Full Stack Web Development with Node.js & React TS

ຄອສ໌ຮຽນພື້ນຖານການກ້າວສູ່ Full Stack Web Developer ດ້ວຍພາສາ Node.js + React (TypeScript) + Vite + Ant Design + Tailwind CSS ເປັນການພັດທະນາ Application ຕັ້ງແຕ່ Backend ດ້ວຍ Node.js ແລະ Frontend ດ້ວຍ React TypeScript ພ້ອມໃຊ້ Vite ເພື່ອເພີ່ມປະສິດຕິພາບການພັດທະນາ ແລະ ອອກແບບ UI ດ້ວຍ Ant Design ແລະ ຕົບແຕ່ງດ້ວຍ Tailwind CSS.

📌 ສິ່ງທີ່ຈະໄດ້ຮຽນຮູ້

- ✅ ພັດທະນາ Backend API ດ້ວຍ Node.js + Express
- ✅ ສ້າງ ແລະ ຈັດການ Routing
- ✅ ສ້າງ ແລະ ຈັດການ Middleware
- ✅ ສ້າງ Authentication ດ້ວຍ JSON web token (JWT)
- ✅ ໃຊ້ MySQL ເປັນຖານຂໍ້ມູນຈັດການດ້ວຍ XAMPP ຫຼື Workbench
- ✅ ພັດທະນາ Frontend ດ້ວຍ React TypeScript
- ✅ ໃຊ້ Vite ໃນການເພີ່ມປະສິດຕິພາບການພັດທະນາ
- ✅ ໃຊ້ Ant Design ເປັນ UI Component ແລະ ເພີ່ມການຕົບແຕ່ງດ້ວຍ Tailwind CSS
- ✅ ການເຮັດ protected & public route
- ✅ ຈັດການ Session & LocalStorage
- ✅ ໃຊ້ State, Hook, Context & Redux
- ✅ Deploy ຜ່ານ Goviralhost

🎯 ເໝາະສຳລັບຜູ້ທີ່ຕ້ອງການພັດທະນາເວັບແອັບພລິເຄຊັນແບບ Full Stack ຕັ້ງແຕ່ເລີ່ມຕົ້ນຈົນເຖິງການເອົາຂຶ້ນ Server ຈິງເພື່ມໃຫ້ໃຊ້ງານໄດ້ໃນທົ່ວໂລກທີ່ມີ Internet.

😊 ຜູ້ຮຽນຕ້ອງມີພື້ນຖານ HTML, CSS ແລະ JavaScript ມາກ່ອນ ຫຼື ບໍ່ມີກໍ່ສາມາດເຂົ້າຮ່ວມໄດ້

💻 ຜູ້ຮຽນຕ້ອງມີພີ Computer ເປັນຂອງຕົນເອງ

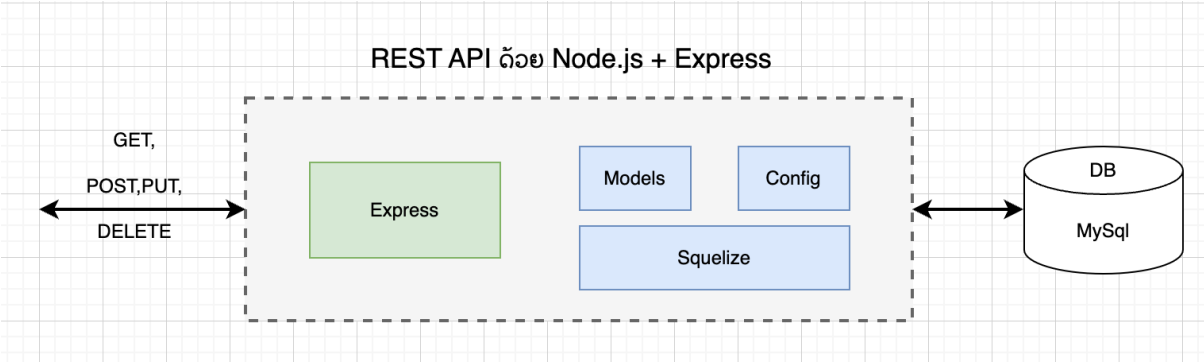
ບົດທີ 1: ປູພື້ນຖານ ແລະ ສ້າງຮາກຖານ Backend

1.1 ເຂົ້າໃຈການເຮັດວຽກຂອງ Backend

Backend ແມ່ນສ່ວນທີ່ເຮັດວຽກຢູ່ຝັ່ງ Server ຄ່ອຍຮັບຟັງຄຳຂໍ (Request) ຈາກຜູ້ໃຊ້ງານ ປະມວນຜົນ ແລະ ສົ່ງຂໍ້ມູນ (Response) ກັບໄປຫາ Client.

💡 **ຕົວຢ່າງ:**

- ເມື່ອເຮົາຄົ້ນຫາສິນຄ້າໃນເວັບໄຊ້ທ໌ E-commerce
- ບຣາວສ໌ເຊີ (Client) ສົ່ງຄຳຂໍໄປຍັງ Server
- Server ຄົ້ນຫາສິນຄ້າໃນຖານຂໍ້ມູນ
- ແລະ ສົ່ງຂໍ້ມູນກັບໄປໃຫ້ Client ເພື່ອສະແດງຜົນ



1.2 ໂຄງສ້າງ Client-Server Model

📌 Client (ບຣາວສ໌ເຊີ / ໂທລະສັບ) → 🔄 HTTP Request → 🖥️ Server (ປະມວນຜົນ) → 🔄 Response → 📌 Client

🔷 ຄຳຂໍ (Request) ປະກອບດ້ວຍ:

- URL ເຊັ່ນ https://example.com/api/users
- HTTP Method ເຊັ່ນ GET (ດຶງຂໍ້ມູນ), POST (ສົ່ງຂໍ້ມູນ)
- Headers (ໃຊ້ຢືນຢັນຕົວຕົນ, ປະເພດຂໍ້ມູນ)

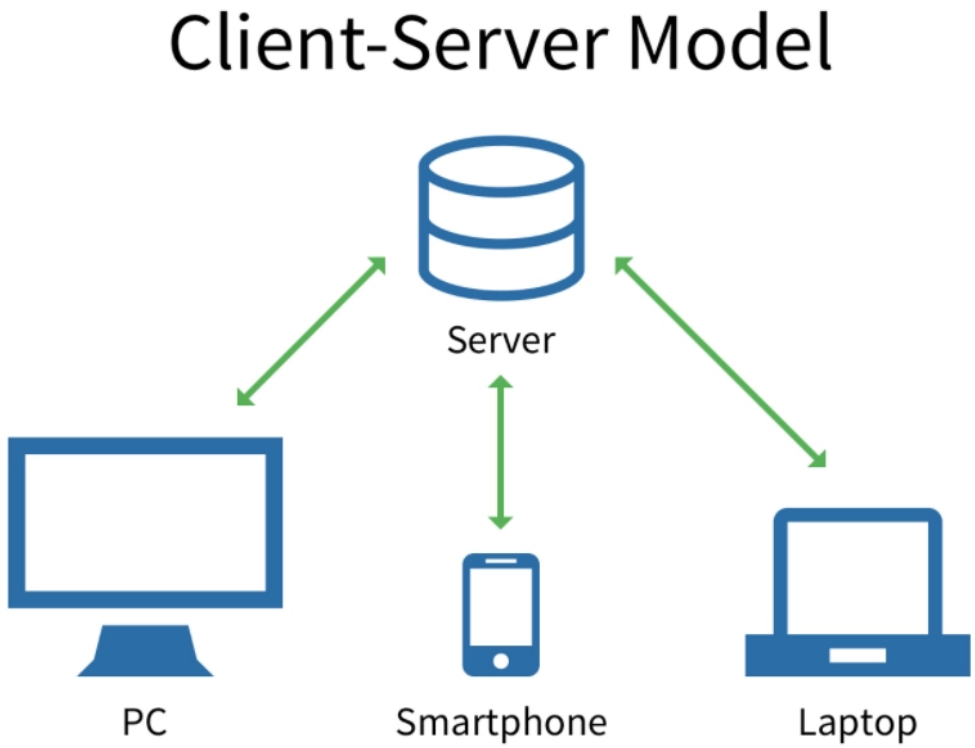
- Body (ຂໍ້ມູນທີ່ຕ້ອງການສົ່ງເຊັ່ນ JSON)

🔷 ຄຳຕອບ (Response) ປະກອບດ້ວຍ:

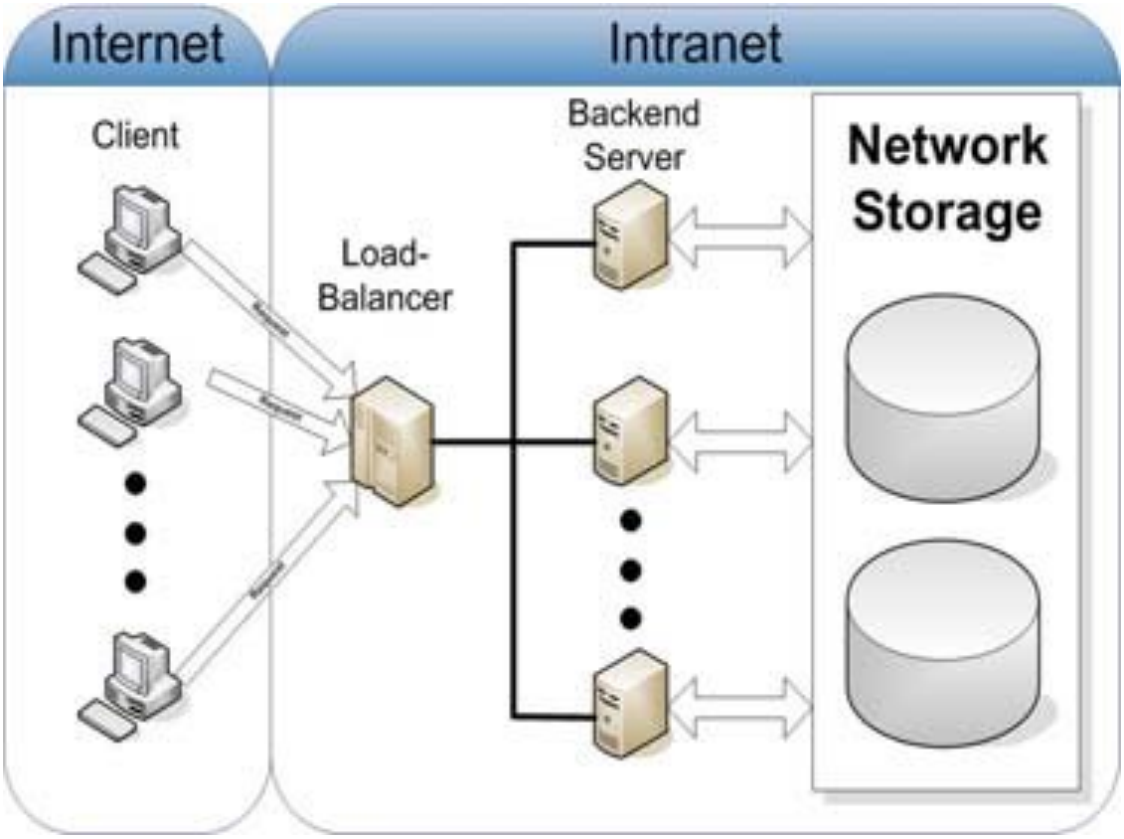
- Status Code ເຊັ່ນ 200 OK, 404 Not Found, etc

  ອ່ານເພີ່ມຕື່ມ: HTTP response status codes - HTTP | MDN
- Headers (ລະບຸປະເພດຂໍ້ມູນເຊັ່ນ JSON)
- Body (ຂໍ້ມູນທີ່ສົ່ງກັບ)

📌 **Model ພື້ນຖານ**



📌 **Model ທີ່ຊັບຊ້ອນ**



1.3 ປะเพด Backend APIs

📌 REST API

- ใช้ HTTP Methods (GET, POST, PUT, DELETE)
- มีหຼายย Endpoint (/users, /products)
- อาดเกิด Over-fetching (ได้ข้ໍมูนเกินที่ต้อງการ)

💡 ตัวอย่าງ

```
app.get('/users', (req, res) => {
  res.json([{ id: 1, name: "ถำพอม วิไลวัม" }]);
});
```

