

โตโต้บิน (TOTO BIN)

Embedded System for Smart Beverage Mixing Vending Machine

จัดทำโดย

6610110509 นายธรรมีนา เพ็งชัย กลุ่มที่ 01

6610110542 เนติวุฒิ เกตุกำพล กลุ่มที่ 01

6610110554 นายปรเมษ แก้วอุบล กลุ่มที่ 01

เสนอ

รศ.ดร. ทวีศักดิ์ เรืองพีระกุล

รศ.ดร. ปัญญยศ ไชยกาฬ

ผศ.ดร. วชรินทร์ แก้วอภิชัย

รายงานเล่มนี้เป็นส่วนหนึ่งของรายวิชา 240-319

EMBEDDED SYSTEM DEVELOPER MODULE

ภาคเรียนที่ 1 ปีการศึกษา 2568

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยสงขลานครินทร์

**สารบัญ**

**สารบัญก**

**ความสำคัญ1**

**โครงสร้างข้อมูล1**

ข้อมูลผู้ใช้ (User)1

ข้อมูลโจทย์ (Problem)1

ข้อมูลการแข่งขัน (Contest)1

ทำไม Tourist Code ต้องเก็บข้อมูลเยอะ3

**การออกแบบระบบและพัฒนา3**

**เทคนิคที่ใช้14**

Category I14

Concurrency14

Category II17

Docker17

Cloud Function21

Virtual Machine (VM)23

Category III27

Terraform27

Git30

Makefile34

Category IV36

Pub/Sub36

Load Balancer38

k8s41

Scheduler45

Category V48

Snapshot48

Backup50

Security52

Category VI63

Multiusers63

**เทคนิคอื่นๆที่ใช้ใน Project64**

Monitoring64

Google Cloud Monitoring64

Web Application Availability | Monitoring & Metrics ของ Cloudflare65

**Tourist Code (Online Judge)**

1. **ความสำคัญ**

"Tourist Code" เป็นเว็บไซต์ Online Judge ที่ออกแบบมาเพื่อฝึกฝนการเขียนโค้ดผ่านโจทย์เกี่ยวกับอัลกอริทึมสำหรับพัฒนาการเขียนโปรแกรม โดยชื่อ "Tourist" ซึ่งแปลว่า "นักท่องเที่ยว" ในภาษาอังกฤษ สื่อถึงแนวคิดของการสำรวจและเดินทางผ่านโลกแห่งโค้ดและอัลกอริทึมที่หลากหลาย เปรียบเหมือนการผจญภัยไปในที่ต่างๆเปรียบเหมือน การท้าทายสิ่งรอบตัว และในที่ ที่เราไม่คุ้นเคยทำให้ได้พบ การท้าทายที่แปลกใหม่ และได้รับความรู้ ความสามารถเพิ่มเติม จากการเรียนรู้ความท้าทายเหล่านั้น โดยเว็บของเราได้มีการแข่งขัน ทำให้ได้เรียนรู้โจทย์ใหม่ๆที่ไม่คุ้นเคย แล้วยังได้แข่งขันกับคนแปลกหน้าอีกด้วย

**2. โครงสร้างข้อมูล**

"Tourist Code" เป็นแพลตฟอร์ม Online Judge ที่ต้องการโครงสร้างข้อมูลที่เรียบง่ายแต่มีประสิทธิภาพเพื่อจัดการโจทย์ ผู้ใช้ และการส่งคำตอบ โครงสร้างข้อมูลหลักประกอบด้วย 3 ส่วน ดังนี้:

**2.1 ข้อมูลผู้ใช้ (User)**

เก็บข้อมูลพื้นฐานของผู้ใช้ เช่น ชื่อผู้ใช้ (username) และการส่งคำตอบ (submissions) ซึ่งบันทึกโจทย์ที่ผู้ใช้เคยแก้และสถานะของคำตอบ เช่น "Accepted" หรือ "Wrong Answer" เพื่อติดตามความก้าวหน้าของผู้ใช้

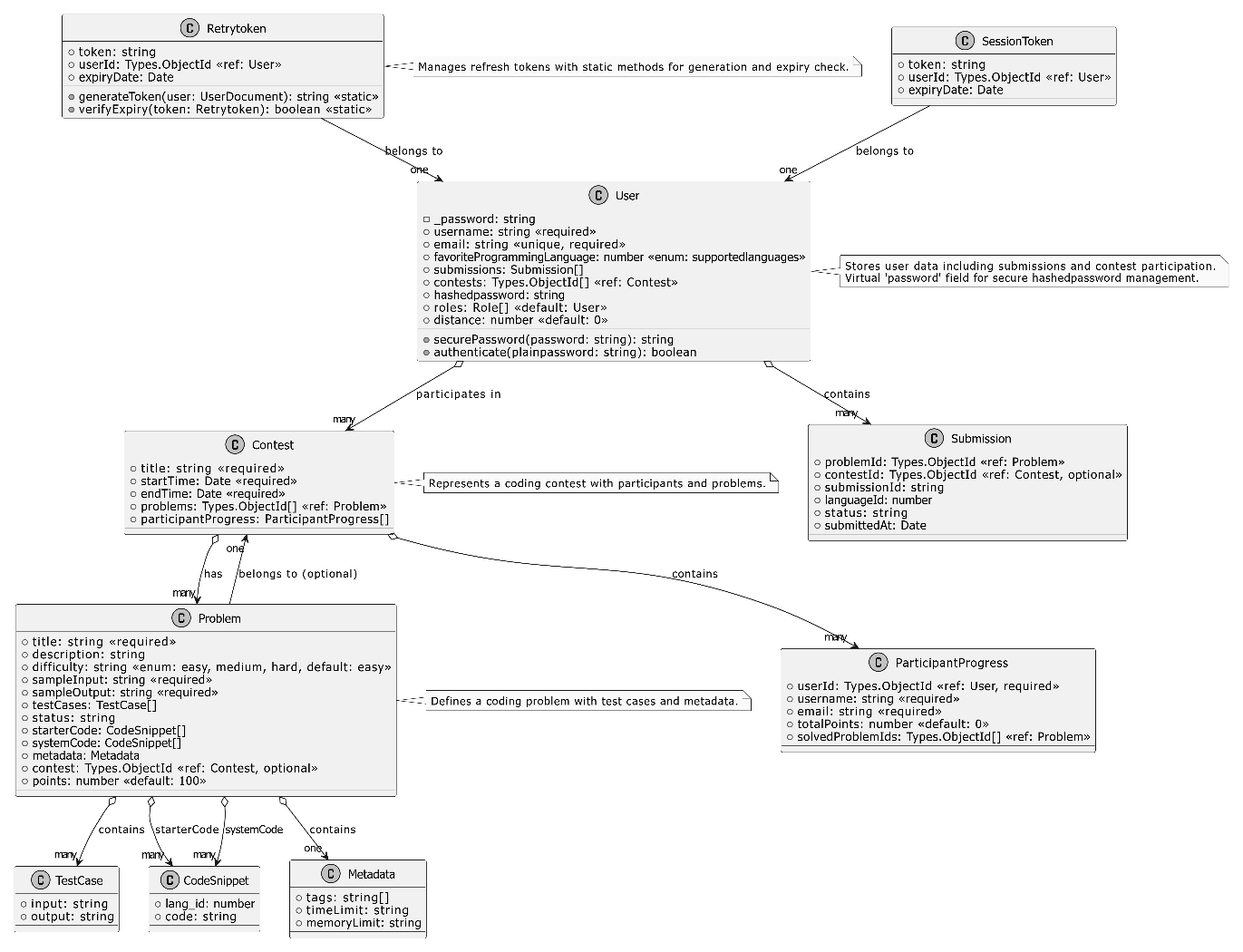
**2.2 ข้อมูลโจทย์ (Problem)**

เก็บรายละเอียดของโจทย์ เช่น ชื่อโจทย์ (title) และเคสทดสอบ (test cases) ที่ใช้ตรวจคำตอบ โดยโจทย์อาจเชื่อมโยงกับการแข่งขัน (contest) เพื่อจัดกลุ่มโจทย์

