Svelte 5 + Hono 🔥

Part 5: company endpoint + upsert company

www.youtube.com/@GetterSethya

constant

```
export const LIMIT = 10
export const SORT = 'created_at'
export const ORDER = 'DESC'
```

```
export type OrderQuery = 'ASC' | 'DESC'
export const idSchema = z.object({ id: z.coerce.number() })
export const queryUrlSchema = z.object({
    q: z.string().default(''),
    page: z.coerce.number().default(1),
    limit: z.coerce.number().default(LIMIT),
    sort: z.string().default(SORT),
    order: z
        .string()
        .default(ORDER)
        .transform((d) => d as OrderQuery),
})
```

CompanyRepository class

```typescript
export class CompanyRepository implements AsyncBaseRepository<CompanyEntity> {
    public async create(args: CreateArgs<CompanyEntity>):
Promise<CompanyEntity> {}
    public async findById(args: FindByIdArgs): Promise<CompanyEntity | null> {}
    public async list(args: ListArgs): Promise<ListReturn<CompanyEntity>> {}
    public async update(args: UpdateArgs<CompanyEntity>):
Promise<CompanyEntity> {}
    public async delete(args: DeleteArgs): Promise<boolean> {}
}
```

CompanyController

```
export const companyController = new Hono()
    .use('*', async (c, next) => {
        //
        return jwtMiddleware(c, next)
    })
    .post('/', async (c) => {})
    .get('/id/:id', async (c) => {})
    .get('/', async (c) => {})
    .patch('/id/:id', async (c) => {})
    .delete('/id/:id', async (c) => {})
```

Update main.ts

```
    // /api/company
    .route('/company', companyController)
```

Update class Client

```
static readonly value = {
    user: hc<typeof userController>('/api/user'),
    authentication: hc<typeof authController>('/api/auth'),
    company: hc<typeof companyController>('/api/company')
};
```

Client routes

```
(protected)
  (table)
    company
       +page.svelte
  (upsert)
    company
      upsert
         +page.svelte
         +page.ts
         upsert-form.svelte
     +layout.svelte
```

Upsert layout

```svelte
<script lang="ts">
    let { children } = $props();
</script>

<div class="flex flex-1 flex-col gap-4 p-4 pt-0">
    {@render children()}
</div>
```

Upsert company load func

```
import { superValidate } from 'sveltekit-superforms';
import type { PageLoad } from './$types';
import { zod } from 'sveltekit-superforms/adapters';
import { companyValidator } from '@root/lib/zod/companyValidator';

export const load: PageLoad = async () => {
    return {
        Form: await superValidate(zod(companyValidator))
    }
};
```

Upsert company page

```svelte
<script lang="ts">
      import { goto } from '$app/navigation';
      import { Button } from '@/components/ui/button';
      import { ICON_SIZE } from '@/constant';
      import { ChevronLeft } from 'lucide-svelte';

      let { data } = $props();
</script>

<div class="flex min-h-[100vh] flex-1 flex-col gap-2.5 rounded-xl md:min-h-min">
      <div class="flex flex-row justify-between">
            <Button onclick={() => goto('/company')} variant="outline" size="sm" class="w-fit">
                  <ChevronLeft size={ICON_SIZE} />
                  <span class="text-sm">Back</span>
            </Button>
      </div>
      <!-- FORM -->
</div>
```

constant.ts

```ts
export const ICON_SIZE = 18;
```

Upsert company form

```
npx shadcn-svelte@next add textarea
```

```
<script lang="ts">
////import…
      type UpsertFormProps = {
            data: SuperValidated<Infer<typeof companyValidator>>;
            id?: string;
      };
      let { data, id }: UpsertFormProps = $props();
      const form = superForm(data, {
            validationMethod: 'auto',
            validators: zodClient(companyValidator),
            SPA: true,
            onUpdate: async ({ form: fd }) => {
                  if (fd.valid) {
                        // upsertt mutation
                  }
            }
      });
      const { form: formData, enhance } = form;
</script>
```

