```
import {
  publicProcedure,
  router,
  userProcedure,
} from '@/app/api/trpc/trpc-router';
import { TRPCError } from '@trpc/server';
import { z } from 'zod';



const explorerOptionsRouter = router({
  addComment: userProcedure
    .input(
      z.object({
        modelId: z.string(),
        modelType: z.string(),
        content: z.string().min(1),
      }),
    )
    .mutation(async ({ input, ctx }) => {
      const { modelId, modelType, content } = input;

      const user_id = ctx.session.data.session?.user?.id ?? '';
      const lastSubmits = await ctx.supabase
        .from('swarms_cloud_comments')
        .select('*')
        .eq('user_id', user_id)
  .order('created_at', { ascending: false })
  .limit(1);

if ((lastSubmits?.data ?? [])?.length > 0) {
  const lastSubmit = lastSubmits.data?.[0] || { created_at: new Date() };
  const lastSubmitTime = new Date(lastSubmit.created_at);
  const currentTime = new Date();
  const diff = currentTime.getTime() - lastSubmitTime.getTime();
  const diffMinutes = diff / (1000 * 20); // 20 secs
  if (diffMinutes < 1) {
    throw 'You can only submit one comment per 20 secs';
  }
}

try {
  const { error } = await ctx.supabase
    .from('swarms_cloud_comments')
    .insert([
      {
        model_id: modelId,
        model_type: modelType,
        content,
        user_id,
      },
    ]);
```

```
      if (error) {
        throw new TRPCError({
          code: 'INTERNAL_SERVER_ERROR',
          message: 'Error while adding comment',
        });
      }

      return true;
    } catch (error) {
      throw new TRPCError({
        code: 'INTERNAL_SERVER_ERROR',
        message: 'Failed to add comment',
      });
    }
  }),

editComment: userProcedure
  .input(
    z.object({
      commentId: z.string(),
      content: z.string().min(1),
    }),
  )
  .mutation(async ({ input, ctx }) => {
    const { commentId, content } = input;
const user_id = ctx.session.data.session?.user?.id ?? '';
try {
  const comment = await ctx.supabase
    .from('swarms_cloud_comments')
    .update({
      content,
      is_edited: true,
      updated_at: new Date() as unknown as string,
    })
    .eq('user_id', user_id)
    .eq('id', commentId)
    .select('*');

  if (comment.error) {
    throw new TRPCError({
      code: 'INTERNAL_SERVER_ERROR',
      message: 'Error while editing comment',
    });
  }

  return true;
} catch (error) {
  console.error(error);
  throw new TRPCError({
    code: 'INTERNAL_SERVER_ERROR',
    message: 'Failed to edit comment',
      });
    }
  }),

getComments: publicProcedure
  .input(
    z.object({
      limit: z.number().default(2),
      offset: z.number().default(0),
      modelId: z.string(),
    }),
  )
  .query(async ({ input, ctx }) => {
    const { limit, offset, modelId } = input;

    try {
      const { count: totalCommentsCount, error: countError } =
        await ctx.supabase
          .from('swarms_cloud_comments')
          .select('id', { count: 'exact' })
          .eq('model_id', modelId);

      if (countError) {
        throw new TRPCError({
          code: 'INTERNAL_SERVER_ERROR',
          message: 'Error while fetching comments count',
    });
  }

  const { data: comments, error } = await ctx.supabase
    .from('swarms_cloud_comments')
    .select(
      `
        id,
        user_id,
        model_id,
        model_type,
        is_edited,
        content,
        updated_at,
        created_at,
          swarms_cloud_comments_replies(id, content, user_id, updated_at, is_edited, created_at,
comment_id, users(full_name, username, avatar_url)),
        users (
          full_name,
          username,
          avatar_url
        )
      `,
    )
    .eq('model_id', modelId)
    .order('created_at', { ascending: true })
```

```
        .range(offset, offset + limit - 1);


      if (error) {
        throw new TRPCError({
          code: 'INTERNAL_SERVER_ERROR',
          message: 'Error while fetching comments',
        });
      }


      return {
        comments,
        count: totalCommentsCount || 0,
      };
    } catch (error) {
      console.error(error);
      throw new TRPCError({
