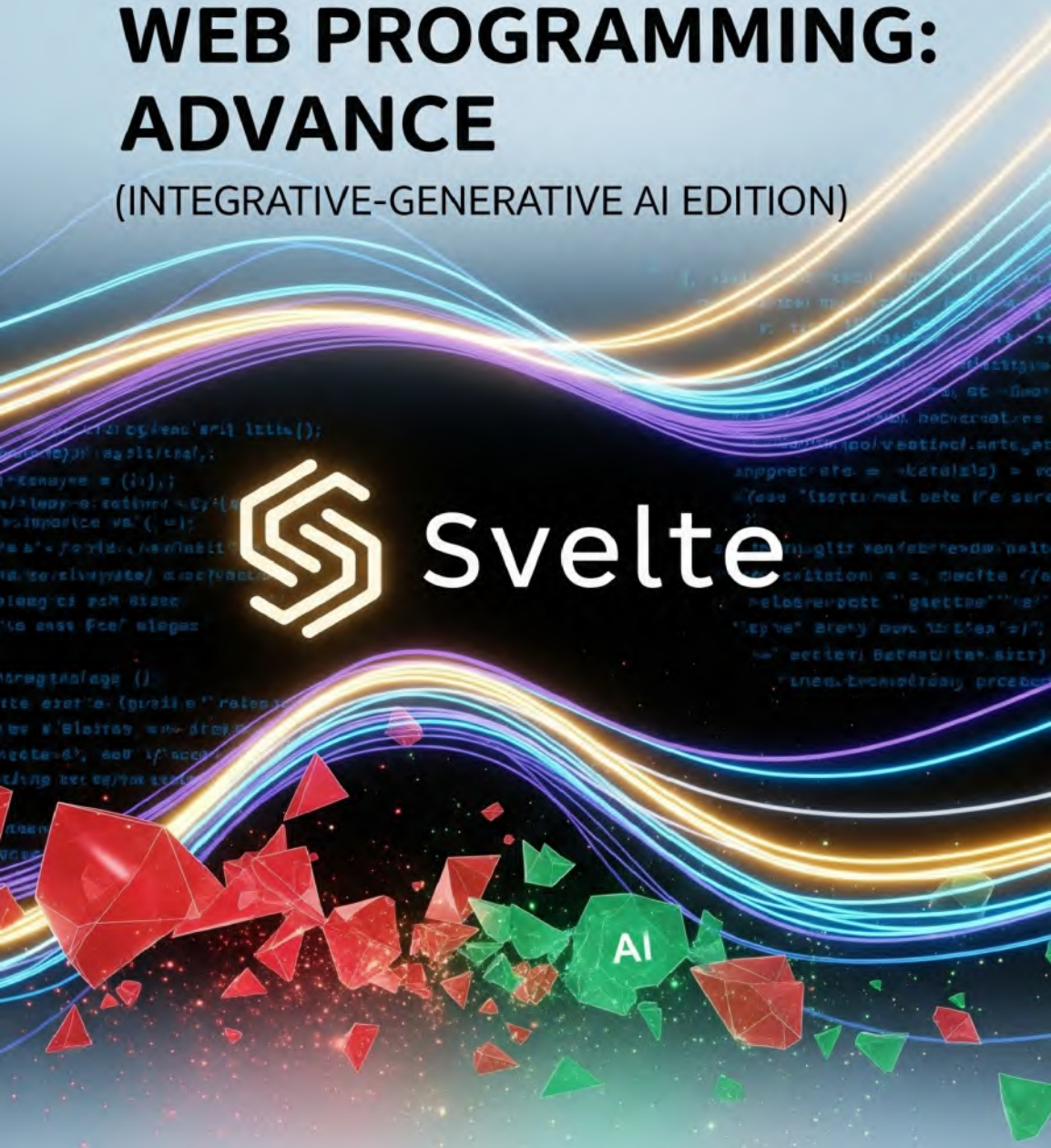
SVELTE WEB PROGRAMMING: ADVANCE

(INTEGRATIVE-GENERATIVE AI EDITION)



Student Price Book Center

คำนำ

ในทศวรรษที่ผ่านมา การพัฒนาเว็บแอปพลิเคชันได้มีการเปลี่ยนแปลงอย่างรวดเร็ว ทั้งในด้านเทคโนโลยี เครื่องมือ และแนวคิดในการออกแบบระบบ Svelte เป็นหนึ่งในเฟรมเวิร์กที่เกิดขึ้นจากความพยายามในการแก้ปัญหาที่พบในเฟรมเวิร์กรุ่นก่อน ๆ โดยนำเสนอแนวทางที่เรียบง่าย ประสิทธิภาพสูง และมีขนาดเล็ก หนังสือ *Svelte Web Programming: Advance* เล่มนี้จึงถือกำเนิดขึ้นเพื่อเสริมความรู้ในระดับลึกต่อจากระดับพื้นฐาน โดยมุ่งเน้นการนำ Svelte และ SvelteKit ไปใช้ในสถานการณ์จริงที่ซับซ้อนมากขึ้น เหมาะสำหรับนักพัฒนาที่ต้องการยกระดับทักษะสู่ระดับมืออาชีพ

