



UNIVERSITÀ
DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Progetto:

Animi



Titolo del documento:

Sviluppo

Gruppo:

T51

Indice

1. User Flows	3
2. Application Implementation and Documentation	12
2.1. Project Structure	12
2.2. Project Dependencies	14
2.3. Project Data or DB	15
2.4. Project APIs	19
2.4.1. Resources Extraction from the Class Diagram	19
2.4.2. Resources Models	20
2.5. Sviluppo API	?
3. API documentation	?
4. FrontEnd Implementation	?
5. GitHub Repository and Deployment Info	?
6. Testing	?

Scopo del documento

Il presente documento fornisce tutte le informazioni necessarie per lo sviluppo di una parte dell'applicazione Animati. In particolare, presenta tutti gli artefatti necessari per realizzare i servizi di Animati. Partendo dalla descrizione degli user flow legati ai tre ruoli presenti nell'applicazione, il documento prosegue con la presentazione delle API necessarie (tramite l'API Model e il Modello delle risorse) per poter visualizzare, inserire e modificare attività del catalogo di Animati; manipolare le liste, le valutazioni e le segnalazioni. Per le API realizzate, oltre ad una descrizione delle funzionalità fornite, il documento presenta la sua documentazione e i test effettuati. Infine, una sezione è dedicata alle informazioni del Git Repository e al deployment dell'applicazione stessa.

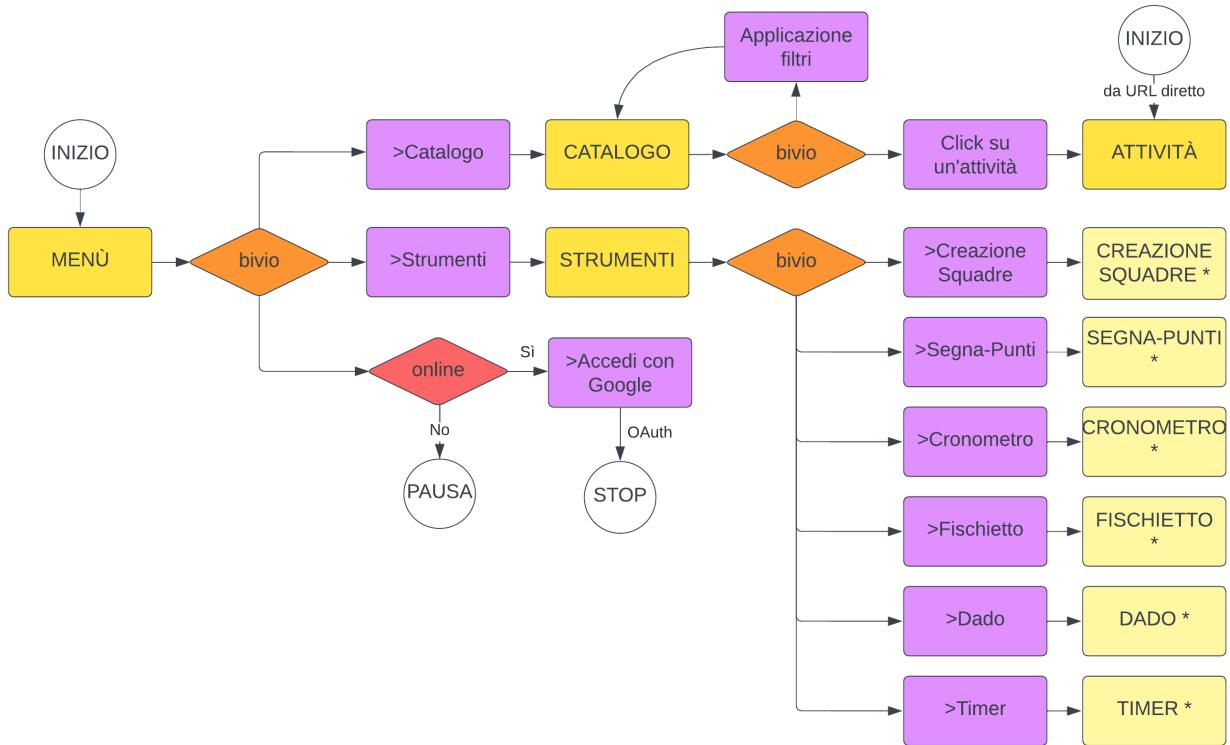
1. User Flows

In questa sezione del documento di sviluppo riportiamo gli "user flows" per i tre diversi ruoli: anonimo, autenticato e amministratore, nonché quelli degli strumenti.

Utente Anonimo

La figura sottostante descrive lo user flow relativo alle azioni che un utente anonimo, ovvero senza autenticazione, riesce ad eseguire. L'utente inizialmente, dal menù, può consultare il catalogo e la lista degli strumenti e può effettuare l'accesso con Google.

- A partire dal catalogo l'utente può applicare dei filtri, facendo in modo di consultare un sottoinsieme del catalogo completo con le caratteristiche desiderate. Oltre a ciò l'utente, sia prima che dopo aver eventualmente applicato dei filtri, può consultare le attività del catalogo.
- A partire dalla lista degli strumenti, l'utente può selezionare uno strumento presente.

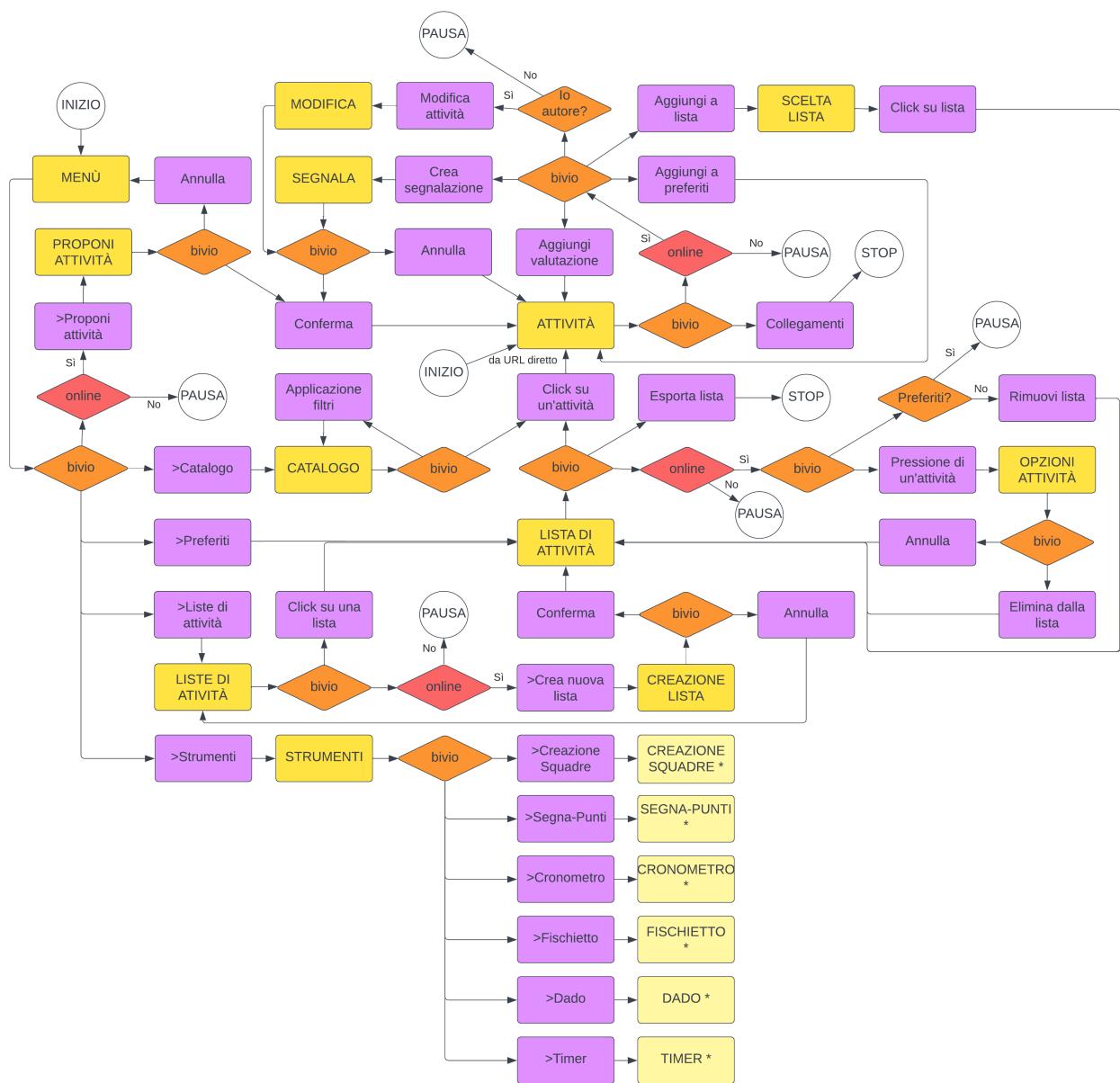


Utente Autenticato

La figura sottostante descrive lo user flow relativo alle azioni che un utente autenticato, ovvero che ha effettuato l'accesso con il suo account Google, riesce ad eseguire. L'utente inizialmente, dal menù, può consultare il catalogo e la lista degli strumenti, andare sul proprio profilo utente, consultare la lista di liste di attività e i preferiti e proporre nuove attività.

- A partire dal catalogo, l'utente autenticato può eseguire le stesse operazioni che può eseguire un utente anonimo, ovvero applicare dei filtri e cliccare sulle attività per consultarle.

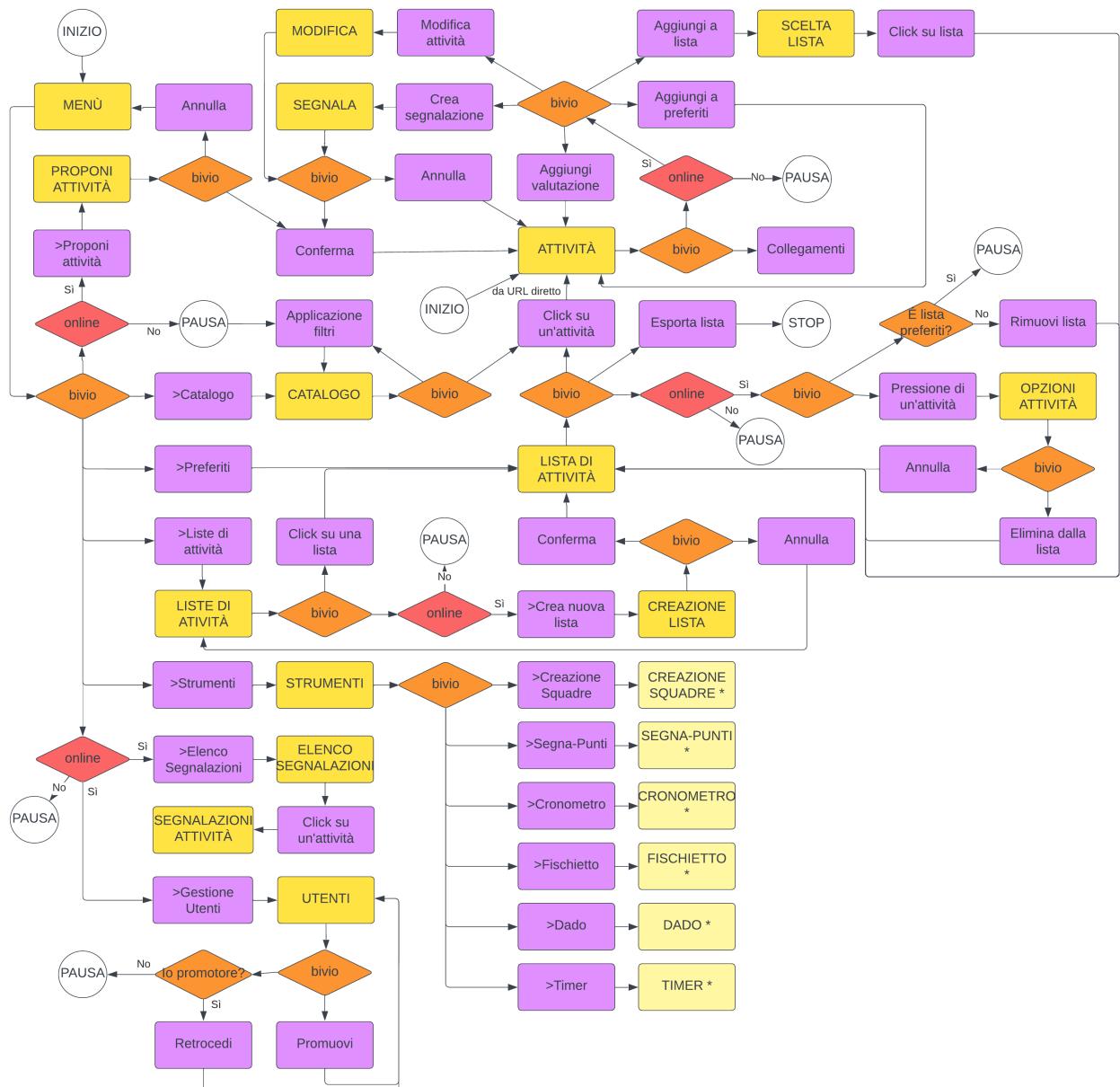
- A partire da un'attività, l'utente autenticato può cliccare sui collegamenti (se esistono), aggiungere l'attività ai preferiti o ad una lista, aggiungere una valutazione, creare una segnalazione e, nel caso di attività con l'etichetta "proposta" e se l'utente è autore dell'attività stessa, modificarla.
- A partire da Segnala, Modifica e Proponi attività, l'utente autenticato può confermare l'azione oppure annullarla, tornando al menù.
- A partire da Preferiti e da Liste di attività (dopo aver selezionato una lista), l'utente autenticato va sulla lista stessa e da lì può cliccare su un'attività, esportare la lista, se la lista non è quella dei preferiti eliminarla, ed eliminare un'attività dalla lista.
- Da Liste di attività, un utente autenticato può anche creare una nuova lista, dovendo poi confermare (andando alla lista di attività) o annullare (tornando a liste di attività).
- A partire dalla lista degli strumenti, l'utente autenticato può eseguire le stesse azioni di un utente anonimo, ovvero selezionare uno strumento presente.



