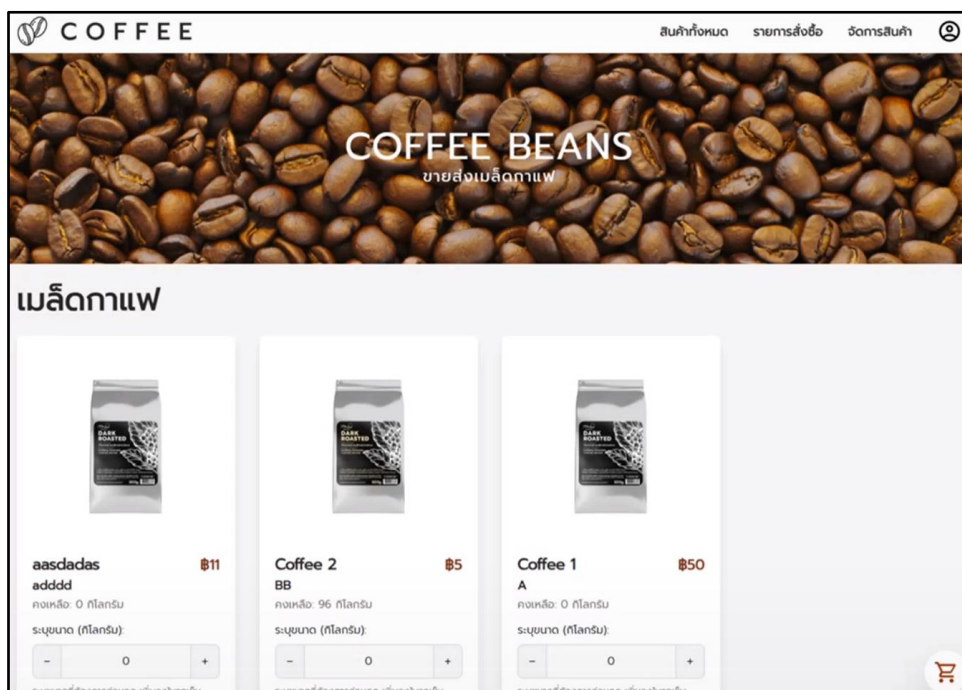
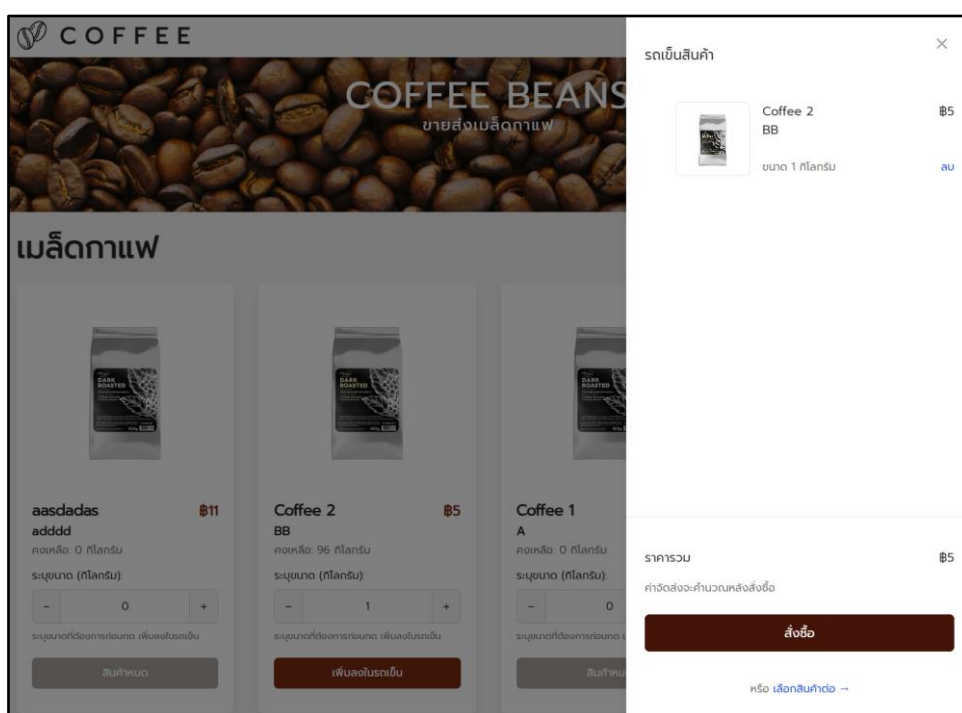


