# Backend Programming Challenge

Imagine a backend server for a chat room. This server knows about two things: user messages and events. For this exercise, we only care about events; we are not implementing anything regarding user messages. The server will store a record of events and will aggregate the event history at various levels of time-based aggregation for reporting purposes. We don't have (or want) a UI for this server, so our implementation will be an API only, responding to HTTP requests. Please make sure that your application persists data across restarts.

The purpose of this challenge is to get an idea of how you think about, and execute on, solving a problem. We are interested in seeing well-crafted, well-factored code. This project should be completed using Node and Express. You should include relevant tests.

- When you start the challenge, please commit a file to the git repo named "started" with the content being the current date/time.
- When you finish, create a git bundle( `git bundle create code_challenge.bdl --all`).
- Email that bundle to the person who sent you this coding challenge or, if the resulting bundle is too big to email, host it on dropbox and email the link to download it.

Please include a README.md (markdown) in the top level folder that includes the following:

- How to install the application, including dependencies, etc.
- How to initialize the data store
- How to start the application
- How to run the test suite
- Any notes that you may want us to know about the application (optional)

To verify correctness (according to the specification), a script will be run against your server to analyze input and output. The script inputs are *very* similar to the examples given in the specification. .

In this challenge we aren't worried about edge cases that would cause errors. For instance,

you can assume that all data posted to `/events` is in the correct format. You can assume that any event posted is a valid event for that time. In a real server, having an exit event before an enter event might be a problem, but we aren't worried about things like that for this challenge. However, if the script encounters an error, or we forget an input field, or something generally unexpected happens, we should fail gracefully. Any unsuccessful request should return a `non-200` HTTP response code with `Content-Type: application/json` and the following data: `{"status": "error"}`

The web service should implement the following endpoints:

## Submit Event

```
POST /events
Content-Type: application/json
```

This allows a chat application to receive an event performed by a user. The following POST bodies are supported:

User enters the room:

```
{"date": "1985-10-26T09:00:00Z", "user": "Doc", "type": "enter"}
```

User comments in a room:

```
{"date": "1985-10-26T09:01:00Z", "user": "Doc", "type": "comment", "message": "I love plutonium"}
```

User highfives another user:

```
{"date": "1985-10-26T09:02:00Z", "user": "Marty", "type": "highfive", "otheruser": "Doc"}
```

User leaves the room:

```
{"date": "1985-10-26T09:03:00Z", "user": "Doc", "type": "leave"}
```

A successful POST should return HTTP response code `200` with `Content-Type: application/json` and the following data: `{"status": "ok"}`.

## Clear Data

```
POST /events/clear
```

This should clear out all data in the data store and will be primarily used in the verification script we run when reviewing your code. A successful POST should return HTTP response code `200` with `Content-Type: application/json` and the following data: `{"status": "ok"}`.

## List Events

**`GET /events?from=DATE&to=DATE`**

DATE is in the ISO 8601 YYYY-MM-DDThh:mm:ssZ format.

This call should return HTTP response code **`200`** with **`Content-Type: application/json`** and an array of events in the same format as is posted to the **`/events`** endpoint above. Only events within the time range should be returned. The output should be sorted by date ascending.

Sample return:

```
{"events": [{"date": "1985-10-26T09:00:00Z", "user": "Doc", "type":
"enter"},{"date": "1985-10-26T09:01:00Z", "user": "Doc", "type": "comment",
"message": "loves plutonium"},{"date": "1985-10-26T09:02:00Z", "user":
"Marty", "type": "highfive", "otheruser": "Doc"},{"date":
"1985-10-26T09:03:00Z", "user": "Doc", "type": "leave"}]}
```

## Event Summary

**`GET /events/summary?from=DATE&to=DATE&by=TIMEFRAME`**

DATE is in the ISO 8601 YYYY-MM-DDThh:mm:ssZ format.
TIMEFRAME is the rollup aggregation: minute, hour or day

Sample return format:

```
{"events": [{"date": "ROLLED_UP_DATE", "enters": NUM_ENTERS, "leaves":
NUM_LEAVES, "comments": NUM_COMMENTS, "highfives": NUM_HIGHFIVES},...]}
```

ROLLED_UP_DATE is a datetime rounded to the preceding time in ISO 8601 format. ***1985-10-26T09:01:55Z*** would roll up to ***1985-10-26T09:01:00Z*** for the minute, ***1985-10-26T09:00:00Z*** for the hour, and ***1985-10-26T00:00:00Z*** for the day. The other **`NUM_*`** entries are integer counts of the number of events in that category within the given timeframe.

This call should return HTTP response code **`200`** with **`Content-Type: application/json.`** We only want to include timeframes in which there is an event.

Sample return:

```
{"events": [{"date": "1985-10-26T09:00:00Z", "enters": 1, "leaves": 1,
"comments": 5, "highfives": 3},{"date": "1985-10-26T11:00:00Z", "enters": 1,
"leaves": 0, "comments": 4, "highfives": 2},{"date": "1985-10-26T15:00:00Z",
"enters": 0, "leaves": 1, "comments": 6, "highfives": 0}]}
```

# Example cURL calls

### Request

```
curl -H "Content-Type: application/json" -X POST -d '{"date": "1985-10-26T09:00:00Z", "user": "Doc", "type": "enter"}' http://localhost:3000/events
```

### Response

```
{"status":"ok"}
```

---

### Request

```
curl -H "Content-Type: application/json" -X POST -d '{"date": "1985-10-26T09:02:00Z", "user": "Marty", "type": "highfive", "otheruser": "Doc"}' http://localhost:3000/events
```

### Response

```
{"status":"ok"}
```

---

### Request

```
curl -H "Content-Type: application/json" -X POST -d '{"date": "1985-10-26T09:03:00Z", "user": "Doc", "type": "leave"}' http://localhost:3000/events
```

### Response

```
{"status":"ok"}
```

---

### Request

```
curl "http://localhost:3000/events?from=1985-10-26T09:00:00Z&to=1985-10-27T09:00:00Z"
```

### Response


```
{"events":[{"date":"1985-10-26T09:00:00Z","user":"Doc","type":"enter"},{"date":"1985-10-26T09:02:00Z","user":"Marty","type":"highfive","otheruser":"Doc"},{"date":"1985-10-26T09:03:00Z","user":"Doc","type":"leave"}]}
```


---

### Request

```
curl "http://localhost:3000/events/summary?from=1985-10-26T09:00:00Z&to=1985-10-27T09:00:00Z&by=hour"
```

### Response


```
{"events":[{"date":"1985-10-26T09:00:00Z","enters":1,"leaves":1,"comments":0,"highfives":1}]}
```