Svelte 5 + Hono 🔥

Part 9: Application backend, upsert application

www.youtube.com/@GetterSethya

Application repository

```typescript
export class ApplicationRepository implements Omit<AsyncBaseRepository<ApplicationEntity>, 'list'> {
    async create(args: CreateArgs<ApplicationEntity>): Promise<ApplicationEntity> {}
    async findById(args: FindByIdArgs): Promise<ApplicationEntity | null> {}
    async list(args: ListArgs) {}
    async update(args: UpdateArgs<ApplicationEntity>): Promise<ApplicationEntity> {}
    async delete(args: DeleteArgs): Promise<boolean> {}
}
```

```typescript
export const APPLICATION_STATUS_KEY = ['sent', 'reply', 'interview', 'ghosting', 'offering'] as const
export const APPLICATION_STATUS: Record<(typeof APPLICATION_STATUS_KEY)[number], string> = {
    sent: 'Terkirim',
    reply: 'Dibalas',
    interview: 'Interview',
    ghosting: 'Ghosting',
    offering: 'Offering',
}
```

```typescript
export const applicationValidator = z.object({
    company: z.number({ message: 'invalid company id' }).positive({ message: 'invalid company id' }),
    position: z.string({ message: 'position cant be blank' }).min(1, { message: 'position cant be blank' }),
    status: z.enum(APPLICATION_STATUS_KEY).default('sent'),
    notes: z.string({ message: 'invalid notes type' }),
})
```

Application controller

```
export const applicationController = new Hono()
       //
       .use('*', jwtMiddleware)
       .use('*', repositoryMiddleware)
       .post('/', appValidator('form', applicationValidator), async (c) => {})
       .get('/', appValidator('query', queryUrlSchema), async (c) => {})
       .get('/id/:id', appValidator('param', idSchema), async (c) => {})
       .patch('/id/:id', appValidator('param', idSchema), appValidator('form', applicationValidator), async (c) => {
              })
       .delete('/id/:id', appValidator('param', idSchema), async (c) => {})
```

```
export type RepositoryMiddleware = {
       Variables: {
              applicationRepo: ApplicationRepository
       }
}
export const repositoryMiddleware = createMiddleware<RepositoryMiddleware>(async (c, next) => {
       c.set('applicationRepo', new ApplicationRepository())
       await next()
})
```

```
//  main.ts
// /api/application
       .route('/application', applicationController)
```

Routes

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

Client

```
    static readonly value = {
            //
            application: hc<typeof applicationController>('/api/application')
    };
```

Table application +page.svelte

```
const client = Client.getCtx();
const applicationEndpoint = $client.application.index.$get;
type ApplicationResponse = NonNullable<InferResponseType<typeof applicationEndpoint>['result']>;
const applicationQuery = createQuery({});
const columns: ColumnDef<ApplicationResponse['items'][number]>[] = [];
const bulkbarMutation = createMutation({
    mutationFn: async (data: Row<ApplicationResponse['items'][number]>[]) => {},
    onSuccess: () => {},
    onError: (err) => {}
});
const bulkbar: TableBulkbar<ApplicationResponse['items'][number]> = {
    onAction: ({ data, table, isMutating }) => {
        isMutating(true);
        $bulkbarMutation.mutate(data, {
            onSettled: () => {
                isMutating(false);
                table.resetRowSelection();
            }
        });
    }
};

afterNavigate(() => $applicationQuery.refetch());
```

```
{#snippet Actions(data: ApplicationResponse['items'][number])}
	<DropdownMenu.Content side="left" align="start">
		<DropdownMenu.Group>
			<DropdownMenu.GroupHeading>Actions</DropdownMenu.GroupHeading>
			<DropdownMenu.Item onclick={() => goto(`/application/upsert?id=${data.applications.id}`)}
				>Edit</DropdownMenu.Item
			>
		</DropdownMenu.Group>
	</DropdownMenu.Content>
{/snippet}

<div class="flex flex-row items-center justify-end gap-2.5">
	<Search />
	<Button size="icon" variant="secondary" onclick={() => goto('/application/upsert')}>
		<PlusIcon size={ICON_SIZE} />
	</Button>
</div>
{#if !$applicationQuery.isSuccess}
	<span>loading..</span>
{:else if $applicationQuery.data.result?.items}
	<DataTable {actions} data={$applicationQuery.data.result.items} {columns} {bulkbar} />
	<div class="flex flex-row justify-between">
		<Limit totalItems={$applicationQuery.data.result.meta.totalItems} />
		<Pagination totalItems={$applicationQuery.data.result.meta.totalItems} />
	</div>
{/if}
```

