Calendly API Integration NestJS

I have to impleement calendly in my Mean Stack project for meeting scheduling?
please tell me the complete implemenation of calendly api in nest js and how the flow willgo and for what purpose calendly is used

What is Calendly?

Calendly is a scheduling tool that allows individuals and teams to schedule meetings with others without back-and-forth emails. It integrates with your calendar (e.g., Google Calendar, Outlook) to check availability and avoid conflicts. In a **MEAN Stack project**, you can use the **Calendly API** to automate scheduling functionalities, such as:

- Embedding scheduling links.
- Managing event types.
- Tracking scheduled events.
- Sending meeting invitations programmatically.

Steps to Integrate Calendly in a NestJS Project

1. Set Up Calendly API Access

- Create a Calendly account if you don't already have one.
- Go to Calendly API documentation and generate an **API key** or use **OAuth** for authentication.
- Copy the API key or client credentials.

2. Install Required Packages

- Install Axios for HTTP requests.

```bash
npm install axios
```

3. Create a Module for Calendly Integration

Create a `CalendlyModule` to handle all API interactions.

Structure:

```
arduino


  src/
  ├── calendly/
      ├── calendly.module.ts
      ├── calendly.service.ts
      ├── calendly.controller.ts
```

4. Calendly Service

Handle API requests to Calendly.

```
typescript


  import { Injectable, HttpException, HttpStatus } from '@nestjs/common';
  import axios, { AxiosInstance } from 'axios';

  @Injectable()
  export class CalendlyService {
    private readonly api: AxiosInstance;

    constructor() {
      this.api = axios.create({
        baseURL: 'https://api.calendly.com',
        headers: {
          Authorization: `Bearer <YOUR_API_KEY>`, // Replace with your API key
          'Content-Type': 'application/json',
        },
      });
    }

    // Get User Details
    async getUserDetails() {
      try {
        const response = await this.api.get('/users/me');
        return response.data;
      } catch (error) {
        throw new HttpException(
          error.response?.data || 'Error fetching user details',
          HttpStatus.BAD_REQUEST,
        );
      }
    }

    // Get Event Types
    async getEventTypes() {
      try {
        const response = await this.api.get('/event_types');
        return response.data;
      } catch (error) {
        throw new HttpException(
          error.response?.data || 'Error fetching event types',
          HttpStatus.BAD_REQUEST,
        );
      }
    }

    // Schedule an Event
    async createEvent(payload: any) {
      try {
        const response = await this.api.post('/scheduled_events', payload);
        return response.data;
      } catch (error) {
        throw new HttpException(
          error.response?.data || 'Error scheduling event',
          HttpStatus.BAD_REQUEST,
        );
```

```
    }
  }
}
```

5. Calendly Controller

Expose API endpoints for the frontend to interact with Calendly.

```typescript
import { Controller, Get, Post, Body } from '@nestjs/common';
import { CalendlyService } from './calendly.service';

@Controller('calendly')
export class CalendlyController {
  constructor(private readonly calendlyService: CalendlyService) {}

  @Get('user')
  async getUserDetails() {
    return this.calendlyService.getUserDetails();
  }

  @Get('event-types')
  async getEventTypes() {
    return this.calendlyService.getEventTypes();
  }

  @Post('schedule')
  async createEvent(@Body() payload: any) {
    return this.calendlyService.createEvent(payload);
  }
}
```

6. Calendly Module

Register the service and controller.

```typescript
import { Module } from '@nestjs/common';
import { CalendlyService } from './calendly.service';
import { CalendlyController } from './calendly.controller';

@Module({
  providers: [CalendlyService],
  controllers: [CalendlyController],
})
export class CalendlyModule {}
```

7. Add the Module to AppModule

Import the `CalendlyModule` in the main `AppModule`.

```typescript
import { Module } from '@nestjs/common';
import { CalendlyModule } from './calendly/calendly.module';
```

```
@Module({
  imports: [CalendlyModule],
})
export class AppModule {}
```

Example API Flow for Scheduling a Meeting

1. **Frontend Integration**:
   - The user selects available time slots from your app.
   - Send the details (event type, time, invitee email, etc.) to the backend.
2. **Backend Process**:
   - Call `createEvent` in `CalendlyService` with the required payload.