Utente Amministratore

La figura sottostante descrive lo user flow relativo alle azioni che un utente amministratore, ovvero un'utente autenticato con privilegi speciali di coordinamento dell'applicazione, riesce ad eseguire. L'utente inizialmente, dal menù, può consultare il catalogo e la lista degli strumenti, andare sul proprio profilo utente, consultare la lista di liste di attività e i preferiti, proporre nuove attività, consultare l'elenco delle segnalazioni e entrare in gestione utenti.

- A partire dal catalogo, l'utente amministratore può eseguire le stesse operazioni che può eseguire un utente anonimo, ovvero applicare dei filtri e cliccare sulle attività per consultarle.
- A partire da un'attività, l'utente amministratore può cliccare sui collegamenti (se esistono), aggiungere l'attività ai preferiti o ad una lista, aggiungere una valutazione, creare una segnalazione e modificare l'attività.
- A partire da Segnala, Modifica e Proponi attività, l'utente amministratore può confermare l'azione oppure annullarla, tornando al menù.
- A partire da Preferiti e da Liste di attività (dopo aver selezionato una lista), l'utente amministratore va sulla lista stessa e da lì può cliccare su un'attività, esportare la lista, se la lista non è quella dei preferiti, eliminarla, ed eliminare un'attività dalla lista.
- Da Liste di attività, un utente amministratore può anche creare una nuova lista, dovendo poi confermare (andando alla lista di attività) o annullare (tornando a liste di attività).
- A partire dalla lista degli strumenti, l'utente amministratore può eseguire le stesse azioni di un utente anonimo, ovvero selezionare uno strumento presente.
- A partire dall'elenco delle segnalazioni, l'utente amministratore può cliccare su un'attività segnalata per vedere le segnalazioni dell'attività stessa.
- A partire da Gestione utenti, l'utente amministratore può promuovere qualsiasi utente e retrocedere gli utenti di cui è stato promotore.



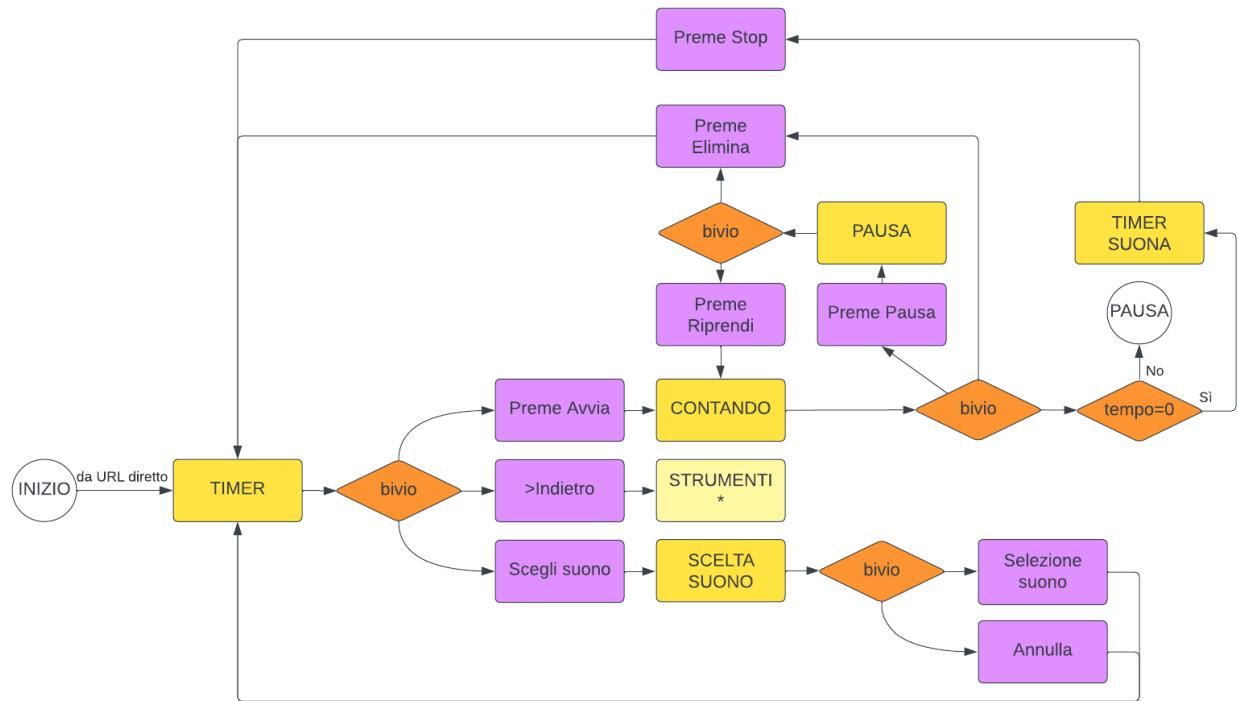
Strumenti

Timer

La figura sottostante descrive lo user flow dello strumento Timer, utilizzabile da tutti gli utenti.

- A partire da Timer, l'utente può premere Avvia, iniziando il conto alla rovescia, tornare indietro alla lista degli strumenti, selezionare ore, minuti e secondi e scegliere un suono.
 - A partire da Contando, l'utente può premere Pausa, fermando momentaneamente il timer, premenre Elimina, cancellando il timer e tornando all'inizio, oppure aspettare lo scadere dello stesso, seguito dalla riproduzione del suono scelto al punto precedente.
 - A partire da Pausa, l'utente può premere Riprendi, continuando il conto alla rovescia precedentemente fermato, o Elimina, cancellando il timer e tornando all'inizio.

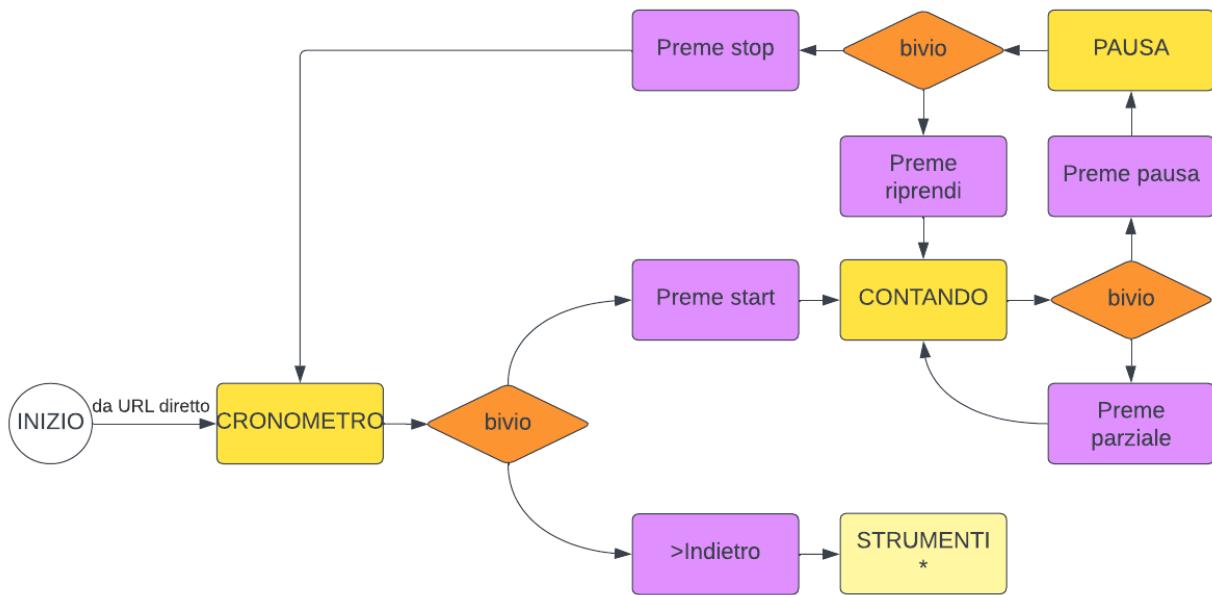
- A partire da Timer suona, l'utente deve premere Stop per poter fermare il suono in riproduzione al termine del timer. Una volta fatto ciò viene riportato all'inizio.



Cronometro

La figura sottostante descrive lo user flow dello strumento Cronometro, utilizzabile da tutti gli utenti.

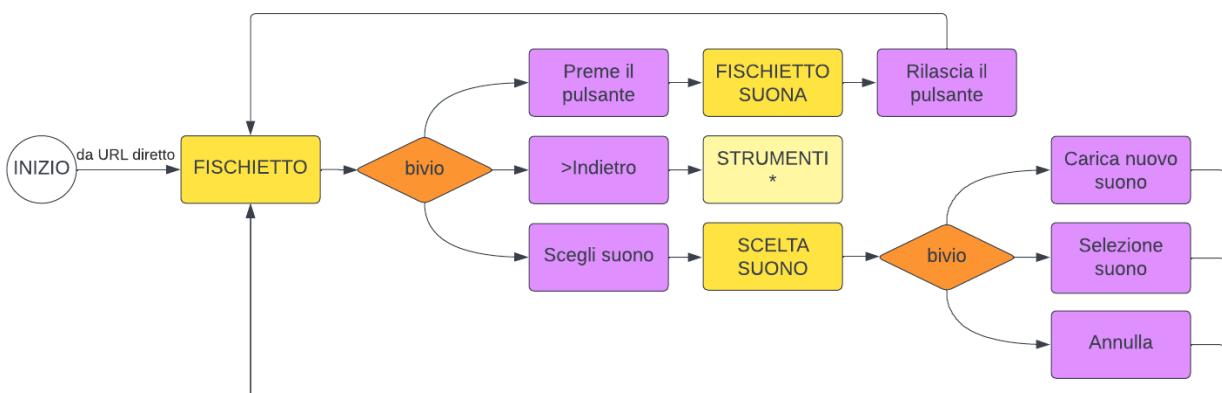
- A partire da Cronometro, l'utente può premere Start, facendo partire il cronometro e tornare indietro alla lista degli strumenti.
 - A partire da Contando, l'utente può premere Parziale, salvando il tempo di quell'istante in una lista visualizzata sotto il cronometro senza fermare il cronometro.



Fischietto

La figura sottostante descrive lo user flow dello strumento Fischietto, utilizzabile da tutti gli utenti.

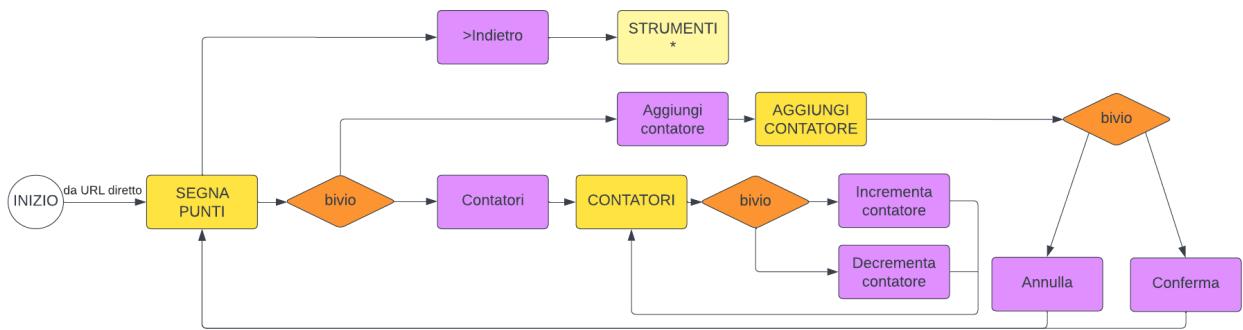
- A partire da Fischietto, l'utente può premere il pulsante per fischiare, tornare indietro alla lista degli strumenti e scegliere un suono.
- A partire da Scelta suono, l'utente può selezionare un suono oppure aggiungerne uno.
- A partire da Fischietto suona, l'utente può premere "Stop" per interrompere la riproduzione.



Segna-Punti

La figura sottostante descrive lo user flow dello strumento Segna-Punti, utilizzabile da tutti gli utenti.

