```
import { AuthApiGuard } from '@/shared/utils/api/auth-guard';
import { supabaseAdmin } from '@/shared/utils/supabase/admin';
import { NextApiRequest, NextApiResponse } from 'next';
import { z } from 'zod';

//schema
const promptSchema = z.object({
  name: z.string().min(2, { message: 'Name should be at least 2 characters' }),
  prompt: z
    .string()
    .min(5, { message: 'Prompt should be at least 5 characters' }),
  description: z.string().optional(),
  useCases: z
    .array(
      z.object({
        title: z.string().min(1, { message: 'Use case title is required' }),
        description: z
          .string()
          .min(1, { message: 'Use case description is required' }),
      }),
    )
    .min(1, { message: 'At least one use case is required' }),
  tags: z.string().min(2, {
    message: 'Tags should be at least 1 characters and separated by commas',
  }),
});
```

```
const addPrompt = async (req: NextApiRequest, res: NextApiResponse) => {
  if (req.method !== 'POST') {
    res.setHeader('Allow', ['POST']);
    return res.status(405).end(`Method ${req.method} Not Allowed`);
  }

  try {
    const apiKey = req.headers.authorization?.split(' ')[1];
    if (!apiKey) {
      return res.status(401).json({ error: 'API Key is missing' });
    }

    const guard = new AuthApiGuard({ apiKey });
    const isAuthenticated = await guard.isAuthenticated();
    if (isAuthenticated.status !== 200) {
      return res
        .status(isAuthenticated.status)
        .json({ error: isAuthenticated.message });
    }

    const user_id = guard.getUserId();
    if (!user_id) {
      return res.status(404).json({ error: 'User is missing' });
    }
```

```
const input = promptSchema.parse(req.body);
const { name, prompt, description, useCases, tags } = input;

// Rate limiting logic
const { data: lastSubmits, error: lastSubmitsError } = await supabaseAdmin
  .from('swarms_cloud_prompts')
  .select('*')
  .eq('user_id', user_id)
  .order('created_at', { ascending: false })
  .limit(1);

if (lastSubmitsError) throw lastSubmitsError;

if (lastSubmits.length > 0) {
  const lastSubmit = lastSubmits[0];
  const lastSubmitTime = new Date(lastSubmit.created_at);
  const currentTime = new Date();
}

//check for existing prompt
const { data: existingPrompts, error: existingPromptsError } =
  await supabaseAdmin
    .from('swarms_cloud_prompts')
    .select('*')
    .eq('prompt', prompt)
    .eq('user_id', user_id);
```

```
if (existingPromptsError) throw existingPromptsError;

if (existingPrompts.length > 0) {
  return res.status(400).json({ error: 'Prompt already exists' });
}

const trimTags = tags
  ?.split(',')
  .map((tag) => tag.trim())
  .filter(Boolean)
  .join(',');

//add new prompt
const { error } = await supabaseAdmin.from('swarms_cloud_prompts').insert([
  {
    name,
    use_cases: useCases,
    prompt,
    description,
    user_id,
    tags: trimTags,
    status: 'pending',
  },
]);
```

```
    if (error) throw error;


    return res.status(200).json({ success: true });
  } catch (e) {
    console.error(e);
    if (e instanceof z.ZodError) {
      return res.status(400).json({ error: e.errors });
    }
    return res.status(500).json({ error: 'Could not add prompt' });
  }
};


export default addPrompt;
```