   Example Payload:

```json
{
  "event_type": "https://api.calendly.com/event_types/<EVENT_TYPE_ID>",
  "invitees": [
    {
      "email": "user@example.com",
      "name": "John Doe"
    }
  ],
  "start_time": "2024-12-10T14:00:00Z",
  "end_time": "2024-12-10T14:30:00Z"
}
```

3. **Calendly API Response**:
   - Returns event details and confirmation.
4. **Frontend Feedback**:
   - Display success/failure to the user.

Common Use Cases for Calendly in MEAN Stack Projects

1. **Automated Meeting Scheduling**: Embed Calendly to allow users to schedule meetings directly.
2. **Team Collaboration**: Allow team members to sync their availability.
3. **Appointment Booking**: Use Calendly for customer support or consultations.
4. **Event Tracking**: Fetch scheduled events to keep track of meetings.

Additional Features You Can Leverage

- **Webhooks**: Set up webhooks to receive real-time updates about events.
- **OAuth Authentication**: Allow users to connect their Calendly accounts securely.
- **Embedding Calendly Widget**: Use Calendly's embed options to display the scheduling interface in your app.

Would you like help with setting up Calendly webhooks or creating the frontend integration?

↳

{ "event_type": "https://api.calendly.com/event_types/<EVENT_TYPE_ID>", "invitees": \[{ "email": "user@example.com", "name": "John Doe" } \], "start_time": "2024-12-10T14:00:00Z", "end_time": "2024-12-10T14:30:00Z" }
but from where the frontend developer will get this payload

The frontend developer will derive this payload from two main sources: **data fetched from the backend** (e.g., available event types) and **user input** (e.g., name, email, and selected time). Below is a detailed explanation of how the payload components are gathered:

---

1. Event Type (`event_type`)

The frontend will fetch a list of available event types from the backend.

- **Backend Endpoint**: `/calendly/event-types` (from the example above).
- **Flow**:
  1. The frontend sends a GET request to `/calendly/event-types`.
  2. The backend (CalendlyController) responds with a list of event types.
  3. The frontend populates a dropdown or list with the event types for the user to select.

  Example response:

```json
{
  "collection": [
    {
      "name": "30-Minute Meeting",
      "slug": "30-min-meeting",
      "uri": "https://api.calendly.com/event_types/ABCDEFG"
    },
    {
      "name": "60-Minute Meeting",
      "slug": "60-min-meeting",
      "uri": "https://api.calendly.com/event_types/HIJKLMN"
    }
  ]
}
```

  The frontend developer will include the `uri` (e.g., `"https://api.calendly.com/event_types/ABCDEFG"`) as the `event_type` in the payload.

---

2. Invitees (`invitees`)

The `invitees` information comes from **user input** in the frontend.

- **Frontend Form**: The frontend will have a form where users provide their **name** and **email**.
  Example form fields:
  - Name: `John Doe`
  - Email: `user@example.com`

  The frontend will map this data into the required structure:

```json
{
  "invitees": [
    {
      "email": "user@example.com",
      "name": "John Doe"
    }
  ]
}
```

---

3. Start Time and End Time (`start_time`, `end_time`)

The time selection typically depends on:

1. **Available Time Slots**: The frontend can use a calendar picker that fetches available time slots from Calendly.
   - Use the `/calendly/availability` (custom backend endpoint calling Calendly API).
   - Response might include:

```json
{
  "available_times": [
    {
      "start_time": "2024-12-10T14:00:00Z",
      "end_time": "2024-12-10T14:30:00Z"
    },
    {
      "start_time": "2024-12-10T15:00:00Z",
      "end_time": "2024-12-10T15:30:00Z"
    }
  ]
}
```

   - The frontend displays these slots in a date-time picker for users to choose.
2. **User Input**: After the user selects a time slot, the frontend populates the payload with the selected start and end times.

---

4. Final Payload in Frontend

Once all the pieces are gathered, the frontend assembles the payload and sends it to the backend.

