Certainly! Let's delve into the integrations/ directory, which is essential for managing third-party API integrations in your NestJS application.

**📁 integrations/ — Third-Party API Integrations**

**📌 Purpose**

The integrations/ directory is designed to handle all interactions with external services and API. By centralizing these integrations, you ensure maintainability, scalability, and a clear separation of concerns within your applicatio.

**🗂️ Suggested Directory Structur**



src/

└── integrations/

├── stripe/

│ ├── stripe.service.ts

│ ├── stripe.controller.ts

│ └── stripe.module.ts

├── sendgrid/

│ ├── sendgrid.service.ts

│ ├── sendgrid.controller.ts

│ └── sendgrid.module.ts

├── google/

│ ├── google.service.ts

│ ├── google.controller.ts

│ ├── google.strategy.ts

│ └── google.module.ts

├── facebook/

│ ├── facebook.service.ts

│ ├── facebook.controller.ts

│ ├── facebook.strategy.ts

│ └── facebook.module.ts

├── bkash/

│ ├── bkash.service.ts

│ ├── bkash.controller.ts

│ ├── bkash.strategy.ts

│ └── bkash.module.ts

├── sslcommerz/

│ ├── sslcommerz.service.ts

│ ├── sslcommerz.controller.ts

│ ├── sslcommerz.strategy.ts

│ └── sslcommerz.module.ts

└── integrations.module.ts

``



- \*\*`stripe/`\*: Handles all Stripe-related functionalities, such as payment processing and subscription managemet.

- \*\*`sendgrid/`\*: Manages email sending capabilities via SendGrd.

- \*\*`google/`\*:: Handles Google authentication logic.

- \*\*`facebook/`\*:: Manages Facebook authentication logic.



- \*\*`integrations.module.ts`\*: Aggregates all integration modules into a single module for easy import into the root modue.

---

### 🧩 Implementation Details

#### 1. \*\*Stripe Integration\*\*

\*\*`stripe.service.ts`\*



```typescript

import { Injectable } from '@nestjs/common';

import Stripe from 'stripe';

import { ConfigService } from '@nestjs/config';

/\*\*

\* Service to handle Stripe payment processing.

\*/

@Injectable()

export class StripeService {

private stripe: Stripe;

constructor(private configService: ConfigService) {

this.stripe = new Stripe(this.configService.get('STRIPE\_SECRET\_KEY'), {

apiVersion: '2020-08-27',

});

}

/\*\*

\* Creates a payment intent for a given amount.

\* @param amount - The amount to charge in cents.

\* @returns The created payment intent.

\*/

async createPaymentIntent(amount: number): Promise<Stripe.PaymentIntent> {

return this.stripe.paymentIntents.create({

amount,

currency: 'usd',

});

}

}

``



\*\*`stripe.controller.ts`\*



```typescript

import { Controller, Post, Body } from '@nestjs/common';

import { StripeService } from './stripe.service';

/\*\*

\* Controller to handle Stripe payment requests.

\*/

@Controller('stripe')

export class StripeController {

constructor(private readonly stripeService: StripeService) {}

/\*\*

\* Endpoint to create a payment intent.

\* @param amount - The amount to charge in cents.

\* @returns The client secret for the payment intent.

\*/

@Post('create-payment-intent')

async createPaymentIntent(@Body('amount') amount: number) {

const paymentIntent = await this.stripeService.createPaymentIntent(amount);

return { clientSecret: paymentIntent.client\_secret };

}

}

``



\*\*`stripe.module.ts`\*



```typescript

import { Module } from '@nestjs/common';

import { StripeService } from './stripe.service';

import { StripeController } from './stripe.controller';

import { ConfigModule } from '@nestjs/config';

/\*\*

\* Module to encapsulate Stripe payment functionalities.

\*/

@Module({

imports: [ConfigModule],

providers: [StripeService],

controllers: [StripeController],

exports: [StripeService],

})

export class StripeModule {}

``



#### 2. \*\*SendGrid Integration\*\*

\*\*`sendgrid.service.ts`\*



```typescript

import { Injectable } from '@nestjs/common';

import { ConfigService } from '@nestjs/config';

import { SendGridMailService } from '@sendgrid/mail';

/\*\*

\* Service to handle email sending via SendGrid.

\*/

@Injectable()

export class SendGridService {

private sendGridMail: SendGridMailService;

constructor(private configService: ConfigService) {

this.sendGridMail = new SendGridMailService();

this.sendGridMail.setApiKey(this.configService.get('SENDGRID\_API\_KEY'));

}

/\*\*

\* Sends an email using SendGrid.

\* @param to - The recipient's email address.

\* @param subject - The subject of the email.

\* @param text - The plain text content of the email.

\* @returns The response from SendGrid.

