Tasky API

This will serve as a documentation for all the API endpoints needed to build Tasky. Please note that this API has been written with care, however if you still find some kind of bug, it would be appreciated if you could report it in WhatsApp or via email to mail@pl-coding.com.

The base url for the Tasky API is https://tasky.pl-coding.com/

You can find the mockups for Tasky here: https://www.figma.com/file/ATBztE2a0RObai1XZVeiVC/Tasky-app?node-id=0%3A1

Authentication

API Key

All endpoints require an API key to be attached. It needs to be attached as `x-api-key` header, like this:

Header name	Header value
x-api-key	<your API key>

JWT

The auth system in the app is implemented using JWT (Json Web Tokens). The API will respond with that token when a user logs in and needs to be attached to all requests, except for registering and logging in. Attach it like the following as Authorization header: `Authorization: Bearer <token>`.

Rate Limiting

To prevent accidental request bursts, the API has a rate limit of **50 requests/min** and **1000 requests/hour.**

Routes

Errors

In case of an error (non 2XX response code), the API will respond with a message for most types of errors like this:

```
{
  "message": "<error message>" (String)
}
```

Registration

Endpoint: `/register`

Method: `POST`

Description: `Registers a new user.`

Body:

```
{
  "fullName": "<full_name>", (String)
  "email": "<email>", (String)
  "password": "<password>" (String)
}
```

No response body.

Login

Endpoint: `/login`

Method: `POST`

Description: `Logs in an existing user.`

Body:

```
{
  "email": "<email>", (String)
  "password": "<password>" (String)
}
```

Response body:

```
{
  "token": "<JWT token>", (String)
  "userId": "<ID of the logged in user>", (String)
  "fullName": "<Full name of the logged in user>" (String)
}
```

Check authentication

Endpoint: `/authenticate`

Method: `GET`

Description: `Used to check if a token is (still) valid`

Response: 200 if valid, 401 if invalid

Logout

Endpoint: `/logout`

Method: `GET`

Description: `Logs out a user and makes the attached token invalid`

Agenda

Endpoint: `/agenda`

Method: `GET`

Description: `Returns the agenda for a given day`

Query parameters:

`timezone`: The timezone of the user, e.g. `Europe/Berlin`

`time`: The time of the agenda as a UTC timestamp (so if you're fetching the agenda for today, this would be `System.currentTimeMillis()`).

Response body:

```
{
  "events": [...events], (List<Event>)
  "tasks": [...tasks], (List<Task>)
  "reminders": [...reminders] (List<Reminder>)
}
```

Sync Agenda

Endpoint: `/syncAgenda`

Method: `POST`

Description: `Synchronizes locally deleted agenda items with the API.`

Body:

```
{
  "deletedEventIds": [...event_IDs],
  "deletedTaskIds": [...task_IDs],
  "deletedReminderIds": [...reminder_IDs],
}
```

```
```

Create Event

Endpoint: `/event`

Method: `POST/Multipart`

Description: `Creates a new event. Since Tasky supports offline mode, the IDs for agenda items need to be generated client-side. Please use UUID.randomUUID().toString() for that.`

Multipart

1. Form item `create_event_request` as serialized JSON:

```
{
  "id": "<UUID of the event>", (String)
  "title": "<Event title>", (String)
  "description": "<Event description>", (String)
  "from": 123456789, (Long)
  "to": 123456789, (Long)
  "remindAt": 123456789, (Long)
  "attendeeIds": [...attendee_IDs] (List<String>)
}
```

2. File items `photo[0-9]` containing bytes for each attached photo.

Response body:

```
{
  "id": "<Event ID>", (String)
  "title": "<Event title>", (String)
  "description": "<Event description>", (String)
  "from": 123456789, (Long)
  "to": 123456789, (Long)
  "remindAt": 123456789, (Long)
  "host": "<Event creator ID>", (String)
  "isUserEventCreator": true/false, (Boolean)
  "attendees": [...attendees], (List<Attendee>)
  "photos": [...photos], (List<Photo>)
}
```

Attendee:

```
{
  "email": "<Attendee email>", (String)
  "fullName": "<Attendee full name>", (String)
  "userId": "<Attendee user ID>", (String),
  "eventId": "<Event the attendee belongs to>", (String)
  "isGoing": true/false, (Boolean)
  "remindAt": 123456789 (Long)
}
```

Photo:

```
{
  "key": "<Unique key of the photo>", (String)
  "url": "https://test.com/photo.png" (String)
}
```

Get Event

Endpoint: `/event`

Method: `GET`

Description: `Returns an existing event`

Query parameters:

`eventId`: The ID of the event

Response body:

```
{
  "id": "<Event ID>", (String)
  "title": "<Event title>", (String)
  "description": "<Event description>", (String)
  "from": 123456789, (Long)
  "to": 123456789, (Long)
  "remindAt": 123456789, (Long)
  "host": "<Event creator ID>", (String)
  "isUserEventCreator": true/false, (Boolean)
  "attendees": [...attendees], (List<Attendee>)
  "photos": [...photos], (List<Photo>)
}
```

Attendee:

```
{
  "email": "<Attendee email>", (String)
  "fullName": "<Attendee full name>", (String)
  "userId": "<Attendee user ID>", (String),
  "eventId": "<Event the attendee belongs to>", (String)
  "isGoing": true/false, (Boolean)
  "remindAt": 123456789 (Long)
}
```