**2.3 ข้อมูลการแข่งขัน (Contest)**

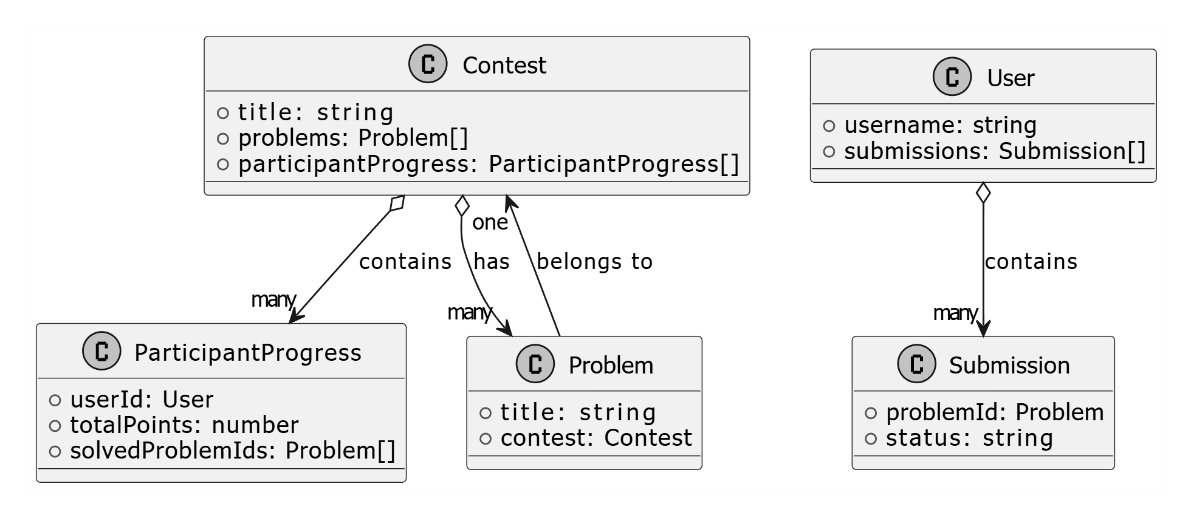
เก็บข้อมูลการแข่งขัน เช่น ชื่อ (title) และรายการโจทย์ (problems) รวมถึงความคืบหน้าของผู้เข้าร่วม (participant progress) เช่น คะแนนรวมและโจทย์ที่แก้ได้ เพื่อใช้ในการจัดอันดับ

โครงสร้างข้อมูลเหล่านี้ถูกออกแบบให้เชื่อมโยงกันผ่านรหัส (ID) เช่น รหัสผู้ใช้ (userId) และรหัสโจทย์ (problemId) เพื่อให้ระบบสามารถจัดการและตรวจคำตอบได้อย่างรวดเร็ว โดยอาจใช้ฐานข้อมูล MongoDB เพื่อความยืดหยุ่นในการจัดการข้อมูล



รูป 2.1 โครงสร้างข้อมูล

**คำอธิบาย:** แสดงโครงสร้างข้อมูลหลัก ซึ่งประกอบด้วย Contest, Problem, User และ Submission ที่เชื่อมโยงกัน สามารถดูรูปเต็มได้ที่ <https://shorturl.at/gYpAM>



รูป 2.2 โครงสร้างข้อมูลโดยย่อ

**2.4 ทำไม Tourist Code ต้องเก็บข้อมูลเยอะ**

เราต้องเก็บข้อมูลหลายอย่าง เช่น ข้อมูลผู้ใช้ โจทย์ การแข่งขัน และการส่งคำตอบ เพราะมันช่วยให้ระบบทำงานได้ดีและปลอดภัยขึ้น การเก็บข้อมูลผู้ใช้ เช่น ชื่อและอีเมล ช่วยให้รู้ว่าใครเป็นใคร ป้องกันคนแปลกหน้าเข้ามายุ่งกับระบบได้ง่าย ๆ ส่วนข้อมูลโจทย์และการส่งคำตอบก็ต้องเก็บเยอะหน่อย เพื่อให้แน่ใจว่าระบบตรวจคำตอบได้ถูกต้องและไม่มีการโกง เช่น ถ้าไม่เก็บประวัติการส่ง เราจะไม่รู้ว่าใครแก้โจทย์ไปแล้วบ้าง การมีข้อมูลเยอะแบบนี้เลยเหมือนเป็นเกราะป้องกัน ให้ทุกอย่างโปร่งใสและปลอดภัยสำหรับทุกคนที่ใช้งาน

**3. การออกแบบระบบและพัฒนา**

Tourist Code ถูกพัฒนาขึ้นโดยทั้งฝั่ง Client และ Server เพื่อให้ได้ประสบการณ์การใช้งานที่ลื่นไหล มีประสิทธิภาพโดยมีการแบ่งส่วนประกอบหลักของระบบดังนี้:

**3.1 ฝั่ง Client: เทคโนโลยีและการออกแบบส่วนติดต่อผู้ใช้**

***React:*** ใช้เป็นเฟรมเวิร์กหลักในการสร้างหน้าเว็บแบบไดนามิก ทำให้การอัปเดตข้อมูลบนหน้าจอเป็นไปอย่างรวดเร็วโดยไม่ต้องโหลดหน้าใหม่

***Zustand:*** จัดการสถานะ (state management) ของแอปพลิเคชันอย่างมีประสิทธิภาพ ช่วยให้การจัดการข้อมูล เช่น สถานะของผู้ใช้หรือการแข่งขัน เป็นไปอย่างง่ายดายและไม่ซับซ้อน

***TailwindCSS:*** ใช้สำหรับการออกแบบสไตล์ของหน้าเว็บ ช่วยให้สามารถปรับแต่ง UI ได้อย่างยืดหยุ่นและรวดเร็วด้วยคลาสสำเร็จรูป

***Material UI:*** เพิ่มส่วนประกอบ UI ที่สวยงามและใช้งานง่าย เช่น ปุ่ม กล่องข้อความ หรือเมนู เพื่อยกระดับประสบการณ์ผู้ใช้ให้ดูทันสมัยและเป็นมืออาชีพ

***TanStack Query:*** ใช้จัดการการดึงข้อมูลจากเซิร์ฟเวอร์และแคชข้อมูล เพื่อให้การโหลดข้อมูลโจทย์หรือผลการแข่งขันรวดเร็วและลดการเรียกซ้ำที่ไม่จำเป็น

***TanStack Table:*** ช่วยในการแสดงข้อมูลในรูปแบบตาราง เช่น อันดับการแข่งขัน (leaderboard) หรือประวัติการส่งคำตอบ ด้วยฟีเจอร์ที่ปรับแต่งได้ เช่น การเรียงลำดับหรือการกรองข้อมูล

***Monaco Editor:*** การเขียนโค้ด (code editor) ที่ใช้ใน Visual Studio Code ถูกนำมาใช้ใน Tourist Code เพื่อให้ผู้ใช้สามารถเขียนและแก้ไขโค้ดได้สะดวก

**3.2 ฝั่ง Server: การจัดการข้อมูล**

เซิร์ฟเวอร์ ถูกออกแบบมาเพื่อรองรับการประมวลผลที่รวดเร็วและปลอดภัย โดยใช้เทคโนโลยีดังนี้:

***NestJS:*** เฟรมเวิร์กฝั่งเซิร์ฟเวอร์ที่เขียนด้วย JavaScript/TypeScript ช่วยให้โค้ดมีโครงสร้างชัดเจนและง่ายต่อการบำรุงรักษา ใช้ในการจัดการ API เช่น การส่งโจทย์ ตรวจคำตอบ หรืออัปเดตคะแนนการแข่งขัน