📌 GraphQL

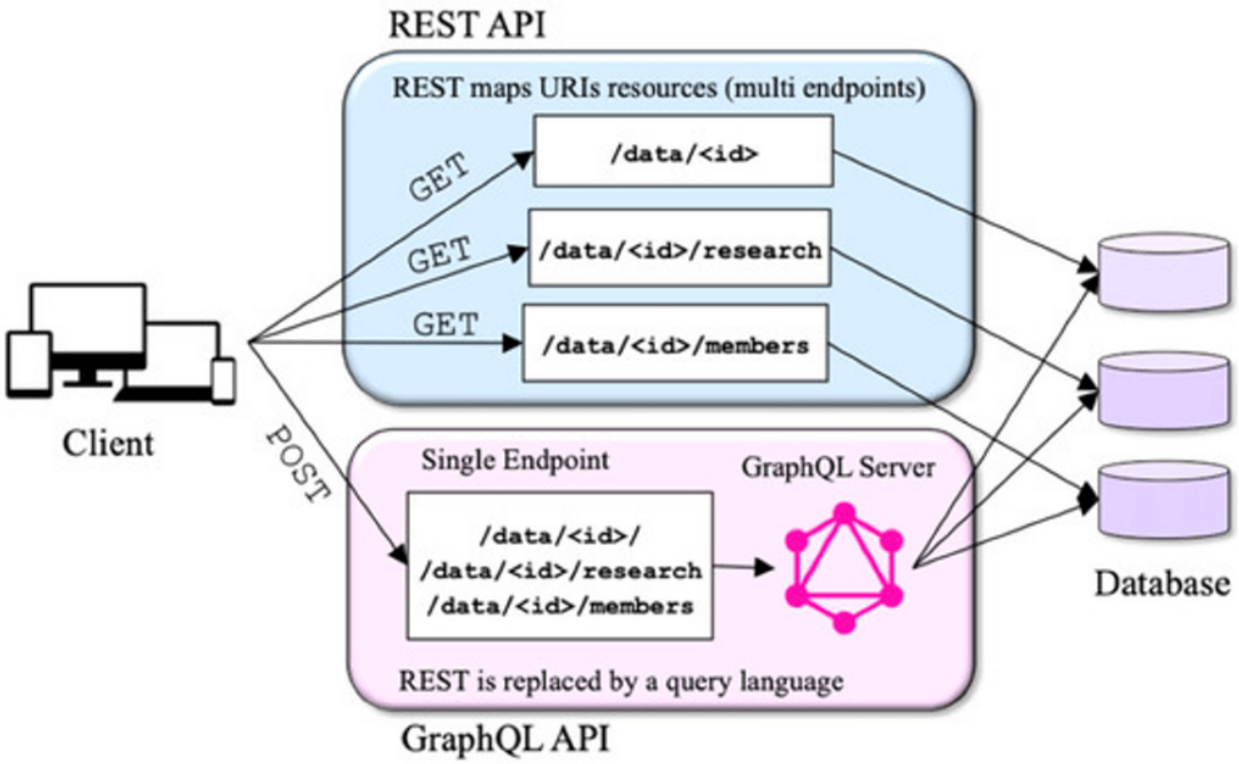
- ใช้ Query Language ดึงข้ໍมูนที่ต้อງการ
- ใช้พรึง 1 Endpoint

- ຢຶດໜຸ່ນມາກຂຶ້ນບໍ່ເກິດ Over-fetching

💡 ຕົວຢ່າງ GraphQL Query

```
query {
  user(id: 1) {
    name
    email
  }
}
```

ສະແດງກຣາບການສີ່ສານກັບ Backend ລະຫວ່າງ RESTful API ແລະ Graphql



1.4 พื้นฐาน HTTP และ HTTPS

📌 HTTP (HyperText Transfer Protocol)

- ใช้ในการส่งข้อมูลระหว่าง Client ↔️ Server

📌 HTTPS (HTTP Secure)

- ใช้ในการส่งข้อมูลระหว่าง Client ↔️ Server แต่ปลอดไพกว่า http เพราะมีการเข้ารหัสข้อมูล

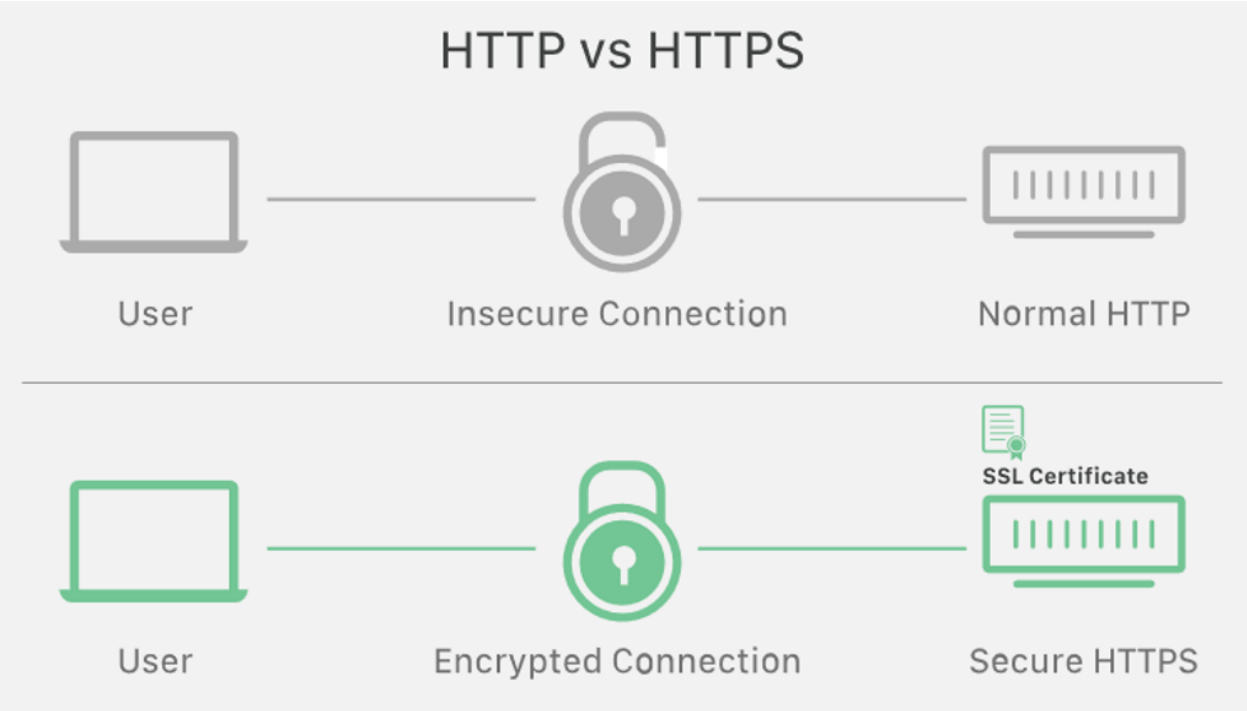
📌 HTTP Methods ที่สำคัญ

Method	จุดประสงค์
GET	ดึงข้อมูล
POST	ส่งข้อมูลใหม่
PUT	อัปเดตข้อมูล
DELETE	ลบข้อมูล

💡 ตัวอย่าง HTTP Request (POST)

```
fetch('https://example.com/api/users', {
    method: 'POST',
    body: JSON.stringify({ name: "John" }),
    headers: { 'Content-Type': 'application/json' }
});
```

ความแตกต่างระหว่าง HTTP & HTTPS



1.5 ຕິດຕັ້ງເຄື່ອງມືສຳລັບການພັດທະນາ

✅ Node.js → ໃຊ້ຣັນໂຄດ JavaScript ຝັ່ງ Server

💡 ຕິດຕັ້ງ Node.js

ດາວນ໌ໂຫລດຈາກ https://nodejs.org

ຄຳສັ່ງກວດເວີຊັ່ນ Nodejs

```
node -v
npm -v
```

✅ VS Code → ເຄື່ອງມືຂຽນໂຄດ

https://code.visualstudio.com

✅ Postman → ທົດສອບ API

https://www.postman.com/downloads/

✅ Git → ใช้ควบคุม Version Code

https://git-scm.com/downloads

บทที่ 2: เริ่มต้นสร้าง Backend ด้วย Node.js & Express

2.1 Setup Node.js และ Express.js

📌 **สร้างโปรเจกใหม่**

```
mkdir backend-product && cd backend-product
code .
npm init -y
npm install express
```

📌 **สร้างไฟล์ index.js**

```
const express = require('express');
const app = express();

app.use(express.json());

app.listen(3000, () => console.log('Server running on port 3000'));
```

📌 **แก้ไข file package.json:** เพิ่ม Code ในแถบสีฟ้าลุ่มนี้เพื่อสั่งใช้ Run code โดยเริ่มจาก file index.js

```
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node index.js"
  },
```

📌 **ສາມາດ Run program ດ້ວຍຄຳສັ່ງ**

```
npm start
```

📌 **ແລະ ນີ້ເປັນຜົນລັບທີ່ໄດ້ໃນ Terminal**

```
mac@kicks-me backend-product % npm start

> backend-product@1.0.0 start
> node index.js

Server running on port 3000
```

2.3 Module ທີ່ຄວນຮູ້ສຳລັບຜູ້ເລີ່ມຕົ້ນ Nodejs:

a. Routing:

- ວິທີການສ້າງ routes ໃນ Node.js ໂດຍໃຊ້ Express

ການສ້າງ Route ແມ່ນການກຳນົດເສັ້ນທາງໃນການເຂົ້າເຖິງຂໍ້ມູນທີ່ກ່ຽວຂ້ອງ ຫຼື ການຈັດການຂໍ້ມູນໃດຂໍ້ມູນໜຶ່ງເຊັ່ນ: ຂໍ້ມູນພະນັກງານ, ຂໍ້ມູນສິນຄ້າ ເປັນຕົ້ນ ຊຶ່ງການເຂົ້າເຖິງຂໍ້ມູນນັ້ນສາມາດຈັດການຜ່ານ HTTP methods (GET, POST, PUT, DELETE) ແລະ path parameters

```
// Route GET: ດຶງຂໍ້ມູນ Users
app.get('/users', (req, res) => {
  res.json([{ id: 1, name: 'John Doe' }, { id: 2, name: 'Jane Doe' }]);
});

// Route POST: ເພີ່ມ User ໃໝ່
app.post('/users', (req, res) => {
  const newUser = req.body;
  res.status(201).json({ message: 'User ຖືກເພີ່ມແລ້ວ!', user: newUser });
});
```

- ການໃຊ້ query parameters ແລະ body data

```
// Route PUT: ອັບເດດ User
app.put('/users/:id', (req, res) => {
    const userId = req.params.id;
    res.json({ message: `User ${userId} ຖືກອັບເດດແລ້ວ!` });
});

// Route DELETE: ລຶບ User
app.delete('/users/:id', (req, res) => {
    const userId = req.params.id;
    res.json({ message: `User ${userId} ຖືກລຶບແລ້ວ!` });
});
```

- ສ້າງ Route ແບບ Nested route

ສ້າງ Route ກຸ່ມທີ່ຕ້ອງມີ Route ຢູ່ທາງໜ້າສະເໝີດັ່ງຕົວຢ່າງລຸ່ມນີ້:

```
const userRouter = express.Router();

// ສ້າງ router ສຳລັບ users ທັງໝົດ
userRouter.get('/', (req, res) => {
    res.send('All Users');
});

 // ສ້າງ router ສຳລັບ user ຕາມລະຫັດທີ່ຕ້ອງການ
userRouter.get('/:id', (req, res) => {
    res.send(`User ID: ${req.params.id}`);
});


// ສ້າງ router ທີ່ຢູ່ພາຍໃຕ້ '/users' ແບບນີ້ເອີ້ນວ່າ Nested routes
app.use('/users', userRouter);
```

Route ທີ່ສ້າງດ້ວຍ userRouter ຈະຖືກເອີ້ນໃຊ້ພາຍໃຕ້ '/users' ສະເໝີ ແລະ Route ສ້າງ userRouter ເອີ້ນວ່າ: Nested routes.

b. ການຈັດການ Error:

ໃນ **Node.js + Express**, ການຈັດການ Error ເປັນສິ່ງສຳຄັນເພື່ອຮັບປະກັນວ່າ API ຫລື Web App ຂອງພວກເຮົາຈະບໍ່ເກີດການຢຸດສະງັດແມ່ນແຕ່ຈະເກີດຂໍ້ຜິດພາດກໍ່ຕາມ. ທາງດ້ານລຸ່ມນີ້ເປັນຈະເປັນວິທີຈັດການ Error ທີ່ມີນິຍົມຢ່າງກວ້າງຂວາງ ແລະ ມີປະສິດທິພາບ.