Example payload:

```json
{
  "event_type": "https://api.calendly.com/event_types/ABCDEFG",
  "invitees": [
    {
      "email": "user@example.com",
      "name": "John Doe"
    }
  ],
  "start_time": "2024-12-10T14:00:00Z",
  "end_time": "2024-12-10T14:30:00Z"
}
```

Sample Flow of Communication

1. **Fetch Event Types**:
   - Frontend calls `/calendly/event-types` to get event types and displays them to the user.
2. **Fetch Available Time Slots**:
   - Frontend calls a custom backend endpoint (`/calendly/availability`) to get available time slots based on the selected event type.
3. **User Selects Details**:
   - User selects an event type, provides name/email, and picks a time slot.
4. **Send Payload to Backend**:
   - Frontend sends the completed payload to the backend (`/calendly/schedule`).
5. **Backend Schedules Event**:
   - Backend calls Calendly's `scheduled_events` API and returns the confirmation details.

Additional Notes for the Frontend Developer

- **Form Validation**: Ensure user inputs (email, name, time slot) are validated before sending the request.
- **UI Elements**: Use libraries like a date-time picker to make selecting time slots easier.
- **Error Handling**: Display error messages for any issues with scheduling (e.g., time conflict or missing fields).

Would you like me to help create the frontend form flow or the custom `/availability` backend endpoint?

↪

{ "collection": \[{ "name": "30-Minute Meeting", "slug": "30-min-meeting", "uri": "https://api.calendly.com/event_types/ABCDEFG" }, { "name": "60-Minute Meeting", "slug": "60-min-meeting", "uri": "https://api.calendly.com/event_types/HIJKLMN" } \] }
couldn't understand what are these event typesetc

please explain like I am 10 year old child explain whole process

Let me simplify this step by step! 🎉

What Are Event Types?

Imagine you have a lemonade stand. You sell:

1. **A small lemonade (30 minutes to make)**.
2. **A large lemonade (60 minutes to make)**.

When a customer wants to order, they need to tell you **what size** they want (small or large). Similarly, in Calendly:

- **Event Types** are like menu options for meetings.
  Example:
  - A **30-minute meeting**.
  - A **60-minute meeting**.

Each "type" has:

1. A **name** (e.g., "30-Minute Meeting").
2. A **unique link** (e.g., `"https://api.calendly.com/event_types/ABCDEFG"`) so Calendly knows which event type you're talking about.

The Whole Process Like a Story

Let's say you're building a website where people can book a time to talk to you. Here's how it works step-by-step:

---

1. Set Up Event Types

You, as the "meeting owner," go to Calendly and set up what types of meetings people can book. For example:

- **Option 1**: A 30-minute chat.
- **Option 2**: A 60-minute deep dive.

These are your **event types**, and Calendly gives you a **unique link** for each one.

---

2. Frontend Asks Backend: "What Are the Options?"

Your website needs to show visitors the available meeting options. So:

1. **Frontend (Your Website)**: It sends a request to your NestJS backend:
   `"Hey backend, what types of meetings can I show to visitors?"`

2. **Backend**: It asks Calendly:
   `"Calendly, what event types do we have?"`
   And Calendly replies with this:

```json
{
  "collection": [
    {
      "name": "30-Minute Meeting",
      "slug": "30-min-meeting",
      "uri": "https://api.calendly.com/event_types/ABCDEFG"
    },
    {
      "name": "60-Minute Meeting",
      "slug": "60-min-meeting",
      "uri": "https://api.calendly.com/event_types/HIJKLMN"
    }
  ]
}
```

3. **Frontend**: The website shows the visitor these options:
   - A 30-minute meeting.
   - A 60-minute meeting.

---

3. User Chooses a Meeting

Now the visitor selects "30-Minute Meeting." Next, they need to pick:

1. **Their Name** (e.g., John Doe).
2. **Their Email** (e.g., john@example.com).
3. **A Time** (e.g., "Tuesday, 2 PM").