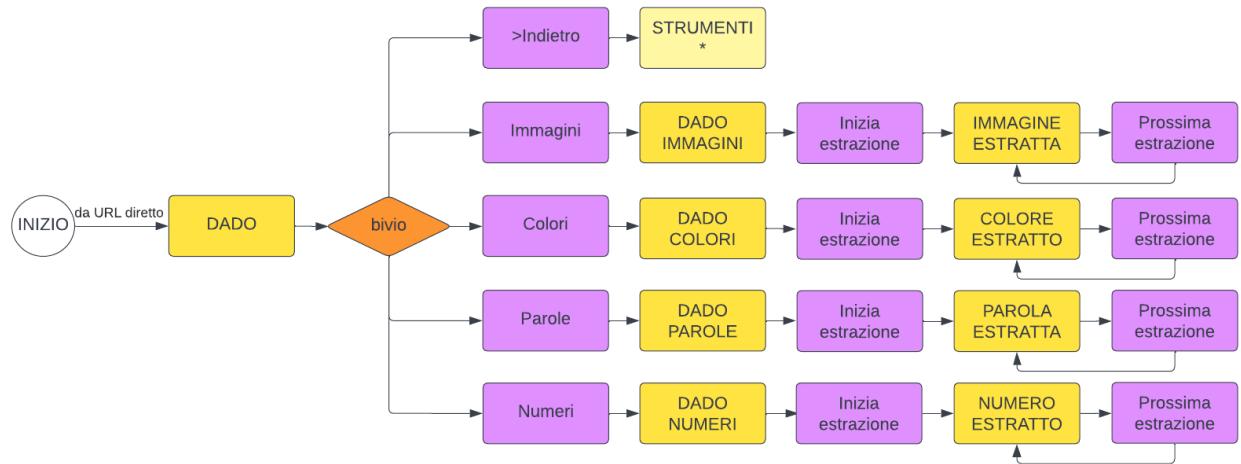
- A partire da Segna Punti, l'utente può aggiungere un contatore, tornare indietro alla lista degli strumenti o procedere al contatore.
- A partire da Aggiungi contatore, l'utente può confermare o annullare l'azione.
- A partire da Contatori, l'utente può decrementare o incrementare i contatori esistenti.



Dado

La figura sottostante descrive lo user flow dello strumento Dado, utilizzabile da tutti gli utenti.

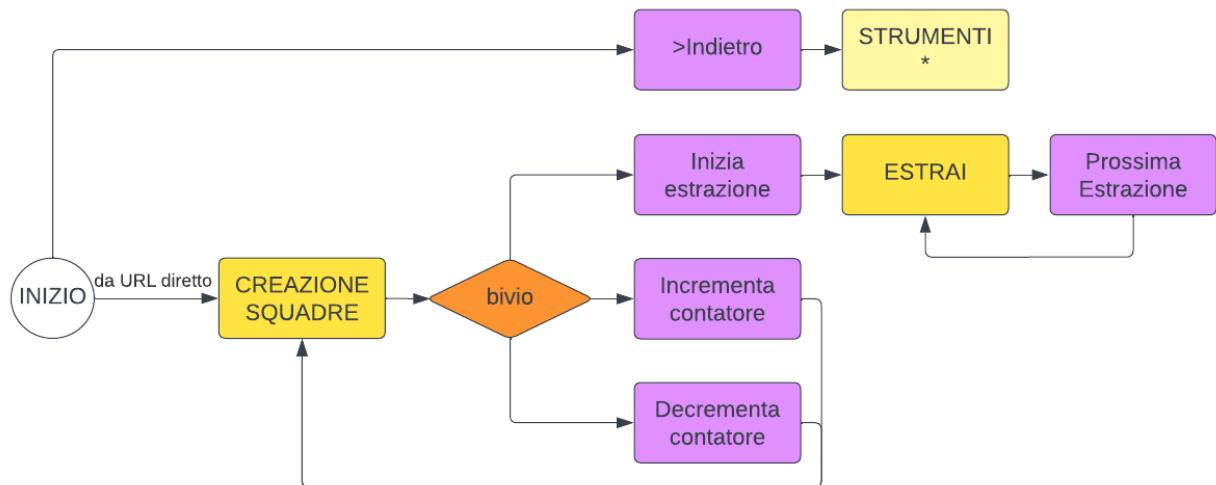
- A partire da Dado, l'utente può scegliere la tipologia di faccia usata e tornare indietro.
- A partire da Dado Immagini, Dado Colori, Dado Parole e Dado Numeri, l'utente può iniziare l'estrazione fino a che non finisce.



Creazione Squadre

La figura sottostante descrive lo user flow dello strumento Creazione Squadre, utilizzabile da tutti gli utenti.

- A partire da Creazione squadre, l'utente può cambiare i parametri grazie a incrementa e decrementa contatore o può decidere di iniziare l'estrazione, che andrà avanti fino a che non finisce.



2. Application Implementation and Documentation

Nelle sezioni precedenti abbiamo identificato le varie features che devono essere implementate per la nostra applicazione con uno schema preciso di come i nostri utenti finali possono utilizzarle nel suo flusso applicativo. Abbiamo individuato un sottoinsieme variegto di queste features, sia tra le API sia tra le funzionalità delegate al solo frontend, e lo abbiamo sviluppato nella nostra applicazione; le sezioni seguenti illustreranno le funzionalità scelte e la loro implementazione.

Il server è stato sviluppato utilizzando NodeJS e OAS3-Tools, il client è stato sviluppato utilizzando Bootstrap e JQuery, mentre per la gestione dei dati abbiamo fatto affidamento a MongoDB.

2.1 Project Structure

Server

Il codice del server è contenuto nella repository "animi-server". La sua struttura è la seguente:

api/ - cartella per la configurazione delle api

openapi.yaml - file di configurazione di OpenAPI per la generazione automatica delle route per le API e per la documentazione delle API tramite Swagger*

index.js - file di avvio del server, responsabile dell'inizializzazione del server http e della connessione iniziale al database

package.json - file di configurazione del progetto NodeJS

vercel.json - file di configurazione di Vercel, responsabile del deployment

src - cartella contenente il codice in esecuzione sul server, le API, i test e tutta la logica del sistema

app.js - file principale del server, responsabile dell'inizializzazione e dell'orchestrazione delle varie componenti

controllers/ - cartella contenente i controllori per le richieste in ingresso

<componente>.js - files responsabili per il collegamento fra route e services

models/ - cartella contenente la definizione dei modelli usati dalla libreria Mongoose per interfacciarsi con MongoDB

<modello>.js - file contenente la definizione del modello

schemas/ - cartella contenente la definizione delle strutture dati usate durante la definizione dei modelli

<schema>.js - file contenente la definizione della struttura del documento

service/ - cartella contenente l'implementazione della logica del server **<servizio>.js** - file con l'implementazione dei vari servizi offerti da Animi

test/ - cartella contenente i file di test

.test.js - file responsabile del testing dei servizi offerti dal server Animi

utils/ - cartella contenente file d'utilità

database.js - file contenente funzioni di utilità per il collegamento ed inizializzazione del database
writer.js - file contenente funzioni di utilità per la scrittura delle risposte HTTP

Client

Il codice del frontend è contenuto nella repository "animi". La sua struttura è la seguente:

assets/ - cartella contenente i file statici

bootstrap/ - cartella contenente i file di Bootstrap

css/ - cartella contenente i file CSS di Bootstrap

bootstrap.min.css - file CSS di Bootstrap

js/ - cartella contenente i file JavaScript di Bootstrap

bootstrap.bundle.min.js - file JavaScript di Bootstrap

css/ - cartella contenente i file CSS dell'intero client

style.css - file responsabile della definizione del tema del client

markdown-editor.min.css - file CSS del Markdown Editor

fonts/ - cartella contenente i file dei font utilizzati nel progetto

**** - file del font

html/ - cartella contenente alcuni componenti HTML comuni all'intero client

header.html - file contenente la definizione della barra di navigazione

modulo.html - file contenente la definizione del modulo per la creazione, proposta o modifica di un'attività

img/ - cartella contenente le immagini utilizzate nel progetto

logo.svg - l'immagine vettoriale del logo

logo<dim>.png - l'immagine del logo in formato PNG

js/ - cartella contenente i file JavaScript dell'intero client

jquery-3.6.1.min.js - file della libreria JQuery

markdown-editor.min.js - file JavaScript del Markdown Editor

markdown-it.min.js - file JavaScript del renderizzatore Markdown

purify.min.js - file JavaScript per la pulizia del codice HTML

script.js - file JavaScript contenente le funzioni di utilità per il client

<route>/ - cartella contenente i file relativi alla pagina di una specifica route

index.html - file contenente la definizione della pagina

script.js - file JavaScript contenente la logica della pagina

style.css - file CSS contenente la definizione del tema della pagina

index.html - file contenente la definizione della pagina principale del client

manifest.webmanifest - file contenente la definizione del manifest del client

sw.js - file contenente la definizione del service worker del client

2.2 Project Dependencies

Server

Le dipendenze del server sono le seguenti:

Nome	Versione
connect	^3.2.0
cors	^2.8.5
dotenv	^16.0.3
google-auth-library	^8.7.0
googleapis	^110.0.0
js-yaml	^3.3.0
jsonwebtoken	^9.0.0
mongoose	^6.9.1
oas3-tools-cors	^2.3.5

Inoltre per eseguire i test sono necessarie anche le dipendenze di sviluppo:

Nome	Versione
jest	^29.4.0
supertest	^6.3.3

("^" indica che è possibile utilizzare qualsiasi versione compatibile con quella indicata) Tali moduli NodeJS sono stati utilizzati e aggiunti al file `package.json`.

Client

Le dipendenze del client sono le seguenti:

Nome	Versione
bootstrap	v5.3.0-alpha1
jquery	v3.6.1
markdown-it	v12.2.0
krajee-markdown-editor	v1.0.1

purify	v0.7.4
fontawesome-free	v5.12.0

2.3 Project Data

Per la gestione dei dati, Animati utilizza un database MongoDB. Tale database è stato creato su MongoDB Atlas, un servizio cloud offerto da MongoDB. Il database è stato creato tramite il servizio Atlas e contiene le seguenti collezioni:

Nome	Descrizione
Catalogo	Contiene i dati delle attività
Etichette	Contiene le etichette delle attività
Liste	Contiene le liste di attività
Segnalazioni	Contiene le segnalazioni di attività
Utenti	Contiene i dati degli utenti registrati
Valutazioni	Contiene le valutazioni delle attività

Per rappresentare i dati, Animati utilizza i seguenti schemi:

Attività

```
{
  banner: String,
  informazioni: InformazioniSchema,
  collegamenti: [CollegamentoSchema],
  ultimaModifica: {
    type: Date,
    default: Date.now
  },
  autore: {
    type: mongoose.Types.ObjectId,
    ref: 'Utenti',
    default: mongoose.Types.ObjectId('000000000000000000000000')
  },
  mediaValutazioni: Number,
  numeroSegnalazioni: Number
}
```

Collegamento

```
{
  nome: {
    type: String,
```

```

        required: true
    },
    link: {
        type: String,
        required: true
    }
}

```

Etichetta

```

{
    nome: {
        type: String,
        required: true
    },
    descrizione: {
        type: String,
        required: true
    },
    categoria: {
        type: String,
        required: true
    }
}

```

Informazioni

```

{
    titolo: String,
    descrizione: String,
    etàMin: Number,
    etàMax: Number,
    durataMin: Number,
    durataMax: Number,
    giocatoriMin: Number,
    giocatoriMax: Number,
    giocatoriPerSquadra: Number,
    giocatoriPerSquadraSet: Boolean,
    numeroSquadre: Number,
    numeroSquadreSet: Boolean,
    etichette: [EtichetteSchema]
}

```

Lista

```
{
    nome: {
        type: String,

```

```

        required: true
    },
    autore: {
        type: mongoose.Types.ObjectId,
        ref: 'Utenti',
        required: true
    },
    'attività': {
        type: [mongoose.Types.ObjectId],
        ref: 'Catalogo',
        default: []
    },
    ultimaModifica: {
        type: Date,
        default: Date.now
    }
}

```

Segnalazione

```

{
    messaggio: {
        type: String,
        required: true
    },
    titolo: {
        type: String,
        required: true
    },
    'attività': {
        type: mongoose.Types.ObjectId,
        ref: 'Catalogo',
        required: true
    },
    autore: {
        type: mongoose.Types.ObjectId,
        ref: 'Utenti',
        required: true
    }
}

```

Utenti

```

{
    email: {
        type: String,
        required: true
    },
    ruolo: {

```

```
        type: String,
        default: 'autenticato'
    },
    promossoDa: {
        type: mongoose.Types.ObjectId,
        ref: 'Utenti',
        default: mongoose.Types.ObjectId('0000000000000000000000000000')
    },
    immagine: {
        type: String,
        default: 'https://animi.app/assets/img/logo512.png'
    }
}
```