หนังสือเล่มนี้ประกอบด้วย 5 บทหลัก เริ่มต้นที่บทที่ 11 ซึ่งเจาะลึก **การจัดการ Routing และโครงสร้างภายใน SvelteKit** ผู้อ่านจะได้เรียนรู้รูปแบบของโฟลเดอร์ src/routes การทำงานของ Layout และ Nested Layout รวมถึงการใช้ Dynamic Route ซึ่งเป็นหัวใจของเว็บแอปสมัยใหม่ นอกจากนี้ยังมีการอธิบายการใช้งาน $page และ $params เพื่อรับข้อมูลจาก URL และการโหลดข้อมูลด้วย load() function อย่างมีประสิทธิภาพ

บทที่ 12 กล่าวถึง **Store ขั้นสูง และ Context API** ซึ่งมีบทบาทสำคัญในการจัดการสถานะของแอปพลิเคชันในระดับโครงสร้าง การใช้ setContext() และ getContext() ช่วยให้การส่งข้อมูลระหว่างคอมโพเนนต์มีความยืดหยุ่นและลดการพึ่งพา props ที่ลึกเกินไป รวมถึงตัวอย่างการใช้งานจริงในการจัดการ Theme, ระบบ Auth และการสร้างระบบหลายภาษา (Multi-language) ซึ่งเป็นองค์ประกอบสำคัญของเว็บแอปที่ใช้งานจริง

บทที่ 13 พาผู้อ่านไปรู้จัก **โลกของ Animation และ Transition** ที่ทำให้เว็บแอปมีชีวิตชีวา โดยเริ่มจาก transition พื้นฐาน เช่น fade, slide, fly ไปจนถึงการสร้าง Custom Transition และการใช้ animate:flip สำหรับการจัดการการเคลื่อนไหวของรายการที่เปลี่ยนลำดับ นอกจากนี้ยังแนะนำเทคนิคการสร้าง animation แบบ reusable ที่สามารถแชร์การใช้งานระหว่างหลายคอมโพเนนต์ เพื่อเพิ่มความสอดคล้องในการออกแบบ UI

ในบทที่ 14 ผู้อ่านจะได้เรียนรู้การใช้ **Actions หรือไดเรกทีฟ use:** ซึ่งเปิดโอกาสให้เราสามารถเขียน logic เพื่อจัดการพฤติกรรมของ DOM โดยตรง เช่น การสร้าง Custom Action อย่าง use:longpress หรือการเชื่อมโยงกับไลบรารีภายนอก เช่น D3.js ที่ต้องเข้าถึง DOM โดยตรง บทนี้จะอธิบายอย่างละเอียดถึง lifecycle ของ Action และแนวทางการจัดการให้เหมาะสมกับการใช้งานจริงในเว็บแอป

บทสุดท้าย บทที่ 15 นำเสนอหัวข้อที่สำคัญมากในเชิงสถาปัตยกรรมเว็บแอปคือ **การทำ SSR และ Prerender ด้วย SvelteKit** ซึ่งช่วยให้หน้าเว็บโหลดได้รวดเร็วขึ้น และรองรับการจัดทำ SEO ได้ดียิ่งขึ้น โดยมีการอธิบายการตั้งค่า prerender = true การใช้ load() ทั้งฝั่งเซิร์ฟเวอร์และไคลเอนต์ ตลอดจนการใช้ <svelte:head> เพื่อกำหนด meta tag สำหรับ SEO อย่างเหมาะสม เพื่อสร้างเว็บไซต์ที่ค้นหาเจอได้ง่ายและมีประสิทธิภาพสูง

หนังสือเล่มนี้ถูกออกแบบให้เป็นสื่อการเรียนรู้สำหรับนักพัฒนาเว็บระดับกลางถึงสูง ที่ต้องการเข้าใจการใช้งาน Svelte และ SvelteKit ในระดับเชิงลึก โดยเนื้อหาแต่ละบทประกอบด้วยแนวคิดหลัก ตัวอย่างโค้ด และคำอธิบายอย่างเป็นระบบ พร้อมทั้งสอดแทรกแนวทางปฏิบัติที่ดี (best practices) เพื่อให้ผู้อ่านสามารถนำไปประยุกต์ใช้ในโปรเจกต์จริงได้อย่างมั่นใจ

หวังเป็นอย่างยิ่งว่าผู้อ่านจะได้รับความรู้ ความเข้าใจ และแรงบันดาลใจจาก *Svelte Web Programming: Advance* เพื่อก้าวสู่การเป็นนักพัฒนาเว็บแอปพลิเคชันยุคใหม่ที่มีทักษะเชิงลึก พร้อมรับมือกับความท้าทายของระบบที่ซับซ้อนและเปลี่ยนแปลงอยู่เสมอ.

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

บทที่ 11

SvelteKit และ Routing ขั้นสูง

(Advanced SvelteKit and Advanced Routing)

เนื้อหา

บทนำบทที่ 11: SvelteKit และ Routing ขั้นสูง

การจัดการเส้นทาง (Routing) คือหัวใจสำคัญของเว็บแอปพลิเคชันที่มีหลายหน้า การกำหนดเส้นทางไม่เพียงแต่เป็นการระบุว่าเมื่อผู้ใช้เข้าถึง URL หนึ่งแล้วจะต้องแสดงหน้าจอใด แต่ยังเกี่ยวข้องกับการจัดการข้อมูลที่ต้องโหลด การกำหนดโครงสร้างส่วนร่วมระหว่างหน้า และการควบคุมพฤติกรรมของแอปพลิเคชันในระดับสูง SvelteKit ซึ่งเป็นเฟรมเวิร์กที่สร้างขึ้นบนพื้นฐานของ Svelte ได้ออกแบบระบบ Routing ใหม่ที่ทรงพลังโดยยึดตามแนวคิด file-based routing ซึ่งผสานเข้ากับความสามารถเชิงลึกอย่าง layout ซ้อน (nested layout), dynamic route และ server-side data loading ได้อย่างกลมกลืน