ใบงาน



รูปที่ 1 ตัวอย่างแอปพลิเคชันสั่งกาแฟ



รูปที่ 2 ส่งคำสั่งซื้อ

COFFEE สินค้าทั้งหมด รายการสั่งซื้อ จัดการสินค้า

สั่งซื้อ

Coffee 2 BB
จำนวน 1 กิโลกรัม
ราคาต่อหน่วย(กิโลกรัม) 5

5

รายละเอียดการสั่งซื้อ
ตรวจสอบรายละเอียดการสั่งซื้อให้ครบถ้วน

ชื่อ นามสกุล

นายทอง ใจดี

ที่อยู่ในการจัดส่ง

รหัสไปรษณีย์ เลือกจังหวัด เลือกอำเภอ เลือกตำบล เลือกหมู่บ้าน

ราคารวม 5

ค่าจัดส่ง 0

ราคาส่ง 5

สั่งซื้อสินค้า

รูปที่ 3 สั่งซื้อเมล็ดกาแฟ

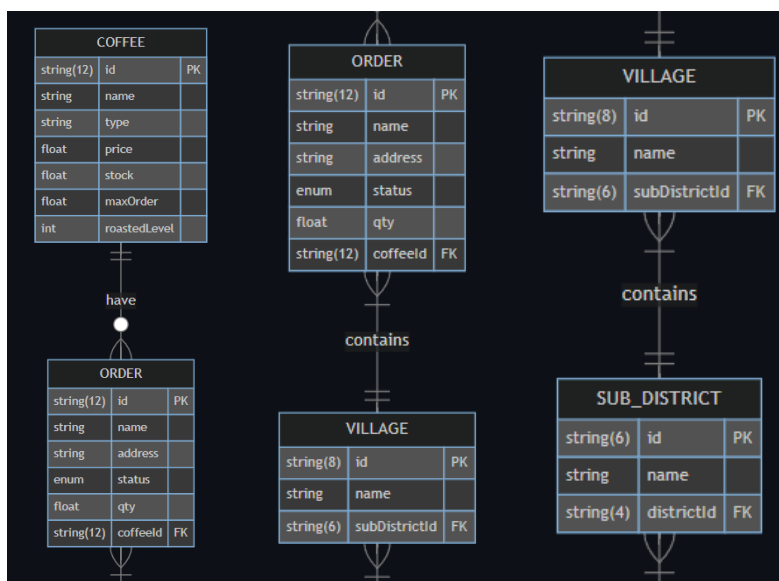
COFFEE สินค้าทั้งหมด รายการสั่งซื้อ จัดการสินค้า

รายการสั่งซื้อ

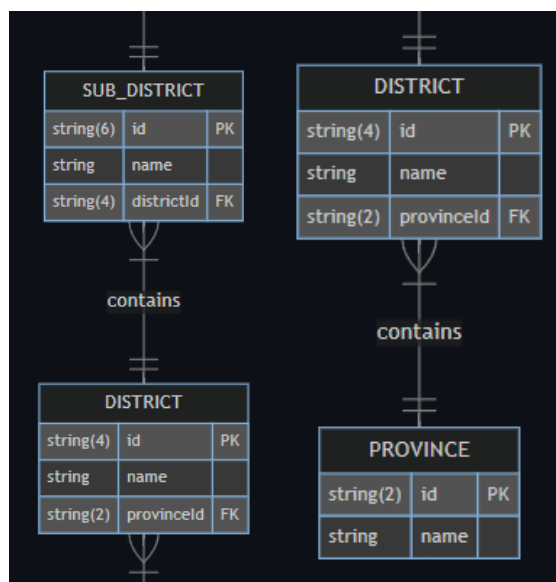
ชื่อ	ที่อยู่	สินค้า	ขนาด (กิโลกรัม)	สถานะ
edm	บางนาแผล ด.กานัง อ.กานัง จ.ยะลา	Coffee 2	2	PENDING
asd	ด.โละปาเนะ ด.บุโง อ.กรงปินัง จ.ยะลา	Coffee 2	2	PENDING
asdasd	บ้านลิ้นด ด.สำโรงเหนือ อ.เมืองสมุทรปราการ จ.สมุทรปราการ	Coffee 2	2	CANCELED
Customer	บ้านลิ้นด ด.สำโรงเหนือ อ.เมืองสมุทรปราการ จ.สมุทรปราการ	Coffee 1	5	ACCEPTED

รูปที่ 4 จัดการคำสั่งซื้อ

ในงานนี้จัดทำเพื่อให้เข้าใจหลักการทำงานของ Front-End, Back-End และการใช้งาน PrismaORM และควบคุม Racing Conditions ในขณะติดต่อกับฐานข้อมูลโดยใช้ Transaction เพื่อให้ข้อมูลสอดคล้องกัน เมื่อมีใช้งานพร้อมกัน โดยระบบจะรองรับระบบ เพิ่ม ลบ แก้ไข เมล็ดกาแฟ ส่งคำสั่งซื้อและจัดการคำสั่งซื้อ และใช้ Web Socket เพื่อให้สามารถแสดงผลการเปลี่ยนแปลงของข้อมูลให้เป็นปัจจุบัน โดยมี ER-Diagram ดังนี้



รูปที่ 5 ER-Diagram (1)



รูปที่ 6 ER-Diagram (2)

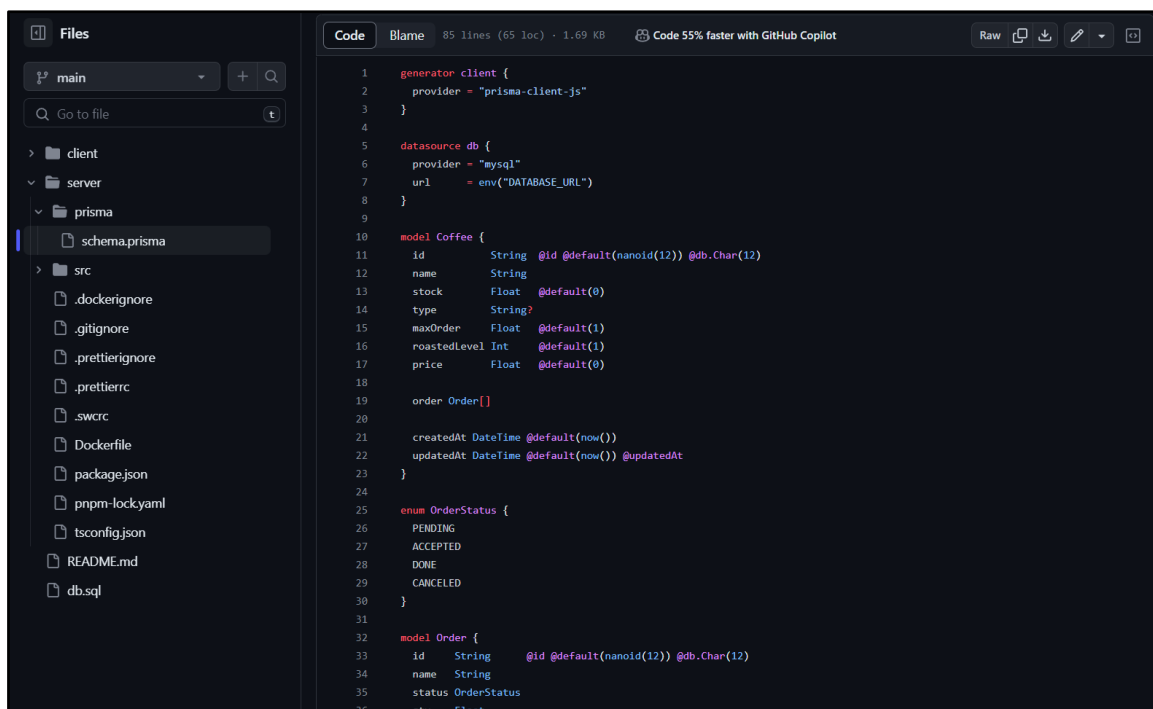

```
MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0.001 sec)

MariaDB [(none)]> create database coffee;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| coffee |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.002 sec)

MariaDB [(none)]> |
```