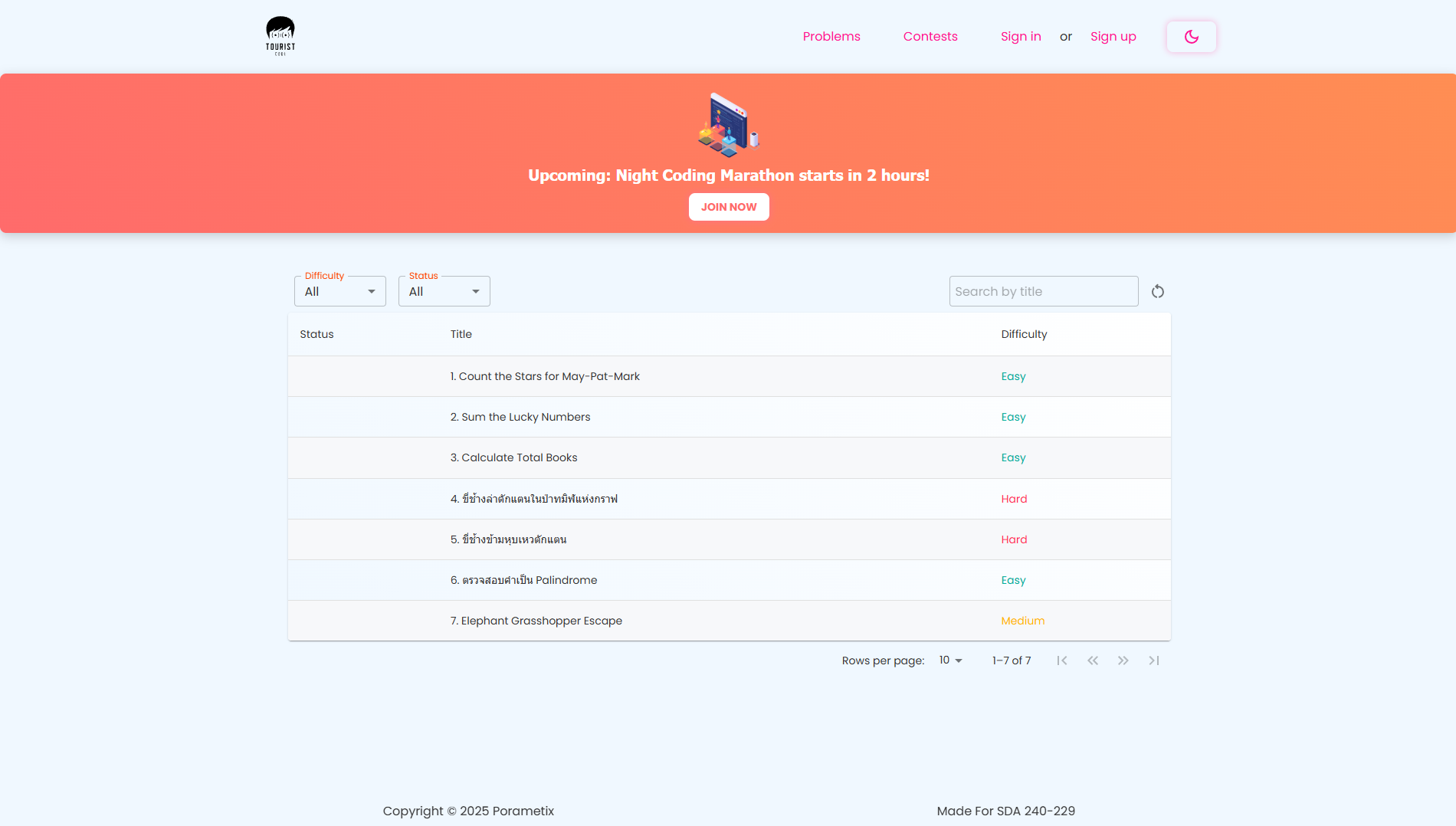
***MongoDB:*** ฐานข้อมูลแบบ NoSQL ที่เลือกใช้เพื่อความยืดหยุ่นในการเก็บข้อมูลที่หลากหลาย เช่น ข้อมูลผู้ใช้ โจทย์ และผลการส่งคำตอบ ซึ่งสามารถขยายขนาดได้ตามจำนวนผู้ใช้ที่เพิ่มขึ้น

***TypeScript:*** ใช้ในทั้งฝั่ง Client และ Server เพื่อเพิ่มความปลอดภัยของโค้ด ลดข้อผิดพลาดจากประเภทข้อมูล (type errors) และช่วยให้ทีมพัฒนาทำงานร่วมกันได้ง่ายขึ้น

**3.3 ภาพรวมการออกแบบเว็บไซต์**

ภาพรวมของการออกแบบหน้าเว็บ เช่น หน้าแสดงโจทย์ปัญหาทั้งหมด, หน้าโจทย์ที่มี Monaco Editor และตารางอันดับจากการแข่งขัน

เปลี่ยน Theme ของเว็บไซต์



แสดงโจทย์ปัญหาทั้งหมด

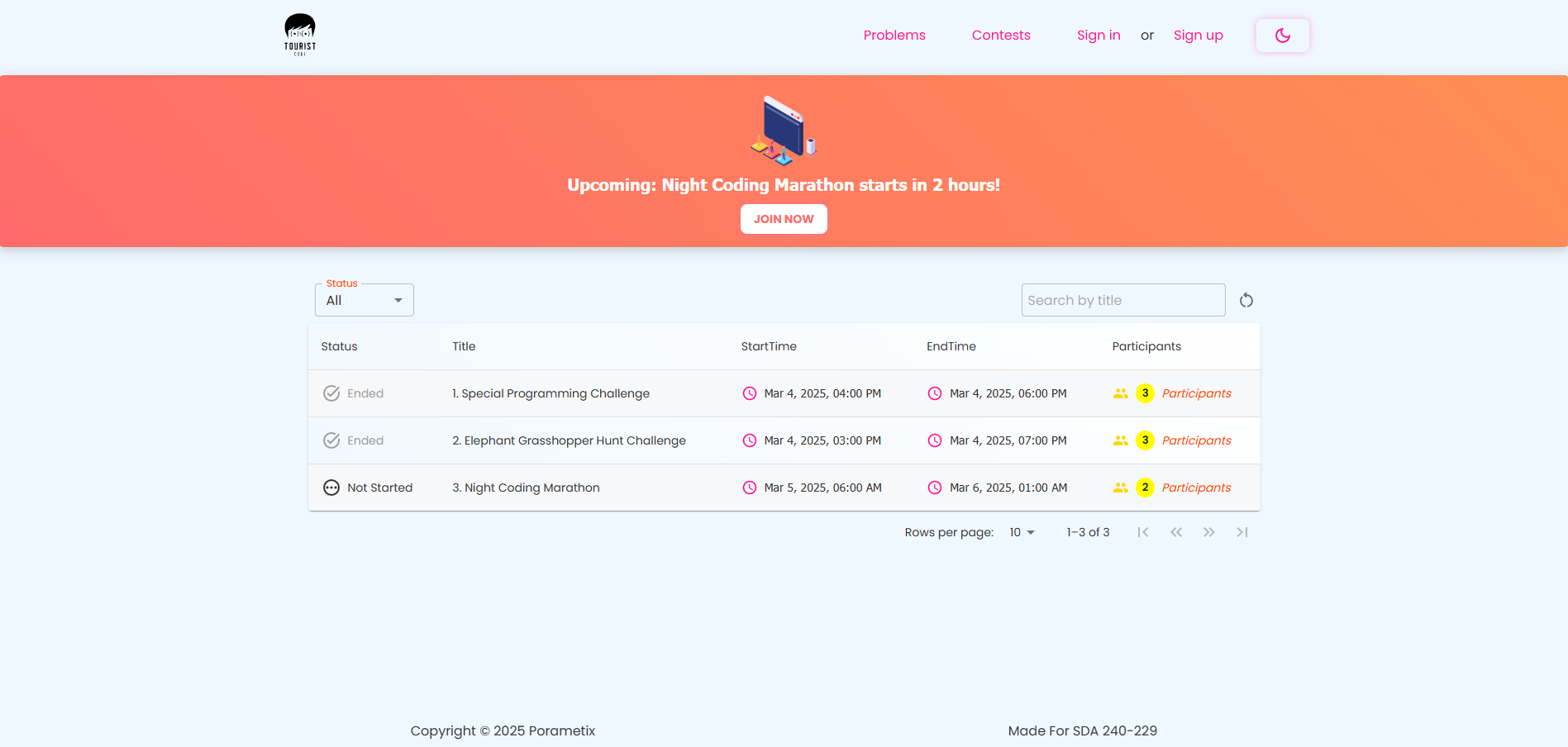
Banner Alert เตือนว่าตอนนี้มีการแข่งขันหรืออนาคต

กรองข้อมูล(ความยากและสถานะ)