---

4. Frontend Sends Info to Backend

Once the visitor fills in their details, the website sends this information to your backend, something like this:

```json
{
  "event_type": "https://api.calendly.com/event_types/ABCDEFG",
  "invitees": [
    {
      "email": "john@example.com",
      "name": "John Doe"
    }
  ],
  "start_time": "2024-12-10T14:00:00Z",
  "end_time": "2024-12-10T14:30:00Z"
}
```

---

5. Backend Talks to Calendly

Your NestJS backend now sends this information to Calendly's API:

- It says:
  `"Calendly, please book a 30-minute meeting with John Doe at 2 PM on December 10th."`
- Calendly processes the request and responds:
  `"Done! Here's the confirmation link for John Doe: https://calendly.com/your-link/confirmation-id"`

---

6. Backend Replies to Frontend

The NestJS backend tells the frontend:
`"The meeting is scheduled! Here's the confirmation link: https://calendly.com/your-link/confirmation-id"`

---

7. Frontend Shows Success Message

Your website now tells the visitor:
`"Your meeting is confirmed! Check your email for the details or click this link: https://calendly.com/your-link/confirmation-id"`

Recap: The Flow Like a Game 🎮

1. **Calendly**: Sets up meeting options (like a menu).
2. **Frontend**: Asks the backend for available options.
3. **Backend**: Gets the options from Calendly and sends them to the frontend.
4. **Visitor**: Picks a meeting, gives their details, and selects a time.
5. **Frontend**: Sends this info to the backend.
6. **Backend**: Books the meeting with Calendly.
7. **Calendly**: Confirms the booking and provides a link.
8. **Frontend**: Shows the visitor the confirmation.

I hope this explanation makes the process crystal clear! Would you like me to create specific examples for the frontend or backend code? 😊

↪

It says: `"Calendly, please book a 30-minute meeting with John Doe at 2 PM on December 10th."` will the calendly remind both the participatant of the meeting about meeting on fixed meeting date and time?

Yes, Calendly will **automatically remind both participants** (the meeting host and the invitee) about the scheduled meeting on the fixed date and time. Here's how it works:

---

1. Confirmation Email (Sent Immediately After Booking)

Once the meeting is successfully scheduled:

- **Host** (you): Receives a confirmation email with all the meeting details.
- **Invitee** (e.g., John Doe): Also receives a confirmation email with:
  - Meeting time.
  - Details (e.g., link to the meeting, location, etc.).
  - Any additional notes or instructions you included in the event setup.

---

2. Reminder Emails (Before the Meeting)

Calendly sends **automatic reminders** to both the host and invitee at specific intervals before the meeting.
Default reminders are typically sent:

- **24 hours before the meeting**.
- **1 hour before the meeting**.

These reminders can include:

- A link to the meeting (e.g., Zoom/Google Meet link if set up).
- Reschedule or cancellation options (if allowed).

---

3. Custom Notifications

As the meeting host, you can customize how Calendly handles reminders:

- Choose how far in advance reminders are sent (e.g., 2 days, 30 minutes, etc.).
- Decide whether reminders are sent via **email** or **SMS** (if SMS notifications are enabled in your Calendly plan).
- Personalize the reminder message content.

---

4. Calendar Integration

If both participants have their calendars synced (e.g., Google Calendar or Outlook), the event is automatically added to their calendar with all the details. This way:

- Both participants can get calendar app notifications based on their personal settings (e.g., pop-ups or phone reminders).

In Summary:

Yes, Calendly ensures both the host and the invitee are reminded about the meeting. It manages this through:

- **Confirmation emails** after booking.
- **Automatic reminders** (email/SMS) before the meeting.
- **Calendar integration notifications** for additional alerts.

Let me know if you want to customize these notifications or understand how to set up SMS reminders! 😊

okay I got it now please tell the whole procedure with code to implement it in nest js

Here's a complete guide to implementing **Calendly meeting scheduling** in your **NestJS** backend with **code examples**. It covers authentication, fetching event types, scheduling a meeting, and handling reminders.

Step 1: Set Up Calendly API Key

1. **Get Your Personal API Key**:
   - Log in to Calendly.
   - Go to **Integrations** > **API and Webhooks**.
   - Generate or copy your **Personal Access Token**.
2. Store the API Key securely, such as in an `.env` file:

```
env


  CALENDLY_API_KEY=your_calendly_api_key
```

Step 2: Install Necessary Dependencies

In your NestJS project, install the required packages:

```
bash


  npm install @nestjs/axios dotenv
```