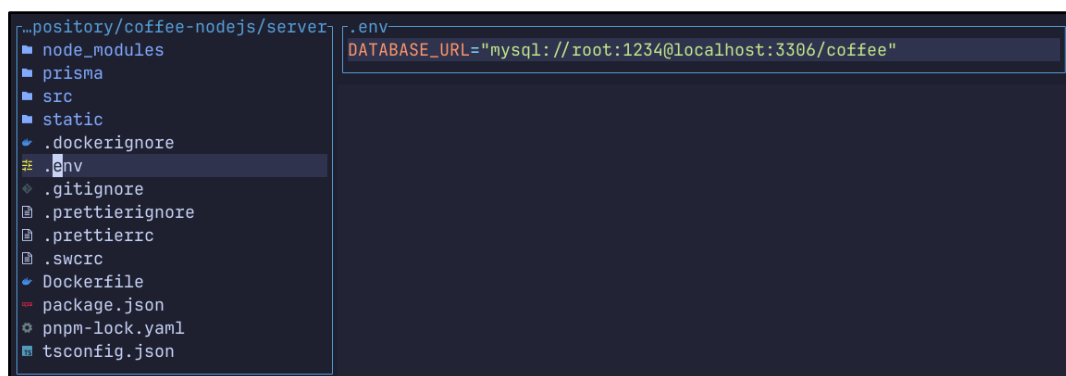
รูปที่ 10 สร้างฐานข้อมูล



รูปที่ 11 โครงสร้างฐานข้อมูล (Database Design)

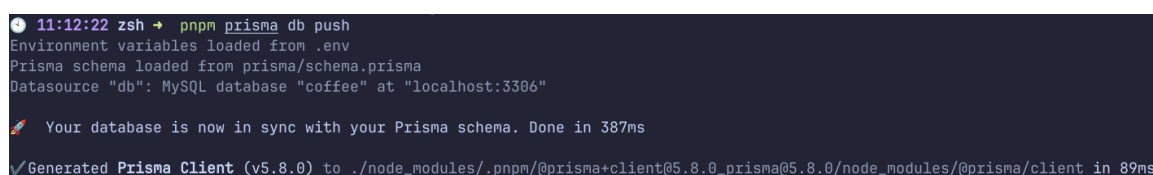
5. เตรียมโครงสร้างให้ตรงกับ Database Design ที่ทำไว้ใน server/prisma/schema.prisma ดังรูปที่ 5 ซึ่งมีขั้นตอนดังนี้

5.1 สร้างไฟล์ server/.env โดยมีเนื้อหาดังรูปที่ 6 โดยเปลี่ยนชื่อผู้ใช้ (root) รหัสผ่าน (1234) และฐานข้อมูล (coffee) ให้ถูกต้อง



รูปที่ 12 ค่าการเชื่อมต่อฐานข้อมูล

5.2 ใช้คำสั่ง `pnpm run db:push` หรือ `pnpm prisma db push` หรือ `npx prisma db push`



รูปที่ 13 สร้างโครงสร้างฐานข้อมูล

6. ทดลองระบบหลังบ้าน ในการทำงานของระบบนั้นจะใช้ Package ที่ชื่อว่า tsx ในการสั่งให้ TypeScript ทำงาน แต่เมื่อต้องการนำไปใช้ใน Production ปกติจะต้องทำการ Build เป็น JavaScript ก่อน จะไม่ใช่ TypeScript แบบตรงๆ

```
methapon in repository/coffee-nodejs/server on  class/partial-remove [$] is  v1.0.0 via  v20.11.0
11:44:44 zsh → pn run dev

> coffee-nodejs@1.0.0 dev /home/methapon/repository/coffee-nodejs/server
> tsx watch ./src/app.ts

Listening on: http://0.0.0.0:3000 11:44:46 AM
```

รูปที่ 14 ระบบหลังบ้านทำงาน

7. การเขียนระบบ API หรือ Rest API ในระบบนี้ตัวอย่างใช้ oak จาก deno (เป็นเพียง 1 ใน Package ตัวเลือกที่สามารถนำมาใช้ได้) ทั้งนี้หากต้องการสร้างขึ้นมาเองก็สามารถเลือกได้จากตัวเลือกต่างๆ เช่น express, koa, fastify, elysia(bun), oak(deno), hono(deno) และอื่นๆ ซึ่งส่วนใหญ่การเลือกใช้งานจะขึ้นอยู่กับ Performance และ Plugins ที่เพิ่มความสามารถให้กับ Framework นั้นๆ

