UNIVERSITÀ DI TRENTO
Dipartimento di Ingegneria e Scienza dell'Informazione

Progetto:

Animati

Titolo del documento:

Sviluppo

Gruppo:

T51

Indice

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.

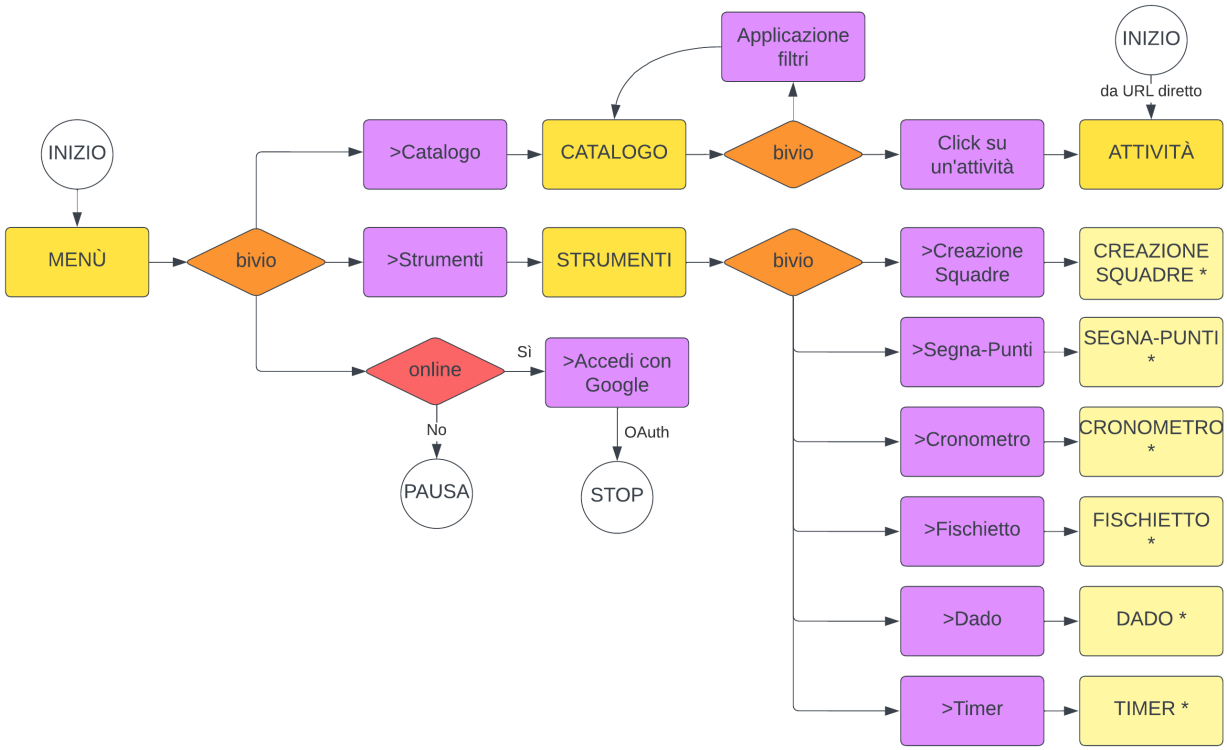
1. User Flows

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.