Step 3: Set Up the Environment

Update your `AppModule` to include `ConfigModule`:

```
typescript


  // app.module.ts
  import { Module } from '@nestjs/common';
  import { ConfigModule } from '@nestjs/config';
  import { HttpModule } from '@nestjs/axios';
  import { CalendlyModule } from './calendly/calendly.module';

  @Module({
    imports: [
      ConfigModule.forRoot({
```

```
        isGlobal: true, // Make .env variables accessible globally
      }),
      HttpModule,
      CalendlyModule,
    ],
  })
  export class AppModule {}
```

Step 4: Create a Calendly Module

Generate a module for Calendly:

```
bash


  nest generate module calendly
  nest generate service calendly
  nest generate controller calendly
```

Step 5: Implement the Service

The **CalendlyService** will handle API requests to Calendly.

Code for `calendly.service.ts`:

```
typescript


  import { Injectable } from '@nestjs/common';
  import { HttpService } from '@nestjs/axios';
  import { firstValueFrom } from 'rxjs';
  import { ConfigService } from '@nestjs/config';

  @Injectable()
  export class CalendlyService {
    private readonly baseUrl = 'https://api.calendly.com';

    constructor(
      private readonly httpService: HttpService,
      private readonly configService: ConfigService,
    ) {}

    // Fetch Event Types
    async getEventTypes(): Promise<any> {
      const apiKey = this.configService.get<string>('CALENDLY_API_KEY');
      const response = await firstValueFrom(
        this.httpService.get(`${this.baseUrl}/event_types`, {
          headers: {
            Authorization: `Bearer ${apiKey}`,
          },
        }),
      );
      return response.data;
    }

    // Schedule a Meeting
    async scheduleMeeting(payload: any): Promise<any> {
      const apiKey = this.configService.get<string>('CALENDLY_API_KEY');
      const response = await firstValueFrom(
        this.httpService.post(`${this.baseUrl}/scheduled_events`, payload, {
          headers: {
            Authorization: `Bearer ${apiKey}`,
```

```
            'Content-Type': 'application/json',
          },
        }),
      );
      return response.data;
    }
  }
```

Step 6: Set Up the Controller

The **CalendlyController** will define routes for your NestJS app.

Code for `calendly.controller.ts`:

```typescript
import { Controller, Get, Post, Body } from '@nestjs/common';
import { CalendlyService } from './calendly.service';

@Controller('calendly')
export class CalendlyController {
  constructor(private readonly calendlyService: CalendlyService) {}

  // Get all event types
  @Get('event-types')
  async getEventTypes() {
    return await this.calendlyService.getEventTypes();
  }

  // Schedule a meeting
  @Post('schedule')
  async scheduleMeeting(@Body() body: any) {
    return await this.calendlyService.scheduleMeeting(body);
  }
}
```

Step 7: Frontend Interaction

The **Frontend Workflow** will look like this:

1. **Fetch Event Types**:

```javascript
fetch('/calendly/event-types')
  .then((res) => res.json())
  .then((data) => console.log('Event Types:', data));
```

2. **Schedule a Meeting**: Example payload to send:

```javascript
const payload = {
  event_type: 'https://api.calendly.com/event_types/ABCDEFG',
  invitees: [
```

```
      {
        email: 'john.doe@example.com',
        name: 'John Doe',
      },
    ],
    start_time: '2024-12-10T14:00:00Z',
    end_time: '2024-12-10T14:30:00Z',
  };

  fetch('/calendly/schedule', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(payload),
  })
    .then((res) => res.json())
    .then((data) => console.log('Scheduled Meeting:', data));
```

Step 8: Testing

1. **Run NestJS Server**: Start your backend:

```
bash


  npm run start
```

2. **Test Endpoints**: Use tools like Postman or curl to test:
   - **Get Event Types**:

```
bash


  GET http://localhost:3000/calendly/event-types
```

   - **Schedule a Meeting**:

```
bash


  POST http://localhost:3000/calendly/schedule
  Body:
  {
    "event_type": "https://api.calendly.com/event_types/ABCDEFG",
    "invitees": [
      { "email": "john.doe@example.com", "name": "John Doe" }
    ],
    "start_time": "2024-12-10T14:00:00Z",
    "end_time": "2024-12-10T14:30:00Z"
  }
```

Step 9: Handle Reminders

Calendly handles reminders automatically:

- Emails are sent to both the **host** and **invitee**.
- Ensure the email address provided in the payload is correct.

