di useRef**:
   - Utilizza `useRef` per ottenere un riferimento al pulsante associato al menu a discesa.
5. **Utilizzo di ReactDOM.createPortal**:
   - Utilizza `ReactDOM.createPortal` per montare il componente `DropdownMenu` nel nodo del DOM del `<body>`, garantendo la corretta sovrapposizione del menu agli altri elementi della pagina.

Componenti Associati

- `DropdownButton`: Rappresenta il pulsante per aprire il menu a discesa.
- `DropdownMenu`: Rappresenta il menu a discesa che mostra le opzioni disponibili e consente all'utente di selezionarle.

DropDownButton

Il componente `DropdownButton` rappresenta il pulsante per aprire il menu a discesa.

Caratteristiche Principali

1. **Utilizzo di forwardRef**:
   - Utilizza `forwardRef` per ottenere un riferimento al pulsante e renderlo disponibile al componente genitore.
   - Questo consente al componente genitore di interagire direttamente con il pulsante, ad esempio per impostare il focus o accedere ad altre proprietà.
2. **Proprietà**:
   - `onClick`: Una funzione da eseguire quando il pulsante viene cliccato. In genere, questa funzione apre il menu a discesa.

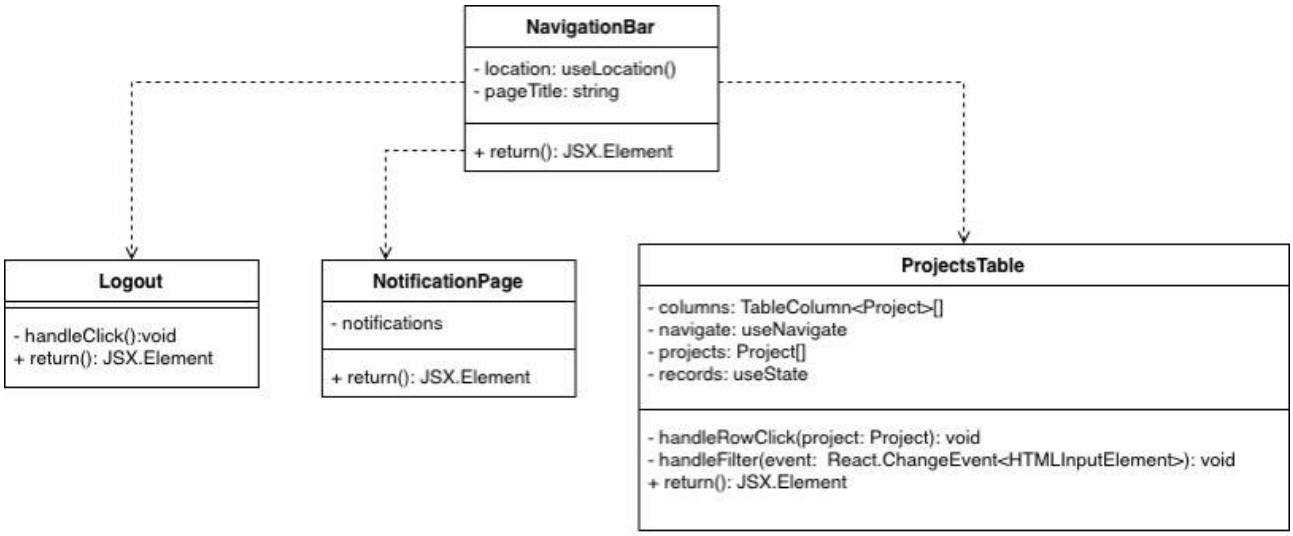
DropDownMenu

Il componente `DropdownMenu` rappresenta il menu a discesa che visualizza gli sviluppatori selezionabili.

Caratteristiche Principali

1. **Proprietà**:
   - `options`: Un array di stringhe rappresentanti le opzioni disponibili nel menu.
   - `selectedOptions`: Un array di stringhe che rappresenta le opzioni attualmente selezionate.
   - `onToggleOption`: Una funzione chiamata quando una opzione viene selezionata o deselezionata.
   - `onConfirm`: Una funzione chiamata quando l'utente conferma le selezioni nel menu.
   - `onAddOption`: Una funzione chiamata quando una nuova opzione viene aggiunta al menu.
   - `onDeleteOption`: Una funzione chiamata quando un'opzione esistente viene eliminata dal menu.
   - `position`: Un oggetto che contiene le coordinate della posizione del menu nella pagina.
2. **Gestione Aggiunta di Opzioni**:
   - Il menu fornisce un'interfaccia per aggiungere nuove opzioni attraverso un campo di input e un pulsante di aggiunta.

4.8 Barra di navigazione



UML NavigationBar

Il componente `NavigationBar` rappresenta una barra di navigazione che mostra un titolo di pagina basato sul percorso corrente e fornisce collegamenti per la navigazione tra diverse pagine.

Caratteristiche Principali

1. **Proprietà**:
   - `routeTitles`: Un oggetto di mapping tra i percorsi e i relativi titoli di pagina.
   - `location`: Un hook di React Router (`useLocation`) che fornisce informazioni sul percorso corrente.
2. **Elementi del DOM**:
   - `<nav>`: Elemento principale della barra di navigazione.
   - `<div className="left">`: Contenitore per il titolo della pagina.
   - `<div className="right">`: Contenitore per i link di navigazione e il pulsante di logout.
   - `<h1>`: Titolo della pagina, dinamicamente aggiornato in base al percorso corrente.
   - `<Link>`: Componenti di React Router per la navigazione tra pagine.
   - `<a>`: Elemento per il componente di logout.
3. **Funzionalità**:
   - Determina il titolo della pagina corrente in base al percorso utilizzando l'oggetto `routeTitles`.
   - Fornisce collegamenti di navigazione alle pagine "Progetti" e "Notifiche".
   - Include un componente per il logout.

Logout

Il componente **Logout** è responsabile della gestione del logout dell'utente. Quando l'utente clicca sul pulsante di logout, il token di autenticazione viene rimosso dal `localStorage` del browser e l'utente viene disconnesso.

Funzioni Principali

- **handleClick**: funzione che viene chiamata quando l'utente clicca sul pulsante di logout. Questa funzione rimuove il token di autenticazione dal `localStorage`.

