```
import { router, userProcedure } from '@/app/api/trpc/trpc-router';
import { generateApiKey } from '@/shared/utils/helpers';
import { User } from '@supabase/supabase-js';
import { TRPCError } from '@trpc/server';
import { z } from 'zod';
import { stripe } from '@/shared/utils/stripe/config';
import Stripe from 'stripe';
import { createOrRetrieveStripeCustomer } from '@/shared/utils/supabase/admin';

const apiKeyRouter = router({
  // api key page
  getApiKeys: userProcedure.query(async ({ ctx }) => {
    const user = ctx.session.data.session?.user as User;
    const apiKeys = await ctx.supabase
      .from('swarms_cloud_api_keys')
      .select('id, name, is_deleted, created_at, key')
      .eq('user_id', user.id)
      .or(`is_deleted.eq.false, is_deleted.is.null`)
      .order('created_at', { ascending: false });

    return apiKeys.data?.map((row) => ({
      ...row,
      key: `${row?.key?.slice(0, 5)}.....${row?.key?.slice(-5)}`,
    }));
  }),
  addApiKey: userProcedure
```

```
.input(z.object({ name: z.string() }))
.mutation(async ({ ctx, input }) => {
  const user = ctx.session.data.session?.user as User;
  const name = input.name.trim();
  if (name === '') {
    throw new TRPCError({
      code: 'BAD_REQUEST',
      message: 'Name is required',
    });
  }

  if (name === 'playground') {
    // error
    throw new TRPCError({
      code: 'BAD_REQUEST',
      message: 'cannot create api key with name "playground"',
    });
  }

  const stripeCustomerId = await createOrRetrieveStripeCustomer({
    email: user.email ?? '',
    uuid: user.id,
  });

  if (!stripeCustomerId) {
    throw new TRPCError({
```

```
        code: 'INTERNAL_SERVER_ERROR',
        message: 'Error while creating stripe customer',
      });
    }

    const paymentMethods = await stripe.paymentMethods.list({
      customer: stripeCustomerId,
      type: 'card',
    });

    if (!paymentMethods.data.length) {
      throw new TRPCError({
        code: 'NOT_FOUND',
        message:
            'Payment method missing. Add valid card to continue and click on added card to set as default',
      });
    }

    try {
      const key = generateApiKey();
      const newApiKey = await ctx.supabase
        .from('swarms_cloud_api_keys')
        .insert({ name: name, key, user_id: user.id });
      if (!newApiKey.error) {
        return {
```

```
          key,
          name,
        };
      }
    } catch (e) {
      throw new TRPCError({
        code: 'INTERNAL_SERVER_ERROR',
        message: 'Error while adding new api key',
      });
    }
  }),
deleteApiKey: userProcedure
  .input(z.string())
  .mutation(async ({ ctx, input }) => {
    const user = ctx.session.data.session?.user as User;
    const apiKey = await ctx.supabase
      .from('swarms_cloud_api_keys')
      .select('*')
      .eq('id', input)
      .eq('user_id', user.id)
      .single();
    // dont allow delete 'playground' api key
    if (apiKey.error) {
      throw new TRPCError({
        code: 'BAD_REQUEST',
        message: 'Invalid api key',
```

```
      });
    }

    if (apiKey.data.name === 'playground') {
      throw new TRPCError({
        code: 'BAD_REQUEST',
        message: 'Cannot delete playground api key',
      });
    }
    const updatedApiKey = await ctx.supabase
      .from('swarms_cloud_api_keys')
      .update({ is_deleted: true })
      .eq('id', input)
      .eq('user_id', user.id);
    if (!updatedApiKey.error) {
      return true;
    }
    throw new TRPCError({
      code: 'INTERNAL_SERVER_ERROR',
      message: 'Error while deleting api key',
    });
  }),
});

export default apiKeyRouter;
```