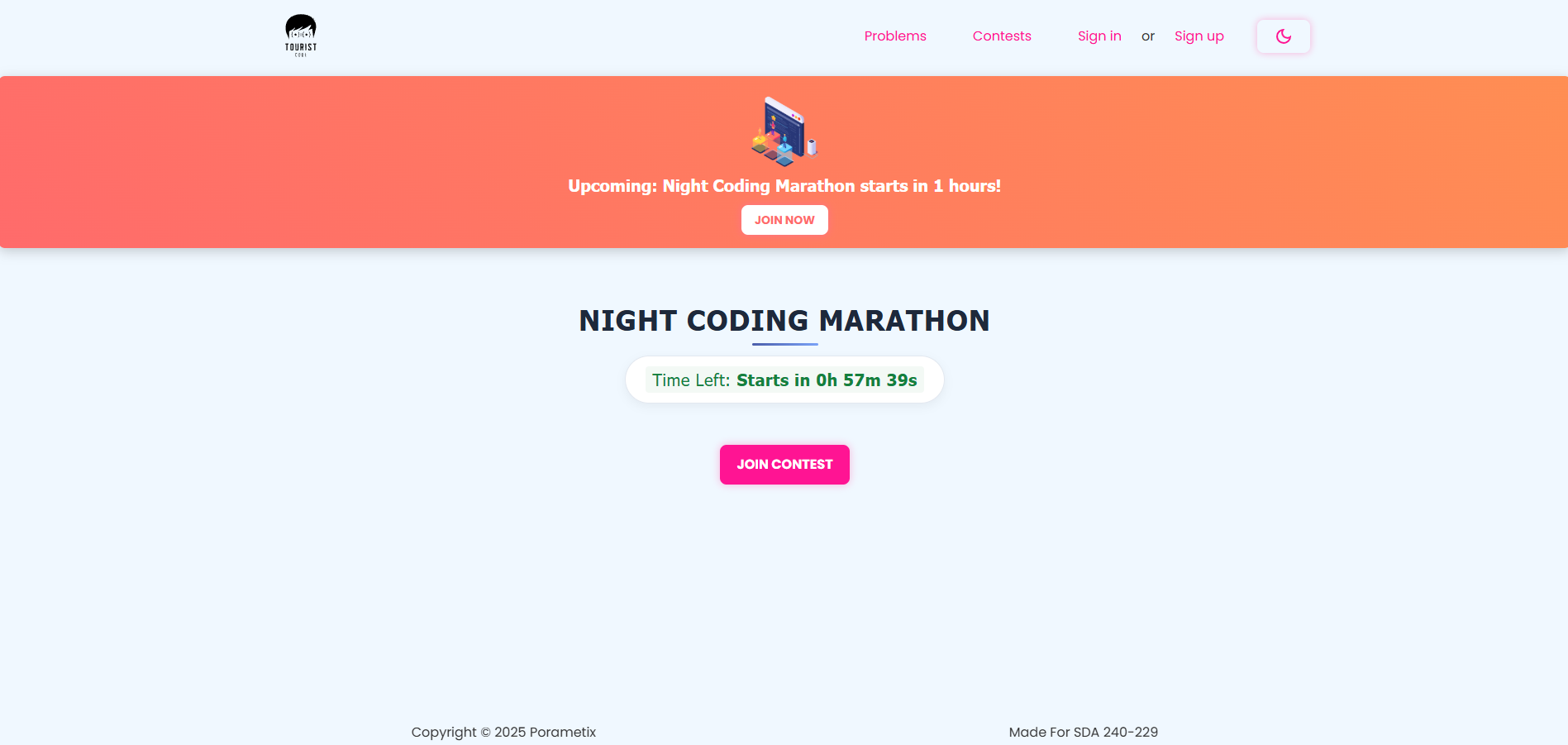
ค้นหาโดยใช้ชื่อ

รูป 3.3.1 แสดงหน้ารวมโจทย์ทั้งหมด

อธิบายเพิ่มเติม: โดยหน้านี้เป็นหน้าที่เมื่อเปิดมาครั้งแรกยังเว็บไซต์

****

รูป 3.3.2 แสดงการแข่งขันทั้งหมด

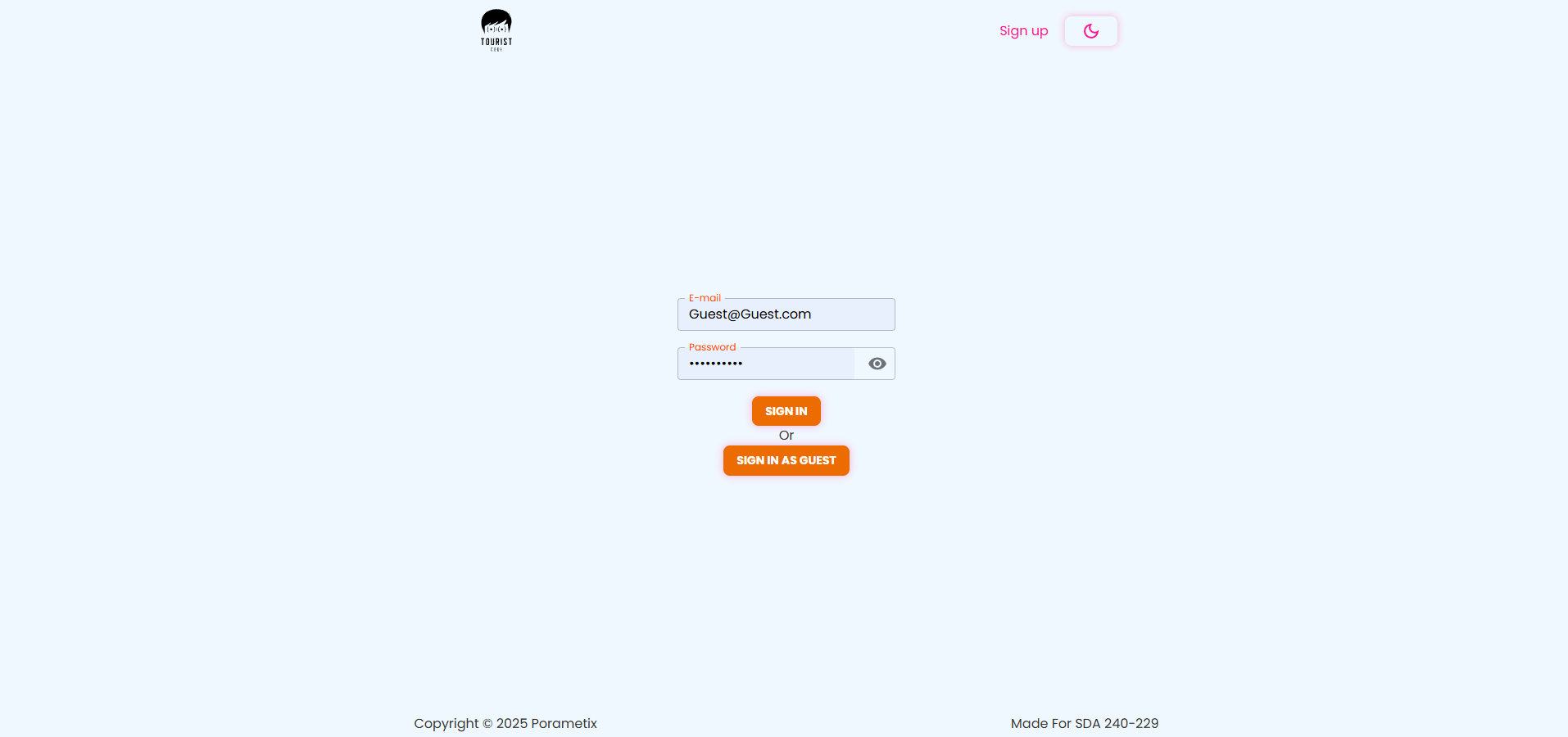


รูป 3.3.3 แสดงหน้าเข้าร่วมการแข่งขัน

อธิบายเพิ่มเติม: ในส่วนของหน้านี้จะแสดงหลังจากเลือกการแข่งเข้ามา แล้วการแข่งขันนั้นๆ   
ยังไม่เริ่มต้นการแข่งขัน

