```
import Decimal from 'decimal.js';
import { supabaseAdmin } from '../supabase/admin';
import { getMonthStartEndDates } from '../helpers';

// User Types
export type DailyCost = {
  date: string;
  totalCost: number;
  invoiceTotalCost?: number;
  model?: {
    [modelId: string]: { tokens: number; requests: number; costs: number };
  };
};

export type UserUsage = {
  totalCost: number;
  invoiceTotalCost: number;
  dailyCosts: DailyCost[];
};

export interface UsageResponse {
  status: number;
  message: string;
  user?: UserUsage;
}
```

```
// Organization Types

type ModelRequestCounts = {
  [modelName: string]: number;
};

interface UserRequestCounts {
  [userId: string]: number;
}

type UserModels = {
  [modelName: string]: number;
};

export type OrganizationUsage = {
  name: string;
  publicId?: string;
  totalReqCount?: number;
  models: ModelRequestCounts;
  users?: {
    email: string;
    totalRequests: number;
    modelsUsed: UserModels;
  }[];
};
```

```
export interface OrganizationResponse extends UsageResponse {
  organization: OrganizationUsage;
}

export async function userAPICluster(
  userId: string,
  month: Date,
): Promise<UsageResponse> {
  try {
    const { start, end } = getMonthStartEndDates(month);

    // Fetch direct user activities (excluding organization activities)
    const { data: userActivities, error: userError } = await supabaseAdmin
      .from('swarms_cloud_api_activities')
      .select(

'invoice_total_cost,total_cost,created_at,input_tokens,output_tokens,request_count,model_id,swarm
s_cloud_models(name)',
      )
      .eq('user_id', userId)
      .is('organization_id', null)
      .gte('created_at', start)
      .lte('created_at', end);

    if (userError) {
      console.error('Error fetching user activities:', userError);
```

```
  return {
    status: 500,
    message: 'Internal server error',
    user: {
      totalCost: 0,
      invoiceTotalCost: 0,
      dailyCosts: [],
    },
  };
}

// Fetch activities for organizations where the user is the owner
const { data: orgActivities, error: orgError } = await supabaseAdmin
  .from('swarms_cloud_api_activities')
  .select(
    `
      invoice_total_cost,
      total_cost,
      created_at,
      model_id,
      input_tokens,
      output_tokens,
      request_count,
      organization_id,
      swarms_cloud_models(name),
      swarms_cloud_organizations!inner(owner_user_id)
```

```
    `,
  )
  .not('organization_id', 'is', null)
  .gte('created_at', start)
  .lte('created_at', end)
  .eq('swarms_cloud_organizations.owner_user_id', userId);

if (orgError) {
  console.error('Error fetching organization activities:', orgError);
  return {
    status: 500,
    message: 'Internal server error',
    user: {
      totalCost: 0,
      invoiceTotalCost: 0,
      dailyCosts: [],
    },
  };
}

// Combine user and organization activities
const allActivities = [...userActivities, ...orgActivities];

const userTotalCost = allActivities.reduce(
  (acc, item) => acc + (item.total_cost || 0),
  0,
```

```
);

const userInvoiceTotalCost = allActivities.reduce(
  (acc, item) => acc + (item.invoice_total_cost || 0),
  0,
);
// Calculate daily costs
const userDailyCosts: DailyCost[] = [];

for (const activity of allActivities) {
  const activityDate = activity.created_at.slice(0, 10);
  const modelName = activity.swarms_cloud_models?.name || '';

  let existingDailyCost = userDailyCosts.find(
    (cost) => cost.date === activityDate,
  );

  if (!existingDailyCost) {
    existingDailyCost = {
      date: activityDate,
      totalCost: 0,
      model: {},
    };
    userDailyCosts.push(existingDailyCost);
  }
```

```
  // Update total costs
  existingDailyCost.totalCost +=
    (activity.total_cost || 0) + (activity.invoice_total_cost || 0);

  // Update model costs, tokens and requests
  if (
    activity.request_count ||
    activity.input_tokens ||
    activity.output_tokens
  ) {
    existingDailyCost.model = existingDailyCost.model || {};