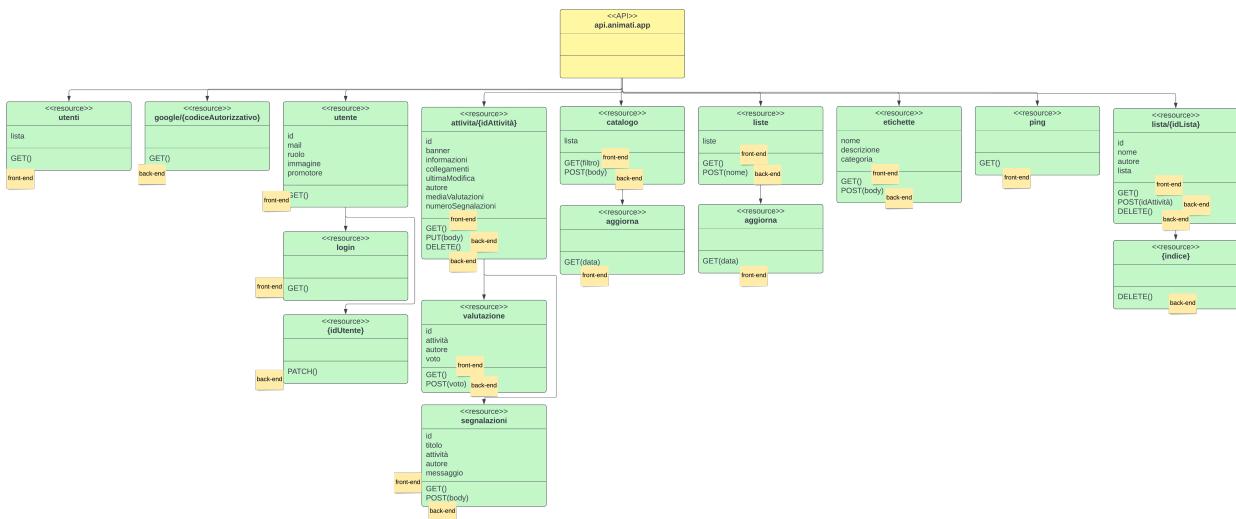
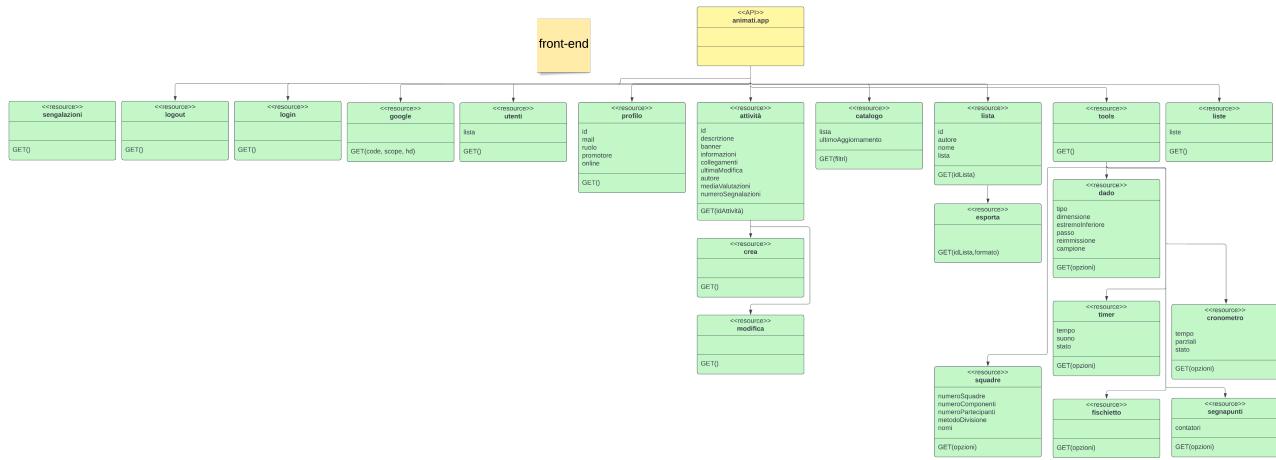
Valutazione

```
{
    voto: {
        type: Number,
        required: true
    },
    'attività': {
        type: mongoose.ObjectId,
        ref: 'Catalogo',
        required: true
    },
    autore: {
        type: mongoose.ObjectId,
        ref: 'Utenti',
        required: true
    }
}
```

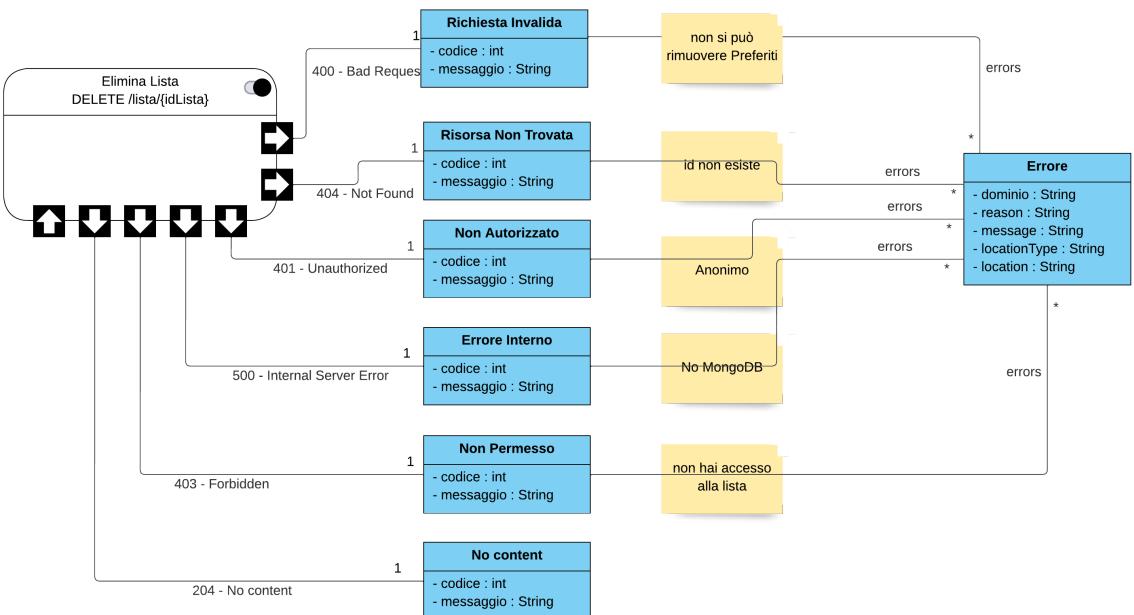
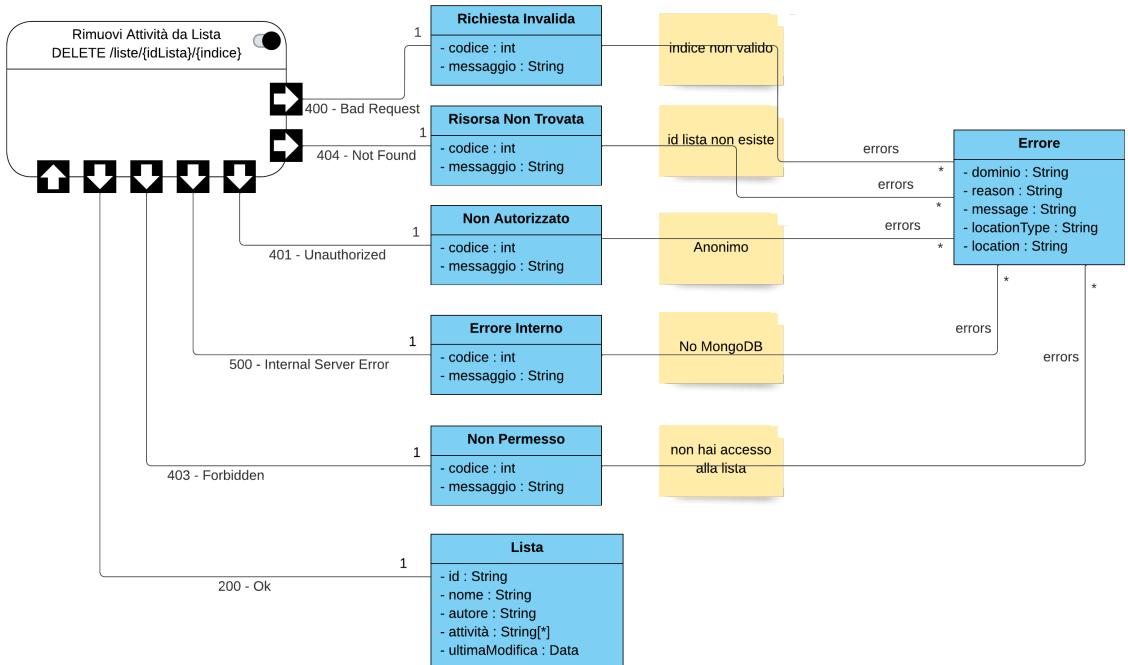
2.4 Project APIs

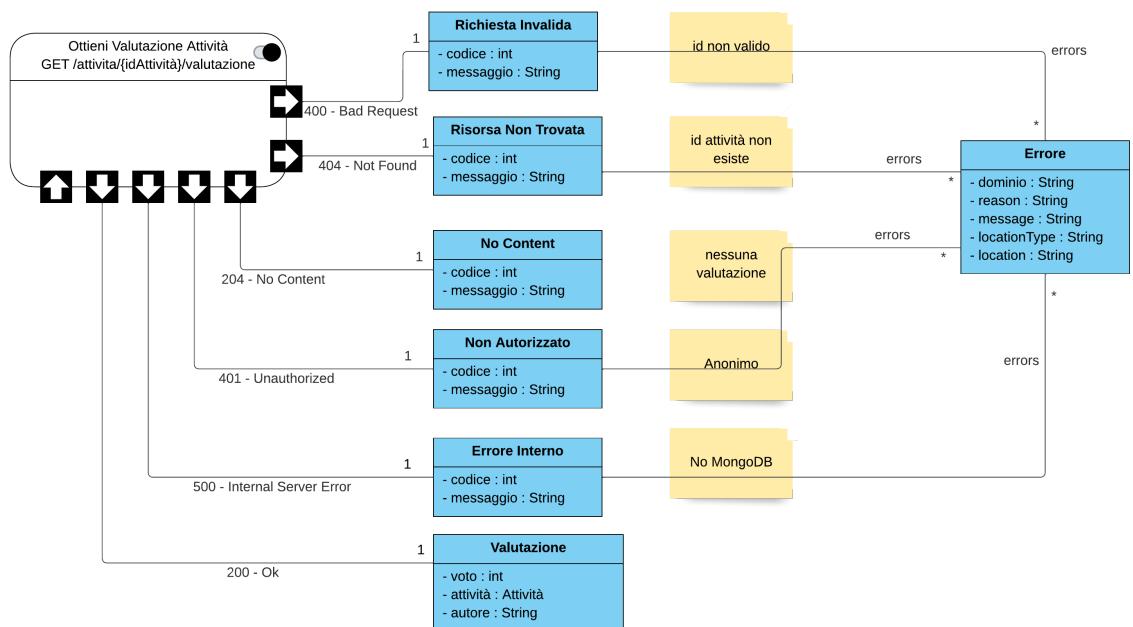
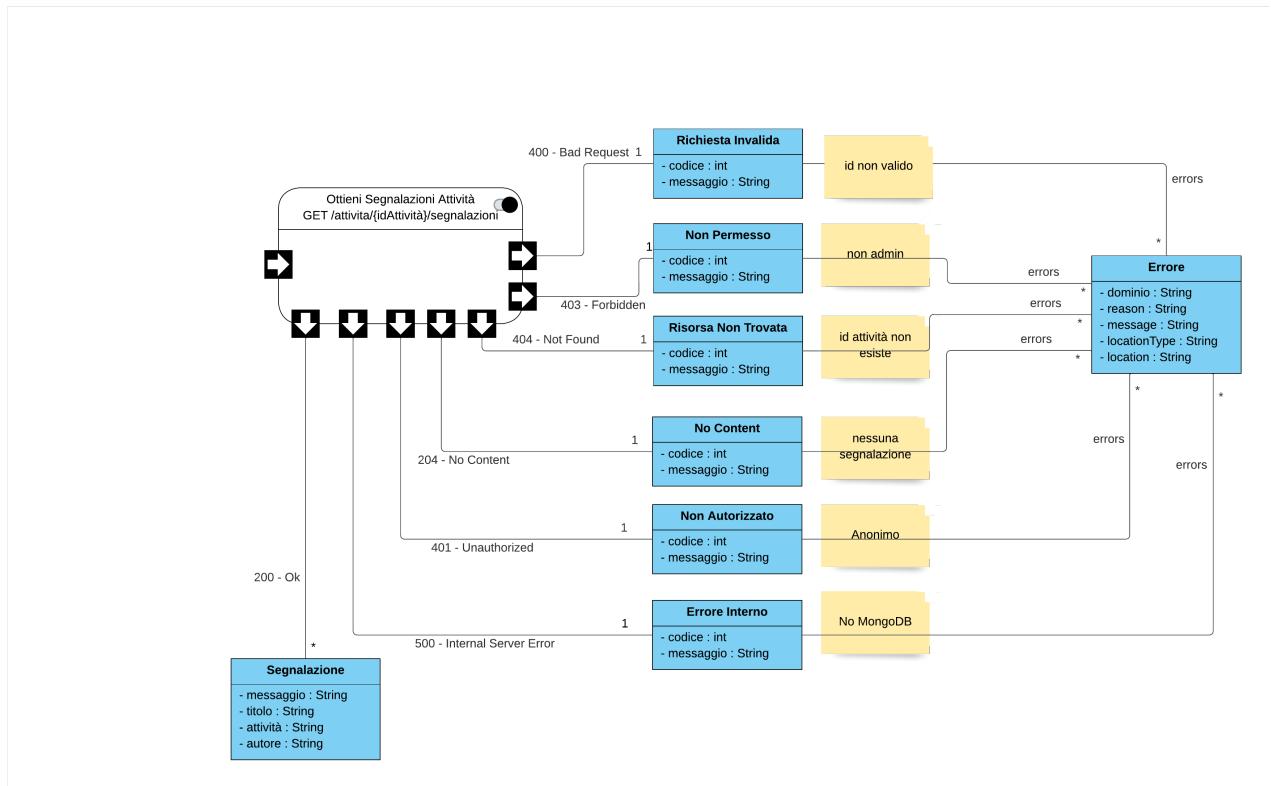
In questa parte del documento vengono descritte le varie API individuate a partire dal diagramma delle classi del documento D3-T51 e quali sono state implementate. Useremo un diagramma per rappresentare l'estrazione delle risorse a partire dal class diagram e uno per rappresentare le risorse sviluppate.