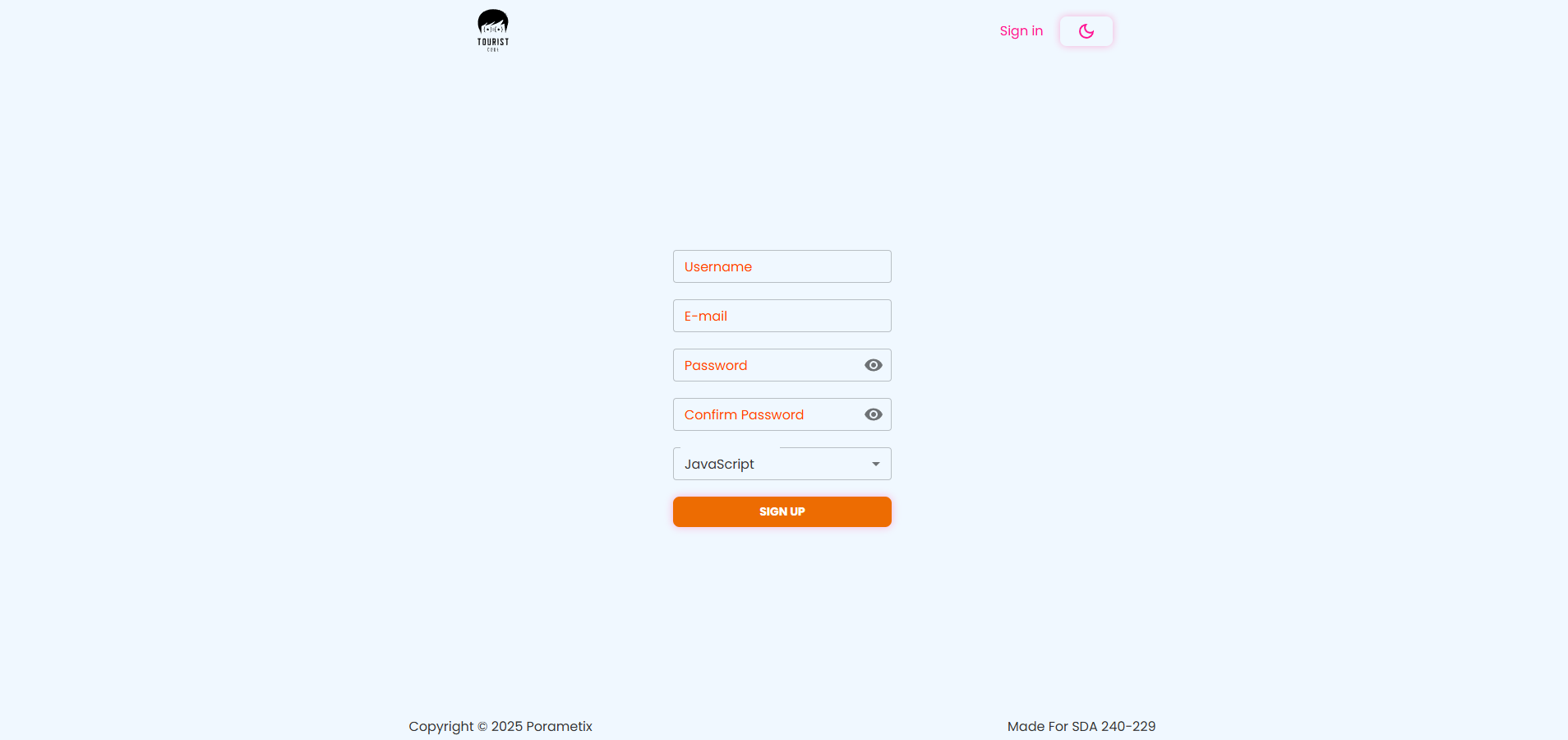


รูป 3.3.4 แจ้งเตือนเมื่อไม่ได้เข้าสู่ระบบ หลังจากคลิก “Join Contest”



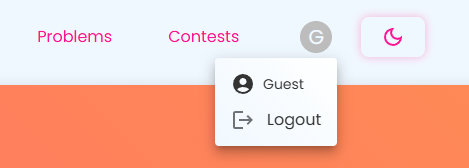
สามารถเข้าสู่ระบบโดยไม่ต้องสมัครสมาชิกก็ได้