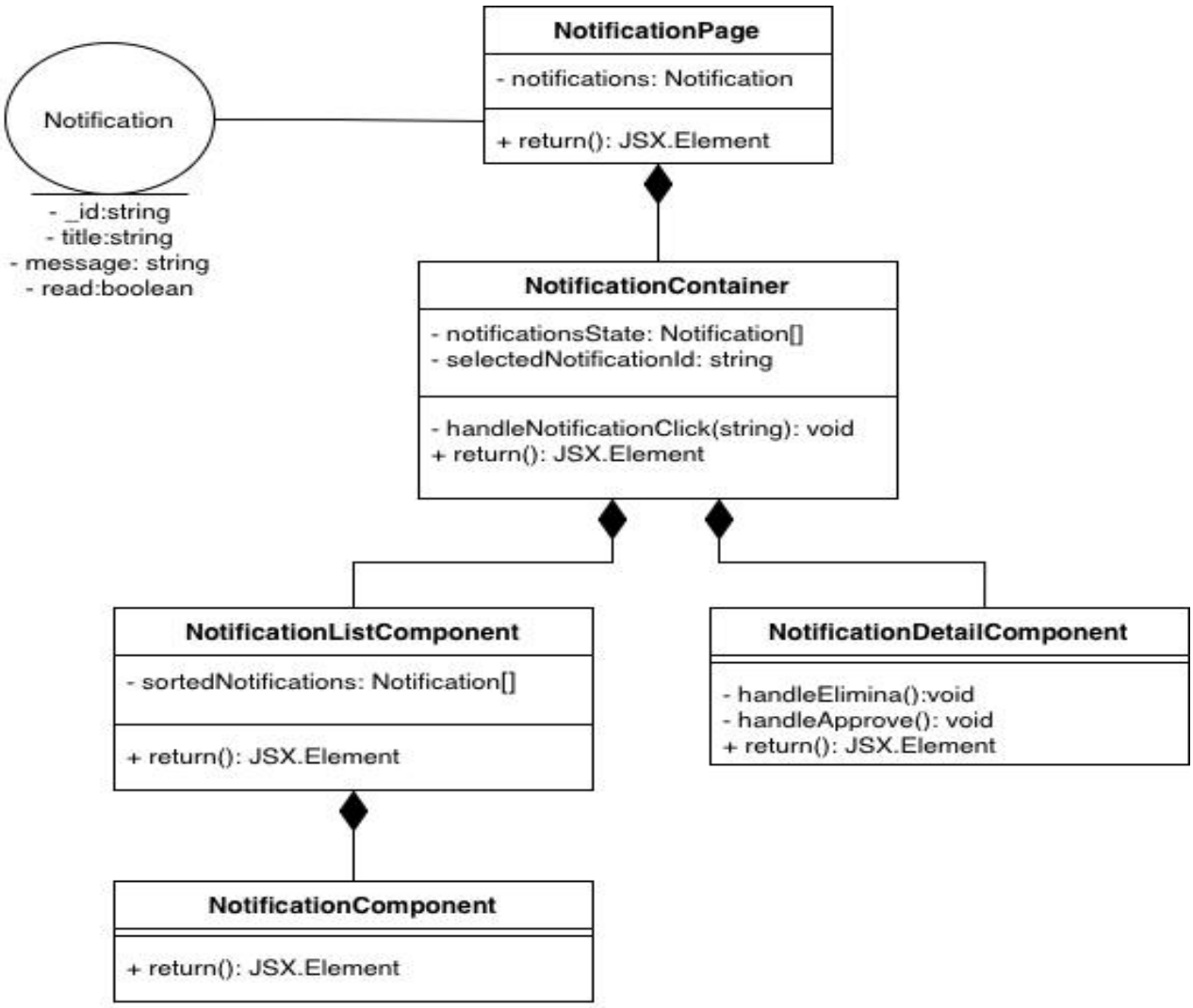
Rendering

Il componente **Logout** rende un pulsante con un'icona di logout. Quando l'utente clicca sul pulsante, viene eseguita la funzione `handleClick`.

Componenti Utilizzati

- **FontAwesomeIcon**:
  - `@fortawesome/react-fontawesome`: Libreria per l'uso delle icone FontAwesome in React.
  - `@fortawesome/free-solid-svg-icons`: Pacchetto di icone gratuite di FontAwesome.
  - `faSignOutAlt`: Icona di uscita utilizzata nel pulsante di logout.

4.9 Notifiche



UML Notifiche

Il componente **NotificationPage** è responsabile della visualizzazione delle notifiche e dei dettagli delle notifiche per un utente. Utilizza diversi componenti presentazionali e container per gestire lo stato e la logica delle notifiche.

Stato Interno

- **notificationsState**: array di notifiche mantenuto dallo stato del componente.
- **selectedNotificationId**: ID della notifica selezionata, inizialmente nullo.

Funzioni Principali

- **handleNotificationClick**: funzione che gestisce il click su una notifica. Marca la notifica come letta e aggiorna lo stato.
- **handleApprove**: funzione che approva una notifica e crea un nuovo progetto basato sulla notifica.

Rendering

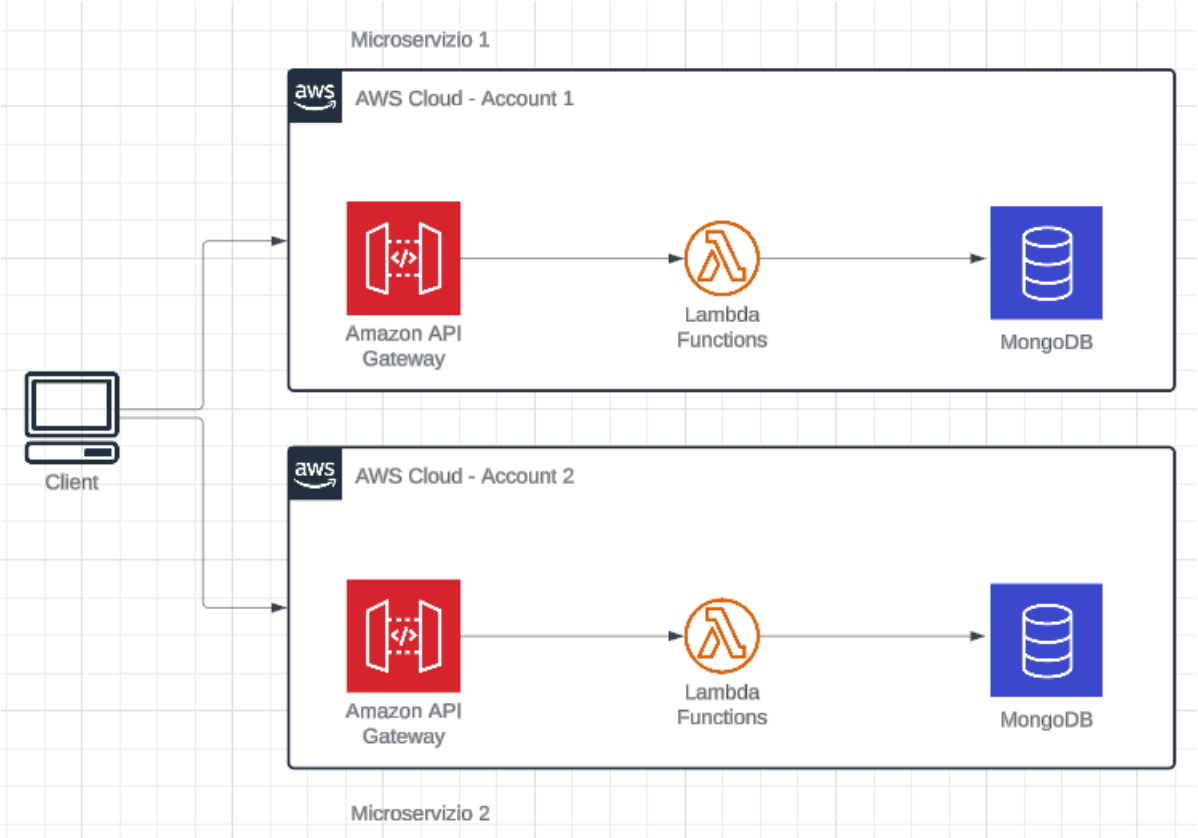
Il componente **NotificationPage** rende un container con una lista di notifiche e i dettagli della notifica selezionata. La lista è ordinata in modo da mostrare prima le notifiche non lette.

Componenti Utilizzati

- **NotificationComponent**:
  - Mostra una singola notifica con il titolo, il messaggio e un indicatore di lettura.
- **NotificationListComponent**:
  - Mostra una lista di notifiche ordinate per stato di lettura.
- **NotificationDetailComponent**:
  - Mostra i dettagli della notifica selezionata con pulsanti per approvare o eliminare.
- **NotificationContainer**:
  - Gestisce lo stato delle notifiche e la logica per la selezione e la lettura delle notifiche.
- **api**:
  - Funzioni per interagire con l'API del backend, come `readNotification` e `addProject`.
- **createEpicStoriesFromBR**:
  - Funzione che utilizza l'IA per creare epic stories da un business requirement.