- **ການໃຊ້ try/catch ເພື່ອຈັດການຂໍ້ມູນຜິດພາດແບບ asynchronous code ເຊັ່ນ:**

```
const fetchDataWithTimeout = async (timeout: number) => {
  return Promise.race([
    new Promise((_, reject) =>
      setTimeout(() => reject(new Error("Request timed out")), timeout)
    ),
  ]);
};

app.get("/data", async (req, res) => {
  try {
    const data = await fetchDataWithTimeout(5000); // ຕັ້ງ timeout 5 ວິນາທີ
    res.json({ success: true, data });
  } catch (error) {
    console.error(error);
    res.status(500).json({
      success: false,
      message:
        error.message === "Request timed out"
          ? "ຄຳຂໍຂອງທ່ານໝົດເວລາ"
          : "ມີຂໍ້ຜິດພາດໃນການດຶງຂໍ້ມູນ",
    });
```

```
  }
});
```

- **ການສ້າງ custom error handling middleware ໃນ Express**

  ✅ ສ້າງ Custom Error Handling Middleware ໃນ Express.js, ສາມາດສ້າງ middleware ສຳລັບຈັດການ error ໂດຍສ້າງ function ທີ່ມີ 4 parameters (err, req, res, next). ນິຍົມໃຊ້ໃນໂປເຈັກທີ່ຕ້ອງການ Performance ດີໆ ແລະ ເນັ້ນຄວາມສະດວກສະບາຍໃນການບຳລຸງຮັກສາ ແລະ ໂຄດສະອາດ (Clean code)

```
// ສ້າງ middleware
const errorHandler = (err, req, res, next) => {
    console.error(err.stack); // Log error ເພື່ອ debug

    res.status(err.statusCode || 500).json({
    success: false,
    message: err.message || 'ມີຂໍ້ຜິດພາດໃນ Server'
    });
 };
// Route ສຳລັບທົດສອບ Error
app.get("/data", async (req, res, next) => {
   try {
    const data = await fetchDataWithTimeout(5000); // ຕັ້ງ timeout 1 ວິນາທີ
    res.json({ success: true, data });
   } catch (error) {
    next(error)
   }
});

 // ໃຊ້ middleware ຈັດການ error ສຸດທ້າຍ
 app.use(errorHandler);
```

✅ ໃຊ້ Error Middleware ຄູ່ກັບ Custom Error Class

ພວກເຮົາສາມາດສ້າງ CustomError ເພື່ອກຳນົດ status code ແລະ message ຂອງ error.

```
class CustomError extends Error {
  constructor(message) {
    super(message);
  }
}

// ນຳໃຊ້ CustomError ຂອງພວກເຮົາ
app.get('/protected', (req, res, next) => {
  const error = new CustomError('ທ່ານບໍ່ມີສິດເຂົ້າໃຊ້');
  next(error);
});
```

- **ການຈັດການ error responses ຢ່າງເໝາະສົມ ເຊັ່ນ ການຕອບກັບດ້ວຍ status code ແລະ ຂໍ້ຄວາມທີ່ຊັດເຈນ)**

ວິທີທີ່ກ່າວມາຂ້າງເທິງນີ້ຈະບໍ່ສາມາດກຳນົດລະຫັດຂໍ້ຜິດພາດໄດ້ (HTTP Status Code). ດັ່ງນັ້ນ, ເຮົາສາມາດແກ້ໄຂດ້ວຍວິທີລຸ່ມນີ້

```
class CustomError extends Error {
  constructor(message, statusCode) {
    super(message);
    this.statusCode = statusCode;
  }
}

// ນຳໃຊ້ CustomError ຂອງພວກເຮົາ
app.get('/protected', (req, res, next) => {
  const error = new CustomError('ທ່ານບໍ່ມີສິດເຂົ້າໃຊ້', 403);
  next(error);
});
```

c. **Middleware:**

Middleware ໃນ Express.js ແມ່ນ ຄອນເຊບ (concept) ຂອງການປ່ອນຂໍ້ມູນເຂົ້າ-ອອກ (Request-Response) ຜ່ານຟັງຊັນຫລາຍໆຊັ້ນ (layers) ກ່ອນທີ່ຈະສົ່ງຜົນລັບໃຫ້ຜູ້ໃຊ້ (Client) ຫຼື middleware ແມ່ນໃຊ້ເພື່ອຈັດການກັບ request ແລະ response ເຊັ່ນ ການກວດສອບ authentication, ການຈັດການ errors ເປັນຕົ້ນ

👉 **Middleware ສາມາດ:**

✅ ປ່ອນ, ປ່ຽນແປງ, ກວດສອບ Request ເຊັ່ນ: ກວດສອບສິດຂອງ JWT

✅ ສົ່ງຄ່າຄືນ Response

✅ ຈັດການ Error

✅ ສົ່ງຂໍ້ມູນໄປຍັງ Route ຖັດໄປ

- **Middleware ມີຫຼາຍຮູບແບບຂຶ້ນກັບຈຸດປະສົງໃນການໃຊ້ງານ ມີທັງ** middleware ທີ່ສ້າງຂຶ້ນມາເພື່ອຈຸດປະສົງສະເພາະເຈາະຈົງຕາມຄວາມໝາຍຂອງຜູ້ສ້າງລະບົບ ແລະ third-party middleware ເຊັ່ນ: body-parser, cors, ຫຼື morgan

d. **ການຈຳກັດເຂົ້າເຖິງດ້ວຍ Core:**

ແມ່ນການການຈັດການ CORS ແລະ ການຈຳກັດການເຂົ້າເຖິງຈາກແຫ່ງທີ່ບໍ່ປອດໄພຊຶ່ງໃນລະບົບຫຼັງບ້ານຕ້ອງໄດ້ປ້ອງກັນ ເປັນອີກວິທີໜຶ່ງທີ່ນິຍົມກັນຢ່າງແພ່ຫຼາຍໃນວົງການ Backend API. ມີຂັ້ນຕອນການຕັ້ງຄ່າດັ່ງລຸ່ມນີ້:

📌 1) ຕິດຕັ້ງ package cors

```
npm install cors
```

📌 2) **ຄີຽ package cors ມາໃຊ້ງານຢູ່ໜ້າທາອິດຂອງລະບົບ**

```
const cors = require('cors');
```

📌 **3) ເອີ້ນໃຊ້ package cors ຢູ່ໜ້າທຳອິດຂອງລະບົບ ແລະ ໄວ້ສ່ວນເທິງສຸດ**

```
// ອະນຸຍາດໃຫ້ກັບທຸກໆແຫ່ງມາສາດເຂົ້າເຖິງລະບົບຫຼັງບ້ານ
app.use(cors({options: "*"}));

// ອະນຸຍາດສະເພາະ www.google.com ນອກນັ້ນແມ່ນບ່ສາມາດເຂົ້າເຖິງໄດ້
app.use(cors({options: "www.google.com"}))
```

e. ການເຊື່ອມຕໍ່ຖານຂໍ້ມູນ:

- ການເຊື່ອມຕໍ່ກັບຖານຂໍ້ມູນເຊັ່ນ: MongoDB ເຊື່ອມຕໍ່ຜ່ານ Mongoose ຫຼື SQL ເຊັ່ນ PostgreSQL ຫຼື MySQL

```
// ຕິດຕັ້ງ MySQL Driver
 npm install mysql2
```

✅ **ການເຊື່ອມຕໍ່ຖານຂໍ້ມູນ** (Connection) ໂດຍໃຫ້ສ້າງໄຟລ໌ config/db.js ສຳລັບການເຊື່ອມຕໍ່:

```
const mysql = require("mysql2");

// ຕັ້ງຄ່າການເຊື່ອມຕໍ່
const connection = mysql.createConnection({
host: "localhost",
user: "root",
password: "yourpassword",
database: "yourdatabase",
});

// ສັ່ງຣັນການເຊື່ອມຕໍ່
connection.connect((err) => {
if (err) {
   console.error("❌ ບ່ສາມາດເຊື່ອມຖານຂໍ້ມູນ:", err);
   return;
}
console.log("✅ ເຊື່ອມຕໍ່ MySQL ສຳເລັດ!");
```

```
});

module.exports = connection;
```

✅ **ທົດລອງການຕິດຕໍ່ຖານຂໍ້ມູນ**

```
    const connection = require("./config/db");


app.get('/connection', (req, res) => {
   // ກວດສອບຂໍ້ມູນຜູ້ໃຊ້ລະບົບ
   connection.query("SELECT * FROM tbusers", (err, results) => {
      if (err) {
         console.error("❌ Query error:", err);
         return res.status(500).json({ message: "Connect to DB failed" });
      }

      console.log("📌 Users:", results);

      // ສົ່ງ response ກັບໄປໃຫ້ Client
      res.status(200).json({ data: results });

      // ປິດ connection ຫລັງຈາກເຮັດວຽກກັບຖານຂໍ້ມູນສຳເລັດ
      connection.end((endErr) => {
         if (endErr) {
            console.error("❌ Error closing DB connection:", endErr);
         } else {
            console.log("✅ Database connection closed.");
         }
      });
   });
});
```

- **ການສ້າງ API (Create, Read, Update, Delete) ດ້ວຍ MySQL**

📌 1) สร้าง routes/userRoutes.js

```
const express = require("express");
const router = express.Router();
const connection = require("../config/db");

// 📌 Create User (POST)
router.post("/", (req, res) => {
const { name, email } = req.body;
const sql = "INSERT INTO users (name, email) VALUES (?, ?)";

connection.query(sql, [name, email], (err, result) => {
    if (err) {
    return res.status(500).json({ error: err.message });
    }
    res.status(201).json({ message: "✅ User added successfully",
id: result.insertId });
});
});

// 📌 Read All Users (GET)
router.get("/", (req, res) => {
connection.query("SELECT * FROM users", (err, results) => {
    if (err) {
    return res.status(500).json({ error: err.message });
    }
    res.json(results);
});
});

// 📌 Read Single User (GET)
router.get("/:id", (req, res) => {
const sql = "SELECT * FROM users WHERE id = ?";
connection.query(sql, [req.params.id], (err, result) => {
    if (err) {
    return res.status(500).json({ error: err.message });
    }
    if (result.length === 0) {
    return res.status(404).json({ message: "❌ User not found"
});
    }
    res.json(result[0]);
});
});
```

```
// 📌 Update User (PUT)
router.put("/:id", (req, res) => {
const { name, email } = req.body;
const sql = "UPDATE users SET name = ?, email = ? WHERE id = ?";

connection.query(sql, [name, email, req.params.id], (err, result)
=> {
    if (err) {
    return res.status(500).json({ error: err.message });
    }
    if (result.affectedRows === 0) {
    return res.status(404).json({ message: "❌ User not found"
});
    }
    res.json({ message: "✅ User updated successfully" });
});
});

// 📌 Delete User (DELETE)
router.delete("/:id", (req, res) => {
const sql = "DELETE FROM users WHERE id = ?";

connection.query(sql, [req.params.id], (err, result) => {
    if (err) {
    return res.status(500).json({ error: err.message });
    }
    if (result.affectedRows === 0) {
    return res.status(404).json({ message: "❌ User not found"
});
    }
    res.json({ message: "✅ User deleted successfully" });
});
});

module.exports = router;
```

📌 2) ທົດສອບການ

```
const express = require("express");

const app = express();
const port = 5000;
```

```
// ນຳໃຊ້ Routes
const userRoutes = require("./routes/userRoutes");
app.use("/users", userRoutes);

// ເປີດ Server
app.listen(port, () => {
console.log(`🚀 Server running at http://localhost: ${port}`);
});
```

- ການຈັດການຖານຂໍ້ມູນດ້ວຍ ORM/ODM ສຳລັບການຈັດການຂໍ້ມູນເຊັ່ນ Sequelize ຫຼື Mongoose.

Sequelize ຄື ORM ສຳລັບ MySQL, ຊ່ວຍໃຫ້ເຂົ້າໃຈງ່າຍກວ່າ SQL

📌 1) ຕິດຕັ້ງ Sequelize

```
npm install sequelize
```

📌 2) ຕັ້ງຄ່າ Sequelize

```
const { Sequelize } = require("sequelize");

const sequelize = new Sequelize("yourdatabase", "root",
"yourpassword", {
host: "localhost",
dialect: "mysql",
});

sequelize
.authenticate()
.then(() => console.log("✅ Connected to MySQL using
Sequelize!"))
.catch((err) => console.error("❌ Connection error:", err));

module.exports = sequelize;
```

📌 3) ສ້າງ models directory ຂຶ້ນມາຍັນເພື່ອເກັບ Sequelize models

📌 4) ສ້າງ models user ຂຶ້ນມາຮັນ user.js

```
const { DataTypes } = require("sequelize");
const sequelize = require("../db"); // ນຳ database connection ມາໃຊ້

const User = sequelize.define("User", {
id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true,
},
name: {
    type: DataTypes.STRING,
    allowNull: false,
},
email: {
    type: DataTypes.STRING,
    allowNull: false,
    unique: true,
},
});

module.exports = User;
```

📌 5) ສ້າງ Route ເພື່ອທົດສອບ API

```
const express = require("express");
const router = express.Router();
const User = require("../models/user");

// 📌 GET ດຶງຂໍ້ມູນທັງໝົດ
router.get("/", async (req, res) => {
try {
    const users = await User.findAll();
    res.json(users);
} catch (error) {
```

```
    res.status(500).json({ error: "❌ ດຶງຂໍ້ມູນບໍ່ສຳເລັດ" });
}
});

// 📌 POST ເພີ່ມຂໍ້ມູນ User
router.post("/", async (req, res) => {
try {
    const { name, email } = req.body;
    const newUser = await User.create({ name, email });
    res.status(201).json(newUser);
} catch (error) {
    res.status(500).json({ error: "❌ ເພີ່ມ User ບໍ່ສຳເລັດ" });
}
});

// 📌 PUT ອັບເດດ User
router.put("/:id", async (req, res) => {
try {
    const { name, email } = req.body;
    const user = await User.findByPk(req.params.id);
    if (!user) return res.status(404).json({ error: "❌ ບໍ່ພົບ User" });

    user.name = name;
    user.email = email;
    await user.save();

    res.json(user);
} catch (error) {
    res.status(500).json({ error: "❌ ອັບເດດບໍ່ສຳເລັດ" });
}
});

// 📌 DELETE ລຶບ User
router.delete("/:id", async (req, res) => {
try {
    const user = await User.findByPk(req.params.id);
    if (!user) return res.status(404).json({ error: "❌ ບໍ່ພົບ User" });
```

```
    await user.destroy();
    res.json({ message: "✅ ລຶບ User ສຳເລັດ" });
} catch (error) {
    res.status(500).json({ error: "❌ ລຶບບໍ່ສຳເລັດ" });
}
});

module.exports = router;
```

f. **Environment Variables:**

Environment Variables (ຕົວປ່ຽນ) ແມ່ນ **ຄ່າທີ່ຖືກກຳນົດໄວ້ໃນລະບົບປະຕິບັດການ (OS) ຫລື ຟາຍ** .env ເພື່ອໃຫ້ໂປຣແກຣມນຳໃຊ້ໃນການປັບແຕ່ງຄວາມປອດໄພ, ຕັ້ງຄ່າ, ແລະ ຈັດການຄວາມລັບ ຫຼື ເອີ້ນສັ້ນໆວ່າໄຟລ์ສຳລັບຂໍ້ມູນຄວາມລັບເຊັ່ນ:

- ການໃຊ້ງານ .env files ເພື່ອເກັບຂໍ້ມູນທີ່ສຳຄັນເຊັ່ນ API keys, JWT secret, ຫຼື database credentials ເປັນຕົ້ນ

```
//ຊື່ຕົວປ່ຽນ = ຄ່າຕົວປ່ຽນ

SECRET_KEY = 123!@#%%$i
```

- ການໃຊ້ dotenv ເພື່ອໂຫລດຄ່າ environment variables ມາໃຊ້ງານໃນໂປເຈກ

📌 1) ຕິດຕັ້ງ Package

```
npm install dotenv --save
```

📌 2) ການໃຊ້ງານ

```
//Javascript
const dotenv = require('dotenv');
```

```
// ES Module
import * as dotenv from 'dotenv';


// Load environment variables from a .env file
dotenv.config();

// ການເຂົ້າເຖິງ environment variables
const mySecret = process.env.MY_SECRET_KEY;

console.log(mySecret);  // ສະແດງຜົນລັບຂອງ MY_SECRET_KEY
```

g. JWT (JSON Web Token):

- ການໃຊ້ງານ JWT ສຳລັບການກວດສອບຜູ້ໃຊ້ (authentication) ແລະ ການຄວບຄຸມສິດ (authorization)
- ການຕັ້ງຄ່າ expiration ໃນ JWT

📌 1) ຕິດຕັ້ງ jwt package

```
npm install jsonwebtoken
```

📌 2) ສ້າງໄຟລ์ utils/jwt.js

```
const jwt = require("jsonwebtoken");

exports.genAccessJWT = (payload)=>{
    const accessOptions = {
        expiresIn: 'JWT_EXPIRY_ACCESS',
        algorithm: "RS256",
    }

    return jwt.sign(payload, 'PRIVATE_KEY', accessOptions);
}
```

📌 2) ສ้างๆ Middleware ສำลับการกวดสอบ JWT ใน middleware/jwt.js

```
const jwt = require("jsonwebtoken");


exports.verifyJWT = (req, res, next) => {
    const auth_key = req?.headers?.authorization;

    if(!auth_key || !auth_key?.startsWith("Bearer "))
    {
        console.log('unauthorized => ',req?.headers);
        return res.status(401).json({resultCode: 401, message:
"Unauthorized"});
    }

    const token = auth_key?.split(" ")[1];

    jwt?.verify(token, 'PUBLIC_KEY', {
        algorithms: "RS256",
    },
    (err, plainCode)=> {
        if(err)
        {
            if(err?.name === 'TokenExpiredError')
            {
                console.log(`token is expired`);
                return res.status(401).json({resultCode: 401,
message: "Session is expired"});
            }
            else
            {
                console.log(`${req?.headers}\t Forbidden on jwt
decode error`);
                return res.status(403).json({resultCode: 403,
message: "Forbidden"});
            }
        }

        req.id = plainCode?.id;
        req.username = plainCode?.username;
        req.role = plainCode?.role;

        next();

    });
```

```
  }
```

- ການປ້ອງກັນ routes ແລະ ຂໍ້ມູນການເຂົ້າເຖິງທີ່ບໍ່ໄດ້ຮັບອະນຸຍາດ

```
import { verifyJWT } from '../../middleware/jwt.js';

app.get('/protected-route', verifyJWT, (req, res) => {
  // ສະເພາະ user ໄດ້ຮັບອະນຸຍາດ ຫຼື ຖືຂໍ້ມູນທີ່ຖືກຕ້ອງມາເທົ່ານັ້ນຈຶ່ງສາມາດເຂົ້າເຖິງ
  Route ນີ້ໄດ້
  res.json({ message: 'This is a protected route', user: req.user });
});
```

h. Export, module.export ແລະ Export Default

ເຄີຍສົງໄສບໍ່ເວລາເຫັນຄົນອື່ນໃຊ້ Export ຫຼື Export Default ? ຮູ້ຄວາມໝາຍບໍ ? ຄວາມຈິງແລ້ວເລືອກໃຊ້ອັນໃດອັນໜຶ່ງກໍ່ໄດ້ ແລະ ບໍ່ມີຖືກ ຫຼື ຜິດໃຫ້ພິຈາລະນາຕາມສະພາບແວດລ້ອມການໃຊ້ງານຕົວຈິງ ແລະ ຕໍ່ໄດ້ນີ້ແມ່ນຄວາມແຕກຕ່າງກັນໃນການໃຊ້ງານ:

📌 1) ການໃຊ້ Export (Name export)

export ແບບ Named Export ຈະໃຫ້ເຮົາສາມາດສົ່ງອອກ (export) **ຫຼາຍຟັງຊັນ/ຂໍ້ມູນ ຈາກໄຟລ์ດຽວ** ໄດ້ ແລະ ຕ້ອງລະບຸຊື່ໃນຂະນະ import

✅ **ຕົວຢ່າງ Named Export**

```
// utils.js

export const add = (a, b) => a + b;

export const subtract = (a, b) => a - b;
```

📌 **ວິທີ import Named Export**

```
// main.js
```

```
import { add, subtract } from './utils.js';


console.log(add(5, 3));      // 8
console.log(subtract(5, 3)); // 2
```

🔷 **ຂໍ້ດີຂອງ Named Export**

- ສາມາດ export **ຫຼາຍຄ່າ/ຫຼາຍຟັງຊັ່ນ** ໃນໄຟລ์ດຽວ
- ຕ้ອງໃຊ้ {} ໃນການ import
- ຊ่ວຍໃຫ้ໂຄງສ้າງຂອງໂປຣແກຣມຊัดเจน

📌 2) Export default (Default Export)

export default ຈะสົ่งออกค่าหลัก (Main Export) ของไฟล์ และ **สามาถ import ได้โดยบ่จำเป็นต้องใช้ { }**

✅ **ต๊วอย่างๆ Default Export**

```
// utils.js
export default function multiply(a, b) {
   return a * b;
}
```

📌 **วิธี import Default Export**

```
// main.js
import multiply from './utils.js';
```

```
console.log(multiply(5, 3));  // 15
```

🔷 **ຂໍ້ດີຂອງ Default Export**

- ບໍ່ຈຳເປັນຕ້ອງໃຊ້ {} ຂະນະ import
- ສາມາດຕັ້ງຊື່ໃໝ່ໃນຂະນະ import ຂຶ້ນຢູ່ກັບຜູ້ນຳໃຊ້
- ຄວນໃຊ້ເມື່ອໄຟລ์ນັ้นສົ่งออกสิ่งที่สำคัญซักอย่างดຽว

📌 **3) ການໃຊ้ export ແລะ export default ร่วมกัน**

ພວกเธ้าสามาดใช้ export ແລะ export default ร่วมกันได้ในไฟล์ดຽว

✅ ตัวอย่าง

```
// utils.js
export const add = (a, b) => a + b;
export const subtract = (a, b) => a - b;
export default function multiply(a, b) {
    return a * b;
}
```

📌 **วิธี import**

```
// main.js
import multiply, { add, subtract } from './utils.js';
```

```
console.log(multiply(5, 3));   // 15
console.log(add(5, 3));        // 8
console.log(subtract(5, 3));   // 2
```

ຄວາມແຕກຕ່າງລະຫວ່າງ export vs export default

Feature	export (Named Export)	export default (Default Export)
ຈຳນວນທີ່ export ໄດ້	ຫຼາຍຄ່າໄດ້	ຄ່າດຽວເທົ່ານັ້ນ
Syntax ຂະນະ import	ຕ້ອງໃຊ້ {}	ບໍ່ຈຳເປັນຕ້ອງໃຊ້ {}
ປ່ຽນຊື່ຂະນະ import	ໄດ້ (ໃຊ້ as)	ຕັ້ງຊື່ໃໝ່ໄດ້ໂດຍກົງ
ເໝາະກັບ	ໂປຣແກຣມທີ່ມີຫຼາຍຟັງຊັນ	ໂປຣແກຣມທີ່ມີເພິ່ງໆໜຶ່ງສິ່ງທີ່ສຳຄັນ

📌 4) ການໃຊ້ module.exports ແລະ exports ໃນ CommonJS (Node.js ເກົ່າ)

ຖ້າໃຊ້ CommonJS (require ແທນ import), module.exports ແລະ exports ຈະເຮັດໜ້າທີ່ຄືກັບ export ແລະ export default

✅ **ຕົວຢ່າງ module.exports (ຄືກັບ export default)**

```
// utils.js (CommonJS)
```

```
module.exports = function multiply(a, b) {
    return a * b;
};
```

📌 ການ import (CommonJS)

```
// main.js
const multiply = require('./utils');

console.log(multiply(5, 3));  // 15
```

✅ ຕົວຢ່າງ exports (ຄືກັບ Named Export)

```
// utils.js (CommonJS)
exports.add = (a, b) => a + b;
exports.subtract = (a, b) => a - b;
```

📌 ການ import (CommonJS)

```
// main.js
const { add, subtract } = require('./utils');

console.log(add(5, 3));       // 8
```

```
console.log(subtract(5, 3));   // 2
```

Backend ດ້ວຍ Node.js + Express ເມື່ອນັກຮຽນຮູ້ຫົວຂໍ້ເຫຼົ່ານີ້ແລ້ວ

ສາມາດນຳໄປພັດທະນາຕໍ່ຍອດ ແລະ ສຶກສາເລີ້ອຍໆເພື່ອໃຫ້ເຂົ້າໃຈວິທີການ ແລະ ຂະບວນການຂອງລະບົບໃຫ້ສາມາດຈັດການກັບ Application Node.js ທີ່ຊັບຊ້ອນໄດ້

2.4 ຈັດການຖານຂໍ້ມູນ

- **ຕິດຕັ້ງ MySql ໂດຍໃຊ້ XAMPP**

ດາວໂຫລດ Xampp: https://www.apachefriends.org/download.html

- **ສ້າງຖານຂໍ້ມູນ**

```
db_full_stack
```

- **ສ້າງຕາຕະລາງທີ່ຈຳເປັນ**

📌 1) ສ້າງຕາຕະລາງ: tbusers

```
CREATE TABLE tbusers (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    role ENUM('admin', 'user') NOT NULL DEFAULT 'user',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

📌 2) ສ້າງຕາຕະລາງ: tbproducts

```
CREATE TABLE tbproducts (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
```

```
    description TEXT,
    slug VARCHAR(255) NOT NULL UNIQUE,
    expired_at DATE,
    buyPrice FLOAT NULL DEFAULT '0',
    qty INT NOT NULL DEFAULT 0,
    sellPrice FLOAT NOT NULL DEFAULT '0',
    image VARCHAR(255),
    img_ext VARCHAR(10),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_by INT NOT NULL,
    FOREIGN KEY (created_by) REFERENCES tbusers(id) ON DELETE
CASCADE
 );
```

📌 3) ສ້າງຕາຕະລາງ: tbsell