รูป 3.3.5 แสดงหน้าเข้าสู่ระบบ

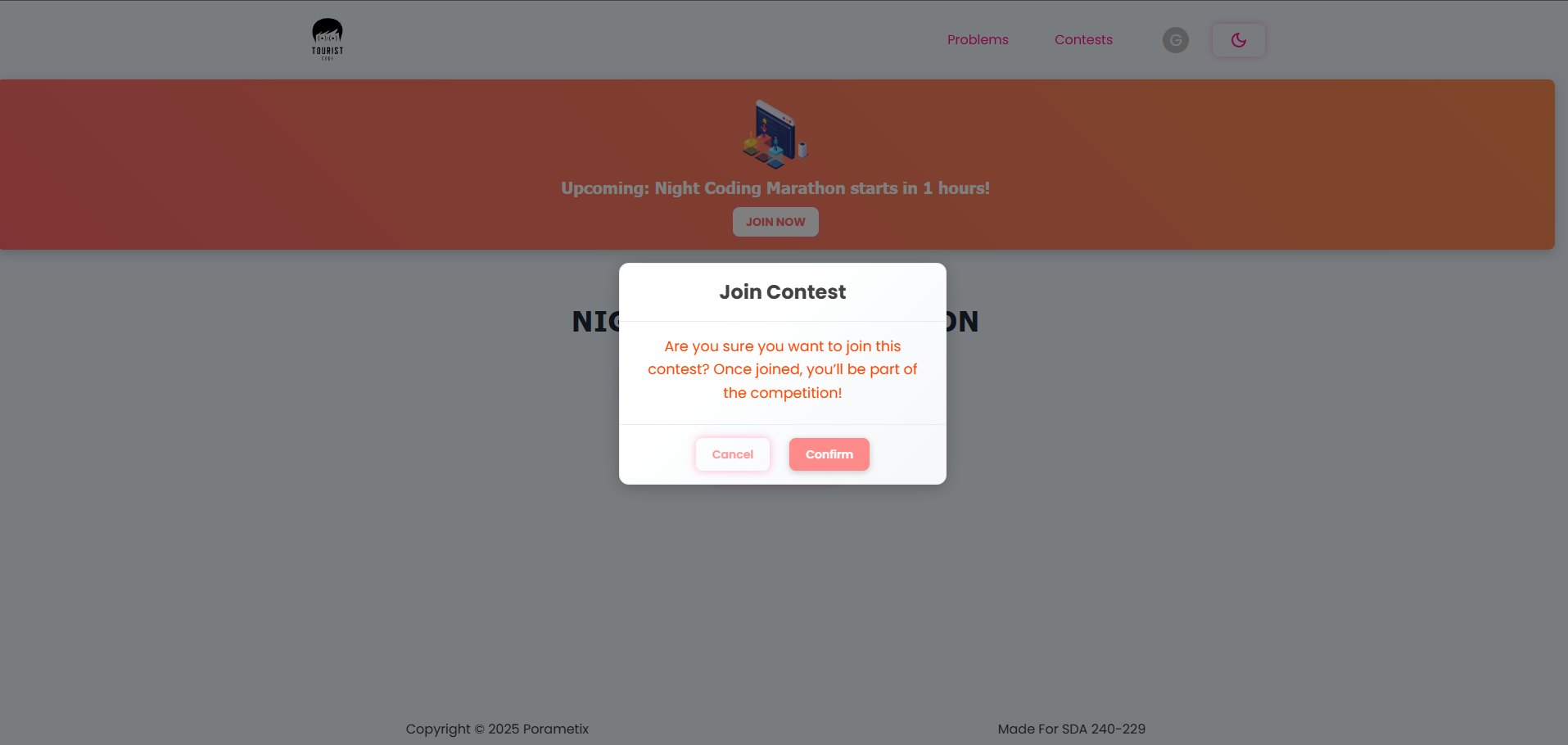


เลือกภาษาที่ชอบ

รูป 3.3.6 แสดงหน้าสมัครสมาชิก

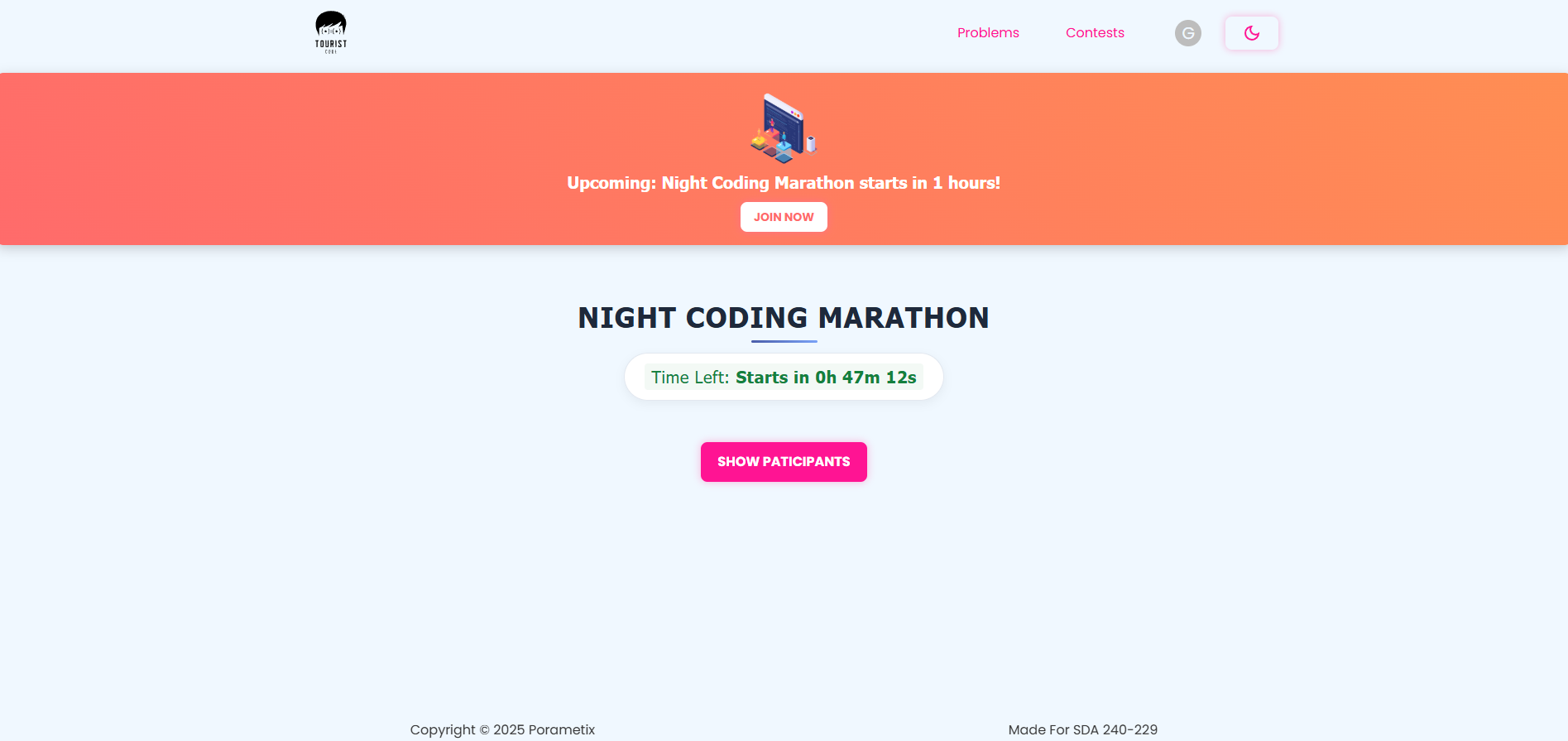


รูป 3.3.7 แสดงแถบเมนูด้านบนหลังจากเข้าสู่ระบบเสร็จสิ้น



รูป 3.3.7 แสดงเมื่อจะเข้าร่วมการแข่งขัน

อธิบายเพิ่มเติม:ในขั้นตอนนี้จะเป็นหลังจากเข้าสู่ระบบเสร็จสิ้น



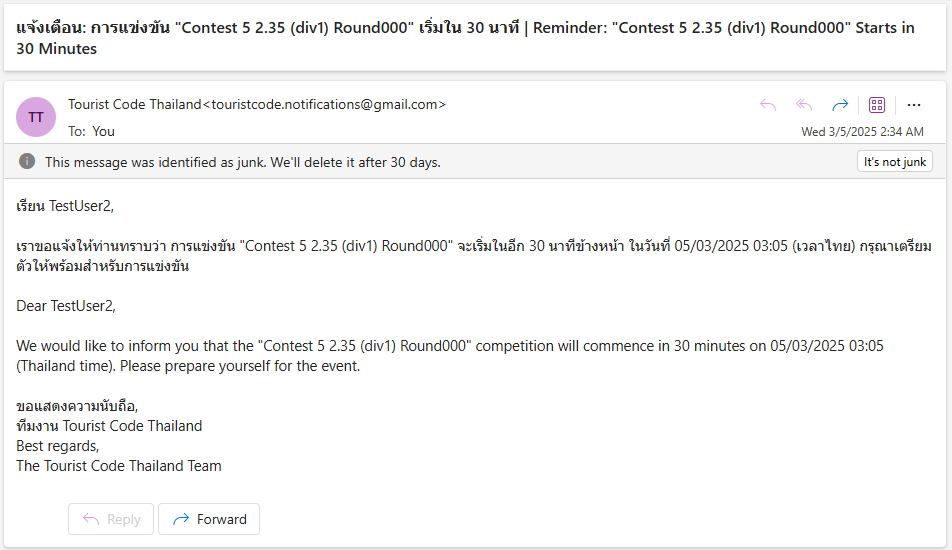
ดูรายชื่อสมาชิกที่ลงทะเบียนการแข่งขัน

รูป 3.3.8 แสดงหลังจากเข้าร่วมการแข่งขัน



รูป 3.3.9 แสดงหน้าแสดงคะแนน หลังจากคลิกเข้ามาดูรายชื่อสมาชิกที่ลงทะเบียนการแข่งขัน

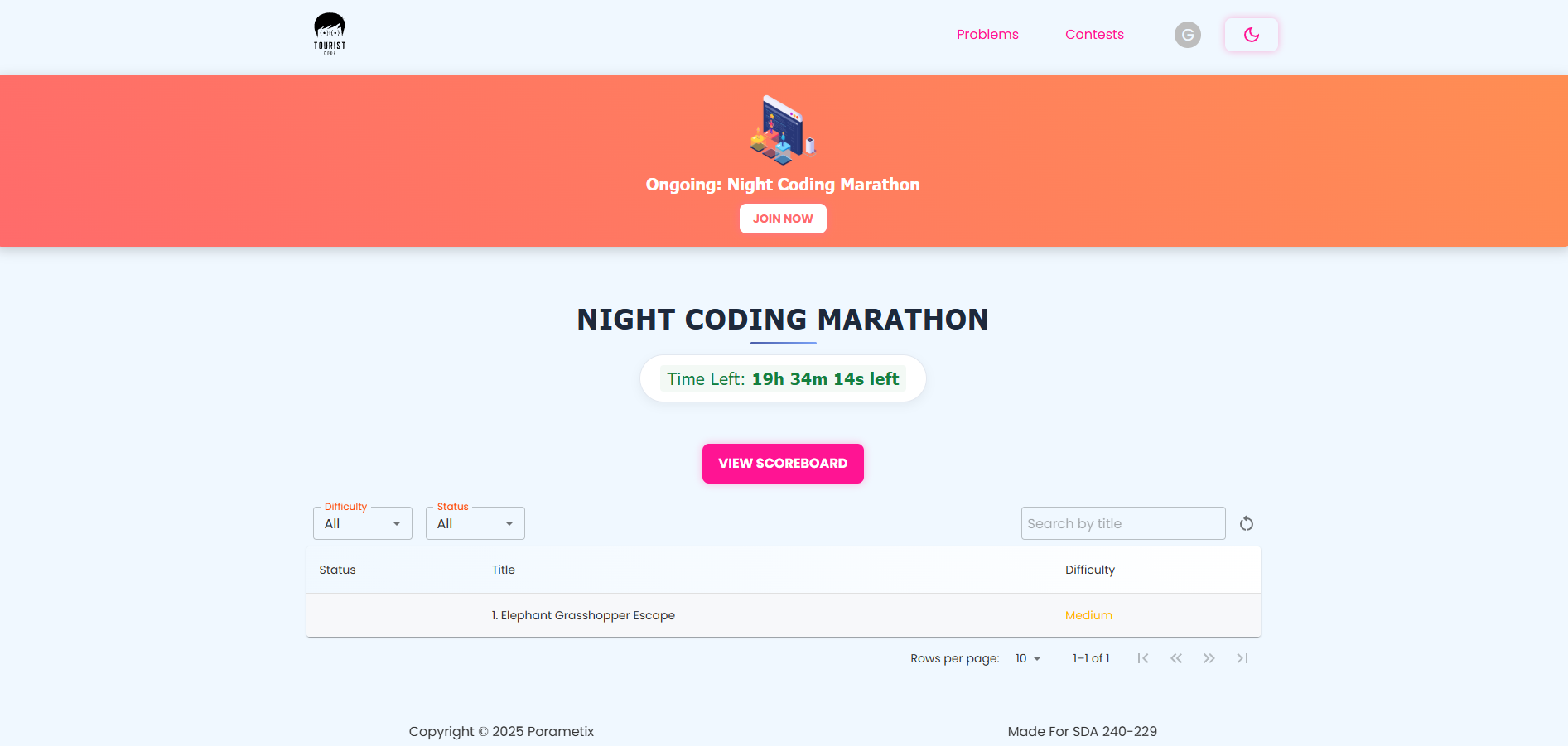
อธิบายเพิ่มเติม**:** เมื่อเข้าร่วมแล้ว คะแนน(Total Distance) ของทุกสมาชิกจะเป็น 0 เท่ากันหมด รวมถึงโจทย์ปัญหาที่แก้ไขไปแล้วด้วย แล้วเมื่อแข่งขันเสร็จสมาชิกทุกคนจะได้ Point ที่เรียกว่า Distance



