API – Progetto Base

# STRUTTURE DATI

## Frequency: enumerable

* “monday”
* “tuesday”
* “wednesday”
* “thursday”
* “friday”
* “saturday”
* “Sunday”
* “every day”
* “every X month” (?)
* “every Y of the month” (?)

## Repetition: enumerable

* “indefinite” (???)
* “N times”
* “until”

## User:

* id: ObjectId
* username: String, Unique
* password: String

(per ora solo questi per l’utente)

Validazione:

* username: lettere e numeri, unico nel database

## Event:

* id: ObjectId (mongoDB)
* title: String,
* owner: reference ObjectId User
* participants: [User]
* frequency: Frequency
* repetitions: Repetition
* location: String
* startTime: Date
* endTime: Date

## UserEvent (linkTable)

* userId: reference User
* eventId: referenceEvent
* status: enum [“accepted”, “unconfirmed”, “declined”]

## Activity

* id: ObjectId
* title: String,
* dueTime: Date,
* status: ActivityStatus
* content: String
* owner: reference ObjectId User

## Pomodoro

* id: ObjectId
* owner: reference ObjectId User
* studyTime: Number,
* pauseTime: Number,
* cycles: Number

Note

* id: objectId
* owner: reference ObjectId User
* title: String
* text: String
* tags: [String]
* accessList: [reference User]
* toDoList: [ToDO]

(per ora no)

# EVENTI (/events)

Tutti i path devono essere autenticati. Tutti gli input devono essere validati (query, params, body).

Res.status = 402 se non autorizzato

Res.message = “Non autorizzato”

## GET /events/?from=:from&to=:to

Ritorna gli eventi dove l’utente autenticato è l’owner o uno dei partecipanti, filtrati per il range di data (nessun range di date -> ritorna tutti).

### Query

query.from = data iniziale dove fare la ricerca degli eventi

query.to = data finale di filtro degli eventi

### Req

{}

### Res 200

{status: “ok”, value: [Eventi]}

## GET /events/:id

Ritorna l’evento con id = params.id, se l’evento è dell’utente

### Params

id = id dell’evento da ritrovare

### Req

req.body = {}

### Res

res.body = [Eventi]

## POST /events/

Inserisce uno o più nuovi eventi per l’utente loggato.

Req

body: Event

Res

{status: ok, value: 1}

## PUT /events/:id

Modifica l’evento con id = params.id, se è dell’utente autenticato.

Req

body: Event modificato

Res

{status: ok, value: 1}

## DELETE /events/:id

Elimina l’evento con id = params.id, se è l’utente autenticato è l’owner

Req

{}

Res

{status: ok, value: 1}

# UTENTI (/users)

Tutti i path devono essere autenticati. Tutti gli input devono essere validati (query, params, body).

Res.status = 402 se non autorizzato

Res.message = “Non autorizzato”

## GET /users/

Ritorna i dati dell’utente autenticato.

### Req

{}

### Res 200

{status: “ok”, value: User}

## POST /users/

Inserisce un nuovo utente.

### Req

body: Event

### Res

{status: ok, value: 1}

## POST /users/login

Esegue il login dell’utente.

### Req

body: {username, password}

### Res (autenticazione ok)

{status: ok, value: true}

### Res (autenticazione fallita)

{status: ko, value: false}

## POST /users/signup

Esegue la registrazione di un nuovo utente.

### Req

body: {username, password}

### Res

{status: ok, value: true}

### Vincolo:

* username non già presente nel database: ritorna status: 400 + body: {status: “ko”, messaggio: “Username non disponibile”}

## PUT /users/

Modifica i dati dell’utente attualmente autenticato.

### Req

body: User modificato

### Res

{status: ok, value: 1}

## DELETE /users/

Elimina l’utente autenticato.

### Req

{}

### Res

{status: ok, value: 1}

### Vincoli

Se l’utente è eliminato, si devono eliminare tutti gli eventi associati all’utente, tutte le attività associate all’utente, tutte le sessioni pomodoro e tutte le note dell’utente dal database. (Possiamo scegliere di non farlo).