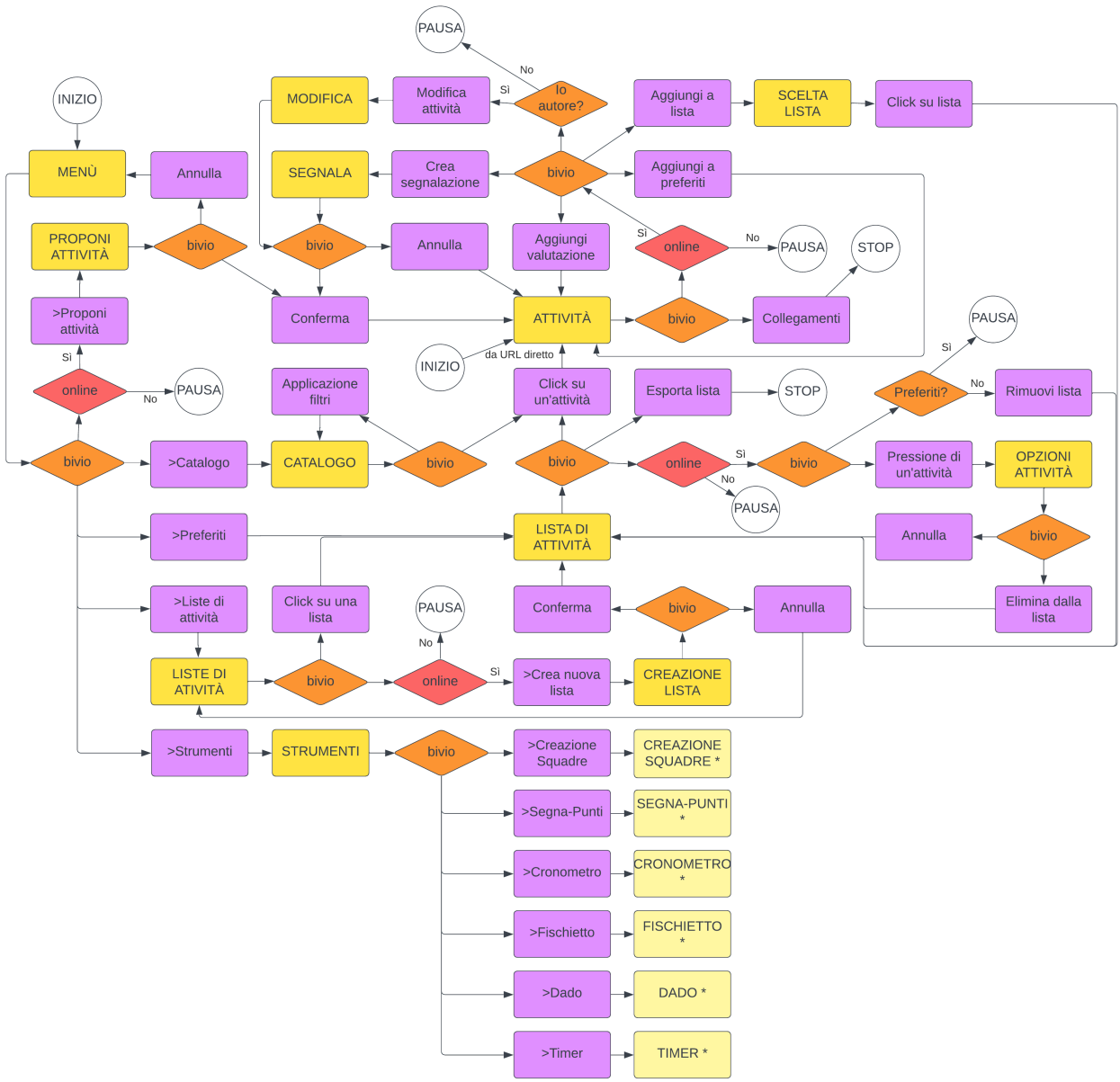


Utente Autenticato

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.