```
 CREATE TABLE tbsell (
    id INT AUTO_INCREMENT PRIMARY KEY,
    sell_id VARCHAR(6) NULL DEFAULT NULL,
    product_id INT NOT NULL,
    user_id INT NOT NULL,
    qty INT NOT NULL CHECK (qty > 0),
    selled_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (product_id) REFERENCES tbproducts(id) ON DELETE
CASCADE,
    FOREIGN KEY (user_id) REFERENCES tbusers(id) ON DELETE
CASCADE
 );
```

ບົດທີ 3: ພື້ນຖານ React Type Script ແລະ ການຈັດການ State

Frontend Programming ຫລື **ການພັດທະນາສ່ວນໜ້າບ່ອນຂອງເວັ້ບ** ແມ່ນຂະບວນການຂຽນໂປຣແກຣມທີ່ຄວບຄຸມການສະແດງຜົນ ແລະ **ປະສົບການຂອງຜູ້ໃຊ້ (User Experience - UX)** ຂອງເວັ້ບໄຊທ໌ ຫລື ແອັບພິເຄຊັ້ນ.

ມັນປະກອບໄປດ້ວຍພາສາທີ່ໃຊ້ໃນການພັດທະນາ:

- **HTML** (HyperText Markup Language) ສຳລັບສ້າງໂຄງສ້າງຂອງເວັ້ບ.

- **CSS** (Cascading Style Sheets) ສຳລັບຕົກແຕ່ງຮູບແບບ ແລະ ຄວາມສວຍງາມ.
- **JavaScript** ສຳລັບເພີ່ມຄວາມແຮງຂັບໃຫ້ເວັບໄຊທ໌ມີຄວາມໂຕ້ຕອບໄດ້.

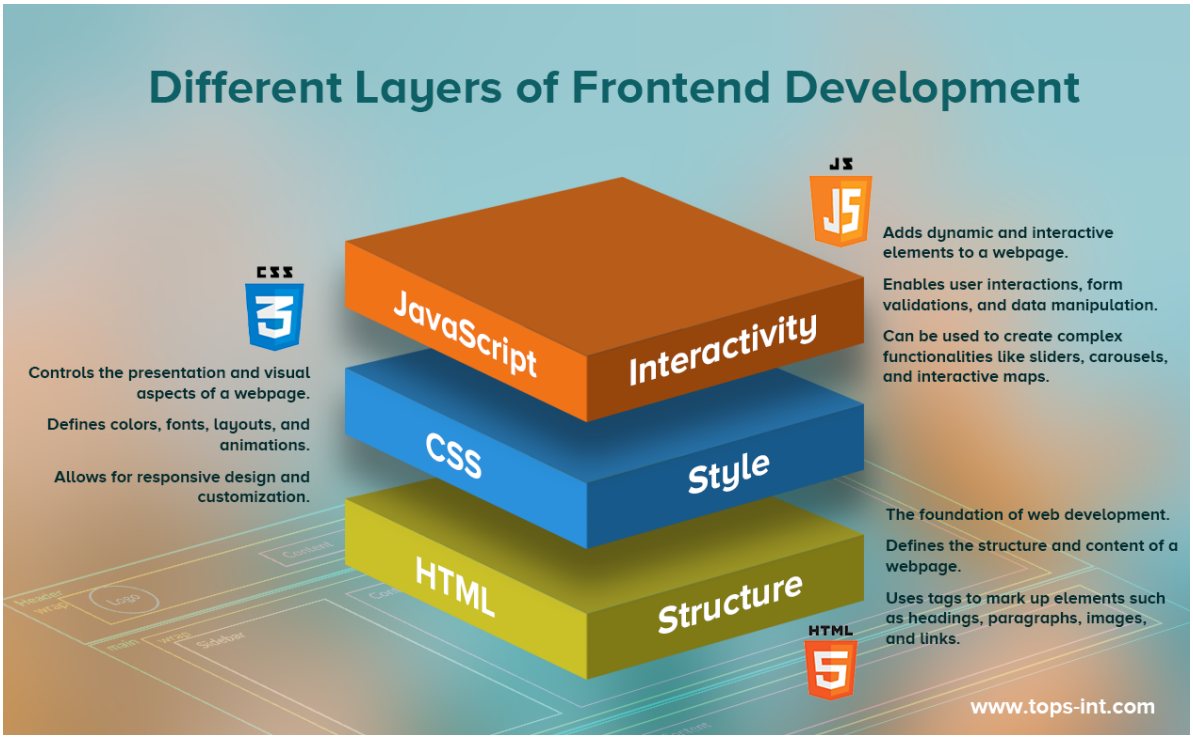
ຫາກໃຊ້ **Frontend Frameworks** ເພື່ອໃຫ້ການພັດທະນາງ່າຍຂຶ້ນ ເຊັ່ນ:

- **React.js**
- **Vue.js**
- **Angular**

ສິ່ງທີ່ Frontend Developer ຈະຮັບຜິດຊອບ

- ການອອກແບບ UI/UX ໃຫ້ສອດຄ່ອງກັບຜູ້ໃຊ້.
- ປະກອບສ່ວນຕ່າງໆ ຂອງເວັບໃຫ້ຄົບຖ້ວນ ແລະ ຄົງທີ່.
- ເຮັດໃຫ້ເວັບໄຊທ໌ແມ່ນ **Responsive** ສາມາດສະແດງຜົນໄດ້ດີທຸກຂະໜາດໜ້າຈໍ.
- ຕິດຕໍ່ກັບ Backend ເພື່ອດຶງ ຫລື ສົ່ງຂໍ້ມູນຈາກ API.

Frontend Programming ມີບົດບາດສຳຄັນໃນການພັດທະນາເວັບໄຊທ໌ ເພາະມັນເປັນສິ່ງທີ່ຜູ້ໃຊ້ສະໜັບສະໜູນຢູ່ໂດຍກົງ. 🚀



3.1 ຕິດຕັ້ງ React.js ດ້ວຍ Vite

📌 ສ້າງໂປເຈກ React ດ້ວຍ Vite

```
npm create vite@latest frontend-product -- --template react-ts
cd frontend-product
npm install
```

📌 ໂຄດ App.jsx

```
import { useState, useEffect } from 'react';
function App() {
    const [message, setMessage] = useState('Loading...');
    useEffect(() => {
        fetch('http://localhost:5000/')
            .then(response => response.json())
            .then(data => setMessage(data.message));
    }, []);

    return <h1>{message}</h1>;
}

export default App;
```

3.2 ການນຳໃຊ້ Tailwind CSS

3.2.1. Tailwind CSS ແມ່ນຫຍັງ? *(5 ນາທີ)*

Tailwind CSS ແມ່ນເຄື່ອງມືທີ່ຊ່ວຍໃຫ້ພວກເຮົາອອກແບບເວັບໄຊຕ໌ໄດ້ໄວ ແລະ ງ່າຍດ້ວຍການໃຊ້ "Utility Class" ໜ້າຍໆຕົວມາປະສົມກັນ. ມັນບໍ່ຄືກັບ Bootstrap ທີ່ມີ Component ສຳເລັດຮູບໃຫ້, ແຕ່ມັນໃຫ້ພວກເຮົາປັບແຕ່ງທຸກຢ່າງໄດ້ຕາມຕ້ອງການ.

- **ຕົວຢ່າງງ່າຍໆ:** ຖ້າຕ້ອງການກ່ອງສີແດງທີ່ມີຕົວອັກສອນສີຂາວ, ພຽງແຕ່ໃຊ້:

```
<div class="bg-red-500 text-white">ສະບາຍດີ</div>
```

- **ຜົນລັບ:** ກ່ອງສີແດງທີ່ມີຄຳວ່າ "ສະບາຍດີ" ສີຂາວຢູ່ໃນນັ້ນ.

⇒ Utility Class ແມ່ນຫຍັງ?

Utility Class ໃນ Tailwind CSS ແມ່ນຊື່ຂອງ "ຄຳສັ່ງສັ້ນໆ" ທີ່ໃຊ້ຄວບຄຸມຮູບແບບ (Style) ຂອງ HTML Element ໂດຍກົງ. ມັນຄືກັບເຄື່ອງມືນ້ອຍໆທີ່ເຮົາເລືອກມາໃຊ້ປະສົມກັນເພື່ອອອກແບບໂຕເວັບໄຊຕ໌ໄດ້ໄວ ແລະ ບໍ່ຕ້ອງຂຽນ CSS ເອງຫຼາຍ.

- **ແຕກຕ່າງຈາກ CSS ແບບດັ້ງເດີມ:**
  - CSS ປົກກະຕິ: ຕ້ອງຂຽນຄຳສັ່ງໃນໄຟລ໌ .css ແຍກ (ເຊັ່ນ: background-color: red;).
  - Utility Class: ໃສ່ຄຳສັ່ງໃນ HTML ໂດຍກົງ (ເຊັ່ນ: bg-red-500).
- **ຈຸດດີ:**
  - ໄວ, ບໍ່ຕ້ອງສະຫຼັບໄປມາໃນຫຼາຍໄຟລ໌.
  - ປັບແຕ່ງງ່າຍ, ເຫັນຜົນທັນທີ.

3.2.2. ການຕິດຕັ້ງ Tailwind CSS *(15 ນາທີ)*

ມີ 2 ວິທີທີ່ງ່າຍທີ່ສຸດສຳລັບຜູ້ເລີ່ມຕົ້ນ:

- **ວິທີ 1: ໃຊ້ CDN (ງ່າຍທີ່ສຸດ)**
  1. ສ້າງໄຟລ໌ HTML (ເຊັ່ນ: index.html).
  2. ໃສ່ໂຄດນີ້ໃນ <head>:

```
<script src="https://cdn.tailwindcss.com"></script>
```

  3. ທົດລອງໃຊ້:

```
<body>
 <h1 class="text-blue-600 text-3xl">ສະບາຍດີ Tailwind!</h1>
</body>
```

    - **ຜົນລັບ:** ໜ້າຈໍຂໍ້ສີຟ້າຂະໜາດໃຫຍ່ທີ່ມີຂໍ້ຄວາມ "ສະບາຍດີ Tailwind!"

- **ວິທີ 2: ຕິດຕັ້ງແບບເຕັ້ມ (ໃຊ້ Node.js)**

1. ຕິດຕັ້ງ Node.js (ດາວໂຫຼດຈາກ nodejs.org).
2. ສ້າງໂຟນເດີໂຄງການ ແລະ ເປີດ Terminal:
   - npm init -y (ສ້າງ package.json)
   - npm install -D tailwindcss (ຕິດຕັ້ງ Tailwind)
   - npx tailwindcss init (ສ້າງໄຟລ໌ tailwind.config.js)
3. ສ້າງໄຟລ໌ input.css ແລະໃສ່:

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

4. ຄຳສັ່ງຄອມໄພລ໌: npx tailwindcss -i ./input.css -o ./output.css --watch
5. ເຊື່ອມໄຟລ໌ output.css ໃນ HTML:

```
<link rel="stylesheet" href="output.css">
```

3.2.3. ການໃຊ້ Utility Class ພື້ນຖານ *(15 ນາທີ)*

Tailwind ໃຊ້ຄຳສັບງ່າຍໆມາປະກອບກັນເພື່ອອອກແບບ.

- **ສີ (Colors):**
  - bg-: ພື້ນຫຼັງ (background)
  - text-: ຕົວອັກສອນ
  - **ຕົວຢ່າງ:**

```
<div class="bg-yellow-300 text-black p-4">ສີເຫຼືອງ</div>
```

  - **ຜົນລັບ:** ກ່ອງສີເຫຼືອງອ່ອນ, ຕົວອັກສອນສີດຳ, ມີ Padding ຮອບດ້ານ.
- **ຂະໜາດ (Sizing):**
  - w-: ຄວາມກວ້າງ (width)
  - h-: ຄວາມສູງ (height)
  - p-: Padding

    - m-: Margin
    - **ຕົວຢ່າງ:**

```
<div class="w-64 h-32 bg-green-500 m-4">ກ່ອງຂະໜາດກຳນົດ</div>
```

    - **ຜົນລັບ:** ກ່ອງສີຂຽວ, ກວ້າງ 16rem (256px), ສູງ 8rem (128px), ມີ Margin 1rem.
- **Typography:**
    - text-: ຂະໜາດຕົວອັກສອນ (ເຊັ່ນ: text-lg, text-2xl)
    - font-: ຮູບແບບ (ເຊັ່ນ: font-bold, font-italic)
    - **ຕົວຢ່າງ:**

```
<p class="text-xl font-bold text-center">ຂໍ້ຄວາມໃຫຍ່</p>
```

    - **ຜົນລັບ:** ຂໍ້ຄວາມຂະໜາດໃຫຍ່, ໜາ, ຢູ່ກາງ.

3.2.4. Responsive Design (10 ນາທີ)

Tailwind ຊ່ວຍໃຫ້ເຮົາປັບແຕ່ງຕາມຂະໜາດຈໍໄດ້ງ່າຍດ້ວຍ Prefix.

- **Prefix:**
    - sm: (≥576px)
    - md: (≥768px)
    - lg: (≥1024px)
- **ຕົວຢ່າງ:**

```
  <div class="bg-blue-500 text-white p-4 sm:bg-red-500 md:text-left
lg:text-2xl">
      Responsive Test
   </div>
```

- **ຜົນລັບ:**
    - ຈໍນ້ອຍ: ກ່ອງສີຟ້າ, ຕົວອັກສອນກາງ.
    - ຈໍ sm: ກ່ອງສີແດງ.

- ຈໍ md: ຕົວອັກສອນຊ້າຍ.
- ຈໍ lg: ຕົວອັກສອນໃຫຍ່ຂຶ້ນ.

3.2.5. ການປັບແຕ່ງ (Customization) *(10 ນາທີ)*

ຖ້າຕ້ອງການສີ ຫຼື ຂະໜາດໃໝ່, ແກ້ໄຂໄຟລ໌ tailwind.config.js.

- **ຕົວຢ່າງ:**

```
module.exports = {
    theme: {
    extend: {
        colors: {
        laoGreen: '#00cc66',
        },
        spacing: {
        '15': '3.75rem', // 60px
        },
    },
    },
};
```

- **ການນຳໃຊ້:**

```
<div class="bg-laoGreen p-15">ສີຂຽວລາວ</div>
```

- **ຜົນລັບ:** ກ່ອງສີຂຽວຕາມທີ່ກຳນົດ, Padding 60px.

3.2.6. ການສ້າງ Component ງ່າຍໆ *(10 ນາທີ)*

- **ຕົວຢ່າງ Button:**

```
 <button class="bg-purple-600 text-white px-6 py-3 rounded-lg hover:bg-purple-800">
  ກົດປຸ່ມນີ້
```

```
</button>
```

- **ຜົນລັບ:** ປຸ່ມສີມ່ວງ, ຕົວອັກສອນສີຂາວ, ມີມຸມໂຄ້ງ, ປ່ຽນສີເມື່ອຊີ້ເມົາສ໌.
- **ຕົວຢ່າງ Card:**

