สร้างโปรเจกต์ Quasar + Express Backend ด้วย Git, Docker และ Docker Compose (ตัวอย่างขั้นสูง)

ตัวอย่างนี้ต่อยอดจากตัวอย่างก่อนหน้า โดยเพิ่ม **backend (Express API)** เพื่อความซับซ้อน ทำให้โปรเจกต์เป็น full-stack app จริง ๆ. เราจะ:

- สร้าง backend Express ที่มี API endpoint ง่าย ๆ (เช่น /api/demo ที่ return ข้อมูล Git/Docker).
- เชื่อม frontend (Quasar) กับ backend ผ่าน HTTP request (ใช้ Axios ใน Quasar).
- ใช้ Git สำหรับทั้ง frontend และ backend (รวมใน repo เดียว).
- เขียน Dockerfile สำหรับ backend.
- ใช้ Docker Compose เพื่อรันทั้ง frontend และ backend พร้อม network สำหรับการสื่อสารภายใน.
- เพิ่ม environment variables (.env) สำหรับ config เช่น API URL.
- เพิ่ม volume สำหรับ backend (สมมติเก็บ log ถาวร).

โครงสร้างโปรเจกต์หลัก (หลังทำเสร็จ):

```
text
my-quasar-express-app/
  .gitignore
  docker-compose.yml
  frontend/          ← Quasar SPA (จากตัวอย่างก่อน)
    Dockerfile
    .dockerignore
    package.json
    src/
      ... (รวม IndexPage.vue ที่อัปเดตเพื่อเรียก API)
  backend/           ← Express API (ใหม่)
    Dockerfile
    .dockerignore
    package.json
    server.js        ← Main file
    logs/            ← สำหรับ volume demo
  docs/              ← เอกสาร (เช่น README.md)
```

เราจะทำให้ปฏิบัติได้จริง โดยเพิ่ม error handling, best practices (เช่น CORS ใน backend), และขั้นตอนทดสอบ.

1. สร้าง Backend (Express API)

จาก root:

Bash

```
cd my-quasar-express-app

# สร้างโฟลเดอร์ backend (ถ้ายังไม่มี)
mkdir -p backend
cd backend
```

Init โปรเจกต์ Express:

Bash

```
npm init -y
npm install express cors dotenv
```

สร้างไฟล์ backend/server.js (API ง่าย ๆ ที่ return ข้อมูล demo):

JavaScript

```
const express = require('express');
const cors = require('cors');
const fs = require('fs');
const path = require('path');
require('dotenv').config();

const app = express();
const PORT = process.env.PORT || 3000;

// Middleware
app.use(cors());  // อนุญาต cross-origin จาก frontend
app.use(express.json());

// สร้างโฟลเดอร์ logs ถ้ายังไม่มี (สำหรับ volume demo)
const logsDir = path.join(__dirname, 'logs');
if (!fs.existsSync(logsDir)) {
```

```
  fs.mkdirSync(logsDir);
}

// Endpoint demo: Return Git + Docker info และ log request
app.get('/api/demo', (req, res) => {
  const logMessage = `Request at ${new Date().toISOString()}: ${req.ip}\n`;
  fs.appendFileSync(path.join(logsDir, 'access.log'), logMessage);

  res.json({
    git: {
      title: 'Advanced Git Workflow',
      detail: 'ใช้ branch protection บน GitHub, code review ใน PR, และ squash merge เพื่อ history สะอาด'
    },
    docker: {
      title: 'Advanced Docker',
      detail: 'ใช้ multi-stage build, healthcheck ใน Dockerfile, และ orchestration ด้วย Compose/Swarm'
    }
  });
});

// Error handling
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Something broke!');
});

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

ทดสอบ Backend ใน Dev Mode

Bash

```
node server.js
```

เปิด browser หรือใช้ curl: curl http://localhost:3000/api/demo ⟶ เห็น JSON response.

2. อัปเดต Frontend (Quasar) เพื่อเชื่อมกับ Backend

กลับไปที่ frontend/src/pages/IndexPage.vue (ต่อยอดจากตัวอย่างก่อน) โดยเพิ่มการเรียก API จาก backend ด้วย Axios.

ก่อนอื่น ติดตั้ง Axios ใน frontend:

Bash