# ATTIVITÀ (/activities)

Tutti i path devono essere autenticati. Tutti gli input devono essere validati (query, params, body).

Res.status = 402 se non autorizzato

Res.message = “Non autorizzato”

## GET /activities/?from=:from&to=:to

Ritorna le attività dove l’utente autenticato è l’owner o uno dei partecipanti, filtrati per il range di data (nessun range di date -> ritorna tutti).

### Query

query.from = data iniziale dove fare la ricerca degli eventi

query.to = data finale di filtro degli eventi

### Req

{}

### Res 200

{status: “ok”, value: [Activity]}

## GET /activities/:id

Ritorna l’attività con id = params.id, se l’evento è dell’utente

### Params

id = id attività da ritrovare

### Req

{}

### Res

{status: “ok”, value: [Activity]}

## POST /activities/

Inserisce una o più nuove attività per l’utente loggato.

Req

body: Activity

Res

{status: ok, value: 1}

## PUT /activities/:id

Modifica l’attività con id = params.id, se è dell’utente autenticato.

Req

body: Activity modificata

Res

{status: ok, value: 1}

## DELETE /activities/:id

Elimina l’attività con id = params.id, se è l’utente autenticato è l’owner

Req

{}

Res

{status: ok, value: 1}

# NOTE (/notes)

Tutti i path devono essere autenticati. Tutti gli input devono essere validati (query, params, body).

Res.status = 402 se non autorizzato

Res.message = “Non autorizzato”

## GET /notes/?from=:from&to=:to&order=:order

Ritorna le note dove l’utente autenticato è l’owner o uno dei partecipanti, filtrati per il range definito da [from, to] e tipo di ordine (order)(nessun range di date -> ritorna tutti).

### Query

query.from = indice elemento iniziale lista ritornata

query.to = indice elemento iniziale lista ritornata

query.order = tipo di ordinamento delle note: [“name”, “date”, “length”]

### Req

{}

### Res 200

{status: “ok”, value: [Note]}

## GET /notes/:id

Ritorna la nota con id = params.id, se l’evento è dell’utente

### Params

id = id della nota da ritrovare

### Req

req.body = {}

### Res

res.body = [Note]

## POST /notes/

Inserisce una o più nuove note per l’utente loggato.

Req

body: Note

Res

{status: ok, value: 1}

## PUT /notes/:id

Modifica la nota con id = params.id, se è dell’utente autenticato.

Req

body: Note aggiornata

Res

{status: ok, value: 1}

## DELETE /notes/:id

Elimina la nota con id = params.id, se è l’utente autenticato è l’owner

Req

{}

Res

{status: ok, value: 1}

# POMODORO (/pomodoro)

Tutti i path devono essere autenticati. Tutti gli input devono essere validati (query, params, body).

Res.status = 402 se non autorizzato

Res.message = “Non autorizzato”

## GET /pomodoro/?from=:from&to=:to&order=:order

Ritorna i pomodoro dove l’utente autenticato è l’owner o uno dei partecipanti, filtrati per il range definito da [from, to] e tipo di ordine (order)(nessun range di date -> ritorna tutti).

### Query

query.from = indice elemento iniziale lista ritornata

query.to = indice elemento iniziale lista ritornata

query.order = tipo di ordinamento: [“name”, “date”, “length”]

### Req

{}

### Res 200

{status: “ok”, value: [Pomodoro]}

## GET /pomodoro/:id

Ritorna il pomodoro con id = params.id, se l’evento è dell’utente

### Params

id = id del pomodoro da ritrovare

### Req

{}

### Res

{status: ok, value: [Pomodoro]}

## POST /events/

Inserisce una o più nuovi pomodoro per l’utente loggato.

Req

body: Note

Res

{status: ok, value: 1}

## PUT /pomodoro/:id

Modifica il pomodoro con id = params.id, se è dell’utente autenticato.

Req

body: Pomodoro aggiornato

Res

{status: ok, value: 1}

## DELETE /pomodoro/:id

Elimina i pomodoro con id = params.id, se è l’utente autenticato è l’owner

Req

{}

Res

{status: ok, value: 1}