7.1 การเขียน Endpoint สำหรับดึงข้อมูลจะมีหลักๆ อยู่ 5 Method คือ GET, POST, PUT, PATCH และ DELETE โดยการใช้งานจะแตกต่างกันดังนี้

7.1.1 GET Method จะใช้สำหรับดึงข้อมูล

7.1.2 POST Method สำหรับให้เพิ่มข้อมูลใน Rest API หรืออาจใช้เป็น Action ใน API

7.1.3 PUT Method สำหรับแทนที่หรือแก้ไขข้อมูล

7.1.4 PATCH Method สำหรับแก้ไขบางส่วนของข้อมูล

7.1.5 DELETE Method สำหรับลบ

เมื่อทำการเขียน Endpoint แล้วสามารถทดสอบได้โดยการใช้ Postman หรือ HTTPie ในการเรียกใช้ API ตาม Method และเส้นทางที่ตั้งค่าไว้ เช่น ในรูปที่ 9 สามารถเข้าได้โดยใช้ Method GET ในโปรแกรม `http://localhost:3000/coffee/<รหัสของข้อมูล>`

สามารถดูเส้นทางทั้งหมดได้ภายใน `server/src/routes`

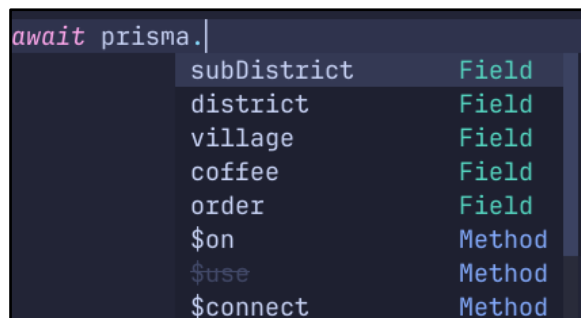
```
1 router.get("/coffee/:id", async (ctx) => {
2   | // do something
3 });
```

รูปที่ 15 Endpoint ของระบบ

7.2 การติดต่อกับฐานข้อมูลด้วย PrismaORM ซึ่งเมื่อเราได้ดำเนินการในขั้นตอนที่ 5 ไปแล้ว เราจะสามารถเขียนติดต่อกับฐานข้อมูลได้โดยใช้ Prisma Client

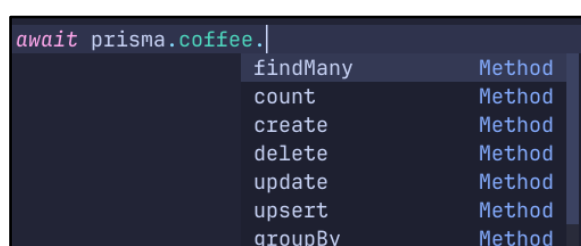
```
1 const prisma = new PrismaClient({
2   | errorFormat: "minimal",
3 });
```

รูปที่ 16 Prisma Client สำหรับติดต่อกับฐานข้อมูล

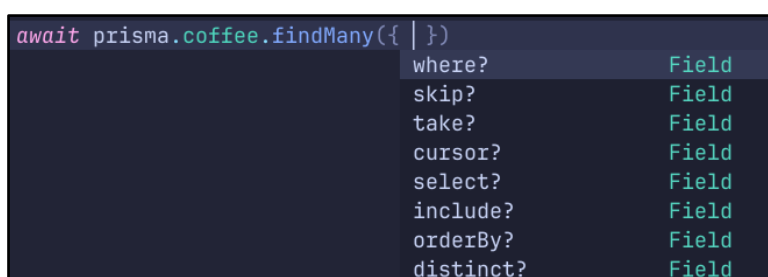


รูปที่ 17 ตัวอย่างการเขียน Prisma

เมื่อเขียนโค้ดใน IDE ที่มี Auto-Completion ให้จะแสดง entity ต่างๆ ที่สามารถเข้าถึงได้มาให้ ซึ่งข้อมูลสำหรับ Auto-Completion นี้ถูกดึงมาจาก Type ที่ได้ในขั้นตอนที่ 6

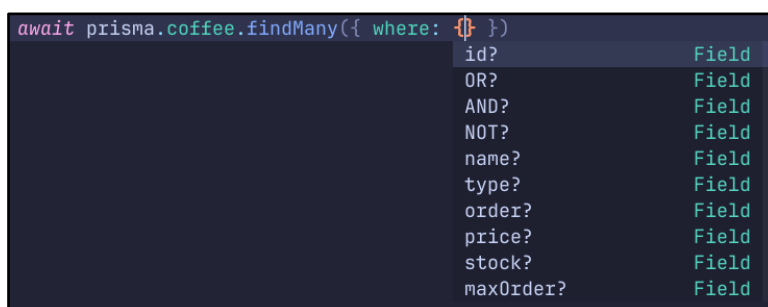


รูปที่ 18 ตัวอย่างคำสั่งต่างๆ ที่สามารถทำได้ใน PrismaORM



รูปที่ 19 ตัวอย่างการดึงข้อมูล

การดำเนินการต่างๆ สามารถทำได้โดยการส่ง Parameter สำหรับควบคุมการกระทำนั้นๆ ตามค่าดังรูปที่ 12



รูปที่ 20 ตัวอย่างการดึงข้อมูลโดยใช้ Where


```

await prisma.coffee.findMany({
  where: {
    name: {},
  },
});

```

gt?	Field
in?	Field
lt?	Field
gte?	Field
lte?	Field
not?	Field
notIn?	Field
equals?	Field
contains?	Field