บทนี้จะเริ่มต้นด้วยการทำความเข้าใจโครงสร้างโฟลเดอร์พื้นฐานของ SvelteKit โดยเฉพาะในส่วนของ src/routes ซึ่งเป็นจุดศูนย์กลางของระบบ Routing ทั้งหมด ไฟล์ในโฟลเดอร์นี้จะถูกแปลงเป็นเส้นทาง (routes) โดยอัตโนมัติ ผู้อ่านจะได้เรียนรู้ว่าไฟล์ .svelte, โฟลเดอร์, และชื่อไฟล์ต่าง ๆ ส่งผลต่อ URL และลำดับของ routing อย่างไร ตลอดจนการกำหนดหน้า fallback เช่น +error.svelte และ +layout.svelte ซึ่งเป็นกลไกสำคัญในการจัดการมุมมองโดยรวม

จากนั้น บทนี้จะนำเสนอแนวคิด Layout และ Nested Layout ซึ่งช่วยให้สามารถกำหนดโครงสร้างหน้าเว็บที่ใช้ร่วมกัน เช่น แถบเมนูด้านบน (Navbar) หรือแถบด้านข้าง (Sidebar) โดยไม่ต้องทำซ้ำในทุกหน้า การทำ Nested Layout ยังเปิดโอกาสให้สร้างระบบ navigation ที่มีหลายระดับได้อย่างมีประสิทธิภาพ นอกจากนี้ยังจะได้เรียนรู้เกี่ยวกับการจัดการ layout หลายชั้นที่ซ้อนกัน และวิธีสื่อสารข้อมูลระหว่าง layout กับหน้าเฉพาะ

ในโลกของเว็บแอปพลิเคชันที่มีความซับซ้อน การใช้ Dynamic Route มีความสำคัญอย่างยิ่ง SvelteKit รองรับ dynamic route ได้โดยใช้ชื่อไฟล์ในรูปแบบ [param].svelte ซึ่งช่วยให้สามารถจับ

ค่าพารามิเตอร์จาก URL ได้อย่างสะดวก เช่น /blog/[slug] สำหรับแสดงบทความตาม slug ที่แตกต่างกัน บทนี้จะสาธิตการใช้งาน dynamic route อย่างชัดเจน พร้อมตัวอย่างที่แสดงถึงความยืดหยุ่นของแนวคิดนี้

เพื่อให้สามารถเข้าถึงข้อมูลที่เกี่ยวข้องกับเส้นทางแบบ dynamic ได้ SvelteKit ได้จัดเตรียมตัวแปรสำคัญอย่าง $params ซึ่งเก็บค่าพารามิเตอร์จาก URL และ $page ซึ่งให้ข้อมูลเกี่ยวกับสถานะของหน้าในปัจจุบัน เช่น path, query, และ status ต่าง ๆ ผู้อ่านจะได้ฝึกใช้งาน $params และ $page เพื่อสร้างเงื่อนไขการแสดงผล และเพื่อส่งข้อมูลไปยังคอมโพเนนต์ต่าง ๆ อย่างมีประสิทธิภาพ

อีกหัวข้อที่สำคัญไม่แพ้กันคือการโหลดข้อมูลในแต่ละหน้า SvelteKit มีฟังก์ชัน load() ที่ทำงานก่อนการ render หน้า เพื่อให้สามารถดึงข้อมูลจาก API หรือฐานข้อมูลได้ล่วงหน้า ฟังก์ชันนี้ทำงานทั้งฝั่งเซิร์ฟเวอร์และไคลเอนต์ และสามารถใช้ร่วมกับ layout เพื่อโหลดข้อมูลที่ใช้ร่วมกันระหว่างหลายหน้าได้อย่างชาญฉลาด ผู้อ่านจะได้เรียนรู้การเขียน load() ในรูปแบบพื้นฐาน ไปจนถึงการประยุกต์ใช้ในรูปแบบที่ซับซ้อน

ท้ายที่สุด บทนี้จะสรุปภาพรวมของการออกแบบ routing ที่ยืดหยุ่นใน SvelteKit ตั้งแต่ระดับโฟลเดอร์ โครงสร้าง layout, พารามิเตอร์, ไปจนถึงกระบวนการโหลดข้อมูลทั้งหมด โดยเน้นแนวทางปฏิบัติที่ดีที่สุดและแนวทางการดีไซน์แอปที่ขยายขนาดได้ง่าย จุดมุ่งหมายของบทนี้ไม่ใช่เพียงแค่ทำให้ผู้อ่านสามารถใช้งาน Routing ได้เท่านั้น แต่ต้องสามารถ "ออกแบบ" Routing ที่รองรับแอปพลิเคชันระดับจริงได้อย่างมั่นใจและยั่งยืน.

