Stripe Api

```
const dotenv = require('dotenv');

dotenv.config();
// const stripe = require('stripe')(process.env.STRIPE_TEST)
// const stripe = require('stripe')
('sk_test_51HucwXKAtKJyGZzfQQlQmtv8zGBdAkdG1EW7K4euboiSRu4xCnFr90vVOCoFTjB1
7QmPHPBQiidIkGichwoBgifY00gTcCmvnP');
const stripe = require('stripe')(process.env.STRIPE_TEST);
const resPattern = require('../helpers/resPattern');
const httpStatus = require('http-status');
const APIError = require('../helpers/APIError');
const db = require('../server')
const PlanColl = db.collection('plan');
const userColl = db.collection('user')
const subColl = db.collection('subscription')
const clientProgramColl = db.collection('clientProgram')
const { ObjectID } = require('mongodb').ObjectID;
const newsubColl = db.collection('newSubscriptioon')
const query = require('../query/query')
const moment = require('moment')

// create user using API
let createCustomer = async (req, res, next) => {
try {
let data = req.body;

let customer = await stripe.customers.create({

name: data.name,
email: data.email,
address: {
line1: data.address.line1,
postal_code: data.address.postal_code,
city: data.address.city,
country: data.address.country
}
});
const obj = resPattern.successPattern(httpStatus.OK, customer, `success`);
return res.status(obj.code).json({
...obj
});
} catch (e) {
return next(new APIError(`${e.message}`, httpStatus.BAD_REQUEST, true));
}
}

// create User function
const create_customer = async (user) => {
const customer = await stripe.customers.create(
{
```

```
name: user.name,
email: user.emailAddress,
phone: user.phonenumber,
// source: 'tok_mastercard',
description: user.name,
}
);

return customer;
};


let createSubscription = async (req, res, next) => {
try {
let data = req.body

try {
await stripe.paymentMethods.attach(data.paymentMethodId, {
customer: data.customerId,
});
} catch (error) {
return res.status('402').send({ error: { message: error.message } });
}
// Change the default invoice settings on the customer to the new payment method
await stripe.customers.update(
data.customerId,
{
invoice_settings: {
default_payment_method: data.paymentMethodId,
},
}
);
// Create the subscription

const subscription = await stripe.subscriptions.create({
customer: data.customerId,
items: [{ price: data.priceId }],
expand: ['latest_invoice.payment_intent'],
});

let insData = {
subscription,
isCancel: false,
}

const insertData = await query.insert(subColl, insData);
const obj = resPattern.successPattern(httpStatus.OK, insertData.ops[0], `success`);
return res.status(obj.code).json({
...obj
});
} catch (e) {
return next(new APIError(`${e.message}`, httpStatus.BAD_REQUEST, true));
```

```
}

}

const cancelSubscription = async (req, res, next) => {
try {
const findSubscription = await stripe.subscriptions.retrieve(
req.body.subscriptionId
)

await stripe.subscriptions.update(
findSubscription.id,
{ cancel_at_period_end: true }
);

let endDate = moment.unix(findSubscription.current_period_end).format("YYYY-MM-
DDThh:mm:ss ")
let subUpdate = await query.findOneAndUpdate(userColl, { subscriptionId: findSubscription.id }, {
$set: { subscriptionExpiryDate: endDate, subscriptionExpired: true } }, { returnOriginal: false })
const obj = resPattern.successPattern(httpStatus.OK, subUpdate, `success`);
return res.status(obj.code).json({
...obj
});
} catch (e) {
return next(new APIError(`${e.message}`, httpStatus.BAD_REQUEST, true));
}
}

const updateScbscription = async (req, res, next) => {
try {

const update_sub = await stripe.subscriptions.update(
req.body.subId,
{
cancel_at_period_end: true,
}
);

// const added_day = moment.unix(update_sub.cancel_at).utc().add(1, 'days');
// const added_day_unix = moment.utc(added_day).unix();

const subscription = await stripe.subscriptionSchedules.create({
customer: req.body.customerId,
start_date: update_sub.cancel_at, //added_day_unix,
end_behavior: 'release',
phases: [
{
items: [{ price: req.body.priceId }],
}],
default_settings: {
billing_cycle_anchor: 'phase_start',
```

```
}
});

await query.findOneAndUpdate(subColl, { 'subscription.id': req.body.subId }, { $set: { isCancel:
true, cancel_at: update_sub.cancel_at } }, { returnOriginal: false })

let allData = {
cancelSubscription: update_sub,
updatedSubscription: subscription
}

const newData = await query.insert(newsubColl, allData)

const obj = resPattern.successPattern(httpStatus.OK, newData, `success`);
return res.status(obj.code).json({
...obj
});
} catch (e) {
console.log(e)
return next(new APIError(`${e.message}`, httpStatus.BAD_REQUEST, true));
}
}

const testWebhook = async (req, res, next) => {
const event = req.body;
const newSUbscription = await stripe.subscriptions.retrieve(
event.data.object.subscription
);

const newData = await query.insert(subColl, newSUbscription)
const obj = resPattern.successPattern(httpStatus.OK, newData, `success`);
return res.status(obj.code).json({
...obj
});
}