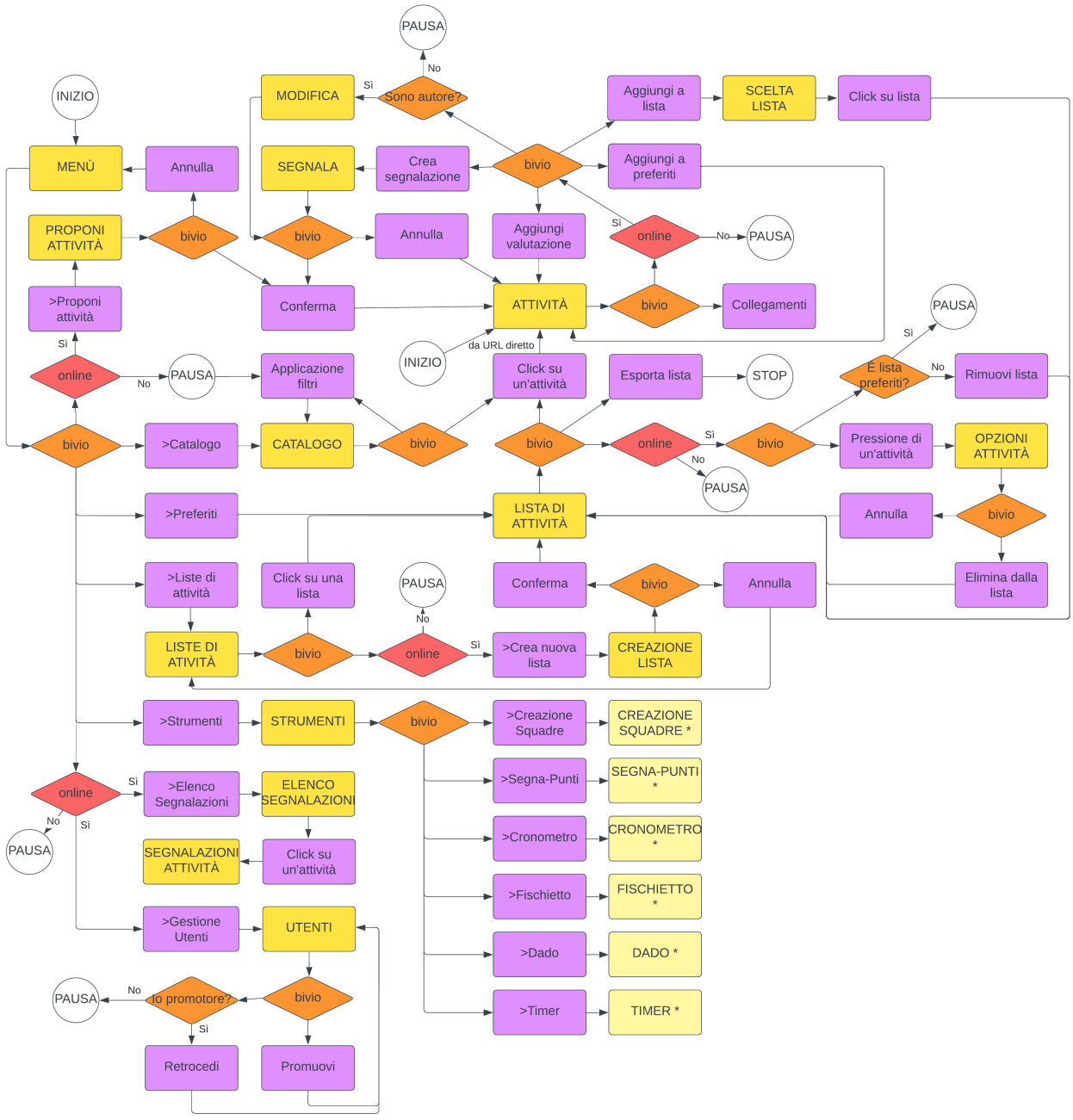
- A partire da un'attività, l'utente autenticato può cliccare sui collegamenti (se esistono), aggiungere l'attività ai preferiti o ad una lista, aggiungere una valutazione, creare una segnalazione e, se l'utente è autore dell'attività stessa, modificarla.
- A partire da Segnala, Modifica e Proponi attività, 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 li 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 li 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.



