# Stripe Node Starter

Démarrez rapidement un SaaS avec Node.js et Stripe.

Vous pouvez:

- soit partir de ce projet tel qu'il est pour créer votre propre projet, il fonctionne directement tel quel avec son propre système d'utilisateur et une base MongoDB.
- soit récupérer et adapter la partie stripe pour l'adapter à votre propre base de données et gestion des utilisateurs.

## getting started

Pour que la démo fonctionne, vous devez avoir préalablement:

- crée un compte Stripe et l'avoir configuré entièrement (voir section "Configurer Stripe en détails")
- Avoir créer une base de données mongodb. Vous pouvez par exemple créer une base de donnée en ligne sur MongoDB Atlas

### installation du code de la démo

```
# installer le serveur node / express
npm install
# éditez les variables d'environnements !
cp .env.example .env
# démarrer l'API
npm run dev


# installer le front-end d'exemple en Vue.js + Tailwind
cd front
npm install
# éditez les variables d'environnements!
cp .env.example .env.local
# démarrer le front-end
npm run dev
```

### Fonctionnement global

Le starter crée principalement 5 routes d'API REST exploitables par n'importe quel front-end (le dossier `front` contient un exemple de client en Vue.js):

- `/api/stripe/plans`: récupére la liste des plans tarifaires à afficher, dont les ids sont spécifiés par la variable d'env `STRIPE_PRICE_IDS`
- `/api/stripe/create-checkout-session`: Commencer un nouveau processus d'achat en créant une nouvelle session Stripe.
- `/api/stripe/webhooks`: Stripe appelera ce endpoint à la fin d'un paiement, moment idéal pour mettre à jour votre propre base de données avec les données renvoyées par Stripe.

- `/api/stripe/create-customer-portal-session` : Ce endpoint créer une url unique qui permettra au client de gérer ses factures, abonnements, moyens de paiement sur le (portail client) [https://stripe.com/docs/billing/subscriptions/customer-portal] de Stripe

- `/api/userinfo` : si l'utilisateur est connecté, ce endpoint renvoie toutes les données de la table utilisateur, dont les données complètes concernant son abonnement.

### Worflow entre front-end et back-end

Pour comprendre l'interfaction entre front-end et back-end, regardez le schéma dans le dossier `./docs/schema-worfklow.pdf` inclus dans ce starter.

### Comment adapter le code à un projet existant

Si vous avez votre propre base de données et votre propre système de gestion des utilisateurs, vous pouvez tout de même ajouter rapidement la gestion des paiements en récupérant le dossier `stripe` de ce starter. Sur votre projet:

- Installer stripe `npm install stripe`
- Ajouter les variables d'environnement du fichier `.env.example` dont le nom commence par `STRIPE_`
- Modifier la config du fichier `src/stripe/config.js` à votre guise.
- Récupérer ou recréez les routes présentes du fichier `src/stripe/routes.js`
- Modifier le fichier `src/stripe/adapter.js` pour mettre à jour votre base de données locales avec du code personnalisé aux bons endroits.

**Le fichier `./src/stripe/adapter.js`**

Il contient des fonctions qui sont appelés automatiquement par les controllers: c'est l'endroit pour mettre votre code personnalisé qui va faire la glue entre Stripe et votre base de données.

1. **`onCreateCheckoutSession()`** (appelé par `src/stripe/controllers/create-checkout-session`):

Un nouvel achat Stripe commence: vous devez passer à Stripe deux variables au moins:

```
// l'id de l'utilisateur qui est en train de s'abonner
checkoutConfig.client_reference_id = currentUser().id;
```

```
// L'id client de votre utilisateur chez Stripe,
// si et UNIQUEMENT SI votre utilisateur a déjà fait un achat précédemment
if (currentUser().customerId) {
  checkoutConfig.customer = customerId;
}
```

2. **`onWehbooks()`** (appelé par `src/stripe/controllers/webhooks`):

Appelé lors des évènements Stripe (quand vous aurez configuré un webhook sur votre Stripe), tel qu'un abonnement acheté avec succès.

On va surtout être intéressé par l'évènement `checkout.session.completed` qui est appelé quand le processus d'abonnement se termine avec succès.

`event.data.object` contiendra la clef `client_reference_id` qui est l'id de votre utilisateur local que vous avez envoyé précédemment.