        code: 'INTERNAL_SERVER_ERROR',
        message: 'Failed to fetch comments',
      });
    }
  }),


deleteComment: userProcedure
  .input(z.string())
  .mutation(async ({ input, ctx }) => {
    const commentId = input;
    const user_id = ctx.session.data.session?.user?.id ?? '';


    try {
      const { error } = await ctx.supabase
        .from('swarms_cloud_comments')
        .delete()
        .eq('user_id', user_id)
        .eq('id', commentId);


      if (error) {
        throw new TRPCError({
          code: 'INTERNAL_SERVER_ERROR',
          message: 'Error while deleting comment',
        });
      }


      return true;
    } catch (error) {
      console.error(error);
      throw new TRPCError({
        code: 'INTERNAL_SERVER_ERROR',
        message: 'Failed to delete comment',
      });
    }
  }),
likeItem: userProcedure
  .input(
    z.object({
      itemId: z.string(),
      itemType: z.enum(['comment', 'reply']),
    }),
  )
  .mutation(async ({ input, ctx }) => {
    const { itemId, itemType } = input;

    const user_id = ctx.session.data.session?.user?.id ?? '';
    try {
      const { data, error } = await ctx.supabase
        .from('swarms_cloud_likes')
        .insert([{ item_id: itemId, item_type: itemType, user_id }]);

      if (error) {
        console.log({ error });
        if (error.code === '23505') {
          throw new TRPCError({
            code: 'INTERNAL_SERVER_ERROR',
            message: 'Careful there, too many requests at once',
          });
        }

        throw new TRPCError({
          code: 'INTERNAL_SERVER_ERROR',
          message: 'Error while liking item',
        });
      }

      return data;
    } catch (error) {
      console.error(error);
      throw new TRPCError({
        code: 'INTERNAL_SERVER_ERROR',
        message: 'Failed to like item',
      });
    }
  }),

unlikeItem: userProcedure
  .input(
    z.object({
      itemId: z.string(),
      itemType: z.enum(['comment', 'reply']),
    }),
  )
  .mutation(async ({ input, ctx }) => {
    const { itemId, itemType } = input;

    const user_id = ctx.session.data.session?.user?.id ?? '';
```

```
try {
  const { error } = await ctx.supabase
    .from('swarms_cloud_likes')
    .delete()
    .eq('item_id', itemId)
    .eq('item_type', itemType)
    .eq('user_id', user_id);

  if (error) {
    if (error.code === '23505') {
      throw new TRPCError({
        code: 'INTERNAL_SERVER_ERROR',
        message: 'Careful there, too many requests at once',
      });
    }

    throw new TRPCError({
      code: 'INTERNAL_SERVER_ERROR',
      message: 'Error while unliking item',
    });
  }

  return true;
} catch (error) {
  console.error(error);
  throw new TRPCError({
        code: 'INTERNAL_SERVER_ERROR',
        message: 'Failed to unlike item',
      });
    }
  }),

getLikes: publicProcedure
  .input(
    z.object({
      itemIds: z.array(z.string()),
      itemType: z.enum(['comment', 'reply']),
      userId: z.string().optional(),
    }),
  )
  .query(async ({ input, ctx }) => {
    const { itemIds, itemType, userId } = input;

    try {
      // Fetch all likes for the given item IDs and type
      const { data: likes, error } = await ctx.supabase
        .from('swarms_cloud_likes')
        .select('item_id')
        .in('item_id', itemIds)
        .eq('item_type', itemType);

      if (error) {
```

```
  throw new TRPCError({
    code: 'INTERNAL_SERVER_ERROR',
    message: `Error while fetching ${itemType} likes`,
  });
}

// Count likes for each item ID
const likeCounts = likes.reduce((acc: Record<string, number>, like) => {
  acc[like.item_id] = (acc[like.item_id] || 0) + 1;
  return acc;
}, {});

// Fetch likes for the specific user
let userLikes: { item_id: string }[] = [];
let userLikesError: any;

if (userId) {
  const response = await ctx.supabase
    .from('swarms_cloud_likes')
    .select('item_id')
    .in('item_id', itemIds)
    .eq('item_type', itemType)
    .eq('user_id', userId);

  userLikes = response.data || [];
  userLikesError = response.error;
```

```
      }

      if (userLikesError) {
        throw new TRPCError({