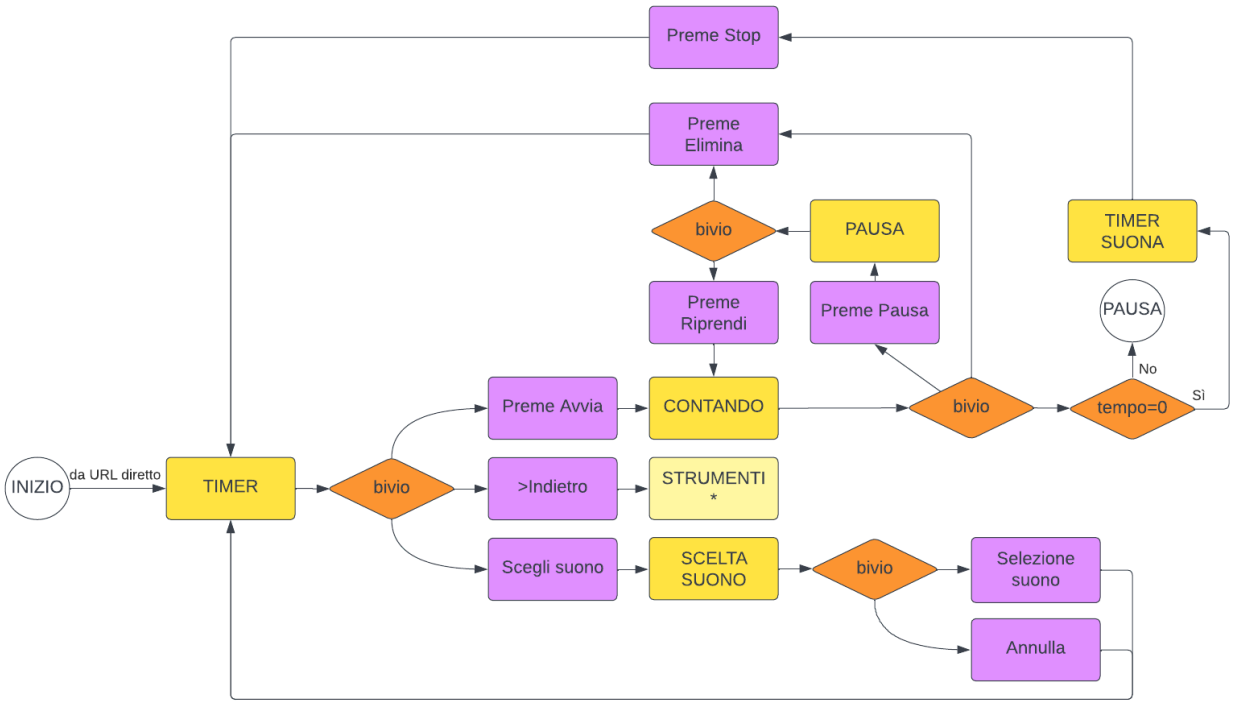
Strumenti

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 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.