รูปที่ 21 ตัวอย่างตัวดำเนินการเปรียบเทียบ

การใช้งานเพิ่มเติมสามารถอ่านเพิ่มเติมได้ที่ <https://www.prisma.io/docs/orm/prisma-client/queries> ทั้งนี้ การเขียนโปรแกรมเราสามารถพึ่งพาระบบ Type และ Auto-Completion ได้ซึ่งช่วยให้ไม่จำเป็นต้องค้นหาการใช้งานบนเว็บ เนื่องจากบาง Function ใน Package ต่างๆ จะมีการอธิบายไว้ว่าใช้ยังไง ทั้งนี้ ขึ้นอยู่กับ Package นั้นๆ ด้วยว่าได้อธิบายไว้หรือไม่

```

router.get("/coffee/:id", async (ctx) => {
  const schema = z.object({ id: z.string().length(12) });
  const validate = await schema.safeParseAsync({ id: ctx.params.id });

  if (!validate.success) return ctx.throw(Status.UnprocessableEntity, "Invalid");

  const found = await prisma.coffee.findUnique({ select: COFFEE_SELECT, where: validate.data });

  if (found) {
    ctx.response.status = 200;
    ctx.response.body = found;
  }
});

router.post("/coffee", async (ctx) => {
  const coffee = z.object({
    name: z.string(),
    price: z.number(),
    stock: z.number().optional(),
    maxOrder: z.number().optional(),
  });
});

```

```

(property) data: {
  id: string;
}

(alias) const COFFEE_SELECT: {
  createdAt: true;
  updatedAt: true;
  id: true;
  name: true;
  stock: true;
  maxOrder: true;
  roastedLevel: true;
  price: true;
  type: true;
}
import COFFEE_SELECT

```

รูปที่ 22 ตัวอย่างการดึงข้อมูลโดยเลือก Field และ เงื่อนไขที่ id ตรงกัน

```

router.post("/coffee", async (ctx) => {
  const coffee = z.object({
    name: z.string(),
    price: z.number(),
    stock: z.number().optional(),
    maxOrder: z.number().optional(),
    roastedLevel: z.number().optional(),
    type: z.string(),
  });
  const schema = z.union([coffee, coffee.array()]);

  const payload = await ctx.request.body().value;
  const validate = await schema.safeParseAsync(payload);

  if (!validate.success) return ctx.throw(Status.UnprocessableEntity, "Invalid Body");
  if (!Array.isArray(validate.data)) {
    await prisma.coffee.create({
      data: validate.data,
    });
  } else {
    await prisma.coffee.createMany({
      data: validate.data.map((property) data: {
        name: string;
        type: string;
        price: number;
        stock?: number | undefined;
        maxOrder?: number | undefined;
        roastedLevel?: number | undefined;
      }
    });
  }
  console.success("In");
  ctx.response.status = 204;
});

```

รูปที่ 23 ตัวอย่างการเพิ่มข้อมูล

8. เมื่อทุกอย่างพร้อมแล้วให้ทำการนำเข้าข้อมูลที่มีอยู่ก่อนแล้ว เข้าสู่ฐานข้อมูล (coffee) โดยสามารถใช้เครื่องมือที่ถนัดเช่น PhpMyAdmin ที่มากับ XAMPP, MySQL Workbench หรือ Beekeeper Studio ก็ได้ ในตัวอย่างจะใช้ CLI โดยใช้คำสั่งดังนี้

```

methapon in ~/repository/coffee-nodejs on main [$]
14:00:46 zsh → mariadb -uroot -p coffee < ./db.sql

```

รูปที่ 24 การนำเข้าข้อมูล

```

MariaDB [coffee]> show tables;
+-----+
| Tables_in_coffee |
+-----+
| Coffee            |
| District          |
| Order             |
| Province          |
| SubDistrict       |
| Village           |
+-----+
6 rows in set (0.000 sec)

MariaDB [coffee]> select count(*) from Village;
+-----+
| count(*) |
+-----+
|      78020 |
+-----+
1 row in set (0.011 sec)

```

รูปที่ 25 ข้อมูลหลังการนำเข้า

ข้อที่ 1) ให้ทำการเขียนโค้ดในการเพิ่มข้อมูลชนิดกาแฟ ใน `server/src/routes/coffee.ts` บรรทัดที่ 54 ถึง 59 ใช้ PrismaORM โดยใช้ Method ที่ชื่อว่า create และ createMany อ่านเพิ่มเติมได้ที่

- <https://www.prisma.io/docs/orm/prisma-client/queries/crud#create>
- <https://www.prisma.io/docs/orm/prisma-client/queries/crud#create-multiple-records>

```
16 router.post("/coffee", async (ctx) => {
17   const coffee = z.object({
18     name: z.string(),
19     price: z.number(),
20     stock: z.number().optional(),
21     maxOrder: z.number().optional(),
22     roastedLevel: z.number().optional(),
23     type: z.string(),
24   });
25   const schema = z.union([coffee, coffee.array()]);
26
27   const payload = await ctx.request.body().value;
28   const validate = await schema.safeParseAsync(payload);
29
30   if (!validate.success) return ctx.throw(Status.UnprocessableEntity, "Invalid Body");
31
32   // check if input is array or not
33   if (true || false) {
34     // Add data to database
35   } else {
36     // Add multiple data to database
37   }
38
39   console.success("Insert Success");
40   ctx.response.status = 204;
41 });
```

การทดสอบ Endpoint สามารถทดสอบได้ตามรูปต่อไปนี้

The first screenshot shows a POST request to `http://localhost:3000/coffee` with a JSON body containing coffee details. The response is a 204 status code, indicating successful creation without content.