```
<div class="max-w-sm bg-white shadow-lg rounded-lg p-6">
  <h2 class="text-xl font-bold">ຫົວຂໍ້</h2>
  <p class="text-gray-700">ນີ້ແມ່ນເນື້ອຫາຂອງ Card.</p>
</div>
```

- **ຜົນລັບ:** ກ່ອງສີຂາວມີເງົາ, ມີຫົວຂໍ້ແລະຂໍ້ຄວາມພາຍໃນ.

3.2.7. ສະຫຼຸບ ແລະ ຄຳແນະນຳ *(5 ນາທີ)*

- Tailwind ເໝາະກັບຄົນທີ່ມັກຄວບຄຸມການອອກແບບທຸກຢ່າງ.
- ຝຶກໃຊ້ Utility Class ຈົນກວ່າຈະຄຸ້ນ (ເບິ່ງ Cheat Sheet ທີ່ tailwindcss.com).
- ຖ້າຕ້ອງການຕົວຢ່າງໂຄງການເພີ່ມ, ລອງເບິ່ງທີ່ Tailwind Components.

> ### ສຶກສາເພີ່ມຕື່ມ:
> https://tailwindcss.com/docs/installation/using-vite

3.3 ການໃຊ້ Ant Design (Antd)

ພາກທີ 1: ພື້ນຖານ ແລະ ການຕິດຕັ້ງ

1. **Antd ແມ່ນຫຍັງ?**
   - Antd ແມ່ນຊຸດເຄື່ອງມື UI ທີ່ໃຊ້ກັບ React ເພື່ອສ້າງໂຕະພື້ນຜູ້ໃຊ້ (UI) ທີ່ສວຍງາມ ແລະ ມີປະສິດທິພາບ.
   - ມັນມີຄອມໂພເນນຕ່າງໆ ເຊັ່ນ Button, Form, Table, Modal ແລະ ອື່ນໆ.
2. **ການຕິດຕັ້ງ**
   - ກ່ອນອື່ນ, ຕ້ອງມີ **Node.js** ແລະ **npm** ຕິດຕັ້ງໃນຄອມພິວເຕີຂອງເຈົ້າ.
   - ສ້າງໂຄງການ React ໂດຍໃຊ້ create-react-app:

```
npx create-react-app my-antd-project
cd my-antd-project
```

- ຕິດຕັ້ງ Antd:

```
npm install antd
```

- ນຳເຂົ້າ CSS ຂອງ Antd ໃນໄຟລ src/index.js ຫຼື src/App.js:

```
import 'antd/dist/antd.css'; // ນຳເຂົ້າ CSS ຂອງ Antd
```

3. **ຕົວຢ່າງພື້ນຖານ: ການໃຊ້ Button**
   - ແກ້ໄຂໃນ src/App.js:

```
import React from 'react';
import { Button } from 'antd';

function App() {
return (
  <div style={{ padding: '20px' }}>
  <h1>ຍິນດີຕ້ອນຮັບກັບ Antd</h1>
  <Button type="primary" onClick={() => alert('ສະບາຍດີ!')}>
    ກົດຂ້ອຍ
  </Button>
  </div>
);
}

export default App;
```

   - ທົດລອງແລ່ນໂຄງການ:

```
npm start
```

- ຜົນ: ຈະມີປຸ່ມສີຟ້າປາກົດຂຶ້ນ ແລະ ເມື່ອກົດຈະມີການ alert "ສະບາຍດີ!".

ພາກທີ 2: ການໃຊ້ຄອມໂພເນນພື້ນຖານ

1. **ການໃຊ້ Layout**
   - Antd ມີຄອມໂພເນນ Layout ສຳລັບການຈັດຮູບແບບໜ້າຈໍ.
   - ຕົວຢ່າງ:

```
import React from 'react';
import { Layout, Menu } from 'antd';

const { Header, Content, Footer } = Layout;

function App() {
return (
  <Layout>
  <Header>
    <Menu theme="dark" mode="horizontal" defaultSelectedKeys={['1']}>
    <Menu.Item key="1">ໜ້າຫຼັກ</Menu.Item>
    <Menu.Item key="2">ກ່ຽວກັບ</Menu.Item>
    </Menu>
  </Header>
  <Content style={{ padding: '20px', minHeight: '80vh' }}>
    <h1>ເນື້ອຫາຫຼັກ</h1>
  </Content>
  <Footer style={{ textAlign: 'center' }}>
    © 2025 ສ້າງໂດຍ Antd
  </Footer>
  </Layout>
);
}
```

```
export default App;
```

- ຜົນ: ໜ້າຈໍທີ່ມີ Header (ມີເມນູ), Content, ແລະ Footer.

2. **ການໃຊ້ Form**
   - ຕົວຢ່າງການສ້າງຟອມລົງທະບຽນ:

```
import React from 'react';
import { Form, Input, Button } from 'antd';

function App() {
const onFinish = (values) => {
  console.log('ຂໍ້ມູນທີ່ໄດ້ຮັບ:', values);
};

return (
  <div style={{ padding: '20px', maxWidth: '400px', margin: '0 auto' }}>
  <h1>ຟອມລົງທະບຽນ</h1>
  <Form name="register" onFinish={onFinish}>
    <Form.Item
    name="username"
    rules={[{ required: true, message: 'ກະລຸນາປ້ອນຊື່ຜູ້ໃຊ້!' }]}
    >
    <Input placeholder="ຊື່ຜູ້ໃຊ້" />
    </Form.Item>
    <Form.Item
    name="password"
    rules={[{ required: true, message: 'ກະລຸນາປ້ອນລະຫັດຜ່ານ!' }]}
    >
    <Input.Password placeholder="ລະຫັດຜ່ານ" />
    </Form.Item>
    <Form.Item>
    <Button type="primary" htmlType="submit">
      ລົງທະບຽນ
    </Button>
    </Form.Item>
  </Form>
```

```
    </div>
);
}

export default App;
```

- ຜົນ: ຟອມທີ່ມີການກວດສອບຂໍ້ມູນ (validation) ແລະ ສົ່ງຂໍ້ມູນໄປ console.

ພາກທີ 3: ການໃຊ້ງານລະດັບກາງ

1. **ການໃຊ້ Table**
   - ຕົວຢ່າງການສະແດງຂໍ້ມູນໃນຕາຕະລາງ:

```
import React from 'react';
import { Table } from 'antd';

const columns = [
{ title: 'ຊື່', dataIndex: 'name', key: 'name' },
{ title: 'ອາຍຸ', dataIndex: 'age', key: 'age' },
{ title: 'ທີ່ຢູ່', dataIndex: 'address', key: 'address' },
];

const data = [
{ key: '1', name: 'ສົມສັກ', age: 32, address: 'ວຽງຈັນ' },
{ key: '2', name: 'ບຸນມີ', age: 25, address: 'ຫຼວງພະບາງ' },
{ key: '3', name: 'ສີລົມ', age: 28, address: 'ສະຫວັນນະເຂດ' },
];

function App() {
return (
    <div style={{ padding: '20px' }}>
    <h1>ຕາຕະລາງຂໍ້ມູນ</h1>
    <Table columns={columns} dataSource={data} />
    </div>
```

```
);
}

export default App;
```

- ຜົນ: ຕາຕະລາງທີ່ສະແດງຂໍ້ມູນຢ່າງເປັນລະບຽບ.

2. **ການໃຊ້ Modal**
   - ຕົວຢ່າງການສ້າງກ່ອງປ໊ອບອັບ:

```
import React, { useState } from 'react';
import { Button, Modal } from 'antd';

function App() {
const [isModalVisible, setIsModalVisible] = useState(false);

const showModal = () => setIsModalVisible(true);
const handleOk = () => setIsModalVisible(false);
const handleCancel = () => setIsModalVisible(false);

return (
  <div style={{ padding: '20px' }}>
  <Button type="primary" onClick={showModal}>
    ເປີດ Modal
  </Button>
  <Modal
    title="ຫົວຂໍ້ Modal"
    visible={isModalVisible}
    onOk={handleOk}
    onCancel={handleCancel}
  >
    <p>ນີ້ແມ່ນເນື້ອຫາພາຍໃນ Modal.</p>
  </Modal>
  </div>
);
}
```

```
export default App;
```

- ຜົນ: ກົດປຸ່ມຈะมิ Modal ปากົดขึ้นมา.

ພากที่ 4: กานใช้งานละดับมืออาชีบ

1. **กานปับแต่ง Theme**
   - ปับแต่งสี และ ธูบแบบของ Antd โดยใช้ less:
     - ติดตั้ง less และ less-loader:

```
npm install less less-loader --save-dev
```

     - สร้างไฟล src/customTheme.less:

```
@primary-color: #ff4d4f; // ปรับสีหลัก
@font-size-base: 16px;   // ปรับขะหนาดตัวอักสอน
```

     - นำใช้ใน src/index.js:

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import './customTheme.less';

ReactDOM.render(<App />, document.getElementById('root'));
```

   - ผึน: UI ของ Antd จะปรับสี และ ธูบแบบตามที่กำนด.

2. **กานเชื่อมโยงกับ API**
   - ตัวอย่างกานดึงข้อมูนจาก API และสะแดงใน Table:

```
import React, { useState, useEffect } from 'react';
```

```
import { Table } from 'antd';
import axios from 'axios';

function App() {
  const [data, setData] = useState([]);
  const columns = [
    { title: 'ID', dataIndex: 'id', key: 'id' },
    { title: 'ຊື່', dataIndex: 'title', key: 'title' },
  ];

  useEffect(() => {
    axios.get('https://jsonplaceholder.typicode.com/posts')
      .then(response => setData(response.data));
  }, []);

  return (
    <div style={{ padding: '20px' }}>
      <h1>ຂໍ້ມູນຈາກ API</h1>
      <Table columns={columns} dataSource={data} rowKey="id" />
    </div>
  );
}

export default App;
```

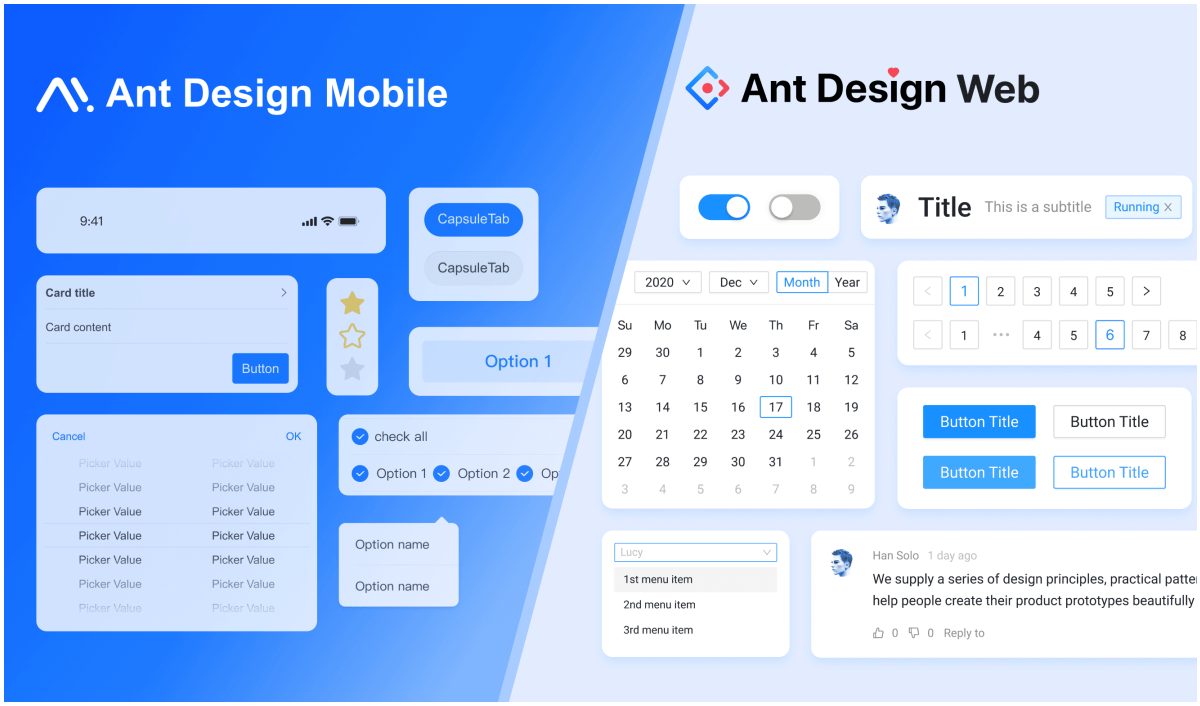
- ຕິດຕັ້ງ axios:

```
npm install axios
```

- ຜົນ: ຕາຕະລາງທີ່ດຶງຂໍ້ມູນມາຈາກ API.

ຄຳແນະນຳສຳລັບມືອາຊີບ

- **ການຈັດການສະຖານະ (State Management):** ໃຊ້ Redux ຫຼື Context API ຮ່ວມກັບ Antd ເພື່ອຈັດການຂໍ້ມູນຂະໜາດໃຫຍ່.
- **ການເພີ່ມປະສິດທິພາບ:** ໃຊ້ React.memo ຫຼື useMemo ເພື່ອປ້ອງກັນການປະມວນຜົນຊ້ຳຊ້ອນ.