รูป 3.3.10 แสดงระบบจะมีการแจ้งเตือนก่อนเริ่มต้น 30 นาทีเมื่อผู้ใช้ได้ทำการลงทะเบียนการแข่งขัน



รูป 3.3.11 แสดงเมื่อถึงเวลาการแข่งขัน



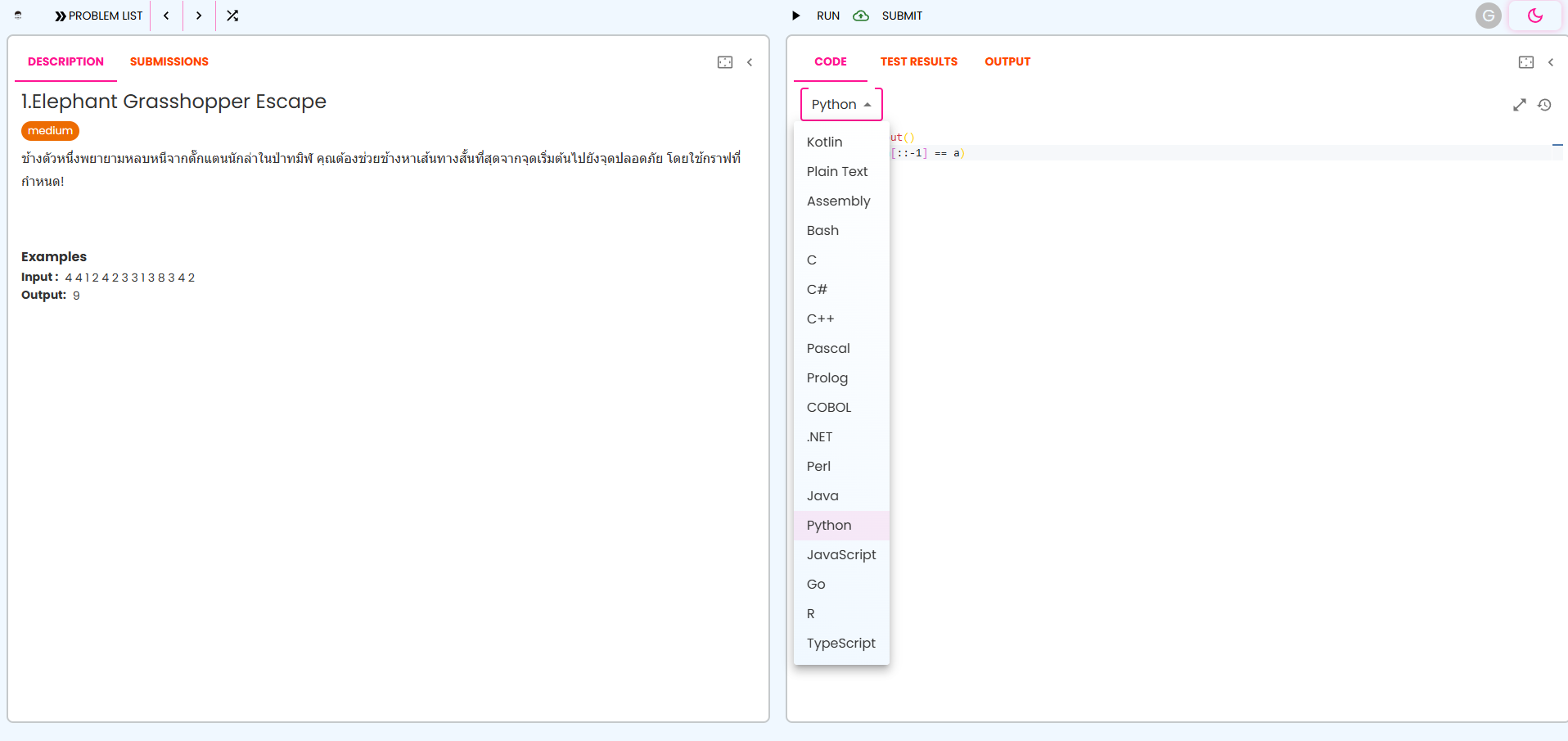
แสดงระยะเวลาที่เหลือก่อนที่การแข่งขันจะจบ

รูป 3.3.12 แสดงเมื่อถึงเวลาการแข่งขัน จะแสดงโจทย์ขึ้นมา

อธิบายเพิ่มเติม: เมื่อแข่งขันเสร็จสิ้นปุ่ม Join จะถูก Disabled แล้วจะแสดงเป็น Scoreboard ของ  
แต่ละ Contest