With this setup, you can integrate Calendly into your NestJS app to fetch event types, schedule meetings, and leverage its built-in reminder features. Let me know if you need help with the frontend or additional configurations! 😊

what about calendly webhooks

Calendly webhooks allow your application to listen for specific events happening in Calendly, such as when a meeting is scheduled, rescheduled, or canceled. By using webhooks, you can automatically trigger actions in your application based on these events.

Here's how to implement Calendly webhooks in a **NestJS** project:

Step 1: What Are Webhooks?

A webhook is an HTTP POST request sent from Calendly to your application whenever a specific event occurs. For example:

- **Invitee Created**: A meeting is scheduled.
- **Invitee Canceled**: A meeting is canceled.

Step 2: Prerequisites

1. **Calendly API Key**: You need your API token (as described earlier).
2. **Publicly Accessible Endpoint**: Your application must have a public URL for Calendly to send webhook notifications. You can use tools like ngrok for local testing.

Step 3: Enable Webhooks in Calendly

1. Register a Webhook URL

Calendly provides an API endpoint to register your webhook. You'll send a POST request to `` `https://api.calendly.com/webhook_subscriptions` ``.

Example payload:

```
json


  {
    "url": "https://your-app.com/calendly/webhook",
    "events": [
      "invitee.created",
      "invitee.canceled"
    ],
    "organization": "https://api.calendly.com/organizations/ORG_ID"
  }
```

Step 4: Implement Webhooks in NestJS

4.1 Create a Controller

This controller will handle webhook events from Calendly.

```typescript
// calendly.controller.ts
import { Controller, Post, Body, Headers, HttpCode } from '@nestjs/common';

@Controller('calendly')
export class CalendlyController {
  @Post('webhook')
  @HttpCode(200) // Calendly requires a 200 OK response
  handleWebhook(
    @Body() payload: any,
    @Headers('x-calendly-signature') signature: string,
  ) {
    // Validate the signature (optional)
    console.log('Webhook received:', payload);

    // Process the event
    if (payload.event === 'invitee.created') {
      console.log('Meeting Scheduled:', payload.payload);
    } else if (payload.event === 'invitee.canceled') {
      console.log('Meeting Canceled:', payload.payload);
    }

    return { message: 'Webhook received successfully' };
  }
}
```

4.2 Service to Register Webhook

Add a service function to register your webhook dynamically.

```typescript
import { Injectable } from '@nestjs/common';
import { HttpService } from '@nestjs/axios';
import { firstValueFrom } from 'rxjs';

@Injectable()
export class CalendlyService {
  private readonly baseUrl = 'https://api.calendly.com';
  private readonly apiKey = 'your_calendly_api_key';

  constructor(private readonly httpService: HttpService) {}

  async registerWebhook() {
    const payload = {
      url: 'https://your-app.com/calendly/webhook',
      events: ['invitee.created', 'invitee.canceled'],
      organization: 'https://api.calendly.com/organizations/ORG_ID',
    };

    const response = await firstValueFrom(
      this.httpService.post(`${this.baseUrl}/webhook_subscriptions`, payload, {
        headers: {
          Authorization: `Bearer ${this.apiKey}`,
          'Content-Type': 'application/json',
        },
      }),
```

```
      );

      return response.data;
    }
  }
```

4.3 Configure Webhook Validation (Optional)

To ensure the webhook is legitimate, validate the `x-calendly-signature` header using your webhook signing key (available in Calendly's webhook configuration).

```typescript
import * as crypto from 'crypto';

function validateSignature(payload: any, signature: string, signingKey: string): boolean {
  const hash = crypto
    .createHmac('sha256', signingKey)
    .update(JSON.stringify(payload))
    .digest('hex');

  return hash === signature;
}
```

Update your webhook handler:

```typescript
if (!validateSignature(payload, signature, process.env.CALENDLY_SIGNING_KEY)) {
  throw new Error('Invalid signature');
}
```

Step 5: Register the Webhook

You can register the webhook either:

- Manually using tools like **Postman**.
- Programmatically by calling the `registerWebhook()` service method.