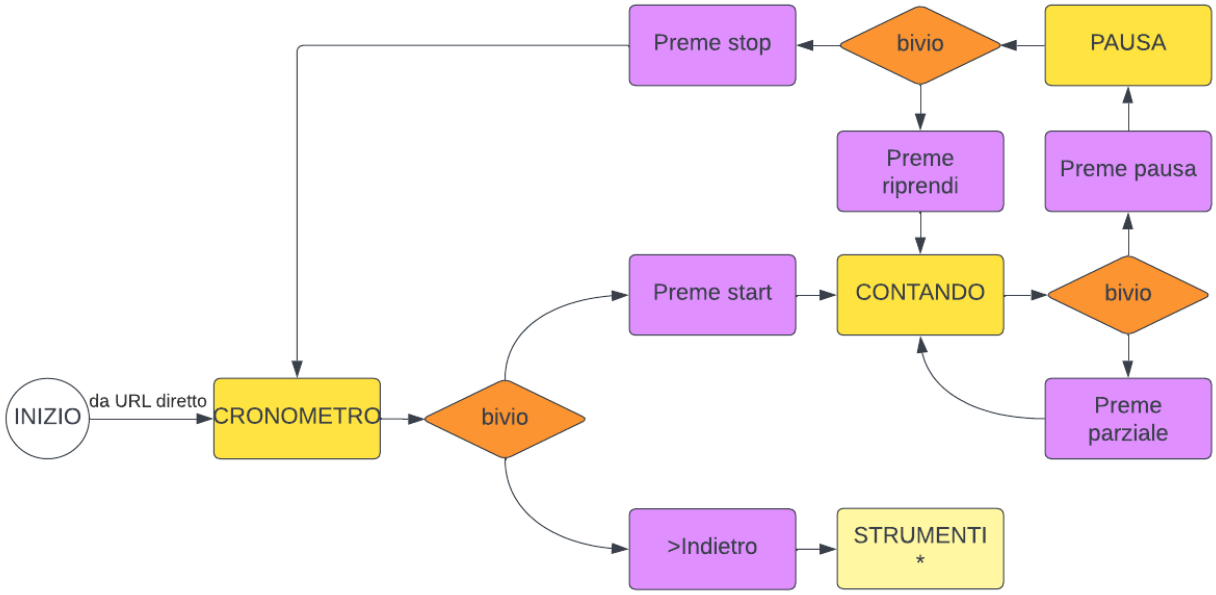
- A partire da Pausa, l'utente può premere Riprendi, continuando il conto alla rovescia precendentemente fermato, o Elimina, cancellando il timer e tornando all'inizio.
- A partire da Timer suona, l'utente deve premere Stop per poter fermare il suono in riproduzione al termine del timer. Una volta fatto ciò viene riportato all'inizio.



Cronometro

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

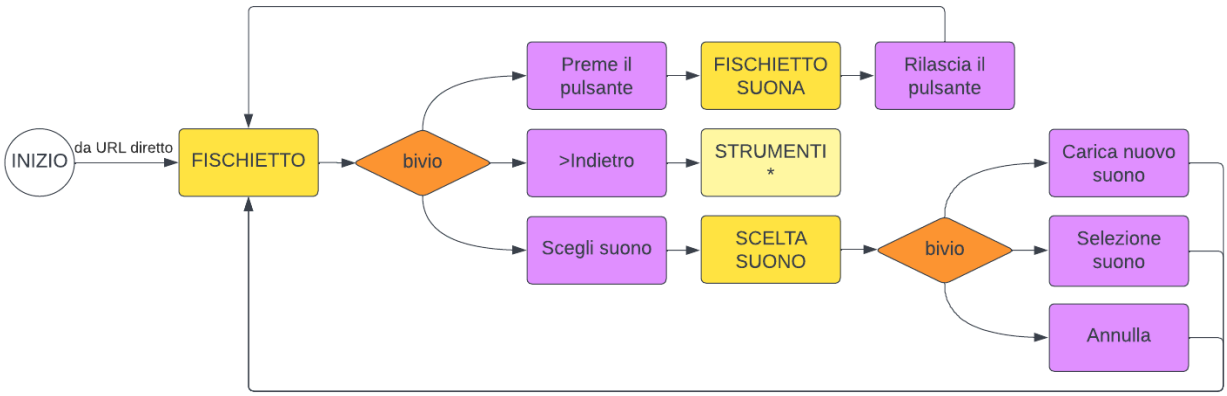
- A partire da Cronometro, l'utente può premere Start, facendo partire il cronometro e tornare indietro alla lista degli strumenti.
- A partire da Contando, l'utente può premere Pausa, fermando temporaneamente il cronometro, o parziale, salvando il tempo di quell'istante in una lista visualizzata sotto il cronometro senza fermare il cronometro.
- A partire da Pausa, l'utente può premere Riprendi per riavviare il cronometro dall'istante in cui era stato fermato, oppure premere Stop, per tornare all'inizio.