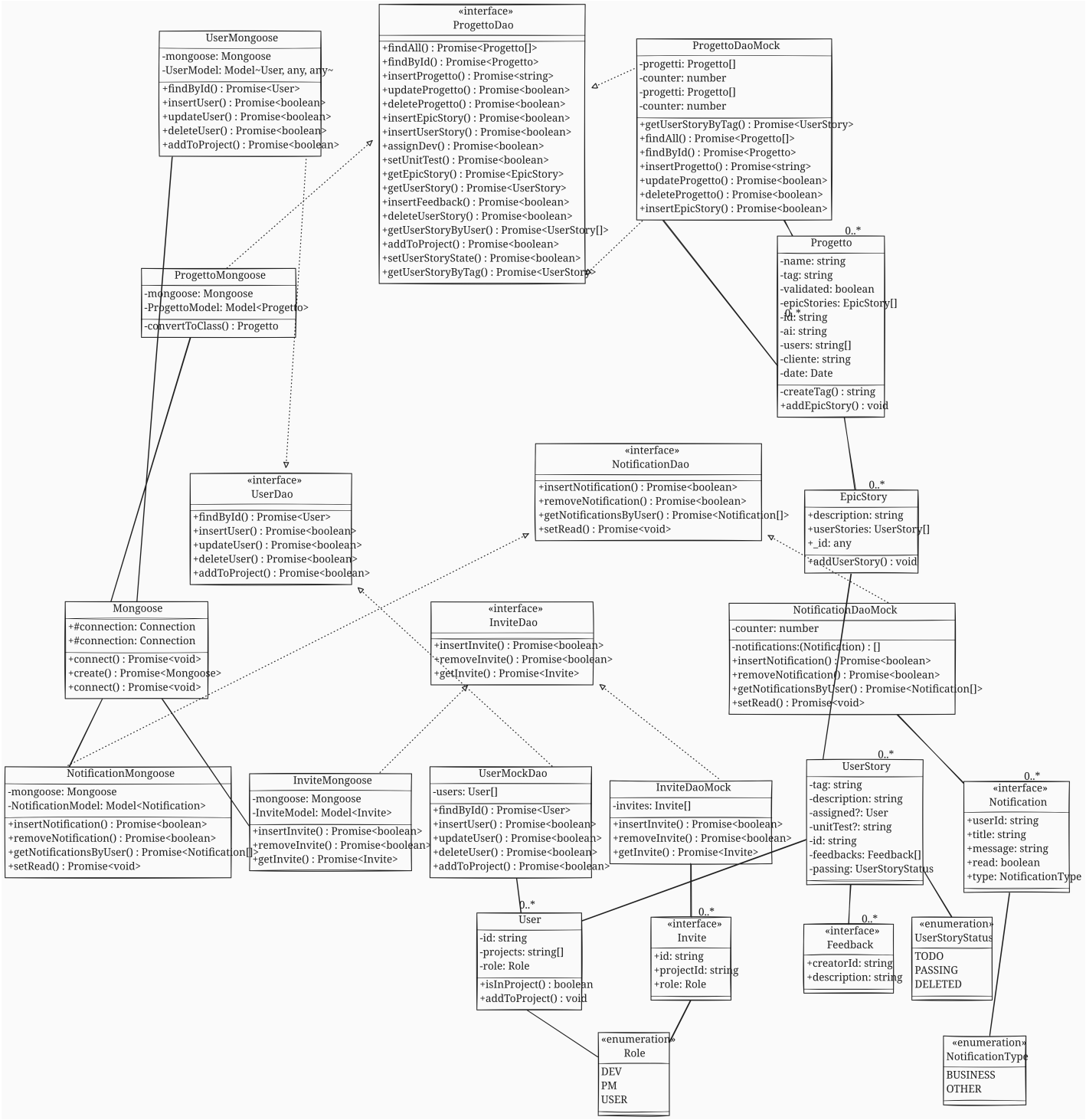
4.10 Architettura Backend

Per il backend del servizio è stata selezionata l'adozione del modello di Serverless Architecture. Questo paradigma di cloud computing consente lo sviluppo e la gestione delle applicazioni senza la necessità di amministrare direttamente i server fisici. Tale modello garantisce una scalabilità elevata, un'alta disponibilità, e costi contenuti, eliminando la preoccupazione relativa all'infrastruttura sottostante. Grazie a questa architettura, sarà infatti possibile effettuare una scalabilità dinamica, adattando le risorse disponibili alle specifiche esigenze del servizio in tempo reale, sia incrementandole che riducendole, a seconda delle necessità operative.



Schema architettura serverless

4.10.1 Diagramma delle classi



Schema delle classi Backend

- **Classe: *Mongoose***
  - Attributi

        - **connection: Connection**: La connessione al database.
    - Metodi
        - **connect() Promise<void>**: Connetti l'istanza di Mongoose al database.
        - **create() Promise<Mongoose>**: Crea una nuova istanza di Mongoose, restituendo l'istanza creata.

- **Classe: *Invite***
    - Attributi
        - **id: string**: L'id dell'invito.
        - **projectId: string**: L'id del progetto.
        - **role: Role**: Il ruolo associato all'invito.

- **Classe: *Notification***
    - Attributi
        - **userId: string**: L'id dell'utente.
        - **title: string**: Il titolo della notifica.
        - **message: string**: Il messaggio della notifica.
        - **read: boolean**: Indica se la notifica è stata letta.
        - **type: NotificationType**: Il tipo di notifica.

- **Classe: *NotificationType***
    - Enumerazione
        - **BUSINESS**: Notifica per i Business Requirements.
        - **OTHER**: Altri tipi di notifica.

- **Classe: *EpicStory***
    - Attributi
        - **description: string**: La descrizione dell'epic story.
        - **userStories: UserStory[]**: Le user story associate all'epic story.
        - **id: string**: L'id dell'epic story.
    - Metodi
        - **addUserStory() void**: Aggiunge una user story all'epic story.

- **Classe: *Progetto***
    - Attributi
        - **name: string**: Il nome del progetto.
        - **tag: string**: Il tag del progetto.
        - **epicStories: EpicStory[]**: Le epic story del progetto.
        - **id: string**: L'id del progetto.
        - **ai: string**: L'intelligenza artificiale utilizzata nel progetto.
        - **users: string[]**: Gli utenti associati al progetto.
        - **cliente: string**: Il cliente del progetto.
        - **date: Date**: La data del progetto.
    - Metodi
        - **createTag() string**: Crea e restituisce un tag per il progetto.
        - **addEpicStory() void**: Aggiunge un'epic story al progetto.

- **Classe: *UserStory***
  - Attributi
    - **tag: string**: Il tag della user story.
    - **description: string**: La descrizione della user story.
    - **assigned: User**: L'utente assegnato alla user story.
    - **unitTest: string**: Il test unitario della user story.
    - **id: string**: L'id della user story.
    - **feedbacks: Feedback[]**: I feedback della user story.
    - **passing: UserStoryStatus**: Lo stato della user story.
- **Classe: *Feedback***
  - Attributi
    - **creatorId: string**: L'id del creatore del feedback.
    - **description: string**: La descrizione del feedback.
- **Classe: *UserStoryStatus***
  - Enumerazione
    - **TODO**: Stato "da fare".
    - **PASSING**: Stato "in corso".
    - **DELETED**: Stato "eliminato".
- **Classe: *User***
  - Attributi
    - **id: string**: L'id dell'utente.
    - **projects: string[]**: I progetti associati all'utente.
    - **role: Role**: Il ruolo dell'utente.
  - Metodi
    - **isInProject() boolean**: Verifica se l'utente è in un progetto, restituendo un booleano.
    - **addToProject() void**: Aggiunge l'utente a un progetto.
- **Classe: *Role***
  - Enumerazione
    - **DEV**: Ruolo sviluppatore.
    - **PM**: Ruolo project manager.
    - **USER**: Ruolo cliente.
- **Classe: *InviteDao***
  - Metodi
    - **insertInvite() Promise<boolean>**: Inserisce un invito.
    - **removeInvite() Promise<boolean>**: Rimuove un invito.
    - **getInvite() Promise<Invite>**: Ritorna un invito dal database.
- **Classe: *InviteDaoMock***
  - Attributi
    - **invites: Invite[]**: Gli inviti simulati.
- **Classe: *InviteMongoose***

  - Attributi
    - **mongoose: Mongoose**: L'istanza di Mongoose.
    - **InviteModel: Model<Invite>**: Il modello Mongoose per gli inviti.

- **Classe: *NotificationDao***
  - Metodi
    - **findByUser() Promise<Notification[]>**: Trova notifiche per utente, restituendo una Promise.
    - **insertNotification() Promise<boolean>**: Inserisce una notifica.
    - **setRead() Promise<boolean>**: Imposta una notifica come letta.