SvelteKit และ Routing ขั้นสูง

---

1. โครงสร้างโฟลเดอร์ SvelteKit (src/routes)

- ใน SvelteKit โฟลเดอร์ src/routes ใช้สำหรับกำหนดเส้นทาง (Routing) ของแอป
- แต่ละไฟล์ .svelte ในโฟลเดอร์นี้จะกลายเป็นหน้าหรือ route อัตโนมัติ เช่น

```
src/routes/
├── index.svelte          -> หน้า /
├── about.svelte          -> หน้า /about
├── blog/
│   ├── index.svelte      -> หน้า /blog
│   └── [slug].svelte     -> หน้า /blog/:slug (Dynamic Route)
└── dashboard/
    └── settings.svelte -> หน้า /dashboard/settings
```

---

2. Layout และ Nested Layout

- ไฟล์ +layout.svelte ใช้สร้าง Layout ให้กับหน้าในโฟลเดอร์เดียวกันและลูกหลาน
- Layout ทำหน้าที่ห่อหุ้มหน้าลูกทั้งหมด เพื่อแชร์โครงสร้างเช่น header, footer หรือ sidebar

ตัวอย่าง

```
src/routes/
├── +layout.svelte        <- Layout หลักของทุกหน้า
├── index.svelte          <- หน้า /
├── blog/
│   ├── +layout.svelte    <- Layout ของ /blog และลูก
│   ├── index.svelte      <- หน้า /blog
│   └── [slug].svelte     <- หน้า /blog/:slug
```

- เมื่อเข้าหน้า /blog/some-post
  - จะใช้ Layout จาก /blog/+layout.svelte
  - และใช้ Layout หลักจาก /+layout.svelte ถ้ามีการเรียก slot ซ้อนกัน

---

3. Dynamic Route

- SvelteKit ใช้สัญลักษณ์ [] เพื่อกำหนด route แบบ dynamic เช่น [id].svelte
- ตัวอย่าง src/routes/blog/[slug].svelte จะจับ route /blog/anything และส่งค่า slug เข้าไปในตัวแปร params

---

4. การใช้ $page และ $params

- $page เป็น store ที่เก็บข้อมูลของหน้าปัจจุบัน เช่น path, params, query
- $params เป็น shortcut สำหรับค่าพารามิเตอร์ route ปัจจุบัน

ตัวอย่างใช้งาน

```
<script>
  import { page } from '$app/stores';

  $: console.log($page.url.pathname); // path ปัจจุบัน เช่น /blog/my-post
  $: console.log($page.params.slug);  // ค่าพารามิเตอร์ เช่น my-post
</script>
```

---

5. การโหลดข้อมูลด้วย load()

- ใน SvelteKit เราใช้ฟังก์ชัน load เพื่อโหลดข้อมูลก่อนแสดงหน้า (server หรือ client)
- ไฟล์ +page.js หรือ +page.ts จะ export ฟังก์ชัน load ที่ return ข้อมูลเป็น props ให้หน้าใช้
- หรือเขียน load ใน +page.svelte ได้ด้วย

ตัวอย่าง +page.js

```
/** @type {import('./$types').PageLoad} */
export async function load({ params, fetch }) {
  const res = await fetch(`/api/posts/${params.slug}`);
  if (res.ok) {
    const post = await res.json();
    return { post };
  }
  return { status: res.status, error: new Error('Not found') };
}
```

ใช้งานใน +page.svelte

```
<script>
  export let data;
</script>

<h1>{data.post.title}</h1>
<p>{data.post.content}</p>
```

สรุป

หัวข้อ	รายละเอียด
โครงสร้างโฟลเดอร์	src/routes ใช้จัดการ routing อัตโนมัติ
Layout	ใช้ +layout.svelte สร้างโครงสร้างหน้าแบบ reusable
Nested Layout	Layout ซ้อนกันได้ตามลำดับโฟลเดอร์
Dynamic Route	ใช้ [param].svelte จับ path แบบ dynamic
$page, $params	Store ดึงข้อมูล path, params, query
load()	โหลดข้อมูลก่อน render หน้า ทำงานทั้ง client/server

รายละเอียดเชิงลึก บทที่ 11: SvelteKit และ Routing ขั้นสูง

---

1. โครงสร้างโฟลเดอร์ src/routes

- SvelteKit ใช้ไฟล์และโฟลเดอร์ภายใน src/routes เป็นตัวกำหนด routing โดยอัตโนมัติ (File-based routing)

- ทุกไฟล์ .svelte ในโฟลเดอร์นี้จะกลายเป็น route หนึ่งหน้า (Page) เช่น
    - src/routes/index.svelte ⟶ / (หน้าแรก)
    - src/routes/about.svelte ⟶ /about
- โฟลเดอร์ย่อยจะเป็น route ที่ซ้อนกัน เช่น
    - src/routes/blog/index.svelte ⟶ /blog
    - src/routes/blog/post.svelte ⟶ /blog/post
- นอกจากนี้ไฟล์และโฟลเดอร์ที่ขึ้นต้นด้วย + เช่น +page.svelte, +layout.svelte, +page.js เป็น naming convention ใหม่ใน SvelteKit (ตั้งแต่ v1) เพื่อแยกประเภทของไฟล์สำหรับ routing, layout, และโหลดข้อมูล

เหตุผล

ระบบนี้ช่วยให้นักพัฒนาสามารถจัดการ routing ได้ง่าย ๆ โดยไม่ต้องเขียน config routing แยกต่างหาก