Cela va permettre de mettre à jour votre base de données, il faudra persister a minima deux données:

- **l'id client** (customer) de l'utilisateur chez Stripe (indispensable pour accéder plus tard à son Portail Client)
- l'id de l'abonnement (subscription) auquel il vient souscrire (indispensable pour pouvoir retrouver les données de son abonnement, tel que le status, le prix, la durée etc)

Vous pouvez persister d'autres données selon vos besoins.

```
if (event.type === "checkout.session.completed") {
  const session = event.data.object;
  // console.log(JSON.stringify(session, 0, 2));

  await db()
    .collection("users")
    .updateOne(
      { _id: oid(session.client_reference_id) },
      {
        $set: {
          stripeCustomerId: session.customer, // set by stripe
          stripeSubscriptionId: session.subscription, // set by Stripe
        },
      }
    );
}
```

3. **onCreateCustomerPortalSession()** (appelé par `src/stripe/controllers/create-customer-portal-session`):

Ici vous devez simplement retourner l'id client Stripe depuis votre base de données, qui va permettre à Stripe de générer un lien d'accès au portal client que pourra utiliser votre front-end. Exemple:

```
  async onCreateCustomerPortalSession({ req }) {
    const fullUser = await db()
      .collection("users")
      .findOne({ _id: oid(req.user.id) });
    customerId = fullUser.stripeCustomerId;
    return customerId;
  },
```

## Configurer Stripe

1 - Créer un compte sur Stripe

2 - Se rendre sur https://dashboard.stripe.com/test/products pour ajouter des abonnements et leur tarifs sur l'environnement de test. Exemple:

- Un produit "Basic" avec:
  - un plan tarifaire mensuel à 9€/mois
  - un plan tarifaire annuel à 90€/an
- Un produit premium avec:
  - un plan mensuel à 19€/mois
  - un plan annuel à €190/an

3 - Récupérer vos clefs d'API

La "clé publique" sera pour votre front-end (variable `VUE_APP_STRIPE_PUBLISHABLE_KEY`) et la clef secrète pour votre serveur node: `STRIPE_SECRET_KEY`.

4 - Configurer un webhook:

Quand une commande Stripe est terminée, on a besoin d'avertir notre serveur. Pour cela il faut créer un webhook: https://dashboard.stripe.com/test/webhooks, qui devra taper sur l'url suivante "http://{{votre-api}}/stripe/webhooks" de votre API Node.

Le starter crée automatiquement un tunnel vers votre localhost pour vous avec `localtunnel`. https://node-stripe-starter-600308910def.loca.lt/api/stripe/webhooks que vous pouvez utilisez à cet effet.

5 - Configurer votre Customer Portal

Vous devez configurer sur cette page d'administration les plans qui apparaitront sur le portail Client: (il sera possible d'upgrader / downgrader un abonnement vers les plans sélectionnés)

https://dashboard.stripe.com/test/settings/billing/portal

## Erreurs fréquentes

### StripeSignatureVerificationError

```
StripeSignatureVerificationError: No signatures found matching the expected
signature for payload. Are you passing the raw request body you received from
Stripe? https://github.com/stripe/stripe-node#webhook-signing
```

Deux possibilités:

1. Vous n'avez pas entré comme il faut votre "STRIPE_WEBHOOK_SECRET" dans les variables d'environnement et donc la vérification de la signature échoue.

Solution: se rendre sur https://dashboard.stripe.com/test/webhooks et vérifier que le webhook secret renseigné par STRIPE_WEBHOOK_SECRET est le bon

2. La requête http POST vers `/api/stripe/webhooks` est modifiée, par exemple par un middleware express qui va transformer automatiquement le corps en JSON avec la fonctin JSON.parse(): cela fait échouer le contrôle de la signature.

Solution: si vous utilisez Express, assurez vous que le middleware JSON n'est pas appliqué à votre route Stripe:

```
app.use((req, res, next) => {
  if (req.originalUrl === "/api/stripe/webhooks") {
    bodyParser.raw({ type: "*/*" })(req, res, next);
  } else {
    bodyParser.json({ limit: "5mb" })(req, res, next);
  }
});
```

### Mon endpoint "/api/stripe/webhooks" ne semble pas appelée par Stripe

1. Vérifier que votre webhook est bien appelé: se rendre sur https://dashboard.stripe.com/test/webhooks et vérifier que l'url renseignée renvoie bien vers une url valide de votre application.

2. Vérifier que ce webhook est correctement configuré. On peut choisir quels évènements sont envoyés: vérifier que vous n'avez pas fait d'erreurs sur la liste d'évènements envoyés. A minima: `checkout.session.completed`

## Checklist globale

- [ ] J'ai bien installé le code (npm install)
- [ ] Stripe: J'ai bien ajouté les produits et leurs plans tarifaires
- [ ] Stripe: J'ai correctement configurer mon webhook
- [ ] J'ai bien configuré le customer Portal: https://dashboard.stripe.com/test/settings/billing/portal
- [ ] J'ai bien ouvert un tunnel en local vers https://{mon-api.com}/api/stripe/webhooks
- [ ] J'ai bien renseignée *TOUTES* les variables d'environnement correctement côté serveur ET côté front-end
- [ ] J'ai adapté le code sur fichier `./src/stripe/adapter.js` selon mes besoins

## Support

Pour éviter tout blocage, vous pouvez profiter du support par tchat à cette adresse: https://discord.gg/6k5tKt2v6B