For example:

```typescript
@Get('register-webhook')
async registerWebhook() {
  const result = await this.calendlyService.registerWebhook();
  return result;
}
```

Step 6: Test Webhook Integration

1. **Set Up a Public URL**: Use ngrok to expose your local server.

```
bash


  ngrok http 3000
```

2. **Register Webhook**: Call the `/register-webhook` endpoint to register your webhook with the public URL.
3. **Trigger Webhooks**:
    - Schedule or cancel a meeting in Calendly.
    - Verify that your NestJS webhook endpoint receives the events.
4. **Check Logs**: View console logs in your application to see the payload data.

Step 7: Process Webhook Events

Handle events as per your business requirements, such as:

- Storing scheduled meetings in your database.
- Sending notifications to users.
- Updating calendars or internal systems.

Webhook Payload Example

When an event occurs, Calendly sends a payload like this:

Invitee Created:

```
json


  {
    "event": "invitee.created",
    "payload": {
      "event": "https://api.calendly.com/scheduled_events/ABCDEFG",
      "invitee": {
        "email": "john.doe@example.com",
        "name": "John Doe"
      },
      "start_time": "2024-12-10T14:00:00Z",
      "end_time": "2024-12-10T14:30:00Z"
    }
  }
```

Invitee Canceled:

```
json


  {
    "event": "invitee.canceled",
    "payload": {
      "event": "https://api.calendly.com/scheduled_events/ABCDEFG",
```

```
      "invitee": {
        "email": "john.doe@example.com",
        "name": "John Doe"
      }
    }
  }
```

This implementation ensures that your NestJS app can handle Calendly webhooks effectively, allowing you to automate meeting-related actions!

↪

organization: 'https://api.calendly.com/organizations/ORG_ID',
what is this origanization url and where to get it?

The **organization URL** in Calendly refers to the unique identifier for your organization within Calendly's system. This is required when registering webhooks because it specifies the context in which the webhook should operate (e.g., your team or account).

Where to Find the Organization URL

You can retrieve the organization URL via Calendly's API using your **API Key** or **OAuth token**. Here's how:

Step 1: Retrieve Organization Information

1. Use the `GET` endpoint:
   URL: `https://api.calendly.com/users/me`
2. This endpoint provides information about the authenticated user, including the organization they belong to.

Example Request (Using cURL)

```bash

curl -X GET https://api.calendly.com/users/me \
-H "Authorization: Bearer YOUR_API_KEY"
```

Example Response

```json

{
  "resource": {
```

```
    "uri": "https://api.calendly.com/users/USER_ID",
    "name": "Your Name",
    "email": "your-email@example.com",
    "organization": "https://api.calendly.com/organizations/ORG_ID",
    ...
  }
}
```

- The `organization` field contains the organization URL, which looks like: `https://api.calendly.com/organizations/ORG_ID`.

Using the Organization URL in Webhook Registration

Once you have the organization URL, include it in your webhook registration payload:

```
json


{
  "url": "https://your-app.com/calendly/webhook",
  "events": ["invitee.created", "invitee.canceled"],
  "organization": "https://api.calendly.com/organizations/ORG_ID"
}
```

Testing the Organization URL

- If you're part of a team on Calendly, the organization URL ensures the webhook listens to all relevant events within your team account.
- For personal accounts, the organization URL will correspond to your individual Calendly account.

If you're unable to retrieve the organization URL or face issues, double-check your API token permissions or contact Calendly support.

Uploaded image
at frontend the set time in my calendly account is not showing should i have to add api key in angular frontend during implementation?

You typically don't need to add the API key directly in the Angular frontend because exposing your API key in the frontend can lead to security vulnerabilities. Instead, you should use the **embedded Calendly widget or iframe**, which directly fetches the available times from your Calendly account.

Why Times Might Not Be Displaying

1. **Account Configuration**: Ensure that your event type in Calendly is active and properly configured with available dates and times.

2. **Calendly Widget Script**: Make sure you're correctly embedding the Calendly widget or using the embed URL.
3. **API Key Not Required for Frontend**: Using the embed method does not require the API key. The widget directly fetches data from Calendly servers.