ສຶກສາເພີ່ມຕື່ມ:

https://ant.design/docs/react/introduce#installation

ບົດທີ 4: ການເຊື່ອມຕໍ່ Frontend ແລະ Backend

4.1 ໃຊ້ Axios ໃນການເຊື່ອມຕໍ່ API

📌 ຕິດຕັ້ງ Axios

```
npm install axios
```

📌 ການໃຊ້ງານ Axios

ການໃຊ້ງານ Axios ໃນບົດຮຽນນີ້ຈະສາທິດພຽງຂັ້ນຕອນການຕິດຕໍ່ກັບ Backend ເທົ່ານັ້ນ ສ່ວນຂະບວນການທາງ Frontend, Render, conditioning ແລະ interactive ອື່ນໆຈະບໍ່ໄດ້

ກຳນົດໄວ້ໃນບົດຮຽນນີ້. ໝາຍຄວາມວ່າຈະໄດ້ຮຽນໃນຫ້ອງຮຽນຕົວຈິງໄປນຳກັນເທົ່ານັ້ນ. ການໃຊ້ງານທັງໝົດຈະຖືກລວບລວມມາຕັ່ງລຸ່ມນີ້:

🔹 ດຶງຂໍ້ມູນຈາກ Backend

```
const getUsers = async () => {
    try {
        const response = await axios.get(
            'http://localhost:3000/app/user/users',
            {
                headers: {
                    Authorization: `Bearer ${token}`,  // ໃຫ້ໃຊ້ jwt token
                },
            }
        );
        console.log(response.data || 'Users retrieved successfully!');
    } catch (error: any) {
        console.log(error?.response?.data?.message || 'Something went wrong');
    }
};
```

🔹 ເພີ່ມຂໍ້ມູນໄປຫາ Backend

```
const createUser = async () => {
    try {
        const response = await axios.post(
        'http://localhost:3000/app/user/create',
        user,
        {
            headers: {
            Authorization: `Bearer ${token}`,  // ໃຫ້ໃຊ້ jwt token
            },
        }
        );
```

```
        console.log(response.data.message || 'User created successfully!');
    } catch (error: any) {
        console.log(error?.response?.data?.message || 'Something went wrong');
    }
};
```

🔷 ສົ່ງຄຳຂໍແກ້ໄຂຂໍ້ມູນໄປຫາ Backend

```
const updateUser = async () => {
    try {
        const response = await axios.put(
            'http://localhost:3000/app/user/update',
            {
                username: "administrator",
                password: "123456",
                role: "user"
            },
            {
                headers: {
                    Authorization: `Bearer ${token}`, // ໃຫ້ໃຊ້ jwt token
                },
            }
        );
        console.log(response.data.message || 'User updated successfully!');
    } catch (error: any) {
        console.log(error?.response?.data?.message || 'Something went wrong');
    }
};
```

🔷 ສົ່ງຄຳຂໍລົບຂໍ້ມູນໄປຫາ Backend

```
const deleteUser = async (userId: string) => {
```

```
    try {
      const response = await axios.delete(
        `http://localhost:3000/app/user/delete`,
        {
          headers: {
            Authorization: `Bearer ${token}`,  // ໃຫ້ໃຊ້ jwt token
          },
          params: { id: userId }
        }
      );
      console.log(response.data.message || 'User deleted successfully!');
    } catch (error: any) {
      console.log(error?.response?.data?.message || 'Something went wrong');
    }
  };
```

4.2 Hooks ແລະ ປະເພດຂອງ Hooks ໃນ React

Hook ໃນ React ແມ່ນ ເຄື່ອງມືໃນການເຂົ້າເຖິງ state ແລະ lifecycle ຂອງ component ໂດຍບໍ່ຈຳເປັນຕ້ອງໃຊ້ class component ເພື່ອເຮັດໃຫ້ການຂຽນໂຄດງ່າຍ, ສັ້ນ ແລະ ສະອາດ.

1. **ຄວາມຈຳເປັນຂອງ Hook**
   - ເຮັດໃຫ້ໃຊ້ function components ເພື່ອຄວບຄຸມ state, effect, context ໄດ້ຄືກັບ class component
   - ສະອາດກວ່າການໃຊ້ class, ໂດຍບໍ່ມີຄຳວ່າ: this
   - ສາມາດ reuse ໂລຈິກໄດ້ (ຜ່ານ custom hooks)

2. **ປະເພດຂອງ Hook ພື້ນຖານ (Built-in Hooks)**
   - **useState:** ແມ່ນ React Hook ທີ່ໃຊ້ເພື່ອເກັບຄ່າຂໍ້ມູນພາຍໃນ component ແລະ ສາມາດ ປ່ຽນແປງໄດ້ເມື່ອຈຳເປັນ.

```
import { useState } from 'react';

function Counter() {
```

```
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>จำนวน: {count}</p>
      <button onClick={() => setCount(count + 1)}>เพิ่ม</button>
    </div>
  );
}
```

- **useEffect:** ແມ່ນ React Hook ທີ່ໃຊ້ເພື່ອ ຈັດການຜົນຂ້າງເຄັຽ (side effects) ໃນຟັງຊັນ component ເຊັ່ນ:
  - ເອີ້ນໃຊ້ງານ API
  - ດຶງຂໍ້ມູນ
  - ຕັ້ງຄ່າ event listener
  - ຈັດການ ການ render ຄັ້ງທຳອິດ (on mount) ຫຼື ເມື່ອ dependency ປ່ຽນແປງ

```
useEffect(() => {
  console.log("Fetching data");
  fetchData();
}, [userId]);
```

- **useContext**: ແມ່ນ React Hook ທີ່ໃຊ້ເພື່ອເຂົ້າເຖິງຂໍ້ມູນຈາກ Context API ໂດຍບໍ່ຈຳເປັນຕ້ອງສົ່ງ props ຜ່ານຫຼາຍໆຊັ້ນ.

```
import React, { useContext, createContext } from 'react';

// 1. ສ້າງ context
```

```
const ThemeContext = createContext<'light' | 'dark'>('light');

// 2. ใช้ context
function ThemedComponent() {
  const theme = useContext(ThemeContext);
  return <p>ຫົວຂໍ້ຂອງ theme: {theme}</p>;
}

// 3. ส่งค่า context ผ่าน Provider
export default function App() {
  return (
    <ThemeContext.Provider value="dark">
      <ThemedComponent />
    </ThemeContext.Provider>
  );
}
```

- **useRef:** ແມ່ນ React Hook ທີ່ໃຊ້ເພື່ອ:

  ⇒ ເຂົ້າເຖິງ DOM element ໂດຍກົງ (ເຊັ່ນ: focus, scroll)

  ⇒ ເກັບຄ່າທີ່ບໍ່ກ່ຽວຂ້ອງກັບ re-render (ເຊັ່ນ: count, timer)

```
import { useRef } from "react";

function FocusInput() {
  const inputRef = useRef<HTMLInputElement>(null);

  const handleFocus = () => {
    inputRef.current?.focus();
```

```
  };

  return (
    <>
      <input ref={inputRef} type="text" placeholder="ພິມຊື່..." />
      <button onClick={handleFocus}>Focus</button>
    </>
  );
}
```

- **useMemo:** ແມ່ນ React Hook ທີ່ໃຊ້ສຳລັບ ເກັບຄ່າທີ່ຄຳນວນແລ້ວ ເພື່ອບໍ່ໃຫ້ຄຳນວນໃໝ່ ທຸກເທື່ອ ຍົກເວັ້ນວ່າ dependencies ປ່ຽນເປັນຄ່າໃໝ່.

```
import { useMemo, useState } from "react";

function ExpensiveComponent({ number }: { number: number }) {
  const expensiveResult = useMemo(() => {
    console.log("ຄຳນວນຄ່າ...");
    let result = 0;
    for (let i = 0; i < 1e6; i++) {
      result += number * 2;
    }
    return result;
  }, [number]);

  return <p>ຜົນລັບ: {expensiveResult}</p>;
}
```

- **useCallback:** ແມ່ນ React Hook ທີ່ໃຊ້ສຳລັບ "ຈົດຈຳ" ຟັງຊັນໃຫ້ເປັນຕົວດຽວ (same reference) ໃນທຸກຮອບການ render ທີ່ dependencies ບໍ່ມີການປ່ຽນແປງ.

```
import { useCallback, useState } from "react";

function ParentComponent() {
  const [count, setCount] = useState(0);

  const handleClick = useCallback(() => {
    setCount(prev => prev + 1);
  }, []);

  return <ChildComponent onClick={handleClick} />;
}

function ChildComponent({ onClick }: { onClick: () => void }) {
  console.log("Child render");
  return <button onClick={onClick}>ຄິດເລກຕໍ່ໄປ</button>;
}
```

- **useReducer:** ແມ່ນ Hook ທີ່ໃຊ້ຈັດການສະຖານະ (state) ທີ່ມີຄວາມຊັບຊ້ອນ ຫຼື ມີຫຼາຍເງື່ອນໄຂໃນການປ່ຽນ state. ເປັນຮູບແບບການຈັດການ State ໃນ Redux ແຕ່ຖືກໃຊ້ໃນ Component ໂດຍກົງ.

```
import { useReducer } from 'react';

type State = { count: number };
type Action = { type: 'increment' } | { type: 'decrement' };

function reducer(state: State, action: Action): State {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
```

```
    case 'decrement':
      return { count: state.count - 1 };
    default:
      return state;
  }
}

export default function Counter() {
  const [state, dispatch] = useReducer(reducer, { count: 0 });

  return (
    <div>
      <p>ຄ່າ: {state.count}</p>
      <button onClick={() => dispatch({ type: 'increment' })}>ເພີ່ມ</button>
      <button onClick={() => dispatch({ type: 'decrement' })}>ຫຼຸດ</button>
    </div>
  );
}
```

ຕາຕະລາງສະຫຼຸບນິຍາມ ແລະ ຄວາມແຕກຕ່າງຂອງທຸກ React Hook ໃຫ້ເຂົ້າໃຈງ່າຍໆ

Hook	ນິຍາມ	ໃຊ້ເມື່ອໃດ	ຄວາມແຕກຕ່າງ
useState	ໃຊ້ສຳລັບຈັດການຄ່າ ສະຖານະໃນ Component.	ເມື່ອຕ້ອງການກຳນົດ/ປ່ຽນຄ່າງ່າຍໆ.	ເປັນແບບ basic ທີ່ສຸດ ໃຊ້ງ່າຍທີ່ສຸດ.
useEffect	ໃຊ້ເພື່ອຈັດການ side effects (fetch, event).	ເມື່ອຕ້ອງການເຮັດ ຫຍັງບາງຢ່າງຫຼັງຈາກ render ແລ້ວ.	ລັກສະນະຈັດການກັບ lifecycle ໃນ component.
useContext	ດຶງຂໍ້ມູນຈາກ context ໂດຍບໍ່ຕ້ອງເຮັດ prop drilling.	ເມື່ອມີ state ຫຼືຄ່າທີ່ ໃຊ້ຫຼາຍບ່ອນ.	ຮ່ວມໃຊ້ຂໍ້ມູນລະຫວ່າງ ຫຼາຍ components.
useRef	ເກັບຄ່າທີ່ບໍ່ເກີດ	ເກັບ DOM ref ຫຼື ຄ່າ	ບໍ່ເຮັດໃຫ້

	ປະຕິກິລິຍາຕໍ່ກັບການປ່ຽນເມື່ອມີການ render ໃຫມ່.	ທີ່ບໍ່ຕ້ອງການ render ໃໝ່.	Component render ໃໝ່ເມື່ອຄ່າປ່ຽນແປງ.
useMemo	ຄືກັບ cache ຜົນຄຳນວນໃນຄັ້ງກ່ອນເພື່ອປ້ອງກັນຄຳນວນໃໝ່.	ເມື່ອມີການຄຳນວນຫນັກ ແລະ ຄ່າບໍ່ປ່ຽນ.	ໃຊ້ໃນການ cache ຄ່າທີ່ເກີດຈາກ function.
useCallback	ຄືກັບ memoize function (ຈຳໄວ້ໃຫ້ຄືເກົ່າ).	ເມື່ອສົ່ງ function ໃຫ້ child ແລ້ວບໍ່ຢາກໃຫ້ child render ໃໝ່ຖ້າບໍ່ຈຳເປັນ.	ຄ້າຍ useMemo ແຕ່ໃຊ້ກັບ function.
useReducer	ແທນ useState ສຳລັບ state ທີ່ຊັບຊ້ອນ ແລະ ຄວບຄຸມງ່າຍ.	ເມື່ອມີຫຼາຍຄ່າຕ້ອງການປ່ຽນ state ຫຼື state ຊັບຊ້ອນ.	ມີ reducer function ຄວບຄຸມ logic ການປ່ຽນແປງ.

3. Custom Hook ຄືຫຍັງ?

ແມ່ນ Hook ທີ່ສ້າງຂຶ້ນໂດຍຜູ້ພັດທະນາເອງເພື່ອງ່າຍຕໍ່ການນຳໃຊ້ໂຄດແບບຊ້ຳໆເຊັ່ນ:

```
export const useWindowWidth = () => {
  const [width, setWidth] = useState(window.innerWidth);

  useEffect(() => {
    const handleResize = () => setWidth(window.innerWidth);
    window.addEventListener('resize', handleResize);
    return () => window.removeEventListener('resize', handleResize);
  }, []);

  return width;
};
```

4.3 Redux State management

4.3.1. ຕິດຕັ້ງ Packages

```
npm install @reduxjs/toolkit react-redux
npm install --save-dev @types/react-redux
```

4.3.2. Create Root Store

Store file ແມ່ນສູນກາງທີ່ເກັບທຸກ state ຈາກ slices ແລະສົ່ງໃຫ້ React app ໃຊ້ງານຕໍ່.