Photo:

```
{
  "key": "<Unique key of the photo>", (String)
  "url": "https://test.com/photo.png" (String)
}
```

Delete Event

Endpoint: `/event`

Method: `DELETE`

Description: `Deletes an existing event`

Query parameters:

`eventId`: The ID of the event

Update Event

Endpoint: `/event`

Method: `PUT/Multipart`

Description: `Updates an existing event. In addition to creating an event, this also takes potentially deleted photos into consideration. Also, the isGoing boolean refers to the isGoing status of the local user.`

Multipart

1. Form item `update_event_request` as serialized JSON:

```
{
  "id": "<UUID of the event>", (String)
  "title": "<Event title>", (String)
  "description": "<Event description>", (String)
  "from": 123456789, (Long)
  "to": 123456789, (Long)
  "remindAt": 123456789, (Long)
  "attendeeIds": [...attendee_IDs], (List<String>)
  "deletedPhotoKeys": [...keys of deleted photos], (List<String>)
  "isGoing": true/false (Boolean)
}
```

2. File items `photo[0-9]` containing bytes for each attached photo.

Response body:

```
{
  "id": "<Event ID>", (String)
  "title": "<Event title>", (String)
  "description": "<Event description>", (String)
  "from": 123456789, (Long)
  "to": 123456789, (Long)
  "remindAt": 123456789, (Long)
  "host": "<Event creator ID>", (String)
  "isUserEventCreator": true/false, (Boolean)
  "attendees": [...attendees], (List<Attendee>)
  "photos": [...photos], (List<Photo>)
}
```

Attendee:

```
{
  "email": "<Attendee email>", (String)
  "fullName": "<Attendee full name>", (String)
  "userId": "<Attendee user ID>", (String),
  "eventId": "<Event the attendee belongs to>", (String)
  "isGoing": true/false, (Boolean)
  "remindAt": 123456789 (Long)
}
```

Photo:

```
{
  "key": "<Unique key of the photo>", (String)
  "url": "https://test.com/photo.png" (String)
}
```

Get Attendee

Endpoint: `/attendee`

Method: `GET`

Description: `Checks if a user with a given email exists and can be added as attendee to an event.`

Query parameters:

`email`: The email of the attendee

Response body:

```
{
  "doesUserExist": true/false, (Boolean)
  "attendee": {
    "email": "<Attendee email>", (String)
    "fullName": "<Attendee full name>", (String)
    "userId": "<Attendee user ID>" (String)
  }
}
```

Delete Attendee

Endpoint: `/attendee`

Method: `DELETE`

Description: `Removes an existing attendee from an event. This is intended to be used when an attendee deletes an event locally.`

Query parameters:

`eventId`: The email of the attendee

Create Task

Endpoint: `/task`

Method: `POST`

Description: `Creates a new task.`

Body:

```
{
  "id": "<Task ID>", (String)
  "title": "<Task title>", (String)
  "description": "<Task description>", (String?)
  "time": 123456789, (Long)
  "remindAt": 123456789, (Long)
  "isDone": true/false (Boolean)
}
```

Update Task

Endpoint: `/task`

Method: `PUT`

Description: `Updates an existing task.`

Body:

```
{
  "id": "<Task ID>", (String)
  "title": "<Task title>", (String)
  "description": "<Task description>", (String?)
  "time": 123456789, (Long)
  "remindAt": 123456789, (Long)
  "isDone": true/false (Boolean)
}
```

Get Task

Endpoint: `/task`

Method: `GET`

Description: `Gets an existing task by ID.`

Query parameters:

`taskId`: The task ID

Response:

```
{
  "id": "<Task ID>", (String)
  "title": "<Task title>", (String)
  "description": "<Task description>", (String?)
  "time": 123456789, (Long)
  "remindAt": 123456789, (Long)
  "isDone": true/false (Boolean)
}
```

Delete Task

Endpoint: `/task`

Method: `DELETE`

Description: `Deletes an existing task.`

Query parameters:

`taskId`: The ID of the task to be deleted.

Create Reminder

Endpoint: `/reminder`

Method: `POST`

Description: `Creates a new reminder.`

Body:

```
{
  "id": "<Reminder ID>", (String)
  "title": "<Reminder title>", (String)
  "description": "<Reminder description>", (String?)
  "time": 123456789, (Long)
  "remindAt": 123456789, (Long)
}
```

Update Reminder

Endpoint: `/reminider`

Method: `PUT`

Description: `Updates an existing reminder.`

Body:

```
{
  "id": "<Reminder ID>", (String)
  "title": "<Reminder title>", (String)
  "description": "<Reminder description>", (String?)
  "time": 123456789, (Long)
  "remindAt": 123456789, (Long)
}
```

Get Reminder

Endpoint: `/reminder`

Method: `GET`

Description: `Gets an existing reminder by ID.`

Query parameters:

`reminderId`: The reminder ID

Response:

```
{
  "id": "<Reminder ID>", (String)
  "title": "<Reminder title>", (String)
  "description": "<Reminder description>", (String?)
  "time": 123456789, (Long)
  "remindAt": 123456789, (Long)
}
```

Delete Reminder

Endpoint: `/reminder`

Method: `DELETE`

Description: `Deletes an existing reminder.`

Query parameters:

`reminderId`: The ID of the reminder to be deleted.