---

2. Layout และ Nested Layout

- Layout คือไฟล์ที่กำหนด UI ส่วนที่ใช้ซ้ำ เช่น Header, Footer, Navigation
- การวางไฟล์ +layout.svelte ในโฟลเดอร์ routes จะทำให้ layout นั้นห่อหุ้มหน้าในโฟลเดอร์นั้นและโฟลเดอร์ลูก
- ถ้ามี nested layout จะซ้อนกันตามลำดับ เช่น

src/routes/+layout.svelte -> Layout หลักของแอปทั้งหมด

src/routes/blog/+layout.svelte -> Layout เฉพาะหน้า /blog และลูก

src/routes/blog/[slug]/+layout.svelte -> Layout เฉพาะหน้า /blog/:slug และลูก

- การส่งข้อมูลจาก layout ลงสู่ page หรือ nested layout ทำได้ผ่าน <slot />
- Nested layouts ทำให้สามารถแบ่งส่วน UI ได้ละเอียด และ reuse code ได้ดีมาก

ตัวอย่าง

```
<!-- src/routes/+layout.svelte -->
<header>
  <nav>เมนูหลัก</nav>
</header>
<main>
  <slot /> <!-- หน้าอื่นจะมาแสดงตรงนี้ -->
</main>
<footer>ลิขสิทธิ์</footer>
```

---

3. Dynamic Route

- ไฟล์หรือโฟลเดอร์ที่ชื่อเป็น [param] จะจับ route แบบ dynamic
- เช่น src/routes/blog/[slug].svelte จับ path /blog/my-first-post แล้วค่า slug คือ my-first-post
- ในไฟล์จะสามารถเข้าถึงพารามิเตอร์นี้ได้ผ่าน $page.params.slug หรือผ่าน argument ใน load function
- สามารถใช้หลายพารามิเตอร์ เช่น [category]/[id].svelte ⟶ /news/123 มี category='news' และ id='123'

---

4. การใช้ $page และ $params

- $page คือ Svelte store ที่เก็บสถานะของหน้า เช่น URL, query parameters, route params, status
- ตัวอย่างข้อมูลใน $page

```
{
  url: URL,
  params: { slug: 'my-first-post' },
  route: {...},
  status: 200,
  error: null
}
```

- ใน component สามารถ subscribe ได้แบบ reactive ด้วย $page
- $params มาจาก $page.params ย่อให้สั้นลง

ตัวอย่างใช้งาน

```
<script>
  import { page } from '$app/stores';
  $: slug = $page.params.slug;
</script>

<h1>โพสต์: {slug}</h1>
```

---

5. การโหลดข้อมูลด้วย load()

- load เป็นฟังก์ชันพิเศษใน SvelteKit ที่ทำหน้าที่โหลดข้อมูลก่อน render หน้า
- ช่วยให้ data fetching เป็นระบบมากขึ้น รองรับทั้ง server-side rendering (SSR) และ client-side navigation
- สามารถกำหนดได้ในไฟล์ +page.js/+page.ts หรือใน <script context="module"> ของ +page.svelte

รูปแบบ

```
export async function load({ params, fetch, url, session, stuff }) {
  // params: พารามิเตอร์จาก route เช่น slug
  // fetch: ฟังก์ชัน fetch ที่รองรับ SSR
  // url: URL ปัจจุบัน
  // session: session ของผู้ใช้
  // stuff: ข้อมูลระหว่าง layout/page

  const res = await fetch(`/api/data/${params.slug}`);
  if (!res.ok) {
    return {
      status: res.status,
      error: new Error('ไม่พบข้อมูล')
    };
  }
  const data = await res.json();
  return { props: { data } };
}
```

- ข้อมูลที่ return จาก load จะถูกส่งเข้าไปยัง component เป็น prop ชื่อ data (หรือชื่ออื่น ตาม return)
- ใน component +page.svelte ต้องรับด้วย export let data;

ทำงานอย่างไร

- เมื่อโหลดหน้าแบบ SSR หรือ client-side navigation จะรัน load ก่อน แล้วจึง render
- สามารถใช้เพื่อ prefetch data, ตรวจสอบสิทธิ์, redirect หรือ handle error ได้

สรุป

Feature	ความหมาย	วิธีใช้/ตัวอย่าง
File-based Routing	สร้าง route จากโฟลเดอร์และไฟล์	src/routes/about.svelte ⟶ /about
Layout	UI ซ้ำในหลายหน้า	+layout.svelte กับ <slot />
Nested Layout	Layout ซ้อนกันตามโฟลเดอร์	src/routes/blog/+layout.svelte
Dynamic Route	จับพารามิเตอร์ใน path	[slug].svelte, $page.params.slug

Feature	ความหมาย	วิธีใช้/ตัวอย่าง
$page Store	ข้อมูล route ปัจจุบัน	Reactive store, ใช้ใน component
load()	โหลดข้อมูลก่อน render	Export function ใน +page.js หรือ <script context="module">

โครงสร้างโฟลเดอร์ src/routes ใน SvelteKit

1. หลักการทำงาน

- SvelteKit ใช้ **File-based Routing**
- ทุกไฟล์ .svelte หรือ +page.svelte ในโฟลเดอร์ src/routes จะกลายเป็น **หน้า (route)** ในเว็บโดยอัตโนมัติ
- ชื่อไฟล์และโฟลเดอร์กำหนดเส้นทาง URL ตามโครงสร้างนั้นเลย
- ไฟล์พิเศษ เช่น +layout.svelte, +page.js กำหนด layout และ load data ของ route นั้น ๆ

2. ตัวอย่างโครงสร้างพื้นฐาน

```
src/
└── routes/
    ├── +layout.svelte          // Layout หลักของเว็บ (wrapper รอบทุก route)
    ├── +page.svelte            // หน้าแรก, route "/"
    ├── about/
    │   └── +page.svelte        // หน้า /about
    ├── blog/
    │   ├── +layout.svelte      // Layout เฉพาะ /blog
    │   ├── +page.svelte        // หน้า /blog
    │   └── [slug]/
    │       └── +page.svelte    // หน้า /blog/:slug (dynamic route)
    └── api/
        └── hello.js            // API route /api/hello
```