const listAllProduct = async (req, res, next) => {
const products = await stripe.products.list();
const obj = resPattern.successPattern(httpStatus.OK, products, `success`);
return res.status(obj.code).json({
...obj
});
}

let createProduct = async (req, res, next) => {
try {

let planName = req.body.planName;
let currencyType = req.body.currencyType;
let price = req.body.price;
let interval_count = req.body.intervalCount;
let duration = req.body.duration;
```

```
let discription = req.body.discription;

const product = await stripe.products.create({
name: planName,
});

let unitprice = price + '00';
const priceData = await stripe.prices.create({
unit_amount: unitprice,
currency: currencyType,
recurring: { interval: duration, interval_count: interval_count },
product: product.id,
});

let addProduct = {
planName: planName,
stripePriceId: priceData.id,
currency: price + " " + currencyType,
duration: interval_count + " " + duration,
discription: discription,
productId: product.id,
bestChoice: false,
isDisable: false
}

const planTblProduct = await query.insert(PlanColl, addProduct)
const obj = resPattern.successPattern(httpStatus.OK, planTblProduct.ops, `success`);
return res.status(obj.code).json({
...obj
});

} catch (e) {
return next(new APIError(`${e.message}`, httpStatus.BAD_REQUEST, true));
}
}

const update_plan = async (req, res, next) => {
try {
let reqData = req.body;
const product = await stripe.products.update(reqData.stripe_product.id, {
name: reqData.title,
description: reqData.content,
});
console.log("product", product);

let userData = {
title: reqData.title,
content: reqData.content,
stripe_product: reqData.stripe_product,
trial_days: reqData.trial_days,
}
```

```
const result = await query.findOneAndUpdate(PlanColl, { _id: reqData.plan_id }, { $set:
userData }, { returnOriginal: false })

const obj = resPattern.successPattern(httpStatus.OK, "Plan Updated !", `success`);
return res.status(obj.code).json({
...obj
});
// const result = await Plan.findByIdAndUpdate(reqData.plan_id, {
// $set: {
// title: reqData.title,
// content: reqData.content,
// stripe_product: reqData.stripe_product,
// trial_days: reqData.trial_days,
// }
// });
// req.body = result;
// await fetch_plans(req, res, next);
} catch (e) {
return next(new APIError(e.message, httpStatus.BAD_REQUEST, true));
}
};

const disableProduct = async (req, res, next) => {
try {
let priceId = req.body.priceId
let productId = req.body.productId

// archive product

await stripe.products.update(
productId,
{ active: false }
);

let subscriptionData = await query.find(subColl, { 'subscription.plan.id': priceId })

await subscriptionData.map(async subId => {
try {
let retriveId = subId.subscription.id

const subRetriveData = await stripe.subscriptions.retrieve(retriveId);

let convertDate = moment.unix(subRetriveData.current_period_end).format("YYYY-MM-
DDThh:mm:ss")

await stripe.subscriptions.update(
subRetriveData.id,
{
cancel_at_period_end: true,
}
);
```

```
await query.updateMany(userColl, { subscriptionId: retriveId },
{
$set: {
subscriptionExpiryDate: convertDate,
isPlanCancle: true
}
})
} catch (e) {
return next(new APIError(`${e.message}`, httpStatus.BAD_REQUEST, true));
}
})

const obj = resPattern.successPattern(httpStatus.OK, 'Product Disable Successfully !! ', `success`);
return res.status(obj.code).json({
...obj
});

} catch (e) {
return next(new APIError(`${e.message}`, httpStatus.BAD_REQUEST, true));
}
}

const customersubscription = async (req, res, next) => {
try {
let data = req.body

try {
await stripe.paymentMethods.attach(data.paymentMethodId, {
customer: data.customerId,
});
} catch (error) {
return res.status('402').send({ error: { message: error.message } });
}
// Change the default invoice settings on the customer to the new payment method
await stripe.customers.update(
data.customerId,
{
invoice_settings: {
default_payment_method: data.paymentMethodId,
},
}
);
// Create the subscription

const subscription = await stripe.subscriptions.create({
customer: data.customerId,
items: [{ price: data.priceId }],
cancel_at_period_end: true,
// expand: ['latest_invoice.payment_intent'],
});
```

```
const startdate = moment(subscription.current_period_start, 'X').format('YYYY-MM-
DDThh:mm:ss')
const enddate = moment(subscription.cancel_at, 'X').format('YYYY-MM-DDThh:mm:ss')

const programdata = await query.findOneAndUpdate(clientProgramColl, { _id:
ObjectID(data.programId) }, { $set: { subStartDate: startdate, subEndDate: enddate, OPstatus: false
} }, { returnOriginal: false })
const obj = resPattern.successPattern(httpStatus.OK, programdata, `success`);
return res.status(obj.code).json({
...obj
});

} catch (e) {
return next(new APIError(`${e.message}`, httpStatus.BAD_REQUEST, true));
}
}

module.exports = {
createCustomer,
create_customer,
createSubscription,
cancelSubscription,
updateScbscription,
testWebhook,
listAllProduct,
createProduct,
update_plan,
disableProduct,
customersubscription
}
```