The second screenshot shows a GET request to `http://localhost:3000/coffee`. The response is a 200 status code with a JSON array containing the created coffee record, including an auto-generated ID and timestamps.

ข้อที่ 2) ให้ทำการเขียนโค้ดในการเพิ่มข้อมูล ใน `server/src/routes/order.ts` บรรทัดที่ 52 ถึง 56 ใช้ **PrismaORM** โดยใช้ **Method** ที่ชื่อว่า **update** และ **create** ภายใน **transaction** โดยมีเงื่อนไขว่าเมื่อทำการสั่งซื้อ (order) แล้วหากจำนวน stock ไม่เพียงพอต่อจำนวนที่สั่งซื้อ คำสั่งซื้อนั้นจะถูกปฏิเสธและการดำเนินการใดๆ ที่เกิดขึ้น เช่นการ update จำนวนสินค้าในสต็อก จะต้องกลับไปเป็นเหมือนเดิมก่อนการสั่งซื้อ และแจ้ง Error ไปโดยมี Status Code เป็น 406 (NotAcceptable) พร้อมข้อความ “Not Enough Quantity”

ตัวอย่างอันกอร์ริมีเช่น

Stock: 5

Order: 10

1. Order < Stock
2. IF TRUE: Stock = Stock – Order
3. ELSE: Throw Error 406

(ตัวอย่างการใช้งาน Transaction อยู่ในบรรทัดที่ 82 ถึง 104 ในไฟล์เดียวกันเมื่อยังไม่มีการแก้ไข ซึ่งในตัวอย่างดังกล่าวเป็นการตรวจสอบสถานะปัจจุบัน)

```

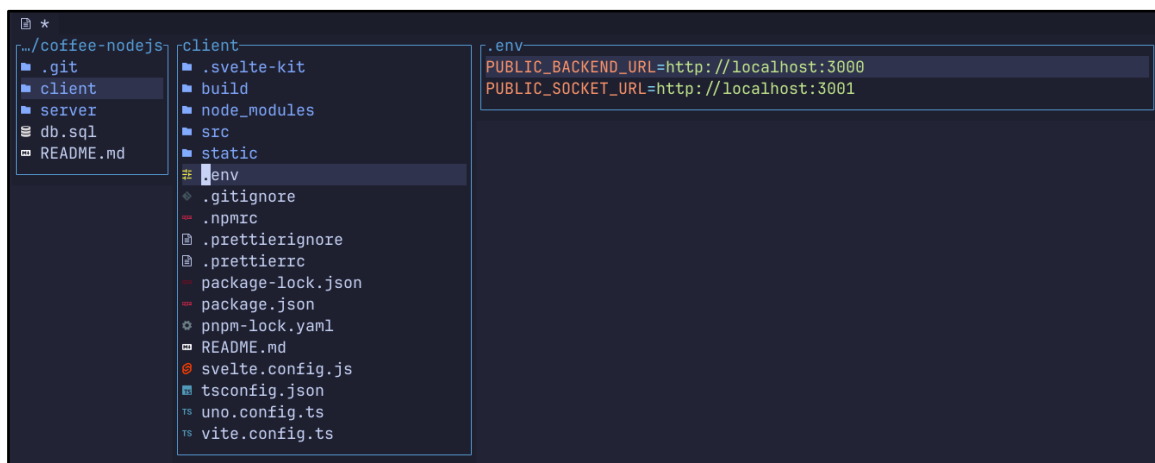
11 router.post("/order", async (ctx) => {
10   const schema = z.object({
9     name: z.string(),
8     qty: z.number().gt(0),
7     coffeeId: z.string().length(12),
6     villageId: z.string().length(8),
5   });
4   const validate = await schema.safeParseAsync(await ctx.request.body().value);
3
2   if (!validate.success) return ctx.throw(Status.UnprocessableEntity, "Invalid Body");
1
52  // prevent racing condition - do transaction You, 2024-01-19 - Initial commit
1   await prisma.$transaction(async (tx) => {
2     // implement transaction
3   });
4
5   ws.io.emit("OrderReceived");
6
7   console.success("Order Success");
8   ctx.response.status = 204;
9 });

```

การใช้ Transaction จะช่วยให้ฐานข้อมูลทำตามคำสั่งย่อยๆ โดยที่เมื่อมีข้อผิดพลาดหรือเงื่อนไขบางอย่างที่ทำให้ Transaction นั้นไม่สำเร็จ จะทำให้ย้อนกลับไปเป็นเหมือนก่อนเกิด Transaction อีกทั้ง Transaction จะเป็นการกระทำคำสั่งทั้งหมด เหมือนเป็นคำสั่งเดียวและไม่ถูกแทรกแซงซึ่งเป็นการป้องกันไม่ให้กระทำกับข้อมูลดังกล่าวในเวลาเดียวกัน (Lock) ซึ่งจะป้องกัน Racing Condition ได้

การทดสอบสามารถทำได้เหมือนข้อแรก ให้ลองทำการ Order และลองใส่จำนวนเกินกว่าที่มีใน Stock และเรียกใช้ Endpoint โดยมีข้อมูลคือ name, qty, coffeeId (ใช้ตัวอย่างจากข้อ 1 ที่ทำเองและได้รับ id มา) และ villageId (ตัวอย่างเช่น 50010701)