Application upsert +page.ts

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

Application upsert upsert-form.svelte

```
type UpsertFormProps = {data: SuperValidated<Infer<typeof applicationValidator>>;id?: string;};
let { data, id }: UpsertFormProps = $props();
const client = Client.getCtx();
const upsertQuery = createQuery({
    queryKey: ['upsert-application', id],
    queryFn: async () => {},
    enabled: !!id
});
const upsertMutation = createMutation({
    mutationFn: async (data: z.infer<typeof applicationValidator>) => {},
    onSuccess: () => {},
    onError: (error) => {}
});
const form = superForm(data, {
    validationMethod: 'auto',
    validators: zodClient(applicationValidator),
    SPA: true,
    onUpdate: async ({ form: fd }) => {}
});
const { form: formData, enhance, reset } = form;
$effect(() => {
    if ($upsertQuery.isSuccess) {
        const res = $upsertQuery.data?.result;
        if (res) {
            reset({
                data: {      }
            });
        }
    }
});
```

```
npx shadcn-svelte@next add popover
npx shadcn-svelte@next add command
npx shadcn-svelte@next add badge
```

Async combobox

```
type AsyncComboboxProps = {
        class?: string;
        items: Writable<DropdownItem[]>;
        placeholder?: string;
        open?: boolean;
        loadFn: (search: string) => Promise<DropdownItem[]>;
        onSelect?: (originalData: { label: string; value: string }) => void;
} & HTMLInputAttributes;

let { open = $bindable(false),items,value = $bindable(''),placeholder,disabled,name,class:
className,loadFn,onSelect,...rest}: AsyncComboboxProps = $props();
let selectedLabel = $state(placeholder);
let isLoading = $state(false);

$effect(() => {
        if ($items.length > 0) {
                const find = $items.find((f) => f.value === value)?.label;
                if (find) {
                        selectedLabel = find;
                } else {
                        selectedLabel = placeholder;
                        value = '';
                }
        }
});
```

Async combobox

```
<input {name} {...rest} type="hidden" bind:value />
<Popover.Root bind:open>
	<Popover.Trigger
		class={'flex w-full flex-row items-center justify-between rounded-lg border border-border'}
		role="combobox"
	>
		<Input
			role="combobox"
			aria-expanded={open}
			class={cn(
				'w-full border-none bg-transparent p-2.5',
				disabled ? 'cursor-not-allowed' : 'cursor-pointer'
			)}
			placeholder={selectedLabel || 'Select item'}
			readonly={true}
			{disabled}
			value={selectedLabel !== placeholder ? selectedLabel : ''}
		/>
		<ChevronsUpDown class="opacity-50" />
	</Popover.Trigger>
	<Popover.Content class="w-full p-0" side="bottom" align="start">
		<Command.Root>
			<Command.Input
				oninput={useDebounce((e) => handleInputChange(e), 500)}
				autofocus
				placeholder="Search..."
				class="h-9"
			/>
```

Async combobox

```
            {#if isLoading}
                <div class="flex min-h-[12rem] items-center justify-center">
                    <span class="text-center text-sm">Loading...</span>
                </div>
            {:else if $items.length === 0}
                <Command.List>
                    <Command.Empty>No Data</Command.Empty>
                </Command.List>
            {:else}
                <Command.List>
                    {#key items}
                        {#each $items as item (item.value)}
                            <div class="p-1">
                                <Button class="line-clamp-1 flex w-full flex-row items-center
justify-center text-left"

                                    variant="ghost"
                                    value={item.label}
                                    onclick={() => {value = item.value;
                                        if (onSelect) {onSelect(item)}}}>
                                    {item.label}
                                    <Check class={cn('ml-auto', value !== item.value &&
'text-transparent')} />
                                </Button>
                            </div>
                        {/each}
                    {/key}
                </Command.List>
            {/if}
        </Command.Root>
    </Popover.Content>
</Popover.Root>
```

Application status badge

```svelte
<script lang="ts">
	import { APPLICATION_STATUS, type APPLICATION_STATUS_KEY } from '@root/lib/constant';
	import { Badge } from '.';

	type ApplicationStatusBadgeProps = {
		status: (typeof APPLICATION_STATUS_KEY)[number];
	};

	let { status }: ApplicationStatusBadgeProps = $props();
</script>

{#if status === 'sent'}
	<Badge class="border border-blue-500 bg-blue-100 dark:bg-blue-900 dark:text-blue-100 text-blue-500
hover:bg-blue-200">{APPLICATION_STATUS[status]}</Badge>
{:else if status === 'reply'}
	<Badge class="border border-emerald-500 bg-emerald-100 dark:bg-emerald-900 dark:text-emerald-100
text-emerald-500 hover:bg-emerald-200">{APPLICATION_STATUS[status]}</Badge>
{:else if status === 'ghosting'}
	<Badge class="border border-border dark:bg-card bg-muted/20 text-foreground hover:bg-muted/20"
	>{APPLICATION_STATUS[status]}</Badge>
{:else if status === 'offering'}
	<Badge class="border border-violet-500 bg-violet-100 dark:bg-violet-900 dark:text-violet-100 text-violet-500
hover:bg-violet-200">{APPLICATION_STATUS[status]}</Badge>
{:else}
	<Badge class="border border-orange-500 bg-orange-100 dark:bg-orange-900 dark:text-orange-100 text-orange-500
hover:bg-orange-200">{APPLICATION_STATUS[status]}</Badge>
{/if}
```

