The Nudge API is organized around REST.

1. Data Model (Object Structure)

Here's what the Nudge object looks like in our database, based on the UI fields. I'm assuming it's a standard SQL, but honestly, this setup works fine for NoSQL too.

Nudge Object:


```
{
  "_id": ObjectId,
  "type": "nudge",
  "uid": 18,
  "tag_type": "event",   "tag_id": "65fa9e...",
  "title": "Join the Backend Development",
  "description": "A deep dive into Backend Development",
  "schedule": {
    "date": "2026-02-10",
    "from": "10:00",
    "to": "12:00",
    "timestamp": "2026-02-10T10:00:00Z"
  },
  "files": {
    "cover_image": "/uploads/nudges/cover.png",
    "icon": "/uploads/nudges/icon.png"
  },
  "invite_text": "Great workshop. Swipe to know more!",
```

```
  "status": "scheduled",
  "createdAt": "2026-01-10T09:00:00Z"
}
```

2. API Documentation

Base URL: https://localhost:5000/api/v3/app/

These are the endpoints the frontend will need:

A. Create a New Nudge

When a user clicks "Publish Now" or saves a draft.

Endpoint: https://localhost:5000/api/v3/app/nudges

Method: POST

Description: Takes all the form data and saves it. For images, upload them to storage first, grab the URL, and send that. It keeps things cleaner.

Request Payload (JSON):

```
{
  "_id": ObjectId,
  "type": "nudge",
  "uid": 18,
  "tag_type": "event",
  "tag_id": "65fa9e...",
  "title": "Join the Backend Development",
  "description": "A deep dive into Backend Development",
  "schedule": {
    "date": "2026-02-10",
    "from": "10:00",
```

```
    "to": "12:00",
    "timestamp": "2026-02-10T10:00:00Z"
  },
  "files": {
    "cover_image": "/uploads/nudges/cover.png",
    "icon": "/uploads/nudges/icon.png"
  },
  "invite_text": "Great workshop. Swipe to know more!",

  "status": "scheduled",
  "createdAt": "2026-01-10T09:00:00Z"
}
```


Response:


```
{
  "nudge_id": "65fb1a9e8c12a9e7b1234567"
}
```


Success (200 OK): Returns the created nudgeId.

Error (400): Happens if the title is too long or you missed a required field.

Error(404): Not Found

B. Get Nudge Details

If someone wants to edit an existing nudge, we need to fetch the current data to pre-fill the form.

Endpoint: https://localhost:5000/api/v3/app/nudges?id=65fb1a9e8c12a9e7b1234567

Method: GET

Description: Returns all the data for a specific nudge ID, using the structure from the Data Model above.

Response:

{

"_id": "65fb1a9e8c12a9e7b1234567",

"title": "Join the Backend Development",

"status": "scheduled"

}

Nudges latest event pagination

Endpoint: https://localhost:5000/api/v3/app/nudges/latest?limit=5&page=1

Merhd: GET

C. Update Nudge

When a user edits a nudge and hits save.

Endpoint: https://localhost:5000/api/v3/app/nudges/:id

Method: POST (I went with POST here .PUT sometimes gets messy with partial updates, but open to discussing.)

Description: Update the fields that changed. Make sure you send the nudgeId so we know what to update.

Request Payload:

{

"_id": "65fb1a9e8c12a9e7b1234567",

"title": "Updated Title Here",

"details": "Updated description text..."

}

D. Delete Nudge

If a user wants to remove a scheduled nudge.

Endpoint: https://localhost:5000//api/v3/app/nudges/:id

Method: DELETE

Description: Hard deletes the nudge from the table. We could do a soft delete (just mark as 'deleted') if we want to keep history up for discussion.

Response:

{

  "message": "Nudge deleted successfully"

}

Success (200)

3. For Frontend Development

Image Uploads: Use the existing /upload/nudges endpoint for both "Choose Image" and "Choose Icon." Don't send binary files directly to /uploads/media it makes the payload huge for no reason.

Validation: Make sure the Title stays under 60 chars on the client side to avoid wasting API calls.

Dates: Always send dates in standard format so the backend doesn't get tripped up by timezones.