3. คำอธิบาย

ไฟล์ / โฟลเดอร์	ความหมาย / เส้นทาง URL	หมายเหตุ
+layout.svelte	Layout สำหรับทุกหน้า	ใช้กำหนด UI ซ้ำ เช่น Header

ไฟล์ / โฟลเดอร์	ความหมาย / เส้นทาง URL	หมายเหตุ
+page.svelte	หน้า / (root)	หน้าแรกของเว็บ
about/+page.svelte	หน้า /about	หน้า About
blog/+layout.svelte	Layout เฉพาะ /blog และลูก route	UI ซ้ำเฉพาะ blog เช่น sidebar
blog/+page.svelte	หน้า /blog	หน้าแสดงรายการ blog
blog/[slug]/+page.svelte	หน้า /blog/:slug (dynamic)	dynamic route รับ slug parameter
api/hello.js	API endpoint /api/hello	Server-side function (Backend)

---

4. จุดเด่นของระบบนี้

- **ไม่ต้องกำหนด route ในโค้ด**
  ไม่ต้องเขียน routing config แบบ manual
- **รองรับ nested route อัตโนมัติ**
  โฟลเดอร์ลูกกลายเป็น path ซ้อนกัน เช่น /blog/2023/first-post
- **รองรับ dynamic parameter ง่าย ๆ**
  ใส่ชื่อโฟลเดอร์/ไฟล์ใน [] เพื่อจับค่าพารามิเตอร์ใน URL
- **แยก layout ตามโฟลเดอร์ได้**
  ช่วยแยก UI ซ้ำได้อย่างมีประสิทธิภาพ

---

5. สรุปตัวอย่าง URL ↔ โครงสร้างโฟลเดอร์

URL	โครงสร้างโฟลเดอร์/ไฟล์
/	src/routes/+page.svelte
/about	src/routes/about/+page.svelte
/blog	src/routes/blog/+page.svelte
/blog/my-post	src/routes/blog/[slug]/+page.svelte
/api/hello	src/routes/api/hello.js (API Route)

นี่คือตัวอย่างโปรแกรม SvelteKit แบบเต็มไฟล์และโครงสร้าง พร้อมคำอธิบายโค้ดและผลการรัน แบ่งเป็น

- 3 โปรแกรมพื้นฐาน
- 3 โปรแกรมแนวประยุกต์

ตัวอย่างโปรแกรมพื้นฐาน (3 โปรแกรม)

---

1. หน้าแรก / - หน้า Home ง่าย ๆ

โครงสร้าง

```
src/
└── routes/
    └── +page.svelte
```

src/routes/+page.svelte

```
<script>
  let message = 'ยินดีต้อนรับสู่ SvelteKit!';
</script>

<h1>หน้าแรก</h1>
<p>{message}</p>
```

---

2. หน้า About /about

โครงสร้าง

```
src/
└── routes/
    └── about/
        └── +page.svelte
```

src/routes/about/+page.svelte

```
<h1>เกี่ยวกับเรา</h1>
<p>นี่คือหน้าข้อมูลเกี่ยวกับเว็บไซต์ของเรา</p>
```

---

3. Dynamic Route /blog/[slug]

โครงสร้าง

```
src/
└── routes/
    └── blog/
        └── [slug]/
            └── +page.svelte
```

src/routes/blog/[slug]/+page.svelte

```
<script>
  import { page } from '$app/stores';

  // ดึง slug จาก params
  export let data;

  // หรือใช้ $page.params.slug ได้ใน component
</script>

<h1>บทความ: {data.slug}</h1>
<p>นี่คือเนื้อหาของบทความ "{data.slug}"</p>
```

src/routes/blog/[slug]/+page.js

```
/** @type {import('./$types').PageLoad} */
export function load({ params }) {
  // params.slug มาจาก dynamic route
  return {
    slug: params.slug
  };
}
```

ตัวอย่างโปรแกรมแนวประยุกต์ (3 โปรแกรม)

---

4. Layout ซ้อนกัน + Navigation

โครงสร้าง

```
src/
└── routes/
    ├── +layout.svelte
    ├── +page.svelte
    ├── about/
    │   └── +page.svelte
    └── blog/
        ├── +layout.svelte
        ├── +page.svelte
        └── [slug]/
```

```
        └── +page.svelte
```

src/routes/+layout.svelte

```
<nav>
  <a href="/">หน้าแรก</a> |
  <a href="/about">เกี่ยวกับ</a> |
  <a href="/blog">บทความ</a>
</nav>
<hr />
<slot />
```

src/routes/+page.svelte

```
<h1>ยินดีต้อนรับสู่หน้าแรก</h1>
<p>นี่คือหน้าแรกของเว็บไซต์</p>
```

src/routes/about/+page.svelte

```
<h1>เกี่ยวกับเรา</h1>
<p>ข้อมูลเกี่ยวกับเว็บไซต์และทีมงาน</p>
```

src/routes/blog/+layout.svelte

```
<h2>บทความ (Blog)</h2>
<slot />
```

src/routes/blog/+page.svelte

```
<h3>รายการบทความ</h3>
<ul>
  <li><a href="/blog/post-1">บทความ 1</a></li>
  <li><a href="/blog/post-2">บทความ 2</a></li>
</ul>
```

src/routes/blog/[slug]/+page.svelte

```
<script>
  export let data;
</script>

<h3>บทความ: {data.slug}</h3>
<p>เนื้อหาของบทความ {data.slug}</p>
```

src/routes/blog/[slug]/+page.js

```
/** @type {import('./$types').PageLoad} */
export function load({ params }) {
```

```
  return {
    slug: params.slug
  };
}
```