9. ให้ทำการสร้างไฟล์ `client/.env` โดยมีเนื้อหา ดังนี้



รูปที่ 26 ไฟล์ .env ในส่วนของ client

10. โครงสร้างไฟล์ของ SvelteKit จะมีการสร้างหน้าใหม่ไว้ใน `routes/+page.svelte` เมื่อต้องการสร้างหน้าใหม่ให้สร้างเพิ่มซึ่งจะเป็น URL หรือ path ที่จะเข้าถึงจากหน้าเว็บ และ `+page.svelte` ที่จะเป็นเนื้อหาภายในหน้านั้นๆ



รูปที่ 27 ไฟล์ .env ในส่วนของ client

```

1 let { ...props } = $props<ActionAdd | ActionEdit>();
2
3 let name = $state((props.action === 'edit' && props.name) || '');
4 let stock = $state<number>();
5 let maxOrder = $state<number>();
6 let roastedLevel = $state<number>();
7 let price = $state<number>();
8 let type = $state<string>();
9 let open = $state<boolean>(false);

```

รูปที่ 28 ตัวอย่าง State ใน Svelte 5

จาก รูปที่ 28 เป็นตัวแปรที่เก็บ State ซึ่งเมื่อตัวแปรมีการเปลี่ยนแปลง และมีการนำตัวแปรดังกล่าว ไปแสดงผลภายใน HTML จะทำให้ HTML Element ที่มีความเกี่ยวข้องกับตัวแปรดังกล่าว มีการเปลี่ยนแปลง ตาม ขึ้นอยู่กับว่าใช้ทำอะไร เช่นใช้ควบคุมแสดงผลด้วย if else หรือใช้ แสดงข้อความ ฯลฯ

ณ เวลาที่เขียน ใช้ Svelte 5 ซึ่งเป็น Version ใหม่ที่ยังไม่พร้อมใช้งานใน Production มีความต่างกับ Version 4 คือต้องประกาศ \$state ในขณะที่ Version 4 ไม่จำเป็น สามารถอ่านเพิ่มเติมได้ที่ <https://svelte-5-preview.vercel.app/docs/introduction>

ใน Framework อย่างเช่น React, Vue และ Svelte จะมีสิ่งที่เรียกว่า State ซึ่งเป็นเหมือนกับ สถานะที่จะสามารถใช้ในการควบคุมการแสดงผลบนหน้าเว็บเมื่อมีการเปลี่ยนแปลงได้

```

1 {#if coffee.length > 0}
2   {#each coffee as item (item.id)}
3     <tr class="bg-white border-t border-stone-200 hover:bg-stone-100">
4       <td class="px-6 py-4"> {item.name} </td>
5       <td class="px-6 py-4"> {item.type} </td>
6       <td class="px-6 py-4"> {item.roastedLevel} </td>
7       <td class="px-6 py-4"> {item.stock} </td>
8       <td class="px-6 py-4"> {item.maxOrder} </td>
9       <td class="px-6 py-4"> {item.price} </td>
10      <td class="px-6 py-4">
11        <CoffeeModal action="edit" {...item} onAction={() => invalidateAll()} />
12        <DeleteCoffeeModal id={item.id} onDelete={() => invalidateAll()} />
13      </td>
14    </tr>
15  {/each}
16 {/if}
17
18 {#else}
19   <tr class="bg-white border-t border-stone-200 hover:bg-stone-100">
20     <td colspan="5" class="px-6 py-4 text-center font-medium whitespace-nowrap">
21       >no items</td>
22   </tr>
23 {/if}

```

รูปที่ 29 ตัวอย่างการแสดงผล ตัวแปร ควบคุมด้วย if else และ loop

จาก รูปที่ 29 จะเป็นการควบคุมการแสดงผลด้วย if else และ each กับตัวแปร coffee ซึ่งเป็น State การแสดงผลจะเปลี่ยนแปลงตามตัวแปรที่เกี่ยวข้องซึ่งก็คือ coffee

ภายใน each ที่เป็น loop จะมีการแสดงผลเปลี่ยนแปลงตามค่าที่อยู่ในตัวแปร item ที่อยู่ใน coffee อีกทีนี้

```

1  <input
2    required
3    bind:value={name}
4    type="text"
5    id="name"
6    name="name"
7    class="w-full rounded-md
8    placeholder="ชื่อบุคคล"
9

```

รูปที่ 30 การนำ State ไปใช้งานกับ Input

ในส่วนของ Input เนื่องจากการทำงานโดยใช้ JavaScript Framework ส่วนใหญ่มักจะมีการนำข้อมูลที่กรอกเก็บไว้ในตัวแปร ซึ่งหากเก็บไว้ในตัวแปรปกติ จะทำให้ ไม่ว่าพิมพ์อะไรเข้าไป ก็จะทำให้ Input กลับมาเหมือนเดิมหรือค่าที่กรอกไม่ถูกเก็บไว้ในตัวแปร เนื่องจาก ตัวแปรนั้นไม่ใช่ State ทำให้ไม่เกิด Reactive

ข้อที่ 3) ให้สร้าง **State** ใน `client/src/routes/manage/coffee-modal.svelte` บรรทัดที่ 26 ถึง 30 สำหรับเก็บข้อมูล Input สำหรับ **สร้าง** และ **แก้ไข** ข้อมูล โดยเมื่อเป็นการแก้ไขข้อมูล จะต้องมีการมีค่าเริ่มต้นให้ โดยรับค่ามาจากข้อมูลที่ต้องการแก้ไข (ข้อมูลที่ส่งเข้ามาสามารถเข้าถึงได้จากตัวแปร props (ตัวอย่าง บรรทัดที่ 25))

```
25 let name = $state((props.action === 'edit' && props.name) || '');
1 let stock = ; // number      ■ ts: Expression expected.
2 let maxOrder = ; // number    ■ ts: Expression expected.
3 let roastedLevel = ; // number ■ ts: Expression expected.
4 let price = ; // number       ■ ts: Expression expected.
5 let type = ; // string        ■ ts: Expression expected.
6 let open = $state<boolean>(false);
```