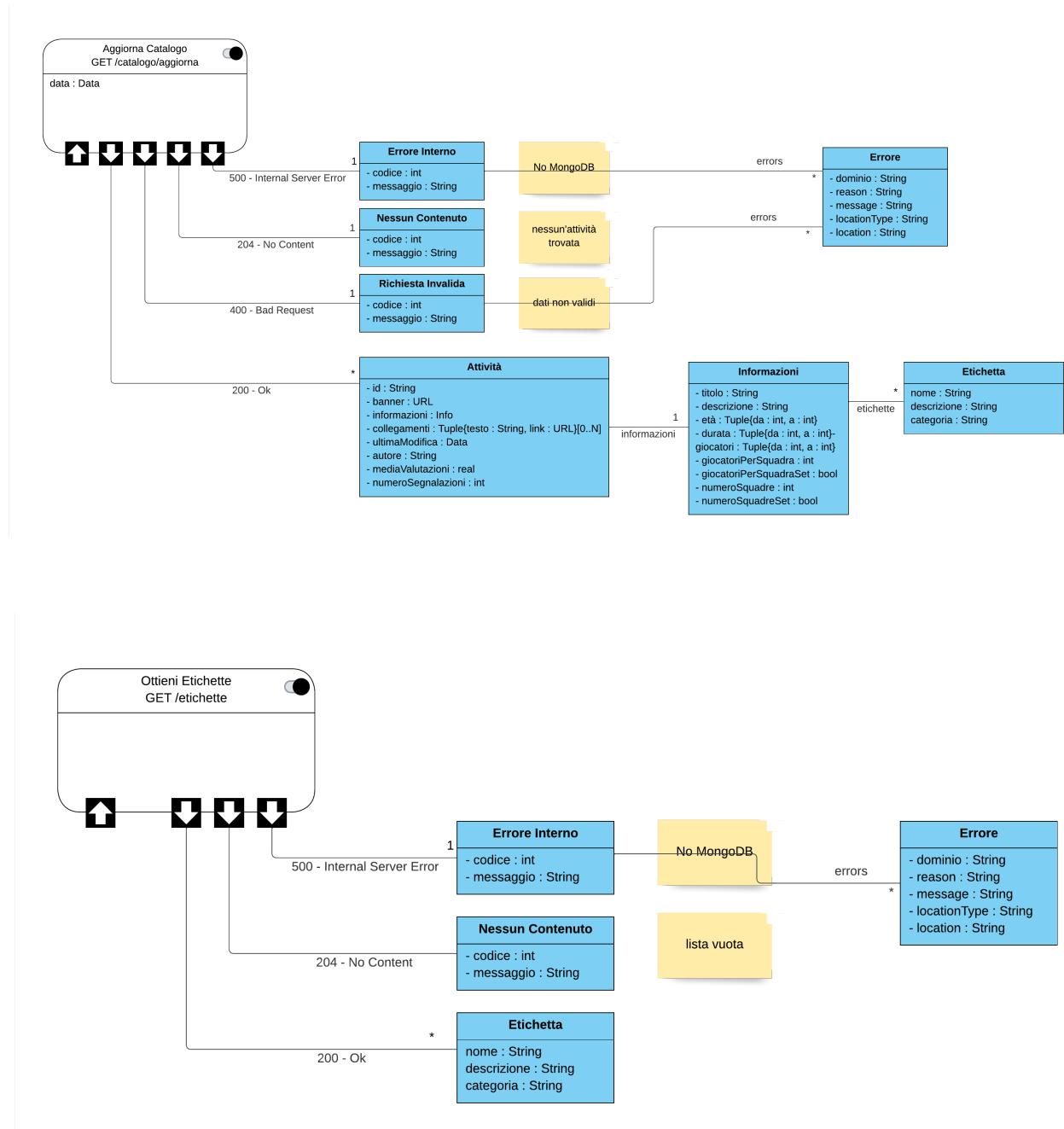
Estrazione risorse

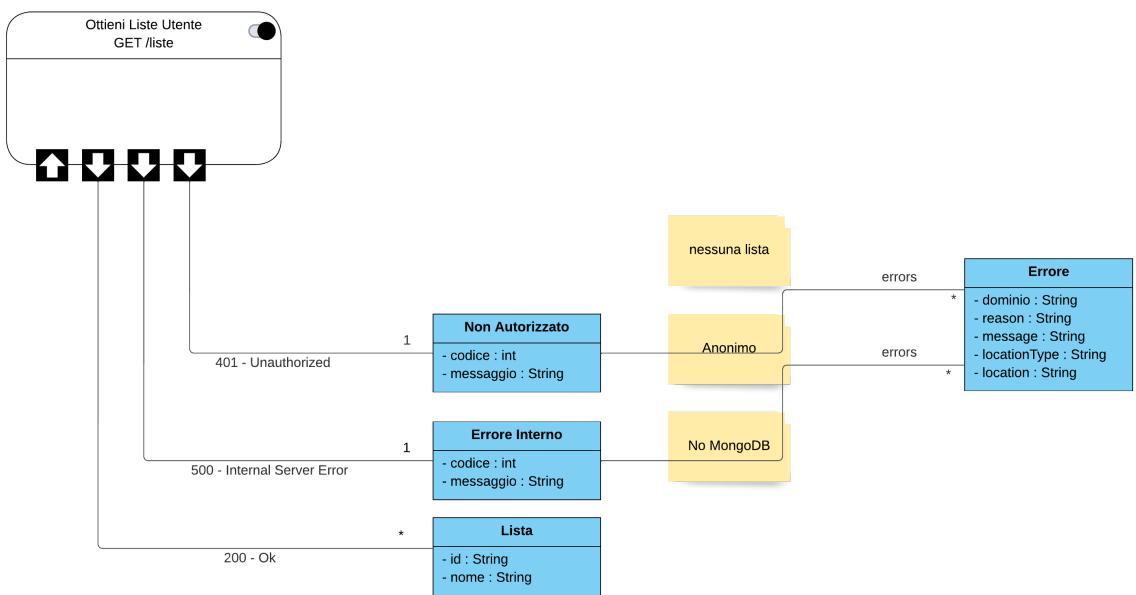
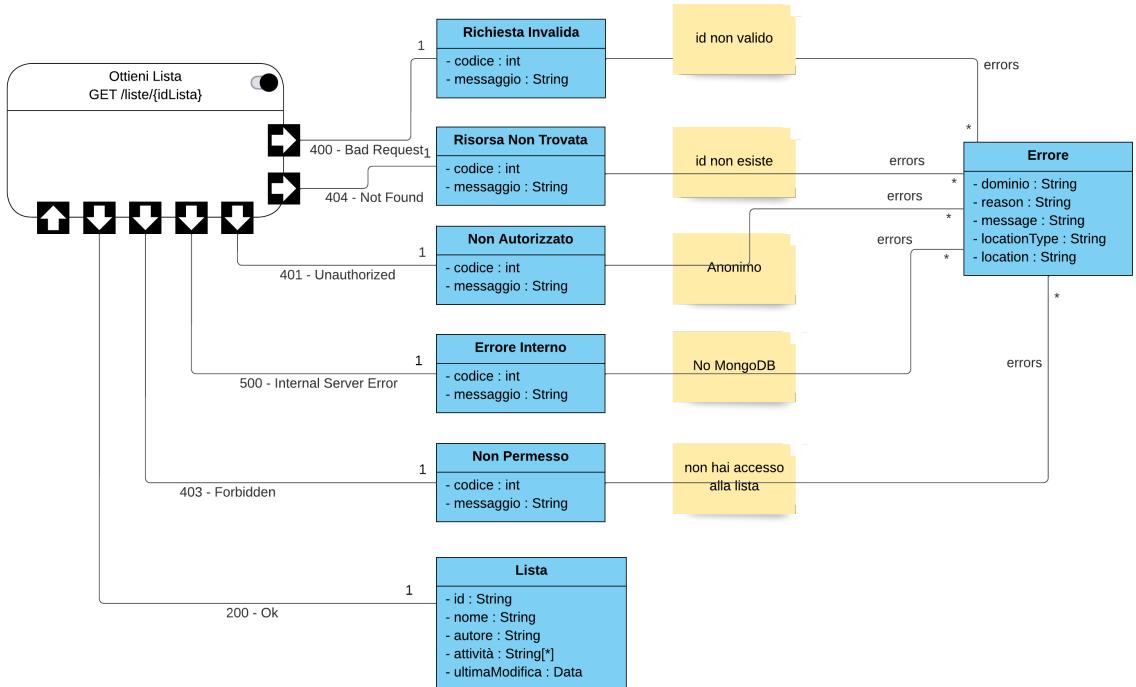


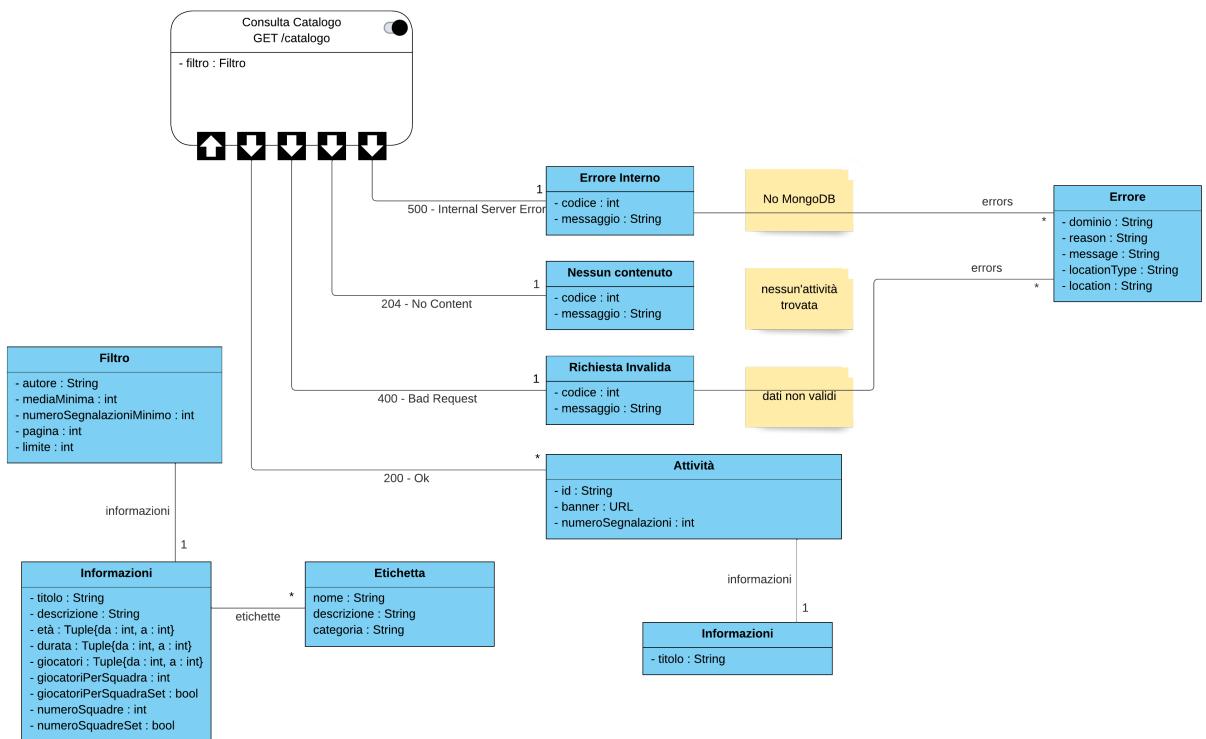
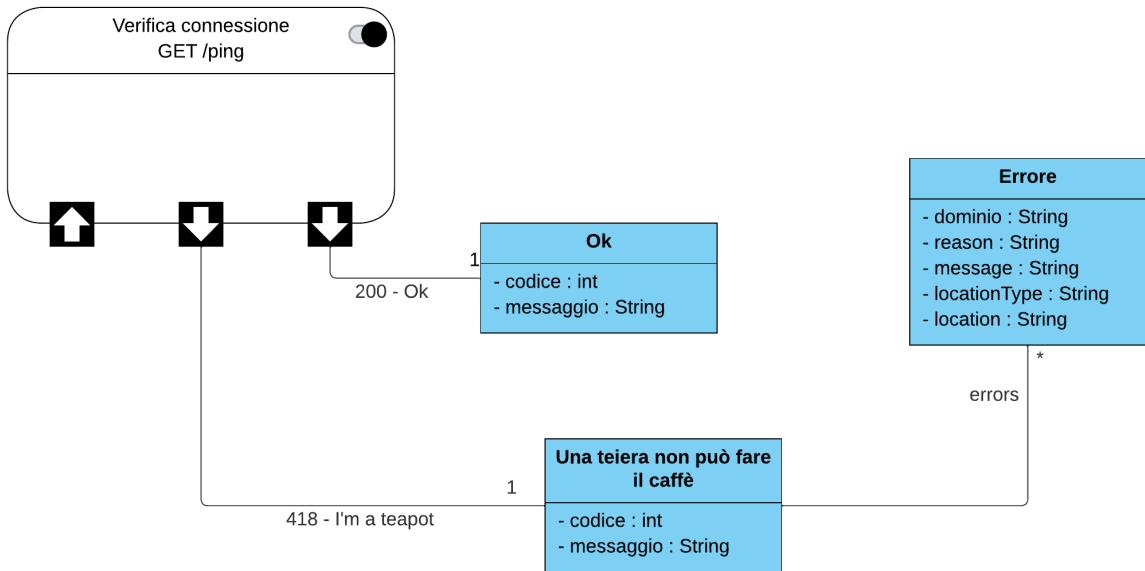
Risorse