---

5. โหลดข้อมูลจาก API ด้วย load()

โครงสร้าง

```
src/
└── routes/
    └── users/
        ├── +page.svelte
        └── +page.js
```

src/routes/users/+page.js

```
/** @type {import('./$types').PageLoad} */
export async function load({ fetch }) {
  const res = await fetch('https://jsonplaceholder.typicode.com/users?_limit=5');
  const users = await res.json();

  return { users };
}
```

src/routes/users/+page.svelte

```
<script>
  export let data;
</script>

<h1>รายชื่อผู้ใช้</h1>

<ul>
  {#each data.users as user}
    <li>{user.name} ({user.email})</li>
  {/each}
</ul>
```

---

6. Dynamic Route + Load API + Error Handling

โครงสร้าง

```
src/
└── routes/
    └── posts/
        └── [id]/
            ├── +page.js
            └── +page.svelte
```

src/routes/posts/[id]/+page.js

```
/** @type {import('./$types').PageLoad} */
export async function load({ params, fetch }) {
  const res = await fetch(`https://jsonplaceholder.typicode.com/posts/${params.id}`);

  if (res.ok) {
    const post = await res.json();
    return { post };
  } else {
    return {
      status: res.status,
      error: new Error('ไม่พบโพสต์นี้')
    };
  }
}
```

src/routes/posts/[id]/+page.svelte

```
<script>
  export let data;
</script>

{#if data.error}
  <p style="color:red">Error: {data.error.message}</p>
{:else}
  <h1>{data.post.title}</h1>
  <p>{data.post.body}</p>
{/if}
```

สรุปผลการรัน

โปรแกรม	ผลลัพธ์
1	หน้าแรกแสดงข้อความต้อนรับ
2	หน้า /about แสดงข้อมูลเกี่ยวกับ
3	หน้า /blog/[slug] แสดง slug ที่ dynamic
4	ระบบ navigation พร้อม layout ซ้อนกัน แสดงเมนูและเนื้อหาแต่ละหน้า
5	โหลดข้อมูลผู้ใช้จาก API แสดงรายชื่อ
6	โหลดโพสต์ตาม id จาก API แสดงโพสต์หรือ error

Layout, Nested Layout, Dynamic Route ใน SvelteKit

---

1. Layout

- คือไฟล์ที่ใช้กำหนดโครงสร้างหน้า (เช่น header, footer, sidebar) ที่จะใช้ซ้ำในหลายหน้า
- ใน SvelteKit ใช้ไฟล์ชื่อ +layout.svelte ในโฟลเดอร์ src/routes หรือโฟลเดอร์ลูกเพื่อสร้าง layout
- หน้าอื่นในโฟลเดอร์นั้นจะถูกแทรกเข้ามาใน <slot /> ของ layout
- ช่วยลดการเขียนซ้ำและจัดการ UI ส่วนกลางง่ายขึ้น

ตัวอย่าง

```
src/routes/
├── +layout.svelte
├── +page.svelte
└── about/
    └── +page.svelte
```

+layout.svelte

```
<header>
  <h1>My Website</h1>
  <nav>
    <a href="/">Home</a> |
    <a href="/about">About</a>
  </nav>
</header>
```

```
<main>
  <slot /> <!-- หน้าอื่น ๆ จะมาแสดงตรงนี้ -->
</main>

<footer>© 2025 My Website</footer>
```

+page.svelte (หน้าแรก)

```
<h2>Welcome to the homepage!</h2>
```

about/+page.svelte

```
<h2>About Us</h2>
<p>This is the about page.</p>
```

---

2. Nested Layout

- SvelteKit รองรับการซ้อน layout กันได้หลายชั้นตามโครงสร้างโฟลเดอร์
- layout ที่อยู่ในโฟลเดอร์ย่อยจะทำหน้าที่เป็น layout เฉพาะสำหรับ route นั้นและลูก
- layout ชั้นบนยังคงทำงานและห่อ layout ชั้นล่างอยู่
- ทำให้แบ่ง UI เป็นส่วน ๆ เช่น layout หลัก (header/footer) และ layout ย่อย (sidebar เฉพาะ section)

ตัวอย่างโครงสร้าง

```
src/routes/
├── +layout.svelte          <-- layout หลัก
├── +page.svelte            <-- หน้า /
└── blog/
    ├── +layout.svelte      <-- layout ย่อยเฉพาะ /blog
    ├── +page.svelte        <-- หน้า /blog
    └── [slug]/
        └── +page.svelte    <-- หน้า /blog/:slug
```

src/routes/blog/+layout.svelte

```
<aside>
  <h3>Blog Sidebar</h3>
  <ul>
    <li><a href="/blog/post-1">Post 1</a></li>
    <li><a href="/blog/post-2">Post 2</a></li>
  </ul>
</aside>
```

```
<section>
  <slot /> <!-- เนื้อหาของ blog จะมาแสดงตรงนี้ -->
</section>
```