เมื่อเข้าหน้า `/manage` แล้ว กดสร้างใหม่ เมื่อกรอกข้อมูลในแต่ละช่อง และดู Console ของ Web Browser เมื่อกดส่ง ข้อมูลที่ขึ้นใน Console ควรจะมีค่าเหมือนกับที่กรอกทุกประการ

ข้อที่ 4) ให้เชื่อมต่อหน้าบ้านกับหลังบ้านใน `client/src/routes/manage/coffee-modal.svelte` บรรทัดที่ 47 ถึง 62 และ `client/src/routes/manage/delete-coffee-modal.svelte` บรรทัดที่ 13 ให้เชื่อมต่อกับหลังบ้านโดยใช้ Fetch API (https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) เพื่อใช้ในการจัดการข้อมูลเมล็ดกาแฟ ตัวอย่างสามารถดูได้ใน `client/src/routes/checkout/+page.svelte` บรรทัดที่ 72 และสามารถตรวจสอบเส้นทาง (Endpoint) ที่อยู่ในระบบหลังบ้านได้จาก รูปที่ 15

```
11 async function submit(e: SubmitEvent) {
10   e.preventDefault();
9     console.log(
8       `name: ${name}\n`,
7       `stock: ${stock}\n`,
6       `maxOrder: ${maxOrder}\n`,
5       `roastedLevel: ${roastedLevel}\n`,
4       `price: ${price}\n`,
3       `type: ${type}\n`,
2       `open: ${open}\n`,
1     );
46 You, 2024-02-05 - Fix querystring on submit
1   switch (props.action) {
2     case 'add': {
3       // send request to backend to add data
4       console.log('Coffee added successfully!');
5       props.onAction();
6       open = false;
7       break;
8     }
9     case 'edit': {
10      // send request to backend to update data
11      console.log('Coffee edited successfully!');
12      props.onAction();
13      open = false;
14      break;
15    }
16  }
17 }
```



```

11
1  async function submit() {
2    // send request to backend to delete data using id
3    console.log('Delete Success');
4    open = false;
5    onDelete();
6  }
7  </script>

```

```

72  async function submit(e: SubmitEvent) {
1  e.preventDefault();
2
3  const res = await fetch(`${env.PUBLIC_BACKEND_URL}/order`, {
4    method: 'POST',
5    headers: {
6      'Content-Type': 'application/json',
7    },
8    body: JSON.stringify({
9      name: cart.order.name,
10     qty: cart.order.qty,
11     coffeeId: cart.order.coffee?.id,
12     villageId: villageSelected,
13   }),
14 }).catch((e) => console.error('Error: ', e));
15
16 if (!res) return;
17 if (!res.ok) return console.error('Failed to submit order:', await res.text());
18
19 console.log('Order submitted successfully!');
20 await goto('/status');
21 }

```

(ตัวอย่าง)

เฉลยใบงาน

ข้อที่ 1)

```

1
2   if (!Array.isArray(validate.data)) {
3       await prisma.coffee.create({
4           data: validate.data,
5       });
6   } else {
7       await prisma.coffee.createMany({
8           data: validate.data,
9       });
10  }
11

```

ข้อที่ 2)

```

1
2   // prevent racing condition - do transaction
3   await prisma.$transaction(async (tx) => {
4       const coffee = await tx.coffee.update({
5           data: {
6               stock: { decrement: validate.data.qty },
7           },
8           where: { id: validate.data.coffeeId },
9       });
10
11      if (coffee.stock < 0) return ctx.throw(Status.NotAcceptable, "Not Enough Quantity");
12
13      await tx.order.create({
14          data: { ...validate.data, status: OrderStatus.PENDING },
15      });
16  });
17

```

ข้อที่ 3)

```

1
2   let name = $state((props.action === 'edit' && props.name) || '');
3   let stock = $state((props.action === 'edit' && props.stock) || 0);
4   let maxOrder = $state((props.action === 'edit' && props.maxOrder) || 1);
5   let roastedLevel = $state((props.action === 'edit' && props.roastedLevel) || 0);
6   let price = $state((props.action === 'edit' && props.price) || 0);
7   let type = $state((props.action === 'edit' && props.type) || '');
8   let open = $state<boolean>(false);
9

```

ข้อที่ 4)

```

case 'add': {
  const res = await fetch(`${env.PUBLIC_BACKEND_URL}/coffee`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      name,
      stock,
      maxOrder,
      roastedLevel,
      price,
      type,
      open,
    }),
  });
  }).catch((e) => console.error('Error: ', e));

  if (!res) return;
  if (!res.ok) return console.error('Failed to add coffee: ', await res.text());

  console.log('Coffee added successfully!');
  props.onAction();
  open = false;
  break;
}

```

```

case 'edit': {
  const res = await fetch(`${env.PUBLIC_BACKEND_URL}/coffee/${props.id}`, {
    method: 'PATCH',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      name,
      stock,
      maxOrder,
      roastedLevel,
      price,
      type,
      open,
    }),
  });
  }).catch((e) => console.error('Error: ', e));

  if (!res) return;
  if (!res.ok) return console.error('Failed to edit coffee: ', await res.text());

  console.log('Coffee edited successfully!');
  props.onAction();
  open = false;
  break;
}

```

```

11
12 async function submit() {
13   // send request to backend to delete data using id
14   const res = await fetch(`${env.PUBLIC_BACKEND_URL}/coffee/${id}`, {
15     method: 'DELETE',
16   });
17   console.log('Delete Success');
18   open = false;
19   onDelete();
20 }

```