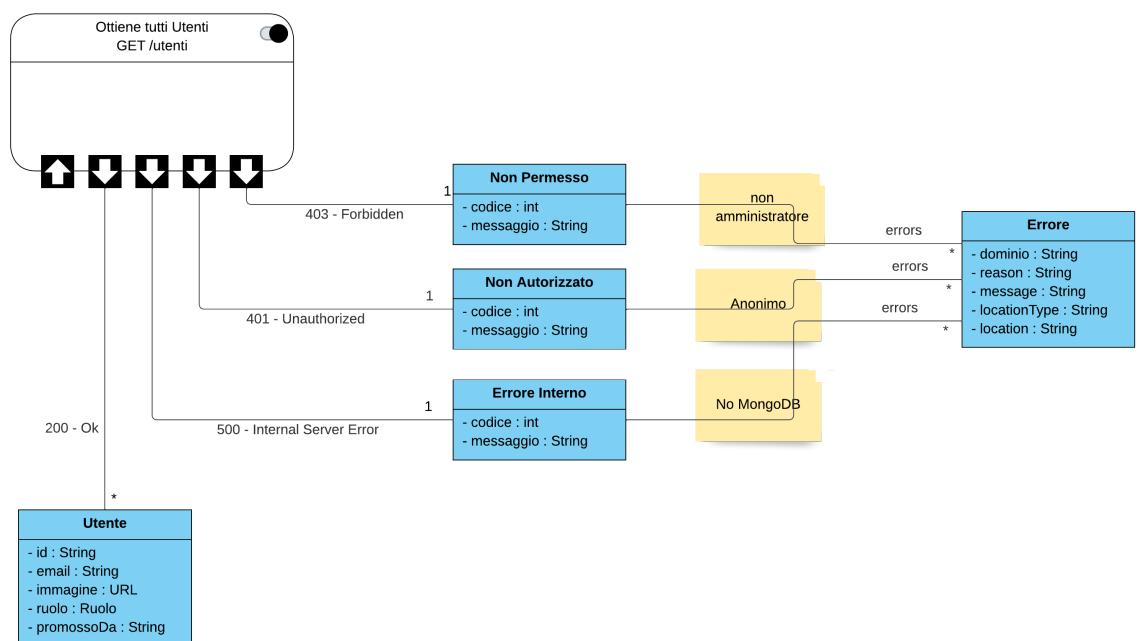
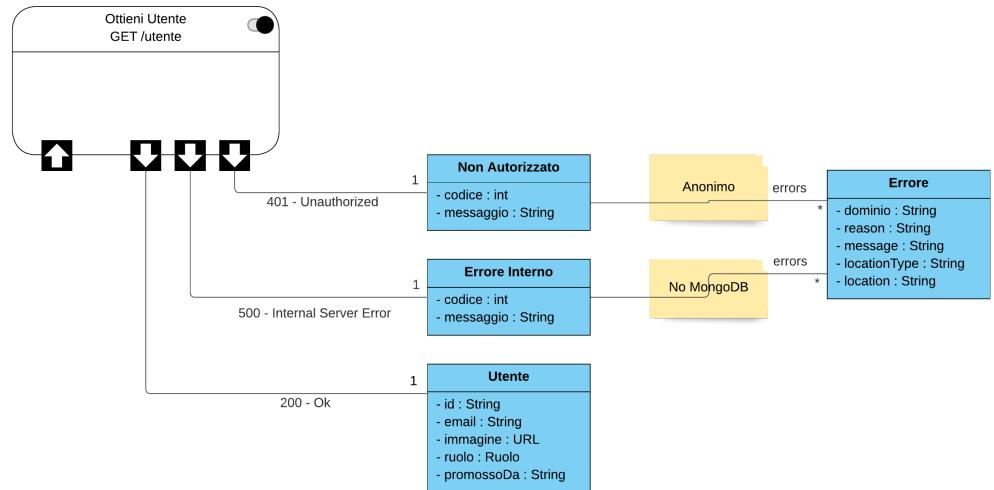


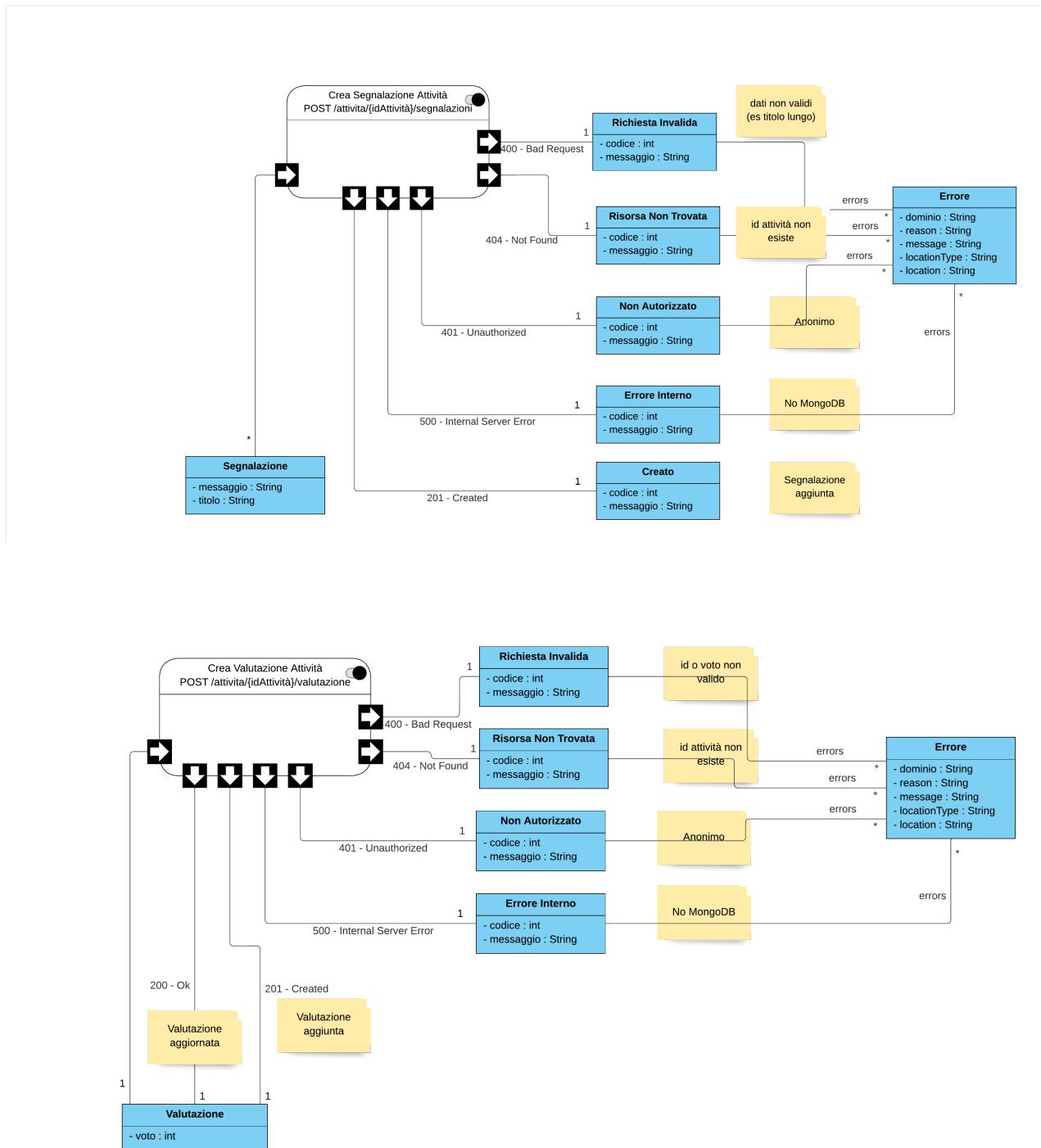


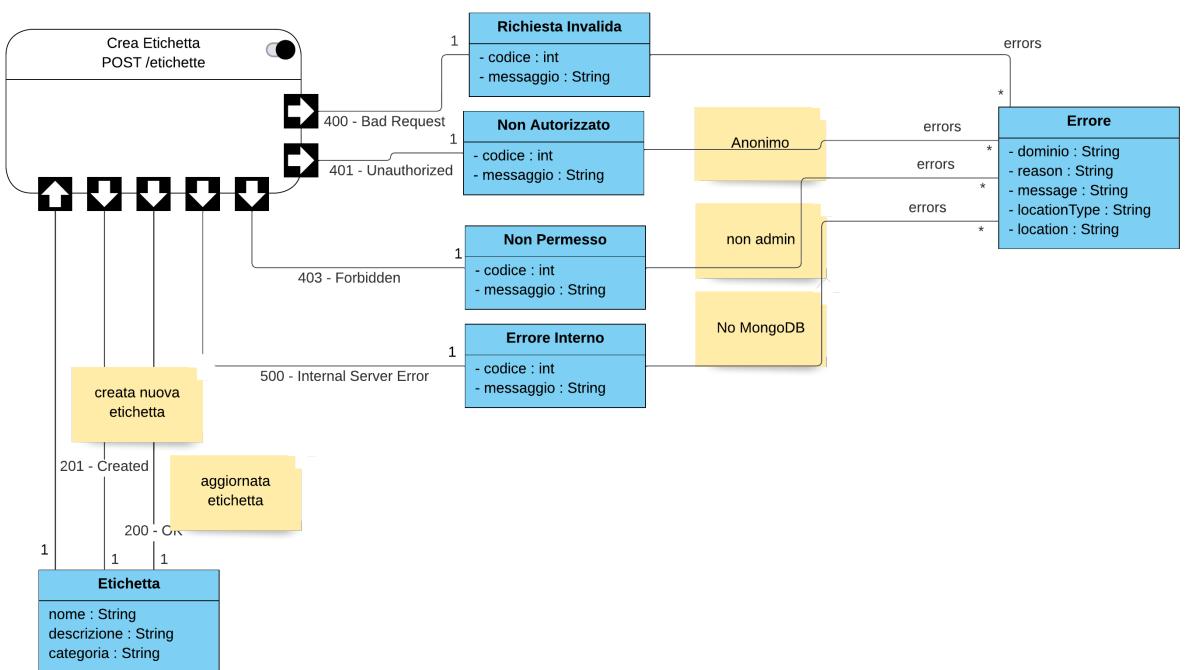
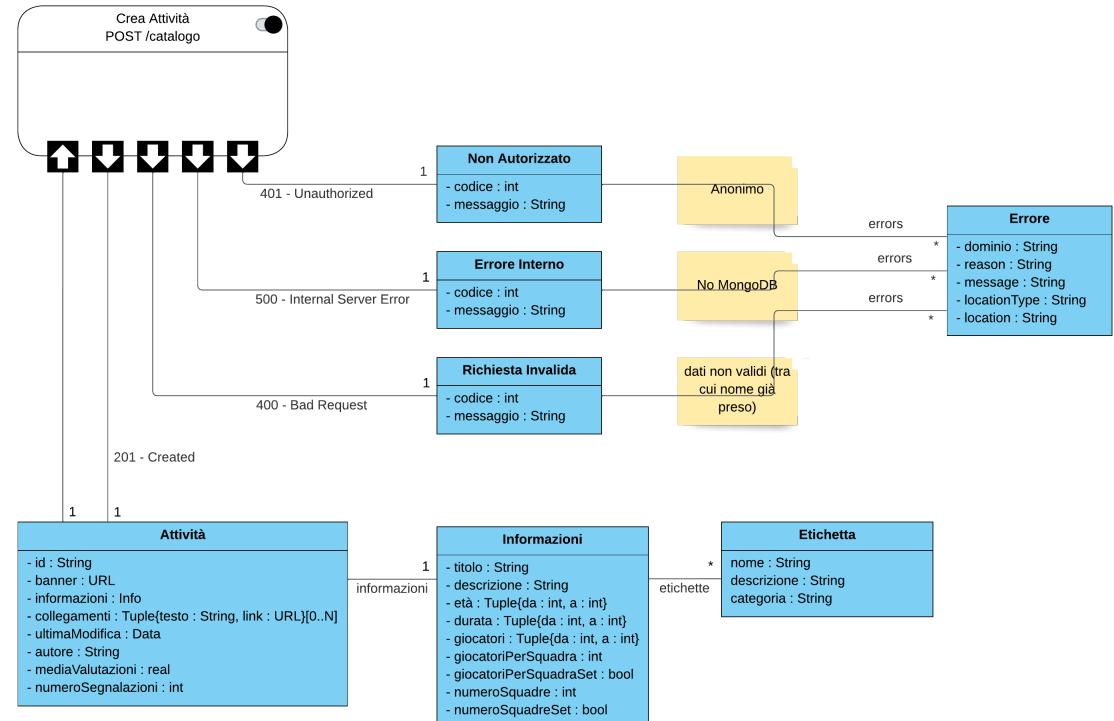