          code: 'INTERNAL_SERVER_ERROR',
          message: `Error while fetching user ${itemType} likes`,
        });
      }

      return {
        likeCounts,
        userLikes: userLikes.map((like) => like.item_id),
      };
    } catch (error) {
      console.error(error);
      throw new TRPCError({
        code: 'INTERNAL_SERVER_ERROR',
        message: `Failed to fetch ${itemType} likes`,
      });
    }
  }),

addReply: userProcedure
  .input(
    z.object({
      commentId: z.string(),
    content: z.string().min(1),
  }),
)
.mutation(async ({ input, ctx }) => {
  const { commentId, content } = input;

  const user_id = ctx.session.data.session?.user?.id ?? '';
  try {
    const { data, error } = await ctx.supabase
      .from('swarms_cloud_comments_replies')
      .insert([{ comment_id: commentId, content, user_id }]);

    if (error) {
      throw new TRPCError({
        code: 'INTERNAL_SERVER_ERROR',
        message: 'Error while adding reply',
      });
    }

    return data;
  } catch (error) {
    console.error(error);
    throw new TRPCError({
      code: 'INTERNAL_SERVER_ERROR',
      message: 'Failed to add reply',
    });
    }
  }),

editReply: userProcedure
  .input(
    z.object({
      replyId: z.string(),
      content: z.string().min(1),
    }),
  )
  .mutation(async ({ input, ctx }) => {
    const { replyId, content } = input;

    const user_id = ctx.session.data.session?.user?.id ?? '';
    try {
      const reply = await ctx.supabase
        .from('swarms_cloud_comments_replies')
        .update({
          content,
          is_edited: true,
          updated_at: new Date() as unknown as string,
        })
        .eq('user_id', user_id)
        .eq('id', replyId)
        .select('*');
      if (reply.error) {
        throw new TRPCError({
          code: 'INTERNAL_SERVER_ERROR',
          message: 'Error while editing reply',
        });
      }

      return true;
    } catch (error) {
      console.error(error);
      throw new TRPCError({
        code: 'INTERNAL_SERVER_ERROR',
        message: 'Failed to edit reply',
      });
    }
  }),

deleteReply: publicProcedure
  .input(z.string())
  .mutation(async ({ input, ctx }) => {
    const replyId = input;

    const user_id = ctx.session.data.session?.user?.id ?? '';

    try {
      const { error } = await ctx.supabase
```

```
        .from('swarms_cloud_comments_replies')
        .delete()
        .eq('id', replyId)
        .eq('user_id', user_id);

      if (error) {
        throw new TRPCError({
          code: 'INTERNAL_SERVER_ERROR',
          message: 'Error while deleting reply',
        });
      }

      return { success: true };
    } catch (error) {
      console.error(error);
      throw new TRPCError({
        code: 'INTERNAL_SERVER_ERROR',
        message: 'Failed to delete reply',
      });
    }
  }),

getReplies: publicProcedure
  .input(
    z.object({
      limit: z.number().default(6),
      offset: z.number().default(1),
      commentId: z.string(),
    }),
  )
  .query(async ({ input, ctx }) => {
    const { commentId, limit, offset } = input;

    try {
      const { data: replies, error } = await ctx.supabase
        .from('swarms_cloud_comments_replies')
        .select(
          `
            id,
            comment_id,
            user_id,
            content,
            is_edited,
            created_at,
            updated_at,
            users (
              full_name,
              username,
              email,
              avatar_url
            )
          `,
     )
     .eq('comment_id', commentId)
     .order('created_at', { ascending: true })
     .range(offset, offset + limit - 1);

    if (error) {
     throw new TRPCError({
      code: 'INTERNAL_SERVER_ERROR',
      message: 'Error while fetching replies',
     });
    }

    return {
     replies,
     count: replies?.length ? replies.length : 0,
    };
   } catch (error) {
    console.error(error);
    throw new TRPCError({
     code: 'INTERNAL_SERVER_ERROR',
     message: 'Failed to fetch replies',
    });
   }
  }),
});
```

```
export default explorerOptionsRouter;
```