Svelte kit

→ Server and client side rendering
→ comes with a routing system
→ Provides a backend API
→ Code splitting

Svelte and other libraries like react are single page based libraries that is a single HTML file is prepared which rendered by browser.

while sveltekit allows server rendering on server first, then pre-rendered html page is sent to browser.
Svelte kit blurs the line b/w frontend and backend.

• Set up sveltekit:

↳ follow official docs
* general installation process includes:
↳ npm create svelte@latest name-of-project
↳ choose skeleton project (demo project is for checking out features of sveltekit and library project is for building reusable libs)
↳ it has pre-built features for to but choose none in case we want to continue in future.
↳ cd name-of-project
↳ npm i
↳ npm run dev -- --open

• File structure in sveltekit project:

→ node_modules
→ .svelte-kit
→ src → routes (consists of pages)
→ html
→ static

This html file is the template html file which intakes the route pages
bunch of configuration files (package.json, package-lock.json, .prettierrc, .prettierignore, vite.config.js, svelte.config.js and others.

Routing

Sveltekit has a file system based routing mechanism. URL paths that users can access in the browser are defined by files and folders. This statement would become more clear after understanding routing conventions:

1. We must have a folder called routes. Inside it we create files with ext. .svelte and '+' sign as prefix.
2. To define routes we need to create folder inside the routes folder with routename. Inside these folders create files with '+' sign prefix followed by name of file (usually page) and then .svelte at end.

By doing above routes are automatically generated. No router configuration is involved.

The home route is directly inside the routes folder.

Nested routes

Naturally or I would intuitively creating folders inside existing folder in routes would create nested routes.

Ex

localhost:5173/hello/world

```
→ routes
    └→ hello
        ├→ +page.svelte
        └→ world
            └→ +page.svelte
```

Dynamic routes

While defining routes is very easy to implement. It is only feasible when routes are clearly defined and are small in number. What if several thousand of routes are possible. An example of it would be product id of

products. There can be millions of products and hence millions of product Ids. If we are willing to access a product by hitting /products/productId then we'd have to create millions of routes which is not feasible. So, we utilize dynamic routing concept. We define one single folder with name "[productId]" (square brackets are necessary). Now automatically it will map to any productId. In order to access the productId, use page store. import it in the page and extract the productId using $page.params.productId

→ products
  ↳ +page.svelte
  ↳ [productId]
    ↳ +page.svelte

```
<script>
import {page} from '$app/stores'
const productId = $page.params.productId
</script>

<h1>Product number - {productId}</h1>
```

Nested Dynamic routes

Straightforward.

Create another folder inside the dynamic route folder. Create page.

Create another folder inside the previous folder and use dynamic route name to create another dynamic route.

Now, this deeply nested route pg can access both the current route params as well as any parent route's dynamic params.

Catch all routes

It may be difficult to implement nested dynamic routes if they become too many. Svelte provides a way to catch all the routes sent at one place. For this purpose we used [...name-of-folder] (preferred name is rest). Then we can track routes no matter how deep it goes.

But it won't make sense if we don't extract route params.

Lets say we name the folder as '[...slug]'. Then to extract route params we use the same $page.params.slug
It return string containing the entire route path.
We can use as it is or use .split('/') to convert route params to contents of an array.
Then we can use it easily to render UI as per the route.

Optional parameters

For cases where optional parameters are to be handled use [[name-of-route]]. It could be useful for cases like multiple language handling.
In case it is at the same level of home route, either delete the home route or more the optional route to any other folder (it's not recommended at all). Default value must be set to avoid 'Not found' error.

Route Navigation

Use <a> to navigate to diff. route
href includes simple url to the route.

Routing programatically

It is done by "goto" functions of '$app/navigation'
Just write goto(route). It will took you the route.
Also, in the goto, we can pass objects as options parameters where we can specify replaceState: true so that it replaces the history instead of adding to it.

beforeNavigate and afterNavigate are two fn of navigation. They take navigation as inputs. They provide info like from, to and type of navigation. Also, beforeNavigate has cancel fn.

Route matchers

To avoid unknown requests we can limit the parameters value to satisfy a few conditions.

For ex → Lets product ID can only be a no. then we can create a separate folder with a js file that applies regex to check if the product Id is a no. of not. Then rename the folder name of the route where it would be applied (of course it would be, to name-of-route ≠ file-name

↳ The js file implementing the checker function.

Layouts

Most of the time pages are required to be carrying some repetitive components. Instead of passing them to each component/page, we can create a layout to be followed in each page in the same directory.

for ex → Every page should contain header and footer. We will create a +layout.svelte page in the same level as the parent/home route. Inside this page, we create slot tag. Then create a footer and header in the page. All the page in the same directory will replace the slot tag to include its own UI.

Layout grouping

We can create separate layout for a few pages without affecting the url. Lets say page1 and page2 are two such routes

~~If they have to be in the same directory~~

Lets say they want some common features apart from the home route. Just move folders of these pages to a new folder with name "(name of folder)". The parantheses makes it a group of layout eliminating the need to include name of folder(group) in the url.

Breaking out of layout

Every route inherits the layout of its parent component.
To break out of layout of any parent change the name of the file to say +page@(...).svelte.

Specify the route above which are allowed to provide layout. A single @ would mean only the home route.

We can keep comps like headers, footers and others into other separate files in the routes folder itself. Just don't put + prefix. Import it and use it. We can move them to a lib folder and then import them by using $lib/path and then use it.

API routes

We can also route to API endpoints.
API routes allow us to create RESTful endpoints giving us full control over the response.
There is no overhead of having to create and configure a separate server.
API routes are also great for making ext. API requests
API routes are perfect as they are server side routes that are never exposed to the browser.

for creating it, simply create a server.js file in a folder inside routes folder. define a GET() fn that returns some response and its done. An API endpoint is created.

We can create GET, POST, PATCH, DELETE, PUT api endpoints.
To test routes, hit url http://[::1]:5173/api/...
localhost is not used