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 deve tenere premuto per l'intera durata del fischio desiderato. Nel momento in cui il pulsante viene rilasciato il suono smette di essere riprodotto.

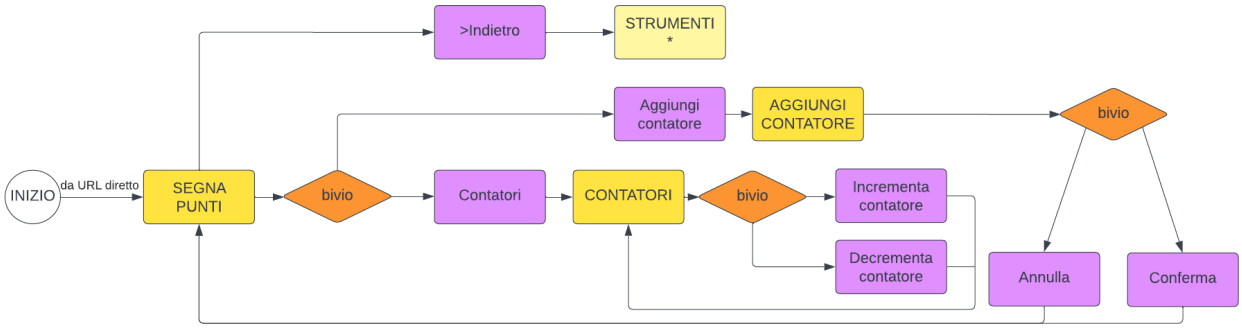


Segna-Punti

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.

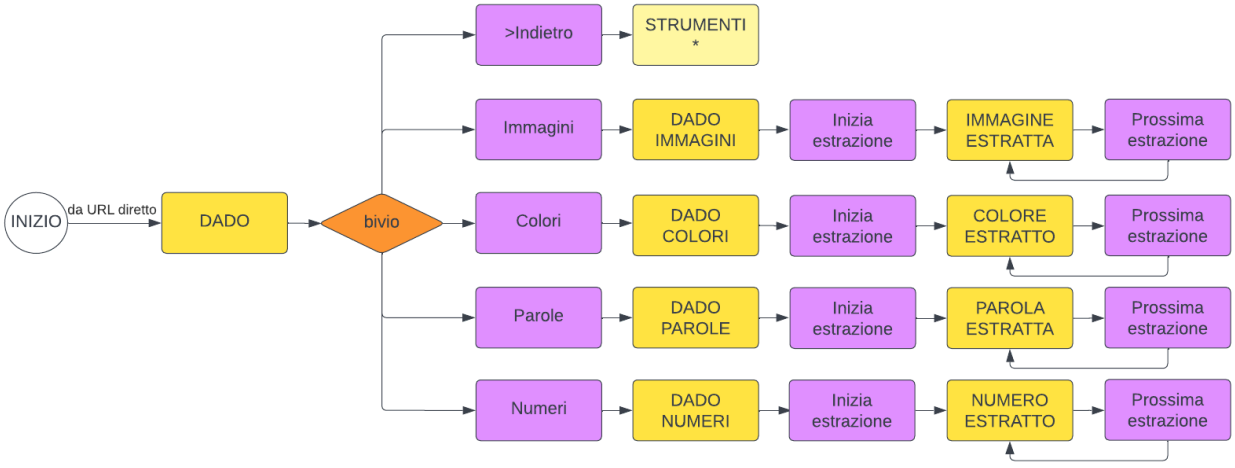
- A partire da Contatori, l'utente può decrementare o incrementare i contatori esistenti.