```
cd ../frontend
npm install axios
```

อัปเดต src/pages/IndexPage.vue (เพิ่ม section สำหรับ API data):

vue

```
<template>
  <q-page padding>
    <div class="text-h4 q-mb-md">
      Advanced Full-Stack Demo (Quasar + Express)
    </div>

    <!-- Git Workflow (จากตัวอย่างก่อน) -->
    <q-card class="q-mb-md">
      <q-card-section>
        <div class="text-h6">Git Workflow</div>
        <q-list bordered separator class="q-mt-sm">
          <q-item v-for="(step, index) in gitSteps" :key="index">
            <q-item-section avatar>
              <q-badge>{{ index + 1 }}</q-badge>
            </q-item-section>
            <q-item-section>
              <q-item-label>{{ step.title }}</q-item-label>
              <q-item-label caption>{{ step.detail }}</q-item-label>
            </q-item-section>
```

```
      </q-item>
    </q-list>
  </q-card-section>
</q-card>

<!-- Docker Concepts (จากตัวอย่างก่อน) -->
<q-card class="q-mb-md">
  <q-card-section>
    <div class="text-h6">Docker Concepts</div>
    <q-list bordered separator class="q-mt-sm">
      <q-item v-for="(item, index) in dockerItems" :key="index">
        <q-item-section>
          <q-item-label>{{ item.title }}</q-item-label>
          <q-item-label caption>{{ item.detail }}</q-item-label>
        </q-item-section>
      </q-item>
    </q-list>
  </q-card-section>
</q-card>

<!-- New: API Data from Backend -->
<q-card>
  <q-card-section>
    <div class="text-h6">Data from Backend API</div>
    <q-spinner v-if="loading" color="primary" size="2em" />
    <q-list v-else bordered separator class="q-mt-sm">
      <q-item>
        <q-item-section>
          <q-item-label>Advanced Git</q-item-label>
          <q-item-label caption>{{ apiData.git.detail }}</q-item-label>
        </q-item-section>
      </q-item>
      <q-item>
        <q-item-section>
```

```
            <q-item-label>Advanced Docker</q-item-label>
            <q-item-label caption>{{ apiData.docker.detail }}</q-item-label>
          </q-item-section>
        </q-item>
      </q-list>
      <q-btn v-if="!loading" color="primary" @click="fetchData">Refresh Data</q-btn>
    </q-card-section>
  </q-card>
 </q-page>
</template>

<script setup>
import { ref, onMounted } from 'vue';
import axios from 'axios';

// จากตัวอย่างก่อน
const gitSteps = [ /* ... (same as before) */ ];
const dockerItems = [ /* ... (same as before) */ ];

const apiData = ref({ git: {}, docker: {} });
const loading = ref(true);

const fetchData = async () => {
  loading.value = true;
  try {
    const response = await axios.get(import.meta.env.VITE_API_URL + '/api/demo');
    apiData.value = response.data;
  } catch (error) {
    console.error('API Error:', error);
  } finally {
    loading.value = false;
  }
};
```

```
onMounted(fetchData);
</script>
```

เพิ่ม Environment Variables สำหรับ Frontend

สร้าง frontend/.env (จะ ignore ใน Git):

text

```
VITE_API_URL=http://localhost:3000
```

ใน production (Docker) จะ override เป็น URL ภายใน network เช่น http://backend:3000.

ทดสอบ: npm run dev ⟶ เห็นข้อมูลจาก backend (รัน backend พร้อมกัน).

3. ใช้ Git สำหรับทั้ง Frontend และ Backend

จาก root (สมมติ repo มีอยู่แล้วจากตัวอย่างก่อน):

Bash

```
git checkout main
git pull origin main

# สร้าง branch สำหรับ feature ใหม่ (backend + integration)
git checkout -b feature/add-express-backend-integration

# เพิ่ม/แก้ไฟล์ backend ทั้งหมด และ frontend ที่อัปเดต
git add backend/ frontend/src/pages/IndexPage.vue frontend/.env  # แต่ .env จะ ignore

git commit -m "feat: add express backend api and integrate with quasar frontend"

git push origin feature/add-express-backend-integration
```

เปิด PR บน GitHub ⟶ Review (เช็ค integration) ⟶ Merge ⟶ ลบ branch.

อัปเดต .gitignore ที่ root ถ้ายังไม่มีส่วน backend:

เพิ่ม:

text

```
# Backend
backend/logs/
```

4. เขียน Dockerfile สำหรับ Backend

สร้าง backend/Dockerfile (multi-stage สำหรับ production):

dockerfile

```
# Stage 1: Build
FROM node:20-alpine AS builder

WORKDIR /app

COPY package*.json ./
RUN npm ci

COPY . .