- **Classe: *NotificationDaoMock***
  - Attributi
    - **notifications: Notification[]**: Le notifiche simulate.

- **Classe: *NotificationMongoose***
  - Attributi
    - **mongoose: Mongoose**: L'istanza di Mongoose.
    - **NotificationModel: Model<Notification>**: Il modello Mongoose per le notifiche.

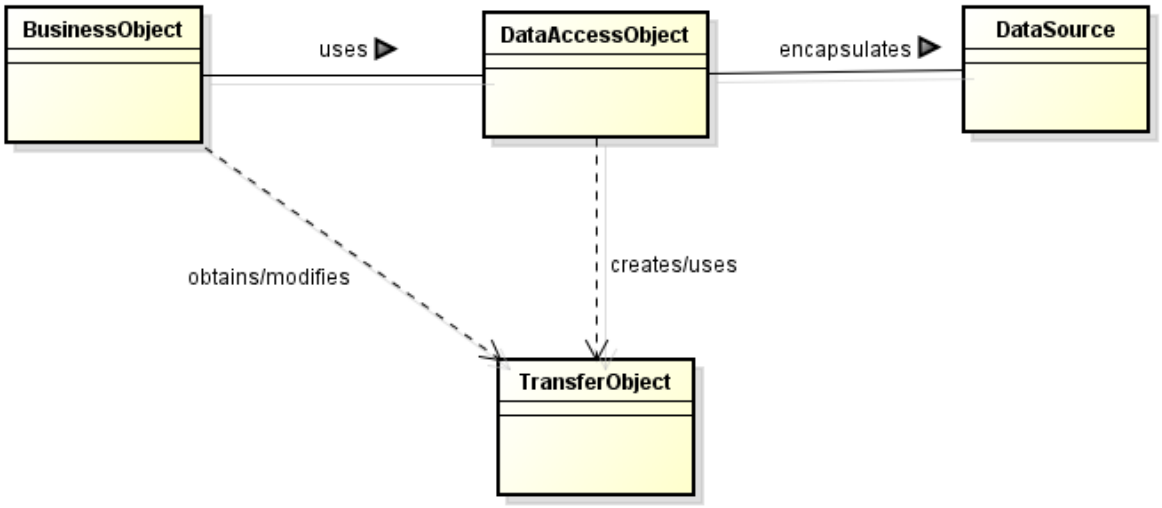
- **Classe: *ProgettoDao***
  - Metodi
    - **findAll() Promise<Progetto[]>**: Trova tutti i progetti.
    - **findById() Promise<Progetto>**: Trova un progetto per ID.
    - **insertProgetto() Promise<string>**: Inserisce un progetto.
    - **updateProgetto() Promise<boolean>**: Aggiorna un progetto.
    - **deleteProgetto() Promise<boolean>**: Elimina un progetto.
    - **insertEpicStory() Promise<boolean>**: Inserisce una EpicStory.
    - **insertUserStory() Promise<boolean>**: Inserisce una UserStory.
    - **assignDev() Promise<boolean>**: Assegna uno sviluppatore a una UserStory.
    - **getEpicStory() Promise<EpicStory>**: Ottiene una EpicStory.
    - **getUserStory() Promise<UserStory>**: Ottiene una UserStory.
    - **insertFeedback() Promise<boolean>**: Inserisce un feedback.
    - **deleteUserStory() Promise<boolean>**: Elimina una UserStory.
    - **getUserStoryByUser() Promise<UserStory[]>**: Trova le UserStory per utente.
    - **addToProject() Promise<boolean>**: Aggiunge un utente a un progetto.
    - **setUserStoryState() Promise<boolean>**: Imposta lo stato di una UserStory.
    - **getUserStoryByTag() Promise<UserStory>**: Trova una UserStory per tag.

- **Classe: *ProgettoMongoose***
  - Attributi
    - **mongoose: Mongoose**: L'istanza di Mongoose.
    - **ProgettoModel: Model<Progetto>**: Il modello Mongoose per i progetti.
  - Metodi
    - **convertToClass(): Progetto**: Converte i dati del database in un'istanza della classe Progetto.

4.11 Pattern utilizzati

4.11.1 Data Access Object

Ogni entità che richiede una rappresentazione nel database è associata a un DAO dedicato. Ciò consente di separare la logica di business dalla logica di accesso ai dati. Inoltre, è stato sviluppato un DAO di mock per agevolare i test di unità.

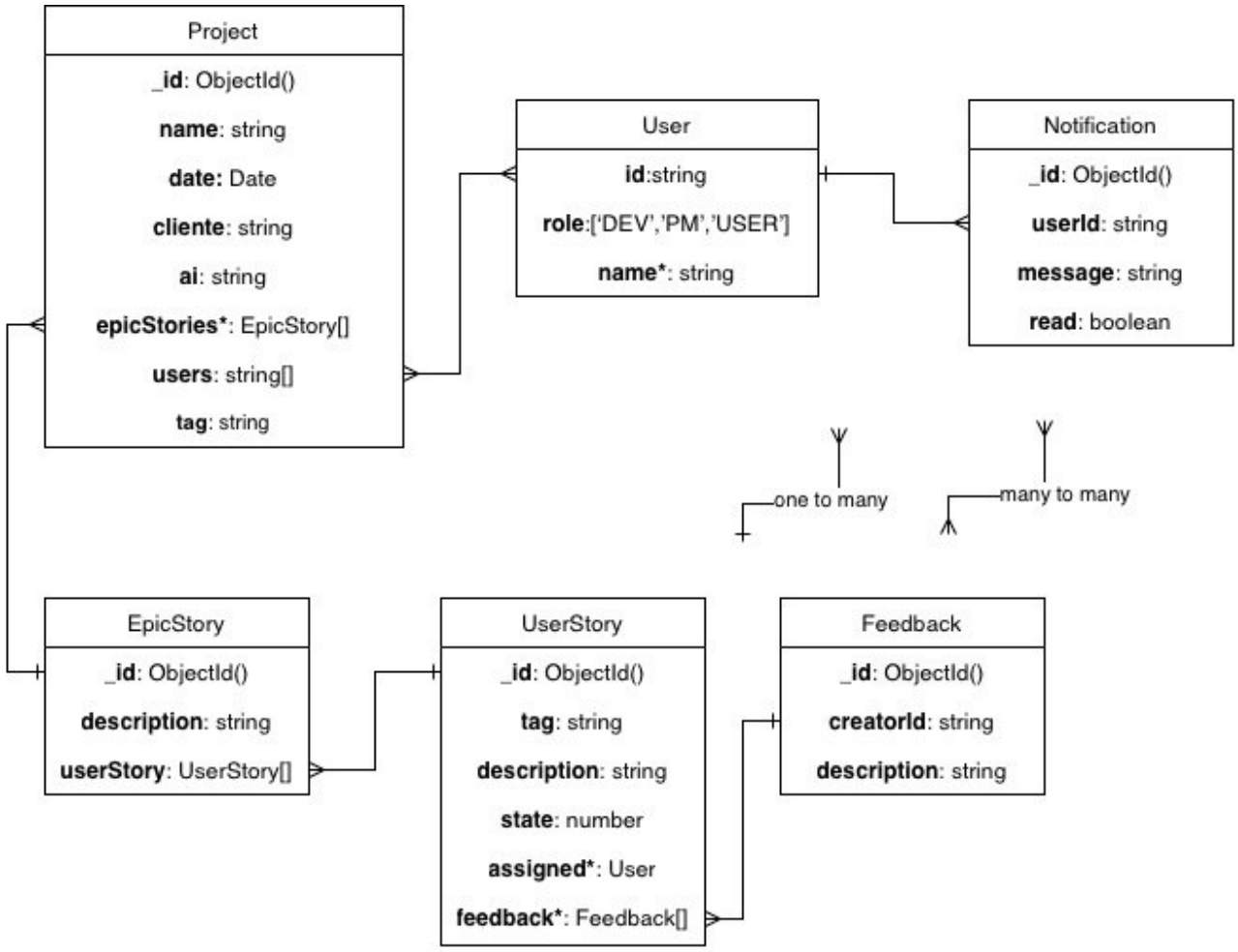


Schema pattern DAO

4.11.2 Dependency Injection

Ogni lambda accetta in input le interfacce di cui necessita, le quali vengono istanziate da una funzione dedicata incaricata di fornire alla lambda i parametri corretti. Durante i test di unità, le classi mock vengono passate come parametri alle lambda.