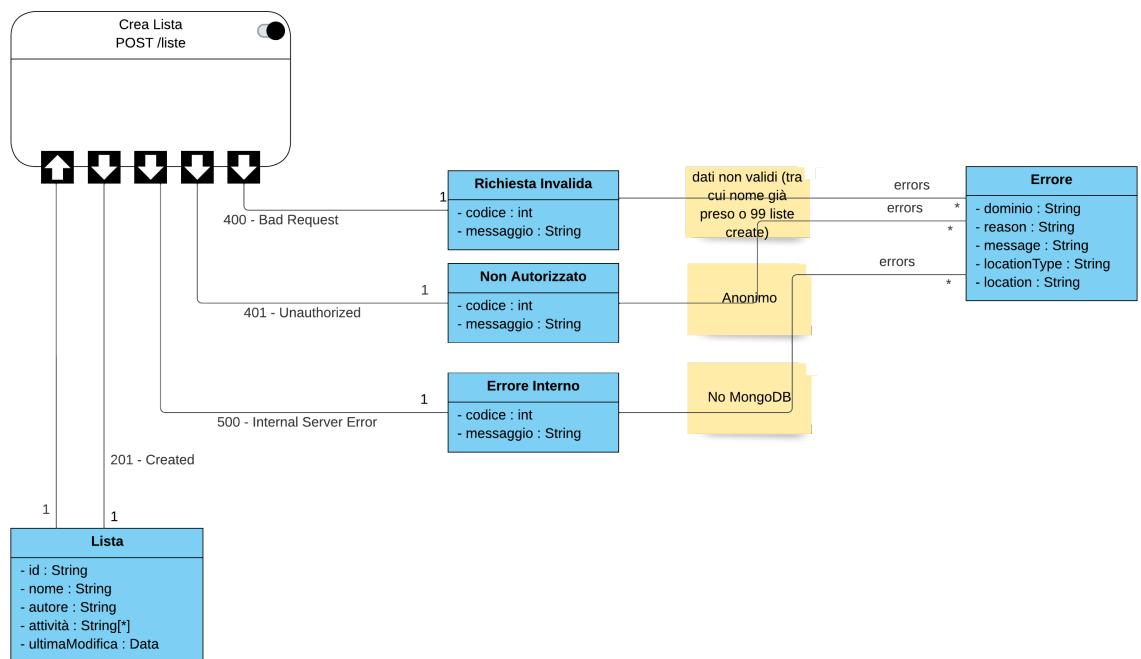
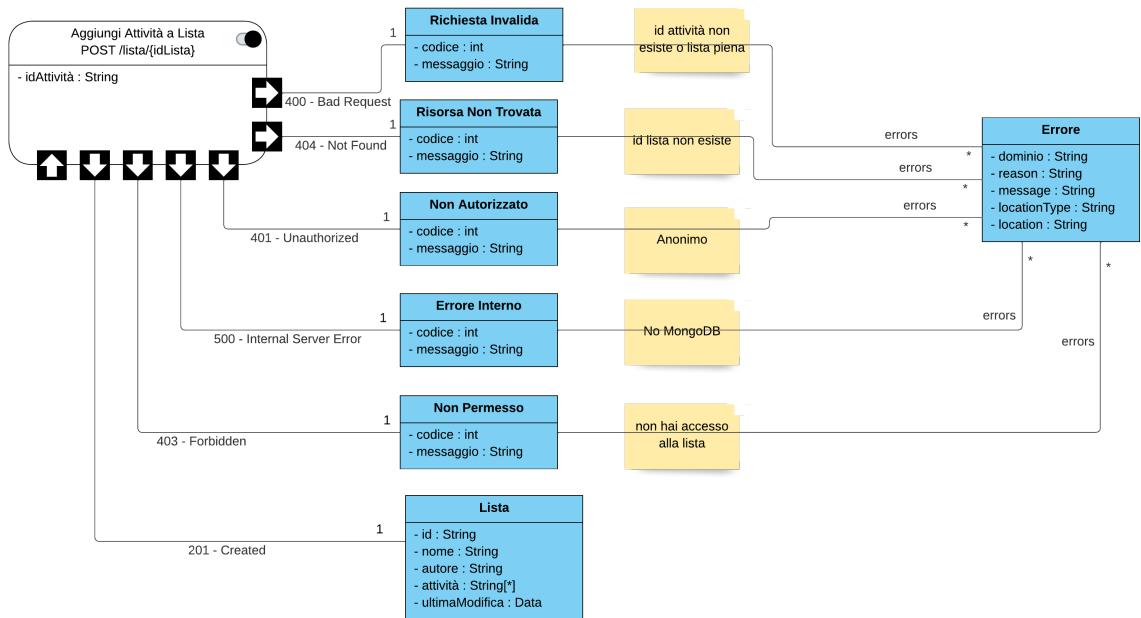












2.5 Sviluppo API In seguito descriveremo il funzionamento delle varie API sviluppate nel progetto.

Il routing viene generato a partire della specifica OpenAPI presente nel file *openapi.yaml* in *animati-server/src/api* grazie al modulo *oas3-tools-cors* di Nodejs. La logica delle API è definita nei file nella cartella *src/controllers*, che si occupano semplicemente dell'inoltro di richiesta e risposta tra i router e le funzioni di logica del sistema, e nei file nella cartella *src/service* che si occupano dell'effettiva elaborazione delle richieste.

Ogni API è progettata per restituire un messaggio di errore 500 in caso di errori interni al server o di comunicazione con MongoDB, tal volta realizzato mediante l'uso di try-catch, tal volta nel .catch delle Promises, o infine nella funzione di utility *writer.js*.

Il codice è reperibile nella repository github <https://github.com/IS-T51/animati-server>.

API per la risorsa Utente

[...]

API per la risorsa Attività

[...]

API per la risorsa Lista

Di seguito descriviamo le API per gestire le varie azioni eseguibili sulla risorsa Lista, la loro logica si articola nel controller *src/controllers/Lista.js* e in *src/service/ListaService.js*.

Ognuna di queste API può essere eseguita solo specificando un token d'accesso come parametro nell'header, in quanto rappresenta una richiesta eseguibile solo da un utente autenticato. Per verificare l'autorizzazione dell'utente le API utilizzano l'API *getUtente* come middleware, quindi in caso di autenticazione non corretta restituiscono un messaggio di errore con codice 401.

API per l'inserimento di un'attività in una lista personale

L'API *aggiungiAttivitàALista* ha lo scopo di aggiungere un'attività all'elenco di attività di una specifica istanza della risorsa Lista e di salvare l'istanza modificata nel database nella collezione Liste. Riceve in input l'id della lista in cui si desidera aggiungere l'attività (passato per percorso) e l'id dell'attività da aggiungere (passato come parametro di query). Viene fornita in output la lista aggiornata se tutto è andato a buon fine. In caso contrario si restituisce un messaggio d'errore, ovvero nel caso in cui la richiesta non sia ben formulata o la lista sia già piena (errore 400), oppure in caso la lista sia inesistente (404) o in caso l'utente che effettua la richiesta non ne sia l'autore (403), oppure l'utente che effettua la richiesta non sia autenticato correttamente (401), oppure in caso si verifichi un errore interno al server o di comunicazione con MongoDB (500).

Creazione di una lista

L'API *aggiungiLista* ha lo scopo di creare una nuova istanza della risorsa Lista e di salvarla nel database nella collezione Liste. Riceve in input alla richiesta un body con il nome da assegnare della risorsa lista che viene creata. Viene fornita in output la risorsa appena creata se tutto è andato a buon fine. In caso contrario si restituisce un messaggio d'errore, ovvero nel caso in cui la richiesta non sia ben formulata o l'utente abbia raggiunto il massimo numero di liste (errore 400), oppure in caso una lista con il nome fornito e l'utente come autore sia già esistente (400), oppure l'utente che effettua la richiesta non sia autenticato correttamente (401), oppure in caso si verifichi un errore interno al server o di comunicazione con MongoDB (500).

Eliminazione di una lista

L'API *eliminaLista* ha lo scopo di eliminare un'istanza della risorsa Lista dal database. Riceve come unico parametro di input l'id della lista che si intende eliminare. Se tutto è andato a buon fine non ritorna niente (messaggio di

successo 204). In caso contrario si restituisce un messaggio d'errore, ovvero nel caso in cui la richiesta non sia ben formulata o l'utente stia cercando di eliminare la propria lista dei Preferiti (errore 400), oppure in caso la lista sia inesistente (404) o in caso l'utente che effettua la richiesta non ne sia l'autore (403), oppure l'utente che effettua la richiesta non sia autenticato correttamente (401), oppure in caso si verifichi un errore interno al server o di comunicazione con MongoDB (500).

Ricerca delle informazioni di una lista

L'API *getLista* ha lo scopo di ricercare una precisa lista nel database, dato il suo id passato come unico parametro in input alla richiesta. Viene fornita in output la precisa risorsa richiesta se presente con titolo ed elenco di attività in lista. In caso contrario si restituisce un messaggio d'errore, ovvero nel caso in cui la richiesta non sia ben formulata, oppure in caso la lista sia inesistente (404) o in caso l'utente che effettua la richiesta non ne sia l'autore (403), oppure l'utente che effettua la richiesta non sia autenticato correttamente (401), oppure in caso si verifichi un errore interno al server o di comunicazione con MongoDB (500).

Ricerca di tutte le proprie liste personali

L'API *getListe* ha lo scopo di ricercare ed elencare tutte le liste di uno specifico utente salvate nel database. Non c'è niente in input alla richiesta, ma viene fornito in output l'elenco di tutte le liste dell'utente presenti, elenco che non sarà mai vuoto perché ogni utente deve avere in ogni momento la lista dei preferiti. In caso la richiesta non vada a buon fine si restituisce un messaggio d'errore, ovvero nel caso in cui la richiesta non sia ben formulata, oppure l'utente che effettua la richiesta non sia autenticato correttamente (401), oppure in caso si verifichi un errore interno al server o di comunicazione con MongoDB (500).

API per la rimozione di un'attività da una lista personale

L'API *rimuoviAttivitàDaLista* ha lo scopo di rimuovere un'attività dall'elenco di attività di una specifica istanza della risorsa Lista e di salvare l'istanza modificata nel database nella collezione Liste. Riceve in input l'id della lista in cui si desidera aggiungere l'attività (passato per percorso) e l'indice all'interno della lista dell'attività da rimuovere (passato come parametro di query). Viene fornita in output la lista aggiornata se tutto è andato a buon fine. In caso contrario si restituisce un messaggio d'errore, ovvero nel caso in cui la richiesta non sia ben formulata (errore 400), oppure in caso la lista sia inesistente (404) o in caso l'utente che effettua la richiesta non ne sia l'autore (403), oppure l'utente che effettua la richiesta non sia autenticato correttamente (401), oppure in caso si verifichi un errore interno al server o di comunicazione con MongoDB (500).

API per la risorsa Etichetta

Di seguito descriviamo le API per gestire le varie azioni eseguibili sulla risorsa Etichetta, la loro logica si articola nel controller *src/controllers/Etichette.js* e in *src/service/EtichetteService.js*.

Creazione di un etichetta

L'API *aggiungiEtichetta* ha lo scopo di creare una nuova istanza della risorsa Etichetta e di salvarla nel database nella collezione Etichette. Riceve in input alla richiesta un body con i vari attributi della risorsa etichetta che viene creata. Viene fornita in output la risorsa appena creata se tutto è andato a buon fine. In caso contrario si restituisce un messaggio d'errore (nel caso in cui la richiesta non sia ben formulata oppure l'utente che effettua la richiesta non sia autenticato correttamente o non abbia i privilegi di amministratore oppure in caso si verifichi un errore interno al server o di comunicazione con MongoDB). Nel caso in cui il nome dell'etichetta da creare coincida con il nome di un'etichetta già presente nel database, l'istanza presente sarà aggiornata. Quest'API può essere eseguita solo specificando un token d'accesso come parametro nell'header, in quanto rappresenta una richiesta eseguibile solo da un utente autenticato e con i privilegi di amministratore. Per verificare l'autorizzazione dell'utente l'API utilizza l'API *getUtente* come middleware.

Ricerca di tutte le etichette

L'API *ottieneEtichette* ha lo scopo di ricercare ed elencare tutte le etichette salvate nel database. Non c'è niente in input alla richiesta, ma viene fornito in output l'elenco di tutte le etichette presenti, una risposta con codice 204 se nessuna etichetta è salvata nel database, oppure un messaggio di errore in caso di errori interni al server o di comunicazione con MongoDB. Per utilizzare questa API non è necessaria autenticazione.

API di default

Il controller *Default.js* e il file di business logic *DefaultService.js* definiscono il funzionamento dell'API di default, ideata per verificare la connessione del client con il server. Qualora il server sia funzionante, la funzione *ping* di *DefaultService.js* restituirà una Promise che può solo avere successo, il messaggio di successo sarà inserito nella risposta da fornire al richiedente dalla funzione *ping* del controller, quindi il metodo GET presso l'endpoint /ping restituirà sempre un messaggio 200 di successo.

3. API documentation

4. FrontEnd Implementation

Il FrontEnd fornisce all'utente un metodo intuitivo per interagire con il sistema.

Sono presenti pagine dedicate la visualizzazione e proposta di attività, la manipolazione di liste e per l'utilizzo degli strumenti.

Data la dimensione del progetto, si è optato per rendere disponibili alcune funzionalità (quali valutazioni, etichette e segnalazioni) solo attraverso API.

The screenshot shows the 'Catalogo' (Catalog) page of the application. At the top, there is a navigation bar with icons for user profile, catalog, and tools, along with a search and filter icon. Below the header, the title 'Catalogo' is displayed in large bold letters, followed by the subtext 'Qui puoi trovare tutte le attività.' (Here you can find all the activities.) On the right side of the header is a 'Filtro' (Filter) button with a magnifying glass icon.