Dado

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

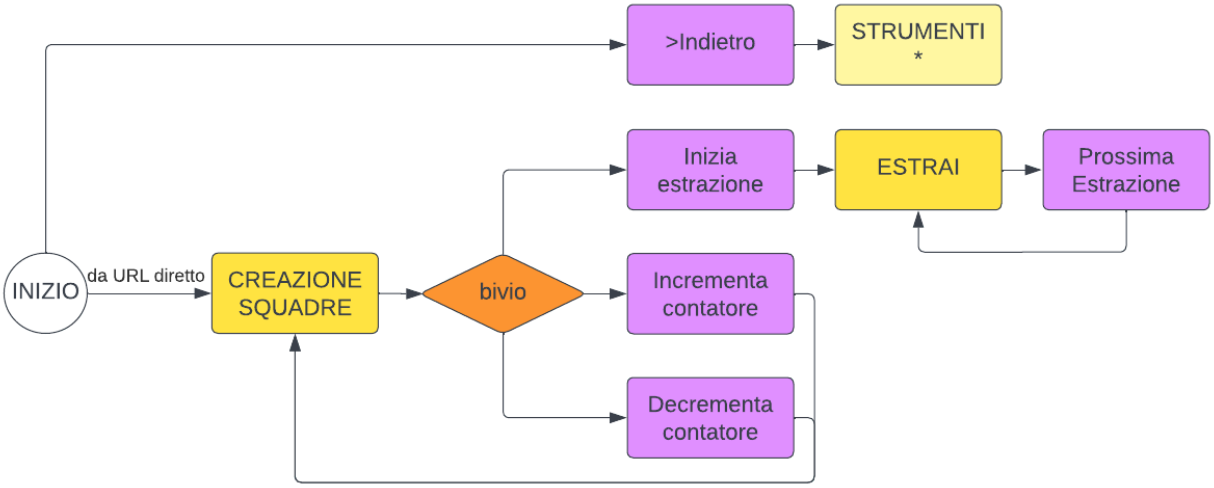
- A partire da Dado, l'utente può scegliere la tipologia di faccia usata e tornare indietro.
- A partire da Dado Immagini, Dado Colori, Dado Parole e Dado Numeri, l'utente può iniziare l'estrazione fino a che non finisce.



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.


>Indietro
STRUMENTI
*
Inizia
estrazione
ESTRAI
Prossima
Estrazione
INIZIO
da URL diretto
CREAZIONE
SQUADRE
bivio
Incrementa
contatore
Decrementa
contatore

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 sottoiensieme 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 "animati-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 conenente 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*
> >
> > ***schema/*** *- 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 Animati*
> >
> > ***test/*** *- cartella contenente i file di test*
> >
> > > ***.test.js*** *- file responsabile del testing dei servizi offerti dal server Animati*
> >
> > ***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 "animati". 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*
> >
> > > ***<font>*** *- file del font*
> >
> > ***html/*** *- cartella contenente alcuni componenti HTML comuni all'intero client*
> >
> > > ***header.html*** *- file conenente 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('000000000000000000000000')
    },
    immagine: {
        type: String,
        default: 'https://animati.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 del- le risorse a partire dal class diagram e uno per rappresentare le risorse sviluppate.

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*.

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 amministartore 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 *ottieniEtichette* 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

5. GitHub Repository and Deployment Info [...]

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.