---

3. Dynamic Route

- ใช้เครื่องหมาย [param] ในชื่อไฟล์หรือโฟลเดอร์เพื่อจับพารามิเตอร์ใน URL
- ค่า param จะถูกส่งเข้า load() function และเก็บใน $page.params
- เหมาะสำหรับ route ที่ข้อมูลไม่ตายตัว เช่น /blog/my-post, /user/123

ตัวอย่าง

src/routes/blog/[slug]/+page.svelte

```
<script>
  import { page } from '$app/stores';
  // ดึง slug จาก params
  $: slug = $page.params.slug;
</script>

<h2>บทความ: {slug}</h2>
<p>เนื้อหาของบทความ {slug} จะถูกแสดงที่นี่</p>
```

สรุป

Feature	อธิบาย	ตัวอย่างไฟล์
Layout	UI ซ้ำในหลายหน้า	src/routes/+layout.svelte
Nested Layout	Layout ซ้อนกันตามโฟลเดอร์	src/routes/blog/+layout.svelte
Dynamic Route	route พารามิเตอร์	src/routes/blog/[slug]/+page.svelte

---

นี่คือตัวอย่างโปรแกรม SvelteKit ที่เน้น **Layout, Nested Layout, Dynamic Route** แบบเต็มไฟล์ พร้อมโครงสร้างและคำอธิบาย แบ่งเป็น

- 3 โปรแกรมพื้นฐาน
- 3 โปรแกรมแนวประยุกต์

ตัวอย่างโปรแกรมพื้นฐาน (3 โปรแกรม)

---

1. Layout หลัก (Simple Layout)

โครงสร้าง

```
src/
└── routes/
    ├── +layout.svelte
    └── +page.svelte
```

src/routes/+layout.svelte

```
<header style="background:#eee;padding:1em;">
  <h1>My Website</h1>
  <nav>
    <a href="/">Home</a> |
    <a href="/about">About</a>
  </nav>
</header>

<main style="padding:1em;">
  <slot />
</main>

<footer style="background:#eee;padding:1em;margin-top:2em;text-align:center;">
  © 2025 My Website
</footer>
```

src/routes/+page.svelte

```
<h2>Welcome to the homepage!</h2>
<p>This is the home page content.</p>
```

---

2. Nested Layout /blog

โครงสร้าง

```
src/
└── routes/
    ├── blog/
    │   ├── +layout.svelte
    │   └── +page.svelte
```

└── +layout.svelte (จากตัวอย่างข้อ 1)

src/routes/blog/+layout.svelte

```
<div style="display:flex;">
  <aside style="width:200px;padding:1em;background:#f4f4f4;">
    <h3>Blog Sidebar</h3>
    <ul>
      <li><a href="/blog/post-1">Post 1</a></li>
      <li><a href="/blog/post-2">Post 2</a></li>
    </ul>
  </aside>

  <section style="flex:1;padding:1em;">
    <slot />
  </section>
</div>
```

src/routes/blog/+page.svelte

```
<h2>Blog Homepage</h2>
<p>รายการบทความล่าสุด</p>
```

---

3. Dynamic Route /blog/[slug]

โครงสร้าง

```
src/
└── routes/
    └── blog/
        ├── [slug]/
        │   └── +page.svelte
        └── +layout.svelte (จากข้อ 2)
```

src/routes/blog/[slug]/+page.svelte

```
<script>
  import { page } from '$app/stores';
  $: slug = $page.params.slug;
</script>

<h2>บทความ: {slug}</h2>
```

```
<p>นี่คือเนื้อหาของบทความ "{slug}".</p>
```

ตัวอย่างโปรแกรมแนวประยุกต์ (3 โปรแกรม)

---

4. Layout + Nested Layout + Dynamic Route + Load Data จาก API

โครงสร้าง

```
src/
└── routes/
    ├── +layout.svelte
    ├── blog/
    │   ├── +layout.svelte
    │   ├── +page.svelte
    │   └── [slug]/
    │       ├── +page.svelte
    │       └── +page.js
```

src/routes/+layout.svelte

(เหมือนข้อ 1)

src/routes/blog/+layout.svelte

(เหมือนข้อ 2)

src/routes/blog/+page.svelte

```
<h2>รายการบทความ</h2>
<ul>
  <li><a href="/blog/post-1">post-1</a></li>
  <li><a href="/blog/post-2">post-2</a></li>
</ul>
```

src/routes/blog/[slug]/+page.js

```
/** @type {import('./$types').PageLoad} */
export async function load({ params, fetch }) {
  const res = await fetch(`https://jsonplaceholder.typicode.com/posts/1`);
  if (!res.ok) throw new Error('ไม่พบโพสต์');

  const post = await res.json();
```

```
  return { post };
}
```

src/routes/blog/[slug]/+page.svelte

```
<script>
  export let data;
</script>

<h2>{data.post.title}</h2>
<p>{data.post.body}</p>
```

---

5. Nested Layout + Authentication Simulation

โครงสร้าง

```
src/
└── routes/
    ├── +layout.svelte
    ├── dashboard/
    │   ├── +layout.svelte
    │   └── +page.svelte
```

src/routes/+layout.svelte

```
<nav>
  <a href="/">Home</a> |
  <a href="/dashboard">Dashboard</a>
</nav>
<hr />
<slot />
```

src/routes/dashboard/+layout.svelte

```
<script>
  let loggedIn = false;

  if (!loggedIn) {
    // ถ้าไม่ login ให้ redirect ไปหน้า home
    window.location.href = '/';
  }
</script>
```