# Stage 2: Production
FROM node:20-alpine

WORKDIR /app

COPY --from=builder /app /app

# สร้าง logs dir
RUN mkdir -p logs

# Healthcheck (เพิ่มความซับซ้อน: เช็คว่า server พร้อม)
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 \
  CMD wget --quiet --tries=1 --spider http://localhost:3000/api/demo || exit 1
```

```
EXPOSE 3000

CMD ["node", "server.js"]
```

สร้าง backend/.dockerignore:

```
text
node_modules
logs
.git
.gitignore
npm-debug.log*
.env
```

5. Build และ Run ด้วย Docker (แยก)

จาก root:

```
Bash
# Build backend
docker build -t my-express-backend:latest -f backend/Dockerfile ./backend
docker build -t my-express-backend:latest .

# Run backend (กับ volume สำหรับ logs)
docker run -d \
  -p 3000:3000 \
  --name my-express-backend-container \
  -v $(pwd)/backend/logs:/app/logs \
  my-express-backend:latest

# Build frontend (จากตัวอย่างก่อน)
docker build -t my-quasar-frontend:latest -f frontend/Dockerfile ./frontend
docker build -t my-quasar-frontend:latest .
# Run frontend (override env สำหรับ API URL)
```

```
docker run -d \
  -p 8080:80 \
  --name my-quasar-frontend-container \
  -e VITE_API_URL=http://host.docker.internal:3000 \  # หรือใช้ IP จริง
  my-quasar-frontend:latest
```

เข้า http://localhost:8080 ⟶ Frontend เรียก API จาก backend ได้ (แต่ใน Docker แยก อาจต้องแก้ network).

6. Docker Compose สำหรับ Full-Stack (หลาย Services)

สร้าง/อัปเดต docker-compose.yml ที่ root (เพิ่ม backend + network):

YAML

```
version: '3.9'

services:
  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    ports:
      - "8080:80"
    environment:
      - VITE_API_URL=http://backend:3000  # ใช้ชื่อ service เป็น URL ภายใน
    networks:
      - app-network
    restart: unless-stopped

  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    ports:
```

```
    - "3000:3000"
  environment:
    - PORT=3000
  volumes:
    - ./backend/logs:/app/logs  # Persist logs
  networks:
    - app-network
  restart: unless-stopped
  healthcheck:  # จาก Dockerfile
    test: ["CMD", "wget", "--quiet", "--tries=1", "--spider", "http://localhost:3000/api/demo"]
    interval: 30s
    timeout: 10s
    retries: 3

networks:
  app-network:  # Network สำหรับสื่อสารระหว่าง services
```

Run:

Bash

```
docker compose up --build -d  # Background
```

- Frontend ที่ http://localhost:8080 จะเรียก API จาก backend ภายใน network.
- เช็ค logs: docker compose logs backend.
- หยุด: docker compose down -v (ลบ volumes ถ้าต้องการ).

7. สรุปและโยงกับวัตถุประสงค์บทเรียน

ตัวอย่างนี้เพิ่มความซับซ้อนด้วย full-stack integration:

1. **Version Control**: ขยาย workflow ให้ครอบคลุมหลายส่วน (frontend + backend) ใน PR เดียว, ใช้ Conventional Commits สำหรับ features ใหญ่.
2. **.gitignore / .dockerignore**: ปรับให้เหมาะกับทั้งสองส่วน, persist logs แต่ ignore ใน Git.
3. **Dockerfile**: Multi-stage สำหรับทั้งคู่, เพิ่ม healthcheck ใน backend สำหรับ reliability.

4. **Docker Usage**: Compose สำหรับ orchestration, networks สำหรับ communication, volumes สำหรับ data persistence, env vars สำหรับ config.
5. **Integration**: Frontend เรียก backend จริง, error handling, CORS.