    existingDailyCost.model[modelName] = {
      tokens:
        (existingDailyCost.model[modelName]?.tokens || 0) +
        (activity.input_tokens || 0) +
        (activity.output_tokens || 0),
      requests:
        (existingDailyCost.model[modelName]?.requests || 0) +
        (activity.request_count || 0),
      costs:
        (existingDailyCost.model[modelName]?.costs || 0) +
        (activity.total_cost || 0) +
        (activity.invoice_total_cost || 0),
    };
  }
}
```

```
    return {
      status: 200,
      message: 'Success',
      user: {
        totalCost: Number(userTotalCost),
        invoiceTotalCost: Number(userInvoiceTotalCost),
        dailyCosts: userDailyCosts,
      },
    };
  } catch (error) {
    console.error('Error calculating total monthly usage:', error);

    return {
      status: 500,
      message: 'Internal server error',
      user: { totalCost: 0, invoiceTotalCost: 0, dailyCosts: [] },
    };
  }
}

export async function getReviews(modelId: string) {
  try {
    const { data: reviews, error: reviewsError } = await supabaseAdmin
      .from('swarms_cloud_reviews')
      .select(
```

```
      `
        id,
        comment,
        model_id,
        user_id,
        model_type,
        created_at,
        rating,
        users (
          email,
        )
      `,
    )
    .eq('model_id', modelId)
    .order('created_at', { ascending: false });

  if (reviewsError) {
    return {
      status: 500,
      message: 'Internal server error',
    };
  }

  return reviews;
} catch (error) {
  console.error(error);
```

```
    return {
      status: 500,
      message: 'Internal server error',
    };
  }
}

export async function getOrganizationUsage(
  userId: string,
  month: Date,
): Promise<OrganizationResponse> {
  try {
    const { start, end } = getMonthStartEndDates(month);

    // Check if the user has an organization
    const { data: organizationData, error: organizationError } =
      await supabaseAdmin
        .from('swarms_cloud_organizations')
        .select('id, name, public_id')
        .eq('owner_user_id', userId)
        .single();

    if (organizationError || !organizationData) {
      if (organizationError.code === 'PGRST116') {
        return {
          status: 200,
```

```
      message: '',
      organization: {} as any,
    };
  }
  console.error('Error fetching organization:', organizationError);
  return {
    status: 500,
    message: 'Internal server error',
    organization: {} as any,
  };
}

const organizationId = organizationData.id;

// Fetch models and activities for the organization
const { data: orgActivities, error: orgError } = await supabaseAdmin
  .from('swarms_cloud_api_activities')
  .select(
    `
    user_id,
    request_count,
    model_id,
    swarms_cloud_models(name),
    users(email)
  `,
  )
```

```
    .eq('organization_id', organizationId)
    .gte('created_at', start)
    .lte('created_at', end);

  if (orgError) {
    console.error('Error fetching organization activities:', orgError);
    return {
      status: 500,
      message: 'Internal server error',
      organization: {} as any,
    };
  }

  // Sum up the total request count for the organization
  const totalReqCount = orgActivities.reduce(
    (acc, activity) => acc + activity.request_count,
    0,
  );

  // Aggregate request counts per model
  const modelRequestCounts: ModelRequestCounts = orgActivities.reduce(
    (acc: ModelRequestCounts, activity) => {
      const modelName = activity.swarms_cloud_models?.name || 'unknown';
      acc[modelName] = (acc[modelName] || 0) + activity.request_count;
      return acc;
    },
```

```
  {} as ModelRequestCounts,
);

// Find the user with the highest request count
const userRequestCounts: UserRequestCounts = orgActivities.reduce(
  (acc: UserRequestCounts, activity) => {
    acc[activity.user_id] =
      (acc[activity.user_id] || 0) + activity.request_count;
    return acc;
  },
  {} as UserRequestCounts,
);

const topUserIds = Object.keys(userRequestCounts)