The main area contains a form for filtering activities:

- Titolo**: A text input field containing 'Contenenti nel titolo...' (Containing in the title...).
- Descrizione**: A text input field containing 'Contenenti nella descrizione...' (Containing in the description...).
- Età**: A dropdown menu with 'Per persone di...' (For people of...) and a 'anni' (years) unit selector.
- Giocatori**: A dropdown menu with '3' selected.
- Durata**: A dropdown menu with '20' selected and 'minuti' (minutes) as the unit.

Below the filter form is an 'Applica filtri' (Apply filters) button.

The page displays a grid of eight activity cards, each with a title, an image, and a 'Vai all'attività' (Go to activity) button:

- Crea il tuo slime**: Shows a colorful swirl of slime. Button: Vai all'attività.
- Antistress**: Shows several cartoonish stress balls. Button: Vai all'attività.
- Origami**: Shows various origami shapes like fish and stars. Button: Vai all'attività.
- Popup art**: Shows a 3D paper craft scene. Button: Vai all'attività.
- Braccialetti**: Shows a hand wearing colorful beaded bracelets. Button: Vai all'attività.
- Sculpture**: Shows various colorful clay or plasticine sculptures. Button: Vai all'attività.
- Glitter Art**: Shows two figures carrying a Christmas tree decorated with glitter. Button: Vai all'attività.
- Disegni nascosti**: Shows a hand drawing on a dark surface with colorful pens. Button: Vai all'attività.

Esempio di catalogo filtrato in base al numero di giocatori e alla durata

Animi Catalogo Strumenti ▾

Attività

Qui puoi visualizzare le informazioni dell'Attività.

Roverino



Autore: #63b2cf5e249a3fa2cf491e1c
Ultima modifica: 2/1/2023 15:54:26

Gioco da svolgere in spazi ampi

Materiali

- 1 roverino
- 2 bastoni

Svolgimento

Lo scopo è lanciare il roverino nel bastone tenuto in mano da un componente della propria squadra, lo si può passare ma non si possono fare più di 3 passi con il roverino in mano.

È vietato strappare di mano il roverino ad un avversario, è possibile solo intercettarlo

Se il roverino cade il possesso passa alla squadra che **NON** ha toccato per ultimo il roverino.

È possibile a propria discrezione delimitare il campo e effettuare delle rimesse stile calcio se il roverino esce da esso.

Vince chi... segna più punti

Come creare un roverino 

Modifica attività

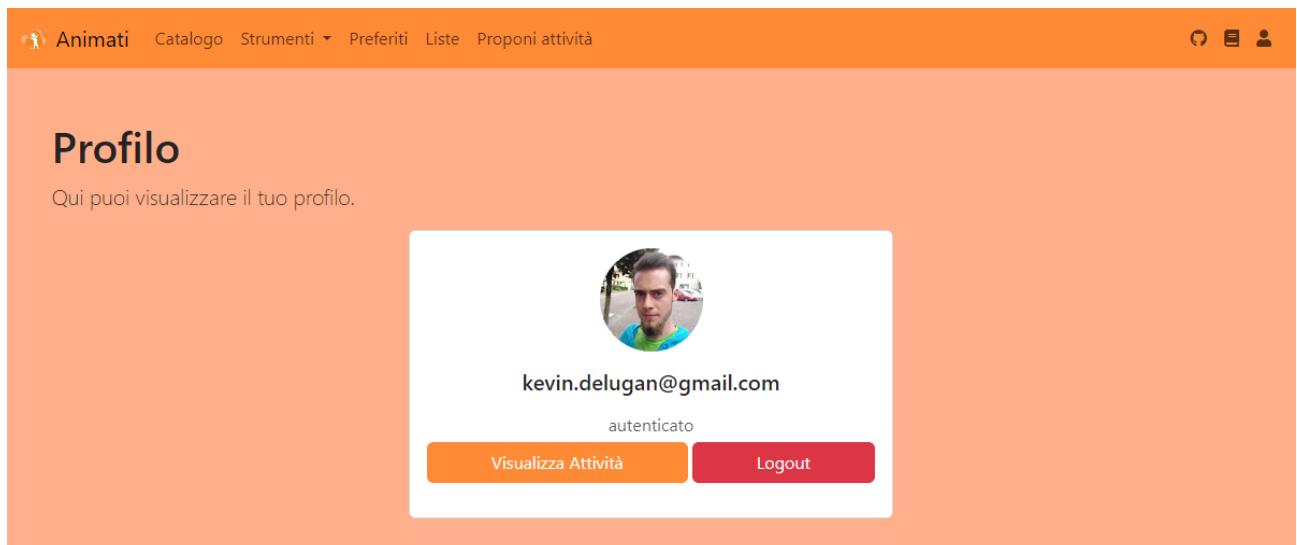
Aggiungi a lista ▾

Esempio di attività proposta da un utente



Esempio di strumento: timer impostato a un minuto e 23 secondi

Nel FrontEnd, attraverso l'icona posta in fondo al menù, è possibile effettuare l'accesso per accedere con Google per rendere visibili le sezioni riservate agli autenticati e amministratori.



Esempio di profilo di autenticato, che presenta nel menù il collegamento alle pagine preferiti, liste e proposta attività

The screenshot shows the 'Profilo' (Profile) section of the Animiati application. At the top, there is a navigation bar with links: Animati, Catalogo, Strumenti, Preferiti, Liste, Crea attività, Segnalazioni, and Gestione utenti. On the far right of the bar are three icons: a gear, a document, and a user profile. Below the navigation bar, the word 'Profilo' is displayed in a large, bold, dark blue font. Underneath it, a smaller text says 'Qui puoi visualizzare il tuo profilo.' (Here you can view your profile). A central white box contains a circular profile picture of a man with a beard, followed by the email address 'kevin.delugan@studenti.unitn.it'. Below the email, the text 'amministratore' (Administrator) is shown, along with the note 'Promosso da #00000000000000000000000000000000'. At the bottom of the box are two buttons: 'Visualizza Attività' (View Activities) in orange and 'Logout' in red.

Esempio di profilo di amministratore, che presenta nel menù il collegamento alle pagine preferiti, liste, creazione attività, segnalazioni e gestione utenti

The screenshot shows the 'Lista' (List) section of the Animiati application. The top navigation bar includes 'Animati', 'Catalogo', 'Strumenti', 'Preferiti', 'Liste', and 'Proponi attività'. On the right side of the bar are three icons: a gear, a document, and a user profile. Below the navigation bar, the word 'Lista' is displayed in a large, bold, dark blue font. Underneath it, a smaller text says 'Qui puoi visualizzare le attività nella lista.' (Here you can view the activities in the list). A central white box titled 'Preferiti' (Favorites) lists four items: 'Braccialetti', 'Acchiappasogni', 'Mosaici', and 'Disegni nascosti'. Each item has a small trash can icon to its right. At the bottom of the box are two buttons: 'Esporta lista' (Export list) in orange and 'Elimina lista' (Delete list) in red.

Esempio di pagina preferiti di un autenticato

Animati Catalogo Strumenti ▾ Preferiti Liste Crea attività Segnalazioni Gestione utenti

Gestione utenti

Qui puoi trovare tutti gli utenti registrati all'app.

 assistenza@animati.app amministratore Visualizza Attività Cambia ruolo	 kevin.delugan@studenti.unitn.it amministratore Visualizza Attività Cambia ruolo	 carmen.casulli@studenti.unitn.it amministratore Visualizza Attività Cambia ruolo
 teodora.panjkovic@studenti.unitn.it amministratore Visualizza Attività Cambia ruolo	 teapanjkovic@gmail.com autenticato Visualizza Attività Cambia ruolo	 kevin.delugan@gmail.com autenticato Visualizza Attività Cambia ruolo

Esempio di pagina di gestione utenti

5. GitHub Repository and Deployment Info

Per permettere la condivisione dei codici sorgente durante lo sviluppo del progetto e sono state realizzate due repository GitHub:

- [IS-T51/Animati](#) - Contenente il codice sorgente del FrontEnd dell'applicativo
- [IS-T51/Animati-Server](#) - Contenente il codice sorgente del BackEnd dell'applicativo

Per effettuare il deployment del FrontEnd è stato usato il servizio gratuito [GitHub Pages](#), mentre per quello del BackEnd è stato usato il servizio gratuito [Vercel](#) (che purtroppo presenta alcuni rallentamenti nell'ultimo periodo).

Per rendere più facilmente accessibile il progetto, si è optato per l'acquisto del dominio `animati.app`, collegando il FrontEnd all'indirizzo web <https://animati.app> e il BackEnd all'indirizzo web <https://api.animati.app>.

6. Testing

Per effettuare il testing abbiamo utilizzato la libreria Jest e il modulo superjest per il testing e l'invocazione delle API.

Il testing è contenuto nella cartella test, all'interno di animati-server.

In questa cartella ci sono 5 file .test.js che costituiscono i file nei quali sono contenuti rispettivamente i test di ciascun modello per cui sono definite delle API.

Un file .test.js è strutturato nel seguente modo:

- all'inizio vi è l'importazione del modulo 'supertest' di jest per il testing delle API, del server di mongoose per effettuare una connessione al database per testare le nostre API e del modulo 'jsonwebtoken' per creare un token per le API che lo necessitano
- nel metodo beforeAll() abbiamo la definizione di tutto ciò che deve essere fatto all'inizio dell'esecuzione del codice del file, ovvero la connessione al database di test, creato apposta per il testing, e l'attivazione del server alla porta 8080
- nel metodo afterAll() abbiamo la definizione di tutto ciò che deve essere fatto al termine dell'esecuzione del codice del file, ovvero chiudere la connessione e disattivare il server, nonché eventualmente svuotare il database di test in alcune occasioni
- in alcuni file è presente pure il beforeEach() dove si svuota, per sicurezza, ancora una volta il database di test

Per ogni test cases viene controllato il corretto status code di ritorno e, se viene tornata una risorsa si verifica che i vari attributi siano quelli desiderati. Sono presenti API di tipo GET, POST, PUT, PATCH e DELETE.

Risultati del testing

Per effettuare il testing abbiamo innanzitutto aggiunto il seguente script al file package.json:

- "test": "jest --coverage --detectOpenHandles"

In questo modo facendo il comando npm test dalla root del progetto verranno eseguiti tutti i file .test.js da noi definiti. Il flag --coverage serve per creare i report finali che testimoniano i risultati del testing, mentre il flag --detectOpenHandles ha lo scopo di identificare se nel corso del testing viene chiamato qualcosa che impedisce al testing di terminare.

Questi sono i risultati dei nostri test:

```
Test Suites: 5 passed, 5 total
Tests:       73 passed, 73 total
Snapshots:   0 total
Time:        47.813 s
Ran all test suites.
```

Tutte e cinque le test suites sono state eseguite e tutti i 73 casi di test (definiti dai metodi test()) risultano passati.

Un report per visualizzare i risultati del test si trova nella cartella coverage/ (nella root del progetto) e successivamente nella cartella lcov-report/. Il file index.html apre una pagina di ipertesto in cui verificare ed analizzare i risultati ottenuti.

D4 - Animati - Sviluppo v0.1 [pre-release]

All files

71.49% Statements 474/663 57.79% Branches 128/218 77.77% Functions 133/171 72.32% Lines 473/654

Press n or j to go to the next uncovered block, b, p or k for the previous block.

Filter:

File	Statements	Branches	Functions	Lines
src	100%	7/7	0/0	100%
src/controllers	88.23%	90/102	0/0	82.6%
src/models	100%	24/24	0/0	100%
src/schema	100%	36/36	0/0	100%
src/service	62.47%	278/445	106/194	73.62%
src/utils	79.59%	39/49	20/24	81.81%

Nella prima schermata che appare abbiamo un riassunto degli statements, dei branches, delle funzioni e delle linee di codice coperte dal nostro testing. La cartella che più interessa è chiaramente la cartella src/controllers in cui sono definite tutte le API (ed infatti è la cartella più grande).

Il testing ci ha permesso di raggiungere il 62.47% di copertura di tutti gli statements e il 63.47% di tutte le linee di codice delle varie API. 57/69 funzioni sono state coperte. La copertura cala un po' quando andiamo a verificare i branches esplorati (ovvero i rami degli if-then-else) attestandosi quasi sul 54.63%, dopo vedremo perché.

Premendo sulla cartella src/controllers otteniamo tutti i dati del testing effettuato su ogni file .js che contiene il codice delle varie API.

All files src/controllers

88.23% Statements 98/102 100% Branches 9/9 82.6% Functions 57/69 88.23% Lines 98/102

Press n or j to go to the next uncovered block, b, p or k for the previous block.

Filter:

File	Statements	Branches	Functions	Lines
Attivita.js	89.47%	34/38	0/0	85.18%
Default.js	83.33%	5/6	0/0	66.66%
Etichette.js	90%	9/10	0/0	83.33%
Liste.js	96.15%	25/26	0/0	94.44%
Utente.js	77.27%	17/22	0/0	66.66%

I numeri sono simili per i vari gruppi di API: le funzioni testate sono intorno all'80%, le linee di codice testate sono intorno al 85% di media e gli statements verificati sull'85%.

La percentuale dei branch vista prima (54.63%) è tale perché in ogni API abbiamo un branch per gestire i vari errori indipendenti da noi che possono accadere (ad esempio errori del server, errori di connessione al database, ecc.).