4.12 Schema del Database



Schema Database

Il Database di questa webApp è stato realizzato con Mongo DB. Lo schema rappresenta il diagramma del database dalla parte di backend. Alcune informazioni, infatti, per la parte di front-end sono superflue, oppure, al contrario, nel front-end saranno presenti alcune informazioni in più, ricavate dai dati salvati del database. Facendo così assicuriamo una chiara divisione delle responsabilità tra le diverse parti del sistema.
Vediamo in dettaglio lo schema, i valori indicati con * possono essere nulli:

- **User**: è la tabella che salva i dati dell'utente
  - **id**: è una stringa con gli id univochi dell'utente che corrisponde all'id che troviamo su Cognito;
  - **role**: è un enum di stringhe che contiene il possibile ruolo degli utenti, DEV (sviluppatore), PM(Project Manager), User (cliente);
  - **name**: è una stringa con il nome dell'utente;
- **Project**: è la tabella che salva i datirelativi ai progetti
  - **_id**: è un oggetto apposito di MongoDB che rappresenta gli id univochi dei progetti;

  - **tag**: è una stringa apposita per identificare e gestire meglio i progetti;
  - **name**: è una stringa che contiene il titolo del progetto;
  - **date**: è un Date che segna la data di creazine del progetto;
  - **cliente**: una stringa con il nome dell'azienda;
  - **ai**: una stringa che specifica l'intelligenza artificiale da usare;
  - **epicStories**: contiene un array di epic storyes collegate al progetto;
  - **users**: è un array di utenti collegati al progetto, che possono lavorarci o visualizzare lo stato;

- **EpicStory**: è la tabella che salva i dati delle epic story
  - **_id**: è un oggetto apposito di MongoDB che rappresenta gli id univochi della epic story;
  - **description**: è una stringa che descrive cosa andrà a fare l'epic story;
  - **userStory**: è un array di user story collegate all'epic story corrispondente;

- **UserStory**: è la tabella che salva i dati delle user story
  - **_id**: è un oggetto apposito di MongoDB che rappresenta gli id univochi delle user story;
  - **tag**: è una stringa univoca per ogni user story che serve per identificare meglio la user story e fare tutte le azioni possibili nel plugin;
  - **description**: è una stringa che descrive cosa andrà a fare la user story;
  - **state**: è un number che rappresenta lo stato della user story; nel nostro caso sarà 0=todo, 1=passing (ha passato tutti i test), 2=deleted (viene eliminata solo nel front end ma rimane nel database, solo se ha passato tutti i test);
  - **assigend**: rappresenta lo sviluppatore a cui è stata assegnata la user story;
  - **feedback**: un array di feedback che contengono i miglioramenti per l'intelligenza artificiale;

- **Feedback**: è la tabella che salva i feedback inviati all'intelligenza artificiale
  - **_id**: è un oggetto apposito di MongoDB che rappresenta gli id univochi dei feedback;
  - **creatorId**: è una stringa con l'id dell'utente che invia il feedback;
  - **description**: è una stringa che descrive il messaggio da inviare all'IA per poter migliorare le prestazioni;

- **Notification**: è la tabella che salva le notifiche tra i diversi utenti
  - **_id**: è un oggetto apposito di MongoDB che rappresenta gli id univochi per le notifiche;
  - **userId**: è una stringa che contiene l'id del mittente;
  - **message**: è una stringa con il corpo del messaggio;
  - **read**: un booleano che segnala se la notifica è stata letta.

Abbiamo utilizzato come id univoco l'id fornito automaticamente da MongoDB per facilitarci le operazioni e le query, dopo per facilitarne l'utilizzo nel front-end lo abbiamo reso una stringa. Mentre per l'id dell'utente abbiamo usato l'id fornito da Cognito.

Inserendo, poi, all'interno di ogni tabella i riferimenti che gli servono (come, ad esempio, dentro a progetto gli id delle epic story collegate) facilita il collegamento e l'elaborazione dei dati, anche se richiede un controllo in più nella fase di inserimento.

4.13 Documentazione API

In questa sezione è descritta ad alto livello la libreria creata per comunicare con l'applicazione, fornendo una panoramica delle API disponibili comprensiva di una breve descrizione della loro funzionalità e suddivise per GET e POST.

Lambda	Descrizione
GET	
/getProgetti	Se l'utente è loggato recupera la lista di tutti i progetti a cui un utente fa parte.
/getProgetto	Dato un id ritorna il progetto relativo a quell'id.
/getEpicStory	Ottiene i dettagli di una specifica epic story.
/getUserStory	Recupera le informazioni relative a una user story specifica.
/getUserStoryByTag	Tramite il tag recupera l'User story relativa.
/getAssignedUserStory	Recupera l'User story assegnata all'utente.
/getNotifications	Recupera le notifiche per l'utente attuale.
/bedrock	Fornisce dati di base per il sistema (non specificato).
/getProgetto	Recupera un progetto secondo un id.
/getProgettoByTag	Recupera un progetto tramite il tag.
POST	
/addProgetto	Aggiunge un nuovo progetto al sistema.
/addEpicStory	Crea una nuova Epic story nel progetto specificato.
/addUserStory	Crea una nuova User story nel progetto specificato.
/acceptInvite	Accetta un invito per un progetto.
/invite	Invia un invito a un utente per unirsi a un progetto.
/assignDev	Assegna uno sviluppatore a una specifica user story.
/setUnitTest	Imposta un test unitario per una funzionalità specifica.
/setUserStoryState	Imposta lo stato di avanzamento della User story.
/readNotifications	Contrassegna le notifiche come lette.
/insertFeedback	Registra il feedback inviato dall'utente.
/login	Permette a un utente di effettuare il login nel sistema.
/change_password	Cambia password dell'utente.
/register	Registra un nuovo utente nel sistema.

Tabella 2: Lista delle Lambda

4.14 Architettura del Plugin

Il plugin di VSCode è stato sviluppato seguendo un approccio ad oggetti.
Di seguito il diagramma delle classi, e la lista dei loro attributi e metodi.