Upsert company form

```svelte
<form method="POST" use:enhance class="flex flex-col">
    <Form.Field {form} name="name">
        <Form.Control>
            {#snippet children({ props })}
                <div class="flex flex-col gap-2.5">
                    <Form.Label>Company Name</Form.Label>
                    <Input {...props} bind:value={$formData.name} placeholder="Input company name" />
                </div>
            {/snippet}
        </Form.Control>
        <Form.FieldErrors class="text-xs font-normal" />
    </Form.Field>
    <Form.Field {form} name="address">
        <Form.Control>
            {#snippet children({ props })}
                <div class="flex flex-col gap-2.5">
                    <Form.Label>Address</Form.Label>
                    <Textarea {...props} bind:value={$formData.address} />
                </div>
            {/snippet}
        </Form.Control>
        <Form.FieldErrors class="text-xs font-normal" />
    </Form.Field>
    <Form.Button class="my-2.5 ms-auto">
        <span class="flex flex-row items-center gap-2.5">
            <span>Save</span>
            <SaveIcon size={ICON_SIZE} />
        </span>
    </Form.Button>
</form>
```

Form query

```
const upsertQuery = createQuery({
    queryKey: ['upsert-company', id],
    queryFn: async () => {
        const response = await $client.company.id[':id'].$get(
            {
                param: { id: id! }
            },
            {
                init: { headers: { Authorization: localStorage.getItem(JWT.ACCESS_TOKEN)! } },
                fetch: appFetch
            }
        );

        return response.json();
    },
    enabled: !!id
});
```

Form mutation

```
      const upsertMutation = createMutation({
            mutationFn: async (data: z.infer<typeof companyValidator>) => {
                  if (id) {
                        const response = await $client.company.id[':id'].$patch(
                              {param: { id: id },form: { ...data, address: data.address || undefined }
                              },
                              {init: { headers: { Authorization: localStorage.getItem(JWT.ACCESS_TOKEN)!
}},
                               fetch: appFetch}
                        );
                        //err handling
                        return resData.result;
                  }
                  const response = await $client.company.index.$post(
                        {form: { ...data, address: data.address || undefined }},
                        {init: { headers: { Authorization: localStorage.getItem(JWT.ACCESS_TOKEN)! } },
                         fetch: appFetch});

                  const resData = await response.json();
                  //err handling
                  return resData.result;
            },
      });
```

Form mutation

```
onSuccess: () => {
      toast.success(id ? 'Update company success' : 'Create company success');
      goto('/company');
},

onError: (error) => {
      if (error instanceof ResponseError) {
            toast.error(error.message);
            return;
      }

      toast.error('Something went wrong');
}
```

```
const form = superForm(data, {
      validationMethod: 'auto',
      validators: zodClient(companyValidator),
      SPA: true,
      onUpdate: async ({ form: fd }) => {
            if (fd.valid) {
                  $upsertMutation.mutate(fd.data);
            }
      }
});


const { form: formData, enhance, reset } = form;

$effect(() => {
      const res = $upsertQuery.data?.result;
      if (res) {
            reset({ data: { ...res } });
      }
});
```

Company table page

```svelte
<script lang="ts">
    import { goto } from '$app/navigation';
    import { Button } from '@/components/ui/button';
    import { Client } from '@/context/client';
    import { appFetch, JWT } from '@/core/jwt';
    import { createQuery } from '@tanstack/svelte-query';

    const client = Client.getCtx();
    const companyQuery = createQuery({
        queryKey: ['company'],
        queryFn: async () => {
            const response = await $client.company.index.$get(
                {
                    query: {}
                },
                {
                    init: { headers: { Authorization: localStorage.getItem(JWT.ACCESS_TOKEN)! } },
                    fetch: appFetch
                }
            );
            return response.json();
        }
    });
</script>

<pre>
    {JSON.stringify($companyQuery.data, null, 2)}
</pre>
<Button class="w-fit" onclick={() => goto('/company/upsert')}>add company</Button>
```