\*/

async sendEmail(to: string, subject: string, text: string) {

const msg = {

to,

from: this.configService.get('SENDGRID\_FROM\_EMAIL'),

subject,

text,

};

return this.sendGridMail.send(msg);

}

}

``



\*\*`sendgrid.controller.ts`\*



```typescript

import { Controller, Post, Body } from '@nestjs/common';

import { SendGridService } from './sendgrid.service';

/\*\*

\* Controller to handle email sending requests.

\*/

@Controller('sendgrid')

export class SendGridController {

constructor(private readonly sendGridService: SendGridService) {}

/\*\*

\* Endpoint to send an email.

\* @param to - The recipient's email address.

\* @param subject - The subject of the email.

\* @param text - The plain text content of the email.

\* @returns A success message.

\*/

@Post('send-email')

async sendEmail(@Body() emailData: { to: string; subject: string; text: string }) {

await this.sendGridService.sendEmail(emailData.to, emailData.subject, emailData.text);

return { message: 'Email sent successfully' };

}

}

``



\*\*`sendgrid.module.ts`\*



```typescript

import { Module } from '@nestjs/common';

import { SendGridService } from './sendgrid.service';

import { SendGridController } from './sendgrid.controller';

import { ConfigModule } from '@nestjs/config';

/\*\*

\* Module to encapsulate SendGrid email functionalities.

\*/

@Module({

imports: [ConfigModule],

providers: [SendGridService],

controllers: [SendGridController],

exports: [SendGridService],

})