How to Fix the Issue

Option 1: Embed Calendly Using a Widget (No API Key Needed)

Ensure you're using the official Calendly embed widget in your Angular app. Example:

1. **Install Calendly Embed Package**: If you want to use Calendly's widget, install their embed script:

```bash
npm install react-calendly
```

(Works for React and Angular)

2. **Add the Embed Script in Your HTML**: Include the Calendly embed code in the component where you want it displayed:

```html
<div id="schedule-appointment">
    <a
      href="https://calendly.com/your-username/event-type"
      onclick="Calendly.initPopupWidget({url: 'https://calendly.com/your-username/event-type'}); return false;"
    >
      Schedule time with me
    </a>
</div>
<script
  type="text/javascript"
  src="https://assets.calendly.com/assets/external/widget.js"
  async
></script>
```

   - Replace `your-username` and `event-type` with your actual Calendly username and event type.

Option 2: Use the Embedded iFrame

If you prefer to use an iframe:

```html
<iframe
  src="https://calendly.com/your-username/event-type"
  width="100%"
  height="630"
  frameborder="0"
></iframe>
```

Option 3: Fetch Data via API (Advanced Approach)

If you want more control (e.g., customizing the display), you can:

1. Fetch the event times from Calendly API in your NestJS backend.
2. Use the backend to serve this data to your Angular frontend.

However, this is only necessary if you need a fully custom solution. Using the widget or iframe is the simplest way to display Calendly.

Security Reminder

- Avoid exposing sensitive API keys in the frontend.
- Use Calendly's embed options for

↪

Add the Embed Script in Your HTML: Include the Calendly embed code in the component where you want it displayed: html Copy code `<div id="schedule-appointment"> Schedule time with me </div> <script type="text/javascript" src="https://assets.calendly.com/assets/external/widget.js" async ></script>`
this href and onclick url will came from the backend implementation?

The `href` and `onclick` URLs (e.g., `https://calendly.com/your-username/event-type`) **do not need to come from the backend** if you are embedding Calendly directly in your Angular app. These URLs are specific to the Calendly event type you have configured and can be hardcoded in the frontend.

How to Get the URL?

1. Log in to your Calendly account.
2. Go to the event type you want to embed (e.g., "30-minute meeting").
3. Copy the event type link, which looks like this:

```arduino
https://calendly.com/your-username/30min
```

4. Replace `your-username` and `30min` in the code snippet with your actual Calendly username and event type.

When Would the URL Come From the Backend?

If you want to **dynamically configure the event type** or **fetch the URL from the backend** for better flexibility (e.g., different users have different Calendly accounts or event types), you can:

1. **Backend Implementation (NestJS)**: Add an endpoint in your backend to return the appropriate Calendly URL for the frontend:

```typescript
import { Controller, Get } from '@nestjs/common';

@Controller('calendly')
export class CalendlyController {
  @Get('event-link')
  getEventLink() {
    return {
      url: 'https://calendly.com/your-username/event-type',
    };
  }
}
```

2. **Frontend Implementation (Angular)**: Fetch this URL from your NestJS backend and use it in the `href` and `onclick` attributes:

```typescript
import { HttpClient } from '@angular/common/http';
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-schedule-meeting',
  template: `
    <div *ngIf="calendlyUrl">
      <a
        [href]="calendlyUrl"
        (click)="openCalendly($event)"
      >
        Schedule time with me
      </a>
    </div>
  `,
})
export class ScheduleMeetingComponent implements OnInit {
  calendlyUrl: string = '';

  constructor(private http: HttpClient) {}

  ngOnInit() {
    this.http
      .get<{ url: string }>('/api/calendly/event-link')
      .subscribe((data) => {
        this.calendlyUrl = data.url;
      });
  }

  openCalendly(event: MouseEvent) {
    event.preventDefault();
    if (this.calendlyUrl) {
      Calendly.initPopupWidget({ url: this.calendlyUrl });
    }
  }
}
```

Which Approach Should You Choose?

- **Static URL in Frontend**: If the Calendly event type is fixed and the same for everyone, hardcode the URL in the frontend. This is simpler and faster.
- **Dynamic URL from Backend**: If the event type or Calendly account is dynamic (e.g., multiple users with unique links), fetch the URL from the backend.

Both approaches will work based on your project's needs! Let me know if you need further clarification.