  .sort((a, b) => userRequestCounts[b] - userRequestCounts[a])
  .slice(0, 3);

// Get details of the top 3 users
const topUsers = topUserIds.map((userId) => {
  const userActivities = orgActivities.filter(
    (activity) => activity.user_id === userId,
  );

  const modelsUsed = userActivities.reduce((acc: UserModels, activity) => {
    const modelName = activity.swarms_cloud_models?.name ?? '';
    acc[modelName] = (acc[modelName] || 0) + activity.request_count;
```

```
      return acc;
    }, {});

    const email = userActivities[0]?.users?.email ?? '';
    const totalRequests = userRequestCounts[userId];

    return {
      email,
      totalRequests,
      modelsUsed,
    };
  });

  return {
    status: 200,
    message: 'Success',
    organization: {
      name: organizationData.name ?? '',
      publicId: organizationData.public_id ?? '',
      models: modelRequestCounts,
      totalReqCount,
      users: topUsers,
    },
  };
} catch (error) {
  console.error('Error fetching organization usage:', error);
```

```
    return {
      status: 500,
      message: 'Internal server error',
      organization: {} as any,
    };
  }
}

export async function getBillingLimit(
  userId: string,
  month: Date,
): Promise<UsageResponse> {
  try {
    const { start, end } = getMonthStartEndDates(month);

    // Fetch user details
    const { data: user, error: userError } = await supabaseAdmin
      .from('users')
      .select('credit_plan, credit_limit')
      .eq('id', userId)
      .single();

    if (userError) {
      console.error('Error fetching user details:', userError);
      return {
        status: 404,
```

```
    message: 'User not found',
  };
}

if (user.credit_plan !== 'invoice') {
  return {
    status: 200,
    message: 'User is not on an invoice plan',
  };
}

// Fetch direct user activities (excluding organization activities)
const { data: userActivities, error: userActivitiesError } =
  await supabaseAdmin
    .from('swarms_cloud_api_activities')
    .select('invoice_total_cost, created_at')
    .eq('user_id', userId)
    .is('organization_id', null)
    .gte('created_at', start)
    .lte('created_at', end);

if (userActivitiesError) {
  console.error('Error fetching user activities:', userActivitiesError);
  return {
    status: 500,
    message: 'Internal server error',
```

```
    };
  }

  // Fetch activities for organizations where the user is the owner
  const { data: orgActivities, error: orgActivitiesError } =
    await supabaseAdmin
      .from('swarms_cloud_api_activities')
      .select(
                                                    'invoice_total_cost,       created_at,       organization_id,
swarms_cloud_organizations!inner(owner_user_id)',
      )
      .not('organization_id', 'is', null)
      .gte('created_at', start)
      .lte('created_at', end)
      .eq('swarms_cloud_organizations.owner_user_id', userId);

  if (orgActivitiesError) {
    console.error(
      'Error fetching organization activities:',
      orgActivitiesError,
    );
    return {
      status: 500,
      message: 'Internal server error',
    };
  }
```

```
  // Combine user and organization activities
  const allActivities = [...userActivities, ...orgActivities];

  // Calculate the total invoice cost
  const userInvoiceTotalCost = allActivities.reduce((acc, item) => {
    return acc.plus(new Decimal(item.invoice_total_cost || 0));
  }, new Decimal(0));

  // Check if the total invoice cost exceeds the user's credit limit
  const creditLimit = new Decimal(user.credit_limit);
  const hasExceeded = userInvoiceTotalCost.greaterThanOrEqualTo(creditLimit);

  if (hasExceeded) {
    return {
      status: 403,
      message: 'You have exceeded your billing limit',
    };
  }

  return {
    status: 200,
    message: 'Within credit limit',
  };
} catch (error) {
  console.error('Error calculating total monthly usage:', error);
```

```
    return {
      status: 500,
      message: 'Internal server error',
    };
  }
}
```