export class SendGridModule {}

``



#### 3. \*\*Integrations Module\*\*

\*\*`integrations.module.ts`\*



```typescript

import { Module } from '@nestjs/common';

import { StripeModule } from './stripe/stripe.module';

import { SendGridModule } from './sendgrid/sendgrid.module';

/\*\*

\* Module to aggregate all third-party integrations.

\*/

@Module({

imports: [StripeModule, SendGridModule],

exports: [StripeModule, SendGridModule],

})

export class IntegrationsModule {}

``



---

### 🔗 Integration into Other Modules or Servies

To utilize the services from the `integrations/` directory in other parts of your application, you can inject them as proviers:

\*\*Example: Using StripeService in an Order Service\*\*

**1. Google Authentication**

To integrate Google authentication, utilize the passport-google-oauth20 strategy.​

**google.strategy.ts**

typescript

CopyEdit

import { Injectable } from '@nestjs/common';

import { PassportStrategy } from '@nestjs/passport';

import { Strategy } from 'passport-google-oauth20';

import { ConfigService } from '@nestjs/config';

@Injectable()

export class GoogleStrategy extends PassportStrategy(Strategy, 'google') {

constructor(private configService: ConfigService) {

super({

clientID: configService.get('GOOGLE\_CLIENT\_ID'),

clientSecret: configService.get('GOOGLE\_CLIENT\_SECRET'),

callbackURL: configService.get('GOOGLE\_CALLBACK\_URL'),

scope: ['email', 'profile'],

});

}

async validate(accessToken: string, refreshToken: string, profile: any) {

// Implement user validation logic here

return { profile, accessToken };

}

}

This strategy handles the OAuth flow and user validation.​

**2. Facebook Authentication**

For Facebook authentication, use the passport-facebook strategy.​

**facebook.strategy.ts**

typescript

CopyEdit

import { Injectable } from '@nestjs/common';

import { PassportStrategy } from '@nestjs/passport';

import { Strategy } from 'passport-facebook';

import { ConfigService } from '@nestjs/config';

@Injectable()

export class FacebookStrategy extends PassportStrategy(Strategy, 'facebook') {

constructor(private configService: ConfigService) {

super({

clientID: configService.get('FACEBOOK\_APP\_ID'),

clientSecret: configService.get('FACEBOOK\_APP\_SECRET'),

callbackURL: configService.get('FACEBOOK\_CALLBACK\_URL'),

profileFields: ['id', 'emails', 'name'],

});

}

async validate(accessToken: string, refreshToken: string, profile: any) {

// Implement user validation logic here

return { profile, accessToken };

}

}

This strategy manages the Facebook OAuth flow and user validation.​

**3. Integrations Module**

The integrations.module.ts aggregates all third-party integrations.​

**integrations.module.ts**

typescript

CopyEdit

import { Module } from '@nestjs/common';

import { GoogleStrategy } from './google/google.strategy';

import { FacebookStrategy } from './facebook/facebook.strategy';

@Module({

providers: [GoogleStrategy, FacebookStrategy],

exports: [GoogleStrategy, FacebookStrategy],

})

export class IntegrationsModule {}

This module ensures that all integration strategies are available throughout the application.​

**🔗 Integration into Other Modules or Services**

To utilize these integrations in other parts of your application, you can inject the respective services or strategies as needed.

**1. bKash Integration**

bKash provides a Node.js library to facilitate payment gateway integration. You can install it using:​

bash

CopyEdit

npm install bkash-payment-gateway

**bkash.service.ts**

typescript

CopyEdit

import { Injectable } from '@nestjs/common';

import { BkashPaymentGateway } from 'bkash-payment-gateway';

import { ConfigService } from '@nestjs/config';

@Injectable()

export class BkashService {

private bkash: BkashPaymentGateway;

constructor(private configService: ConfigService) {

this.bkash = new BkashPaymentGateway({

username: this.configService.get('BKASH\_USERNAME'),

password: this.configService.get('BKASH\_PASSWORD'),

secret: this.configService.get('BKASH\_SECRET'),

key: this.configService.get('BKASH\_KEY'),

});

}

async initiatePayment(amount: number) {

return this.bkash.initiatePayment(amount);

}

async executePayment(paymentId: string) {

return this.bkash.executePayment(paymentId);

}

}

**bkash.controller.ts**

typescript

CopyEdit

import { Controller, Post, Body } from '@nestjs/common';

import { BkashService } from './bkash.service';

@Controller('bkash')

export class BkashController {

constructor(private readonly bkashService: BkashService) {}

@Post('initiate')

async initiate(@Body('amount') amount: number) {

return this.bkashService.initiatePayment(amount);

}

@Post('execute')

async execute(@Body('paymentId') paymentId: string) {

return this.bkashService.executePayment(paymentId);

}

}

**bkash.module.ts**

typescript

CopyEdit

import { Module } from '@nestjs/common';

import { BkashService } from './bkash.service';

import { BkashController } from './bkash.controller';

import { ConfigModule } from '@nestjs/config';

@Module({

imports: [ConfigModule],

providers: [BkashService],

controllers: [BkashController],

exports: [BkashService],

})

export class BkashModule {}

**2. SSLCommerz Integration**

SSLCommerz provides an official Node.js module for integration. Install it using:​

bash

CopyEdit

npm install sslcommerz-lts

**sslcommerz.service.ts**

typescript

CopyEdit

import { Injectable } from '@nestjs/common';

import \* as SSLCommerz from 'sslcommerz-lts';

import { ConfigService } from '@nestjs/config';

@Injectable()

export class SSLCommerzService {

private sslcommerz: SSLCommerz;

constructor(private configService: ConfigService) {

this.sslcommerz = new SSLCommerz({

store\_id: this.configService.get('SSLCOMMERZ\_STORE\_ID'),

store\_password: this.configService.get('SSLCOMMERZ\_STORE\_PASSWORD'),

is\_live: false, // Set to true for production

});

}

async initiateTransaction(data: any) {

return this.sslcommerz.initiateTransaction(data);

}

async validateTransaction(data: any) {

return this.sslcommerz.validateTransaction(data);

}

}

**sslcommerz.controller.ts**

typescript

CopyEdit

import { Controller, Post, Body } from '@nestjs/common';

import { SSLCommerzService } from './sslcommerz.service';

@Controller('sslcommerz')

export class SSLCommerzController {

constructor(private readonly sslcommerzService: SSLCommerzService) {}

@Post('initiate')

async initiate(@Body() transactionData: any) {

return this.sslcommerzService.initiateTransaction(transactionData);

}

@Post('validate')

async validate(@Body() validationData: any) {

return this.sslcommerzService.validateTransaction(validationData);

}

}

**sslcommerz.module.ts**

typescript

CopyEdit

import { Module } from '@nestjs/common';

import { SSLCommerzService } from './sslcommerz.service';

import { SSLCommerzController } from './sslcommerz.controller';

import { ConfigModule } from '@nestjs/config';

@Module({

imports: [ConfigModule],

providers: [SSLCommerzService],

controllers: [SSLCommerzController],

exports: [SSLCommerzService],

})

export class SSLCommerzModule {}

**🔗 Integrating with Other Modules**

To utilize these services in other parts of your application, inject them as providers:​

typescript

CopyEdit

import { Injectable } from '@nestjs/common';

import { BkashService } from '../integrations/bkash/bkash.service';

import { SSLCommerzService } from '../integrations/sslcommerz/sslcommerz.service';

@Injectable()

export class PaymentService {

constructor(

private readonly bkashService: BkashService,

private readonly sslcommerzService: SSLCommerzService,

) {}

// Implement methods to handle payments using bkashService and sslcommerzService

}

**✅ Final Thoughts**

By integrating bKash and SSLCommerz into your NestJS application, you provide your users with trusted and widely-used payment options in Bangladesh. Ensure to handle sensitive data securely and comply with relevant regulations to maintain the integrity and trustworthiness of your application.​

If you need further assistance or have more questions, feel free to ask!