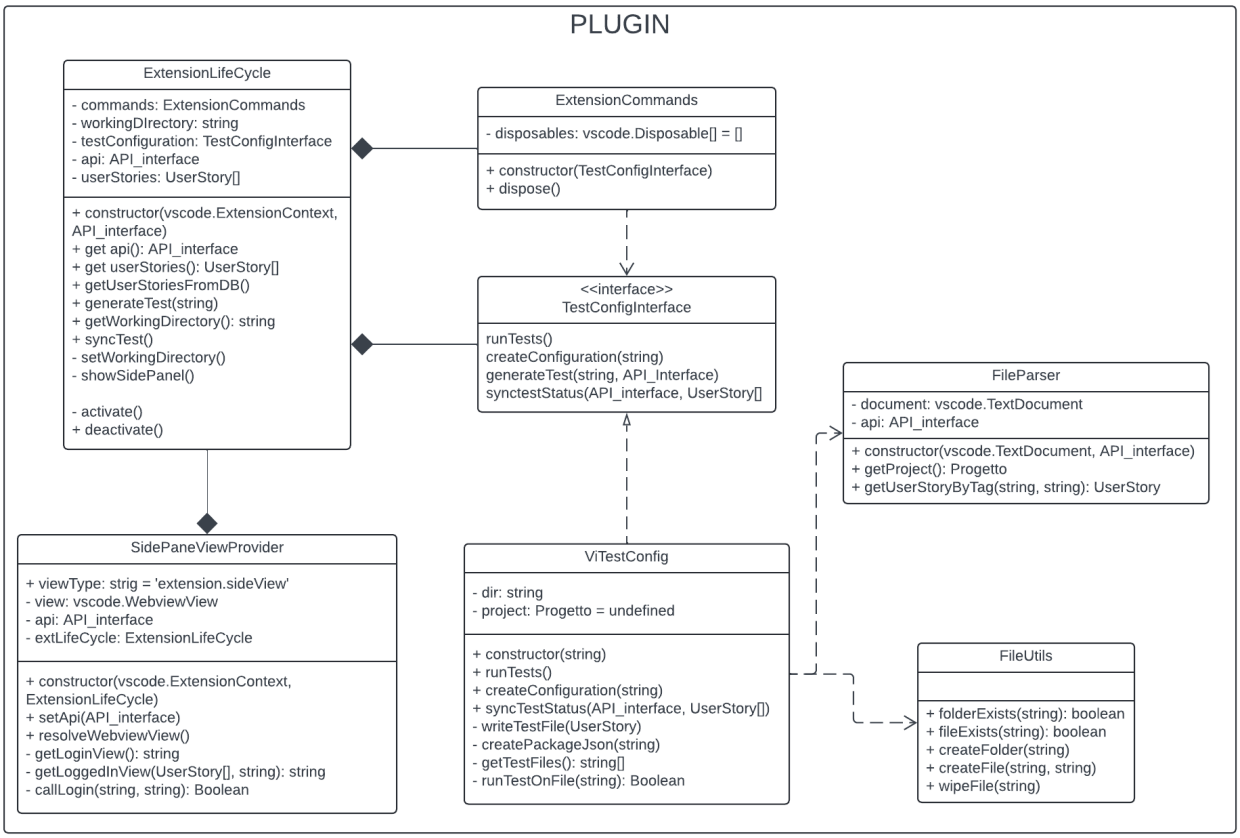


Diagramma delle classi del Plugin

- **Classe: *ExtensionLifeCycle***
  - Attributi
    - **commands: ExtensionCommands**: I comandi dell'estensione;
    - **workingDirectory: string**: La cartella dove si sta lavaronado;
    - **testConfiguration: TestConfigInterface**: L'oggetto che gestisce la configurazione dei test;
    - **api: API_Interface**: L'oggeto api della libreria, che gestisce le chiamate all'API;
    - **userStories: UserStory[]**: L'array di user stories assegnate all'utente che utilizza l'estensione;
  - Metodi
    - **constructor(vscode.ExtensionCommands, API_interface)**: Costruisce un oggetto ExtensionLyfeCycle, prendendo il contesto dell'estensione e un oggetto API_interface;
    - **get api(): API_interface**: Getter per l'attributo api;
    - **get userStories(): UserStory[]**: Getter per l'attributo userStories;
    - **getUserStoriesFromDB()**: Per un utente loggato all'interno dell'estensione, recupera tramite chiamata ad api le user stories a lui assegnate e le assegna all'attributo *UserStories*;
    - **generateTest(string)**: Genera i test una data user story a partire dal suo tag;
    - **getWorkingDirectory: string**: Restituisce il path assoluto della cartella di lavoro;
    - **syncTest()**: Sincronizza i risultati dei test con il database, modificando il campo *state* della user story;

- **setWorkingDirectory()**: Imposta la cartella di lavoro tramite pop-up di sistema;
- **showSidePanel()**: Renderizza il pannello laterale dell'estensione;
- **activate()**: Attiva l'estensione;
- **deactivate()**: Disattiva l'estensione;

- **Classe: *ExtensionCommands***
  - Attributi
    - **disposables: vscode.Disposable[] = []**: Array che contiene i comandi registrati;
  - Metodi
    - **constructor(TestConfigInterface)**: Costruisce un oggetto della classe a partire da una TestConfifInterface;
    - **dispose()**: Elimina i comandi quando un oggetto ExtensionCommand viene distrutto;

- **Interfaccia: *TestConfigInterface***
  - Metodi
    - **runTests()**: Esegue i test relativi alle user stories;
    - **createConfiguration(string)**: Crea i file di configurazione nella cartella di lavoro, prendendo il suo path in input;
    - **generateTest(string, API_Iterface)**: Genera i test nella cartella di lavoro, prendendo in input il suo path;
    - **syncTestStatus(API_Interface, UserStory[])**: Sincronizza lo stato delle user stories in base ai risultati dei test;

- **Classe *VitestConfig***
  - Attributi
    - **dir: string** Rappresenta la cartella dove vengono generati i test;
    - **project: Progetto = undefined** Rappresenta il progetto sul quale l'utente sta lavorando;
  - Metodi
    - **constructor(string)**: Costruisce una nuova configurazione, e imposta dir alla cartella di lavoro;
    - **runTests()**: Esegue i test relativi alla user story;
    - **createConfiguration(string)**:Crea i file di configurazione nella cartella di lavoro, prendendo in input il suo path;
    - **syncTestStatus(API_Interface, UserStory[])**: Sincronizza lo stato delle user stories in base ai risultati dei test;
    - **writeTestFile(UserStory)**: Genera un nuovo file di test per la usertory;
    - **createPackageJson(string)**: Crea il file di configurazione package.json nella cartella passata in input;
    - **getTestFiles(): string[]**: Restituisce un array di string, dove ognuna di esse è il path ad un file di test;
    - **runTestOnFile(string): Boolean**: Dato il path ad un file di test, esegue il file. Restituisce *true* se tutti i test passano, altrimenti restituisce *false*;

- **Classe: *SidePanelViewProvider***
  - Attributi
    - **viewType: string = 'extension.sideView'**: Nome della view, utilizzato per identificarla nel file di configurazione package.json del plugin;

        - **view: vscode.WebviewView**: Contiene il codice per renderizzare la vista;
        - **api: API_Interface**: Istanza di un oggetto api utilizzata per fare le chiamate all'API;
        - **extLifeCycle: ExtensionLifeCycle**: Oggetto ExtensionLifeCycle, contentente le funzionalità e i dati relativi all'istanza della estensione attiva;
    - Metodi
        - **constructor(vscode.ExtensionContext, ExtensionLifeCycle**: Costruice un nuovo oggetto, a partire dal contesto dell'estensione. Imposta il campo extLyfeCycle;
        - **setApi(API_Interface)**: Setter per il campo api;
        - **resolveWebView(vscode.WebviewView, vscode.WebviewViewResolveContext, vscode.CancellationToken)**: Risolve una webView;
        - **getLoginView: string**: Restituisce il codice necessario a renderizzare la pagina di login;
        - **getLoggedInView(UserStory[], string): string**: Restituisce il codice per renderizzare la vista di un utente loggato. Mostra le user story a lui assegnate, un pulsante per ciascuna per generare i test, ed un pulsante generale per sincronizzare lo stato delle user story con il database;
        - **callLogin(string, string): Boolean**: Prendendo in input email e password, gestisce la chiamata al login;

- **Classe: *FileParser***
    - Attributi
        - **document: vscode.TextDocument**: Il documento attivo, dove si esegue il parsing;
        - **api: API_Interface**: L'oggetto api che gestisce le chiamate ad API Gateway;
    - Metodi
        - **constructor(vscode.TextDocument, API_Interface)**: Costruisce un oggetto FileParser inizializzando tutti i suoi campi dati;
        - **getProject(): Progetto**: Restituisce il progetto relativo al file, recuperato dal tag sulla prima riga del file;
        - **getUserStoryByTag(string, string)**: Dato il tag di una userstory, se è presente il suo tag nel file, la restituisce;

- **Classe: *FileUtils***
    - Metodi
        - **folderExists(string): Boolean**: Dato un path di una cartella, verifica se esiste;
        - **fileExists(string): Boolean**: Dato un path di un file, verifica se esiste;
        - **createFolder(string)**: Crea una cartella nel path passato in input;
        - **createFile(string, string)**: Crea un file a partire dal path e dal contenuto passati in input;
        - **wipeFile(string)**: Svuota il file al path passato in input;

4.15 Architettura della libreria

Per raggruppare alcune funzionalità utili sia nel frontend che nel plugin, è stata sviluppata una libreria. Di seguito è presente il suo diagramma delle classi, ed una descrizione della sua struttura.