ປະໂຫຍດ:

- ຮວບຮວມ reducers ທັງໝົດ
- ສ້າງ store ໃຫ້ React ສາມາດເຂົ້າເຖິງ state ໄດ້

📁 ສ້າງ File ພາຍໃຕ້ Folder ດັ່ງນີ້ src/redux /store.ts

```
import { configureStore } from '@reduxjs/toolkit';
import cartReducer from './slice/cart';

export const store = configureStore({
  reducer: {
    cartInfo: cartReducer,
  },
});

export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;

export default store;
```

4.3.3. Create Redux Slice

Slice file ແມ່ນບ່ອນທີ່ກຳນົດ state ແລະ function (reducers) ທີ່ໃຊ້ຈັດການຂໍ້ມູ.

ປະກອບມີ:

- initialState: ຂໍ້ມູນເລີ່ມຕົ້ນ
- reducers: ຟັງຊັນເພື່ອເພີ່ມ, ແກ້ໄຂ, ລົບຂໍ້ມູນ

📁 สร้าง File พายใต้ Folder ดั่งนี้ src/redux/slice/cart.ts

```
import { createSlice, PayloadAction } from '@reduxjs/toolkit';

type CartItem = {
productId: Number;
productName: string;
productSlug: string;
price: Number;
qty: Number;
};

type CartState = CartItem[];

const initialState: CartState = [];

const cartSlice = createSlice({
   name: 'cart',
initialState,
reducers: {
   addCartItem: (state, action: PayloadAction<CartItem>) => {
   const index = state.findIndex(item => item.productId ===
 action.payload.productId);

   if (index !== -1) {
      state[index] = {
      ...state[index],
      qty: Number(state[index].qty) + Number(action.payload.qty),
      };
   } else {
      state.push(action.payload);
   }
   },

   removeCartItem: (state, action: PayloadAction<number>) => {
   return state.filter(item => item.productId !== action.payload);
```

```
    },

    resetCart: () => initialState,
  },
  });

  export const { addCartItem, removeCartItem, resetCart } = cartSlice.actions;
  export default cartSlice.reducer;
```

4.3.4. Setup Provider

ການນຳໃຊ້ໃຫ້ໃຊ້ຜ່ານ Provider ຂອງ Redux ທີ່ຢູ່ຊັ້ນນອກສຸດຂອງ Application ໂດຍສ່ວນຫຼາຍແລ້ວແມ່ນຈະຢູ່ໃນ main.tsx ຫຼື index.tsx

<Provider> ແມ່ນ component ທີ່ມາຈາກ react-redux ມີໜ້າທີ່:

🎁 ໃຊ້ໃນການຫໍ່ (wrap) App ເພື່ອໃຫ້ແອັບທັງໝົດເຂົ້າເຖິງ Redux store ໄດ້

```
import { Provider } from 'react-redux';
import { store } from './store';

ReactDOM.createRoot(document.getElementById('root')!).render(
  <Provider store={store}>
    <App />
  </Provider>
);
```

4.3.5. Use Redux State and Actions in Components

ການໃຊ້ງານແມ່ນຕ້ອງ import useDispatch & useSelector ທຸກໆຄັ້ງ

```
import { useDispatch } from 'react-redux';
import { useSelector } from 'react-redux';
```

- **useDispatch** ແມ່ນຈະໃຊ້ເພື່ອຈັດການຄ່າໃນ State ຂອງ Redux ເຊັ່ນ: ການເພີ່ມກະຕ່າ, ລົບກະຕ່າ ເປັນຕົ້ນ.
- **useSelector** ແມ່ນໃຊ້ສຳລັບການເຂົ້າເຖິງ State ໃນ Redux.

```
import { FaMinus, FaPlus } from 'react-icons/fa';
import { MdOutlineClear } from 'react-icons/md';
import { useDispatch } from 'react-redux';
import { useSelector } from 'react-redux';
import { cartState } from '../redux/slice/cart';


const ReduxUsageComponent = () => {
  const dispatch = useDispatch();
  const selectorCartItem = useSelector(cartState);

  return (
    <>

  //ການຈັດການ State
     <Button icon={<FaPlus />} type='link'
onClick={()=>dispatch(addCartItem(item))}>Add cart</Button>
     <Button icon={<FaMinus />} type='link'
onClick={()=>dispatch(removeCartItem(item))}>Remove cart</Button>
     <Button icon={<MdOutlineClear />} type='link'
onClick={()=>dispatch(resetCart())}>Remove cart</Button>

   //ການເຂົ້າເຖິງ State
  <Badge size="small" count={selectorCartItem.length} overflowCount={99}
color='#1677ff' className='!p-0 !mr-16 !my-auto !mt-5'>
     <Avatar src={<BsMinecartLoaded color='white' size={20}/>} size={"large"}
className='!p-0 cursor-pointer !opacity-90'/>
  </Badge>
    </>
```

```
  )
}
```

ບົດທີ່ 5: ພື້ນຖານການນຳໃຊ້ git

1. ແນະນຳກ່ຽວກັບ Git ແລະ GitHub

Git ແມ່ນຫຍັງ?

Git ແມ່ນ version control system (VCS) ເຊິ່ງໃຊ້ໃນການຄວບຄຸມການປ່ຽນແປງໃນໂປຣແກຣມ ຫຼື ໄຟລ์ໂຄງການຂອງທ່ານ ຊ່ວຍໃຫ้เห็นประวัติการแก้ไข และ กับคืนหาสะพายเก่าได้.

GitHub ແມ່ນຫຍັງ?

GitHub ແມ່ນບໍລິການ cloud ທີ່ໃຫ้ท่านเก็บโครงการที่บ่อยู่ในเครื่องของท่าน และให้ทำงานร่วมกันกับทีมงานได้ง่ายขึ้น.

ຄວາມແຕກต่างละหว่าง Git ແລะ GitHub:

Git	GitHub
แอ๊บ Git ในเครื่อง	แพลตฟอมเอ๊บ (Git repository hosting)
จัดการโครงการในเครื่อง	จัดเก็บโครงการใน cloud
ใช้ได้โดยบ่ต้องเชื่อม internet	ต้องใช้ internet ในการ sync โครงการ

2. Initial Git Project

```
mkdir my-project
cd my-project
git init
```

git init: สร้าง Git repository ที่เป็น local ในไฟล์โฟมเด้ของโครงการ.

3. Push Code to Git (GitHub)

1.ເພີ່ມໄຟລ์:

```
git add .
```

2. ส้างๆ commit:

```
git commit -m "Initial commit"
```

3. ເຊື່ອມຕໍ່ກັບ GitHub repo

```
git remote add origin https://github.com/username/repo-name.git
```

4 Push ໂຄດໄປ Github

```
git push -u origin main
```

4. Pull Code ຈາກ Github

ດึງ code ล่าสุดจาก GitHub:

```
git pull origin main
```

ມັກໃຊ้ເมื่อมีการปรับแปลงอยู่บ่อนอื่น (เช่น: ลูกทีม) และต้องการดึงมาใช้ในเครื่องของตัน.

5. git commands ອื่นๆที่ต้องรู้

Command	ອะທิบาย
git branch	ดูลายชื่อทุก branch
git branch <branch-name>	ส้างๆ branch ใหม่ (บ่ switch)

git checkout -b <branch-name>	ສ້າງແລະສະຫຼັບໄປ branch ໃໝ່ທັນທີ
git merge	ລວມການປ່ຽນແປງຈາກ branch ໜຶ່ງມາອີກ branch
git pull	ດຶງຂໍ້ມູນຈາກ remote ແລະ merge ອັດຕະໂນມັດ
git fetch	ດຶງຂໍ້ມູນຈາກ remote ມາ local (ແຕ່ບໍ່ merge)

ຕົວຢ່າງການໃຊ້ງານ

```
git checkout -b dev
```

🔥 ສະຫຼຸບ

Full Stack Web Development ເປັນລະບົບທີ່ຂຽນເວັບໄຊຕ໌ສະໄໝໃໝ່ ທີ່ນີ້ຍົມກັນຢ່າງກວ້າງຂວາງວົງການພັດທະນາຊັອບແວຣ໌ຊຶ່ງມາຮອດຕອນນີ້ນັກຮຽນ ໄດ້ຮຽນຮູ້ພື້ນຖານໂດຍລວມໃນການສ້າງ CRUD, Authentication & Authorization ດ້ວຍ JWT ລວມເຖິງການເຂົ້າລະຫັດເພື່ອຄວາມປອດໄພ ໃນການສົ່ງຂໍ້ມູນລະຫວ່າງໜ້າບ້ານ ແລະ ຫຼັງບ້ານຕາມ ມາດຕະຖານຕະຫຼອດຮອດການໃຊ້ TailwindCSS Framework ແລະ ການໃຊ້ Antd Component Framework ເພື່ອຈັດການໜ້າເວັບໄຊຕ໌ໃຫ້ສວຍງາມ.

ບົດຮຽນທີ່ໄດ້ຮຽນຜ່ານມາຂ້າງເທິງນັ້ນເປັນສ່ວນໜຶ່ງໃນການພັດທະນາລະບົບເທົ່ານັ້ນ ຫວັງວ່ານັກຮຽນທຸກຄົນຈະຕັ້ງໃຈເອົາໄປເຝິກຝົນຕົນເອງເລື້ອຍໆເພື່ອໃຫ້ຄວາມ ຮູ້ນີ້ສາມາດນຳໄປຕໍ່ຍອດ ແລະ ປະຍຸກໃຊ້ໃນພາກປະຕິບັດຕົວຈິງເພື່ອໃຫ້ແຕກດອກອອກຜົນ.

ຖ້າຫາກມີຄຳຖາມທີ່ກ່ຽວຂ້ອງສາມາດສອບຖາມໄດ້ຕະຫຼອດ

ຂໍຂອບໃຈ

🔥 ພາກປະຕິບັດຕົວຈິງ

ຈົ່ງວິເຄາະ ແລະ ສ້າງລະບົບສົ່ງເສີມການຂາຍໂດຍໃຊ້ Nodejs ເພື່ອຈັດການຝັ່ງ Backend ແລະ React Type Script ເພື່ອຈັດການຝັ່ງ Frontend ຮ່ວມກັບ Framework ທີ່ກ່ຽວຂ້ອງ ແລະ ຈຳເປັນ ແລະ ໃຫ້ແຍກລະບົບອອກເປັນ 2 Modules ໃນໂປເຈັກດຽວກັນເຊັ່ນ:

📌 Module 1: ຈັດການແອັດມິນ (Admin System)

- ສ້າງໜ້າ Authentication (Login) ໂດຍໃຫ້ເຮັດການ Authorisation ເພື່ອແຍກສິດໃນການຈັດການຂໍ້ມູນ
- ສ້າງໜ້າຈັດການຂໍ້ມູນຜູ້ໃຊ້ລະບົບ (CRUD)ໂດຍແບ່ງສິດອອກເປັນ 2 ສິດເຊັ່ນ:

  > User: ສາມາດອ່ານ, ຄົ້ນຫາ ແລະ ສ້າງໄດ້ເທົ່ານັ້ນ

  > Admin: ສາມາດຈັດການໄດ້ທຸກຮູບແບບ (ເພີ່ມ, ລຶບ, ແກ້ໄຂ, ອ່ານ ແລະ ຄົ້ນຫາ)

- ສ້າງໜ້າຕັ້ງຄ່າ Profile ຂອງຕົນເອງ ແລະ ອາດຈະໃຫ້ມີສ່ວນແກ້ໄຂລະຫັດຜ່ານນຳກໍ່ໄດ້
- ໜ້າໜ້າຈັດການ CRUD ຂໍ້ມູນສິນຄ້າ ອັດຈະຕິດຄັດ Link ຫຼື ບໍ່ມີກໍ່ໄດ້ ເພື່ອໃຫ້ສາມາດເຂົ້າໄປສຶກສາເພີ່ມຕື່ມກ່ຽວກັບສິນຄ້າດັ່ງກ່າວ.
- ຈັດການແບ່ງໜ້າສະແດງລາຍສິນຄ້າໄດ້ເທື່ອລະ 10, 15, 25, 50 ແລະ 100 ລາຍການ
- ສາມາດຄົ້ນຫາສິນຄ້າເປັນ ຊື່, ອ.ດ.ປີ ເພີ່ມສິນຄ້າລົງລະບົບ

📌 Module 2: ນຳສະເໜີສິນຄ້າ (Presentation Pages)

- ນຳສິນຄ້າທີ່ແອັດມິນໄດ້ເພີ່ມໄວ້ໃນລະບົບມາສະແດງພ້ອມຂໍ້ມູນທີ່ຈຳເປັນ
- ສາມາດກົດສັ່ງຊື້ຜ່ານ WhatsApp ໂດຍຈຳເປັນຕ້ອງເກັບຂໍ້ມູນການສັ່ງຊື້ໄວ້ໃນລະບົບ