Application upsert form

```svelte
<form method="POST" use:enhance class="flex flex-col">
    <Form.Field {form} name="company">
        <Form.Control>
            {#snippet children({ props })}
                <div class="flex flex-col gap-2.5">
                    <Form.Label>Company</Form.Label>
                    <AsyncCombobox
                        bind:value={selectedCompany}
                        onSelect={handleOnSelect}
                        items={companies}
                        loadFn={companyLoadFn}
                        {...props}
                    />
                </div>
            {/snippet}
        </Form.Control>
        <Form.FieldErrors class="text-xs font-normal" />
    </Form.Field>
    <Form.Field {form} name="position">
        <Form.Control>
            {#snippet children({ props })}
                <div class="flex flex-col gap-2.5">
                    <Form.Label>Position</Form.Label>
                    <Input {...props} bind:value={$formData.position} placeholder="Input position" />
                </div>
            {/snippet}
        </Form.Control>
        <Form.FieldErrors class="text-xs font-normal" />
    </Form.Field>
```

Application upsert form

```svelte
<Form.Field {form} name="status">
    <Form.Control>
        {#snippet children({ props })}
            <div class="flex flex-col gap-2.5">
                <Form.Label>Status</Form.Label>
                <Select.Root type="single" bind:value={$formData.status} name={props.name}>
                    <Select.Trigger {...props}>
                        {#if $formData.status}
                            <ApplicationStatusBadge status={$formData.status} />
                        {:else}
                            <span> Select application status</span>
                        {/if}
                    </Select.Trigger>
                    <Select.Content>
                        {#each APPLICATION_STATUS_KEY as status}
                            <Select.Item value={status} label={APPLICATION_STATUS[status]}>
                                {#snippet children()}
                                    <ApplicationStatusBadge {status} />
                                {/snippet}
                            </Select.Item>
                        {/each}
                    </Select.Content>
                </Select.Root>
            </div>
        {/snippet}
    </Form.Control>
    <Form.FieldErrors class="text-xs font-normal" />
</Form.Field>
```

Application upsert form

```
    <Form.Field {form} name="notes">
        <Form.Control>
            {#snippet children({ props })}
                <div class="flex flex-col gap-2.5">
                    <Form.Label>Notes</Form.Label>
                    <Textarea {...props} bind:value={$formData.notes} placeholder="Add notes" />
                </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>
```

Application upsert form

```
$effect(() => {
    const userCompany = $upsertQuery.data?.result?.companies;
    if ($companyQuery.data && userCompany) {
        const companyIndex = $companies.findIndex((c) => c.value === userCompany.id.toString());
        if (companyIndex === -1) {
            $companies.push({ value: userCompany.id.toString(), label: userCompany.name });
        }
    }
});

async function companyLoadFn(search: string): Promise<DropdownItem[]> {
    const products = await $client.company.index.$get(
        {query: {q: search,limit: '5',sort: 'id'}},
        {
            fetch: appFetch,
            init: {
                headers: { Authorization: localStorage.getItem(JWT.ACCESS_TOKEN) || '' }
            }
        }
    );

    const resData = await products.json();
    if (!resData || !resData.result) return [];

    return resData.result.items.map((c) => ({ value: c.id.toString(), label: c.name }));
}

function handleOnSelect(originalData: { label: string; value: string }) {
    $formData.company = Number(originalData.value);
}
```

Application +page.svelte

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

	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('/')} variant="outline" size="sm" class="w-fit">
			<ChevronLeft size={ICON_SIZE} />
			<span class="text-sm">Back</span>
		</Button>
	</div>
	<UpsertForm data={data.form} id={page.url.searchParams.get('id') || undefined} />
</div>
```

Fix hono static file route

```
.use('*', async (c, next) => {
        const url = new URL(c.req.url)
        if (
                !url.pathname.startsWith('/api') &&
                !url.pathname.match(/\.(js|css|png|jpg|gif|ico|svg|ttf|woff|woff2|eot|otf)$/)
        ) {
                c.req.path = '/'
        }
        await next()
})
```