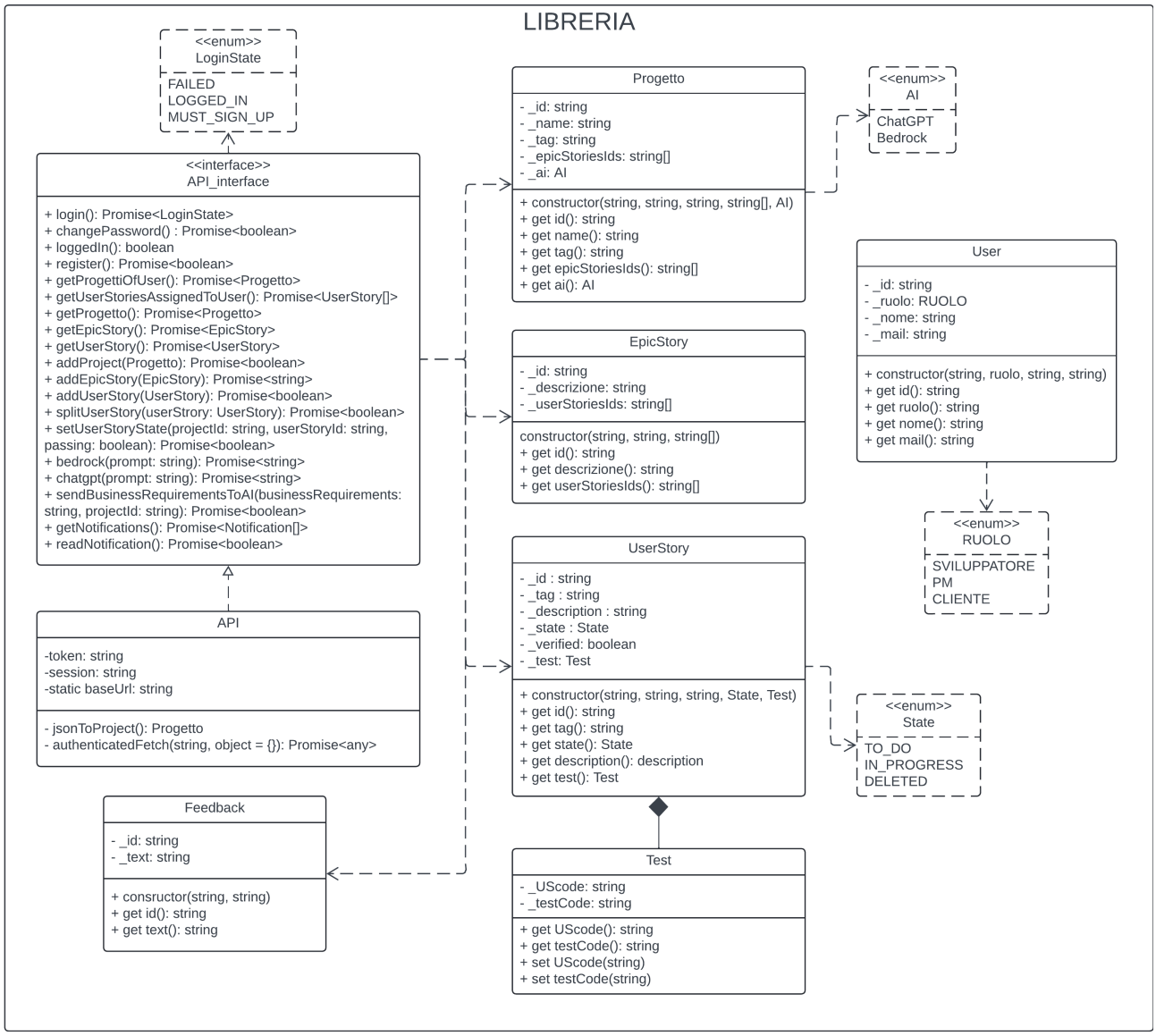


Diagramma delle classi della Libreria

- **Classe: *Progetto***
  - Attributi
    - **_id: string**: Líd del progetto;
    - **_name: string**: Il nome del proggetto;
    - **_epicStoriesIds: string[]**: Le epic stoy che compongono il progetto;
    - **_ai: AI** L'intelligenza artificiale da utilizzare per il progetto;
  - Metodi
    - **constructor(project: ProjectData)** Costruisce un oggetto Progetto prendendo il contesto del project data;
    - **get id(): string**: Getter per l'id;
    - **get name(): string**: Getter per il nome del progetto;
    - **get epicStoriesIds(): string[]**: Getter per la lista di epic story;
    - **get ai(): AI**: Getter per l'intelligenza artificiale da usare;

- **Classe: *EpicStory***
  - Attributi
    - **_id: string**: Líd dellépic story;
    - **_descrizione: string**: La descrizione di cosa deve fare l'epic story;
    - **_userStoriesIds: string[]**: Le user stoy che compongono la specifica epic story;
  - Metodi
    - **constructor(epic: EpicData)** Costruisce un oggetto EpicStory prendendo il contesto di EpicData;
    - **get id(): string**: Getter per l'id;
    - **get descrizione(): string**: Getter per la descrizione della epic story;
    - **set descrizione(descrizione: string): void**: Setter per la descrizione della epic story;
    - **get userStoriesIds(): string[]**: Getter per la lista di epic story;
- **Classe: *UserStory***
  - Attributi
    - **_id: string**: L'id dell'user story;
    - **_tag: string**: Il tag univoco della user story;
    - **description: string**: La descrizione di cosa deve fare l'user story;
    - **_state: State**: un numero che rappresenta 0 se To_DO, 1 se IN_Progress o 2 se DELETED per lo stato della user story;
    - **_verified:boolean**: per veedre se il codice scritto nel pluggin ha passato tutti i test;
    - **_test: Test**: contiene le informazioni per la creazione del test della user story;
  - Metodi
    - **constructor(user: UserData)** Costruisce un oggetto UserStory prendendo il contesto di UserData;
    - **get id(): string**: Getter per l'id;
    - **get tag(): string**: Getter per il tag;
    - **get verified(): boolean**: Getter per lo stato di verifica;
    - **get state(): State**: Getter per lo stato della user story;
    - **get description(): string**: Getter per la descrizione della user story;
    - **get test(): Test**: Getter per il test automatico della user story;
    - **set tag(theTag: string): void**: Setter per il tag;
    - **set description(description: string):**: Setter per la descrizione della user story;
    - **set state(state: State): void**: Setter per lo stato della user story;
    - **set verified(verified: boolean): void**: Setter per la verifica;
    - **set test(test: Test): void**: Setter per il test;
- **Classe: *User***
  - Attributi
    - **_id: string**: L'id dell'utente;
    - **_ruolo: RUOLO**: Il ruolo dell'utente che puo' essere: 'PM', 'CLIENTE' o 'SVILUPPATORE';
    - **_nome: string**: Il nome dell'utente;
    - **_mail: string**: la mail dell'utente;
  - Metodi
    - **get id(): string**: Getter per l'id;
    - **get ruolo(): RUOLO**: Getter per il RUOLO;

        - **get nome(): string**: Getter per il nome dell'utente;
        - **get mail(): string**: Getter per la mail;

- **Classe: *Feedback***
    - Attributi
        - **_id: string**: L'id dell'feedback;
        - **_text: string**: Iltesto del feedback;
    - Metodi
        - **get id(): string**: Getter per l'id;
        - **get text(): string**: Getter per il testo;

- **Classe: *Test***
    - Attributi
        - **_UScode : string**: L'id della user story per fare il test;
        - **_testCode: string**: Il codice del test;
    - Metodi
        - **constructor(code: string)**: Costruttore del test che prende il codice;
        - **get UScode(): string**: Getter per il codice della user story;
        - **get testCode(): string**: Getter per il codice del test;

