Backend

Updated `get_queryset` in views.py:-

```
def get_queryset(self):
    """
    Override to return only events belonging to the current user.
    """
    queryset = Event.objects.filter(user=self.request.user)
    start_time = self.request.query_params.get('start_time')
    end_time = self.request.query_params.get('end_time')

    if start_time and end_time:
        # Include only events that overlap the given range
        queryset = queryset.filter(
            start_time__lt=end_time,  # Event starts before the end_time
            end_time__gt=start_time   # Event ends after the start_time
        )
    elif start_time:
        # Include events that end after the start_time
        queryset = queryset.filter(end_time__gt=start_time)
    elif end_time:
        # Include events that start before the end_time
        queryset = queryset.filter(start_time__lt=end_time)

    return queryset
```

Key changes:-

1. **Time Range Filters**:
    - **Added Parameters**:
        - `start_time` and `end_time` are extracted from the request's query parameters.
        - These parameters allow filtering the events that occur within a specific time range.
    - **Conditional Logic**:
        - If both `start_time` and `end_time` are provided, the queryset filters events that overlap this period.
        - If only one of the parameters is provided, the function adjusts the queryset accordingly to include:

- Events that end after `start_time`.
- Events that start before `end_time`.

2. **Filtering Events by Time**:
   - The new logic ensures that only events occurring within the provided time window are included in the response.
   - This change enables users to request events within a specific time range, improving usability, especially for large datasets.

Frontend

Updated `getEvents()` in Api.js:-

```
export const getEvents = async (startTime, endTime) => {
  console.log('Sending request to fetch events...');
  try {
      const params = {};
      if (startTime) params.start_time = startTime; // Add start_time to
params
      if (endTime) params.end_time = endTime; // Add end_time to params
      const response = await api.get('/api/events/', { params });
      console.log('Events response received:', response.data);
      return response.data; // Return the event data
  } catch (error) {
      console.error('Error fetching events:', error);
      throw error; // Rethrow the error for further handling
  }
};
```

**Key changes:-

1. Passing parameters of `startTime` and `endTime` for proper filtering of events

Updated in `fetchEvents()` inApp.js:-

```
const fetchEvents = async () => {
    try {
        const response = await getEvents(); // Ensure this function is called
with necessary parameters if needed
        console.log('API Response:', response); // Log the entire response for
debugging
        // Check if response is an array
```

```
        if (Array.isArray(response)) {
            // Format the events, converting date strings to Date objects
            const formattedEvents = response.map(event => ({
                ...event,
                start: new Date(event.start_time),
                end: new Date(event.end_time),
            }));
            setEvents(formattedEvents);
            scheduleReminders(formattedEvents);
        } else {
            console.error('Unexpected response structure:', response);
        }
    } catch (error) {
        console.error('Failed to fetch events:', error);
    }
};
```

**key changes :-

1. **Response Structure Validation**:
    - **Conditional Check for Array**:
        - The original code directly accessed `response.data`, assuming the response structure is consistent.
        - The updated code checks if the `response` is an array using `Array.isArray(response)`. This is crucial because the response format might change, and this validation prevents potential errors when trying to map over a non-array structure.

Updated `fetchEvents` in EventList component:-

```
const fetchEvents = async () => {
  try {
    const response = await getEvents(startTime, endTime); // Fetch with date
filters
    setEvents(response || []); // Ensure empty array if response is undefined
  } catch (error) {
    console.error('Failed to fetch events:', error);
    setEvents([]); // Show no events on error
  }
};
```

Key Changes:

- **Fetching and Filtering**: Added ability to filter events based on a date range. The `startTime` and `endTime` states capture user input and pass it to the `fetchEvents` function.
- **Handling Empty State**: Displays "No events found" if no events match the filters.

Updated the jsx in EventList.js and added textfields to pass startime and endtime as query:-

```
{/* Date filters */}
      <div>
        <TextField
          label="Start Time"
          type="datetime-local"
          onChange={(e) => setStartTime(e.target.value)}
        />
        <TextField
          label="End Time"
          type="datetime-local"
          onChange={(e) => setEndTime(e.target.value)}
        />
        <Button variant="contained" onClick={fetchEvents}>Filter</Button>
      </div>
```