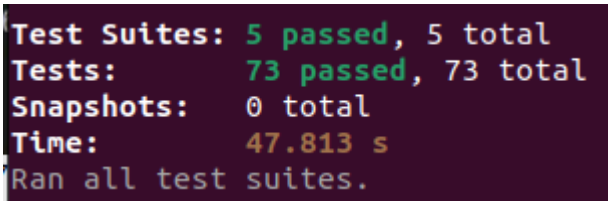
Risultati del testing

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

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

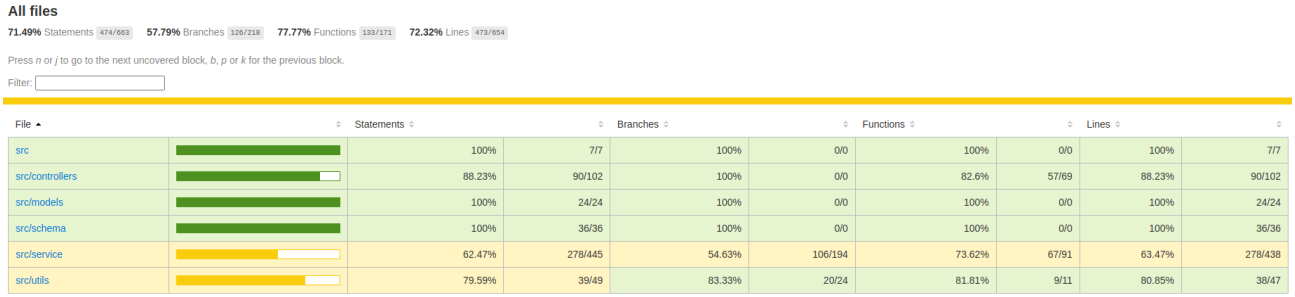
In questo modo facendo il comando npm test dalla root del progetto verranno eseguiti tutti i file .test.js da noi definiti. Il flag --coverage serve per creare i report finali che testimoniano i risultati del testing, mentre il flag --detectOpenHandles ha lo scopo di identificare se nel corso del testing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

Questi sono i risultati dei nostri test:



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.



All files

71.49% Statements 474/663 **57.79%** Branches 126/218 **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.

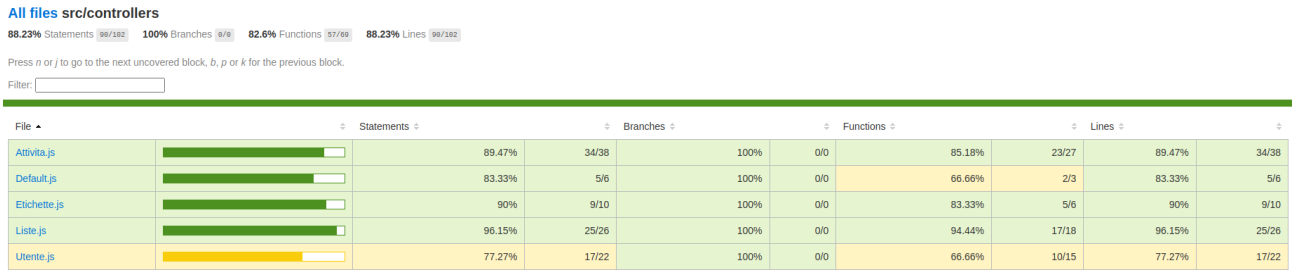
Filter:

File		Statements		Branches		Functions		Lines	
src		100%	7/7	100%	0/0	100%	0/0	100%	7/7
src/controllers		88.23%	90/102	100%	0/0	82.6%	57/69	88.23%	90/102
src/models		100%	24/24	100%	0/0	100%	0/0	100%	24/24
src/schema		100%	36/36	100%	0/0	100%	0/0	100%	36/36
src/service		62.47%	278/445	54.63%	106/194	73.62%	67/91	63.47%	278/438
src/utils		79.59%	39/49	83.33%	20/24	81.81%	9/11	80.85%	38/47

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 90/102 **100%** Branches 0/0 **82.6%** Functions 57/69 **88.23%** Lines 90/102

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	100%	0/0	85.18%	23/27	89.47%	34/38
Default.js		83.33%	5/6	100%	0/0	66.66%	2/3	83.33%	5/6
Etichette.js		90%	9/10	100%	0/0	83.33%	5/6	90%	9/10
Liste.js		96.15%	25/26	100%	0/0	94.44%	17/18	96.15%	25/26
Utente.js		77.27%	17/22	100%	0/0	66.66%	10/15	77.27%	17/22

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.).