- **Interfaccia: *API_Interface***
    - Metodi
        - **loggedIn(): Boolean**: Restituisce true se l'utente è loggato, false altrimenti;
        - **login(string, string): Promise<LoginState>**: Dati email e password, effettua il login, restitusce lo stato di login;
        - **register(string, string): Promise<boolean>**: Dati email e password, registra un nuovo utente. Restituisce true se la registrazione è avvenuta con successo, false altrimenti;
        - **changePassword(string, password):Promise<boolean>**: Data email e nuova password, cambia la password di un utente;
        - **getProgettiOfUser(): Promise<Progetto[]>**: Restituisce un array contenente tutti i progetti a cui l'utente loggato è stato assegnato;
        - **getUserStoriesAssignedToUser(): Promise<UserStory[]>**: Restituisce un array contenente tutte le user stories assegnate all'utente loggato;
        - **getProgetto(projectId: string): Promise<Progetto>**: Dato il suo id, restituisce un progetto;
        - **getEpicStory(epicId: string, projectId: string): Promise<EpicStory>**: Dato il suo id, restituisce una epic story;
        - **getUserStory(userStoryId: string, projectId: string): Promise<UserStory>**: Dato il suo id, restituisce una user story;
        - **addProject(progetto: Progetto): Promise<boolean>**: Aggiunge un progetto al database;
        - **addEpicStory(epic: EpicStory, projectId: string): Promise<string>**: Aggiunge una epic story al database;
        - **addUserStrory(userStory: UserStory, projectId: string, epicStoryId: string): Promise<boolean>**: Aggiunge una user story al database;
        - **splitUserStory(userStrory: UserStory): Promise<boolean>**: Data una user story troppo ampia, la divide in multiple user story;
        - **setUserStoryState(projectId: string, userStoryId: string, passing: boolean): Promise<boolean>**: Modifica lo stato di una user story;

- **AI(prompt: string): Promise<string>**: Invia alla AI un prompt, e ne ritorna la risposta;
- **sendBusinessRequirementsToAI(businessRequirements: string, projectId: string): Promise<boolean>**: Invia all'intelligenza artificiale selezionata per il progetto i requisiti di business;
- **getNotifications(): Promise<Notification[]>**: Restituisce un array di notifiche contenente tutte le notifiche invita all'utente loggato;
- **readNotification(string): Promise<boolean>**: Segna una notifica come letta;

- **Classe: *API***
  - Attributi
    - **token: string**: Il token d'accesso per poter fare richieste all'API;
    - **session: string**: Identificatore di sessione utilizzato per mantenere lo stato tra diverse richieste dall'utente all'API;
    - **baseUrl: string**: Endpoint di base per l'API;
  - Metodi
    Oltre a tutti i metodi implementati richiesti dall'interfaccia abbiamo:
    - **jsonToProject(object): Progetto** : Dato un progetto in formato json, restituisce un oggetto Progetto della libreria;;
    - **async authenticatedFetch(string,object =):Promise<any>**: Per un utente loggato, esegue una fetch passando come header il token di autorizzazione.;

5 Requisiti soddisfatti

In questa sezione è riportata, dal documento di *Analisi dei Requisiti 2.0.0* e seguendo le convenzioni lì illustrate, il soddisfacimento dei requisiti.

Codice	Descrizione	Stato
ROF1	Accesso a web app tramite login composto da email e password.	Soddisfatto
ROF2	Scrittura di richieste di business tramite box testuale da web app.	Soddisfatto
ROF3	Invio delle richieste di business da web app.	Soddisfatto
ROF4	Visualizzazione andamento sviluppo richieste tramite barra di completamento basata sulla percentuale di user stories completate.	Soddisfatto
ROF5	Approvazione o rifiuto del risultato relativo all'implementazione di una user story.	Soddisfatto
RDF6	Ricezione notifiche quando user story completata.	Soddisfatto
ROF7	Funzionalità di tag nel plug-in.	Soddisfatto
ROF8	Lista di user stories assegnate da Project Manager sia su web app che su plug-in.	Soddisfatto
RDF9	Ricezione notifica su web app quando nuova user story è assegnata dal Project Manager.	Soddisfatto
ROF10	Invio del codice sviluppato a IA per richiesta verifica.	Soddisfatto
ROF11	Visualizzazione user stories generate da IA.	Soddisfatto
ROF12	Invio di feedback sulle user stories generate all'IA.	Soddisfatto
ROF13	Suddivisione delle user stories troppo grandi.	Soddisfatto
ROF14	Assegnazione user stories agli sviluppatori.	Soddisfatto
RDF15	Ricezione notifiche quando user story$_G$ viene generata in seguito a richiesta del cliente.	Non soddisfatto
ROF16	Invio richiesta di modifiche relative a user stories a IA prima di approvazione.	Soddisfatto
ROF17	Visualizzazione andamento epic/user stories assegnate.	Soddisfatto
ROF18	Creazione di un plug-in per VSCode.	Soddisfatto
RDF19	Creazione di un plug-in per XCode.	Non soddisfatto
ROF20	I linguaggi supportati dal plug-in sono Typescript e Javascript.	Soddisfatto

RDF21	Altri linguaggi che potrebbero essere supportati in futuro sono Kotlin$_G$ e Swift.	Non soddisfatto
ROF22	Gestione degli input (prevenzione da Injection Cross Site Scripting e sanificazione dell'input.)	Soddisfatto
ROQ1	Il progetto deve essere accessibile pubblicamente su GitHub o su un'altra repository pubblica.	Soddisfatto
ROQ2	Il prodotto deve essere sviluppato conformemente a quanto stabilito nelle *Norme Way of Working*$_G$.	Soddisfatto
ROQ3	Deve essere effettuato il testing delle unità e dell'integrazione con una copertura minima dell'80%.	Soddisfatto
ROQ4	Deve essere fornita una documentazione completa sulle scelte implementative e progettuali effettuate.	Soddisfatto
ROQ5	Deve essere fornito un manuale per l'utilizzo del prodotto.	Soddisfatto
ROQ6	Deve essere fornita una documentazione che compara la capacità di ChatGPT e quella di AWS Bedrock nell'interpretare del codice sorgente ed associare le user stories generate.	Soddisfatto
ROQ7	Deve essere fornita una documentazione che prova un'interpretazione corretta da parte dell'IA che si basa: sulle epic/user stories generate dall'IA, i test generati dall'IA, i criteri di accettazione delle epic/user stories forniti dal proponente.	Soddisfatto
ROV1	L'applicazione per l'interazione con la piattaforma dev'essere sviluppata attraverso l'uso di tecnologie web.	Soddisfatto
ROV2	Le due IA utilizzate per l'analisi sono AWS Bedrock e ChatGPT.	Soddisfatto
RDV3	L'applicazione deve essere utilizzabile tramite browser (Chrome 123.0, Firefox 124.0, Safari 17.0) di dispositivi mobili(Android 14.0, iOS 17.0).	Non soddisfatto
ROV4	Il front-end dell'applicazione verrà sviluppato in React.	Soddisfatto
ROV5	Ogni AWS Lambda-function deve essere sviluppata in Node.js.	Soddisfatto

ROV6	Tutte le API devono essere integrate in AWS API-gateway.	Soddisfatto
ROV7	L'applicativo deve essere compatibile con il browser Google Chrome dalla versione 121.	Soddisfatto
ROV8	L'applicativo deve essere compatibile con il browser Firefox dalla versione 122.	Soddisfatto
ROV9	L'applicativo deve essere compatibile con il browser Microsoft Edge dalla versione 121.	Soddisfatto
ROV10	Il plug-in deve essere compatibile con VSCode dalla versione 1.84.1 .	Soddisfatto
ROV11	L'accesso deve essere controllato da AWS Cognito, con autenticazione univoca.	Soddisfatto
ROV12	I ruoli devono essere definiti all'interno della piattaforma per evitare accessi non autorizzati.	Soddisfatto
ROV13	Protezione delle informazioni trasmesse tra browser e server tramite protocollo SSL/TLS.	Soddisfatto

Tabella 3: Tabella dei requisiti soddisfatti

5.1 Resoconto dei requisiti soddisfatti

Tipologia requisito	Istanze	Totale soddisfatto
Requisito funzionale	22	86,36%
Requisito obbligatorio	36	100%

Tabella 4: Tabella di resoconto dei requisiti soddisfatti