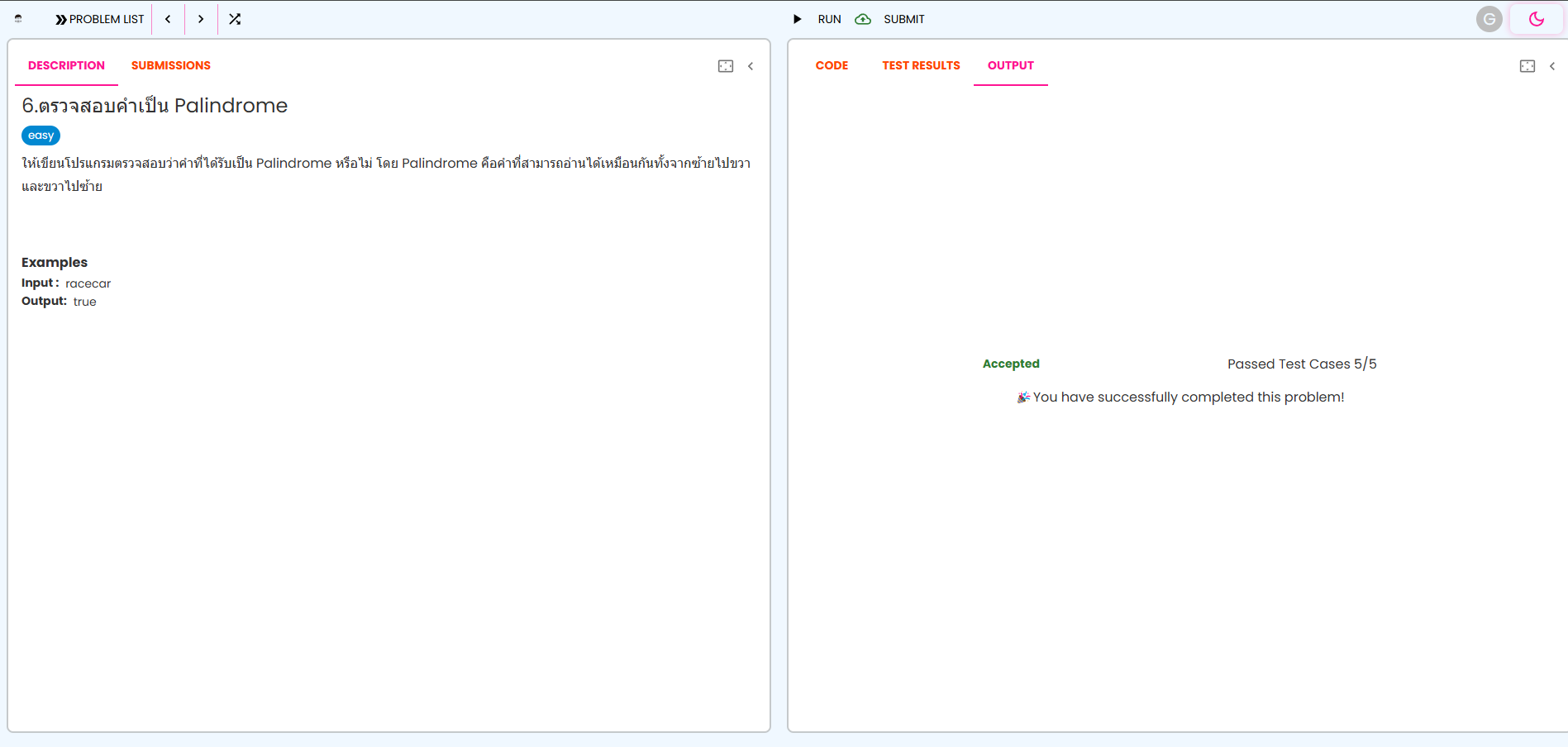
“RUN” ตรวจผลลัพธ์

“SUBMIT” ส่ง Code ไปตรวจ



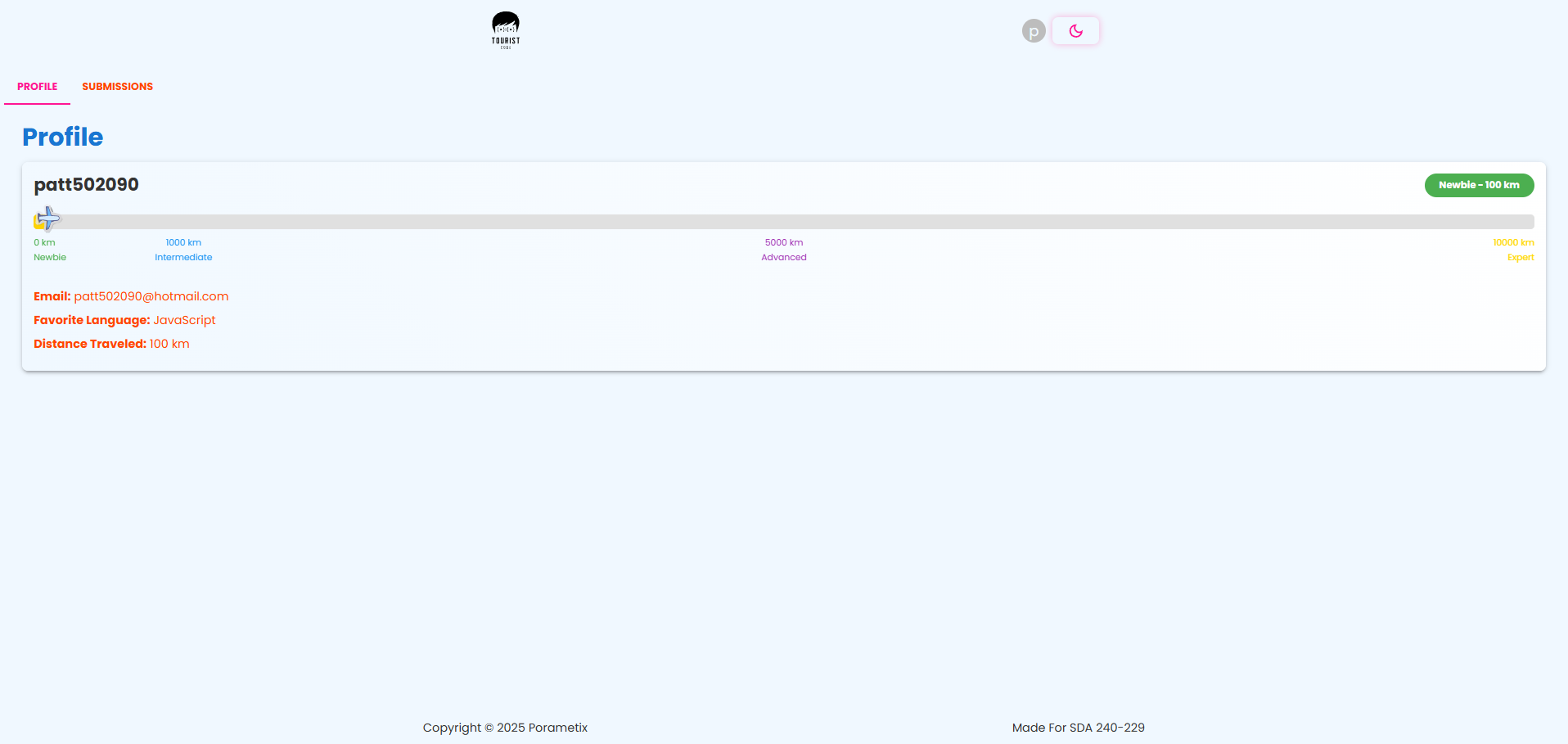
มีภาษาให้เลือกมากมาย

รูป 3.3.13 แสดงตัว Code Editor ที่จะทำการแก้ไขปัญหาจากโจทย์



แสดงสภานะ ผ่านกี่ Testcase

รูป 3.3.14 แสดงตัว Code Editor เมื่อกด “Submit” แล้วผ่านครบทุก Testcase

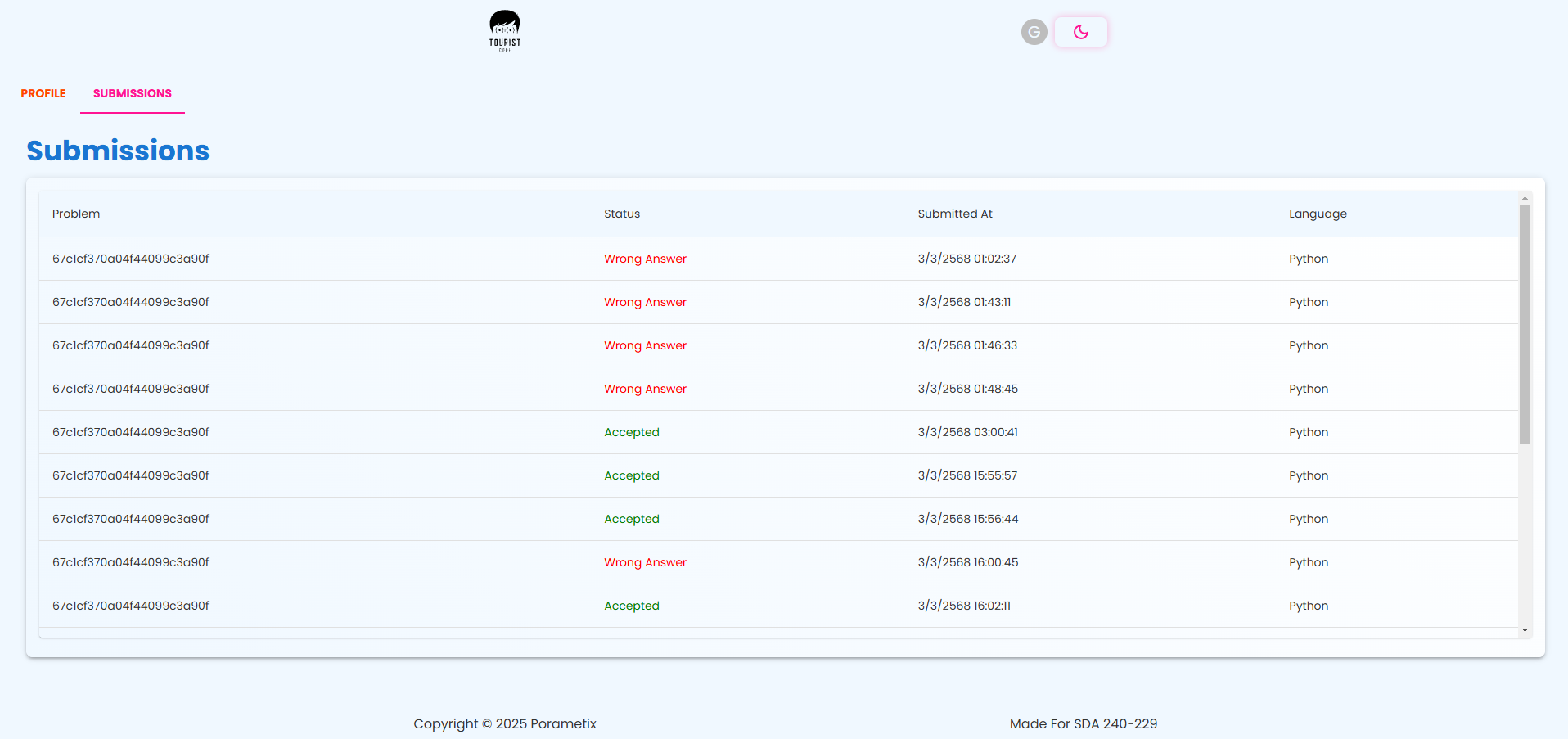


หลอดแสดงแต้ม Distance ที่แสดงให้เห็นว่าอยู่ระดับไหนเมื่อเทียบกับบุคคลอื่นๆ

Distance

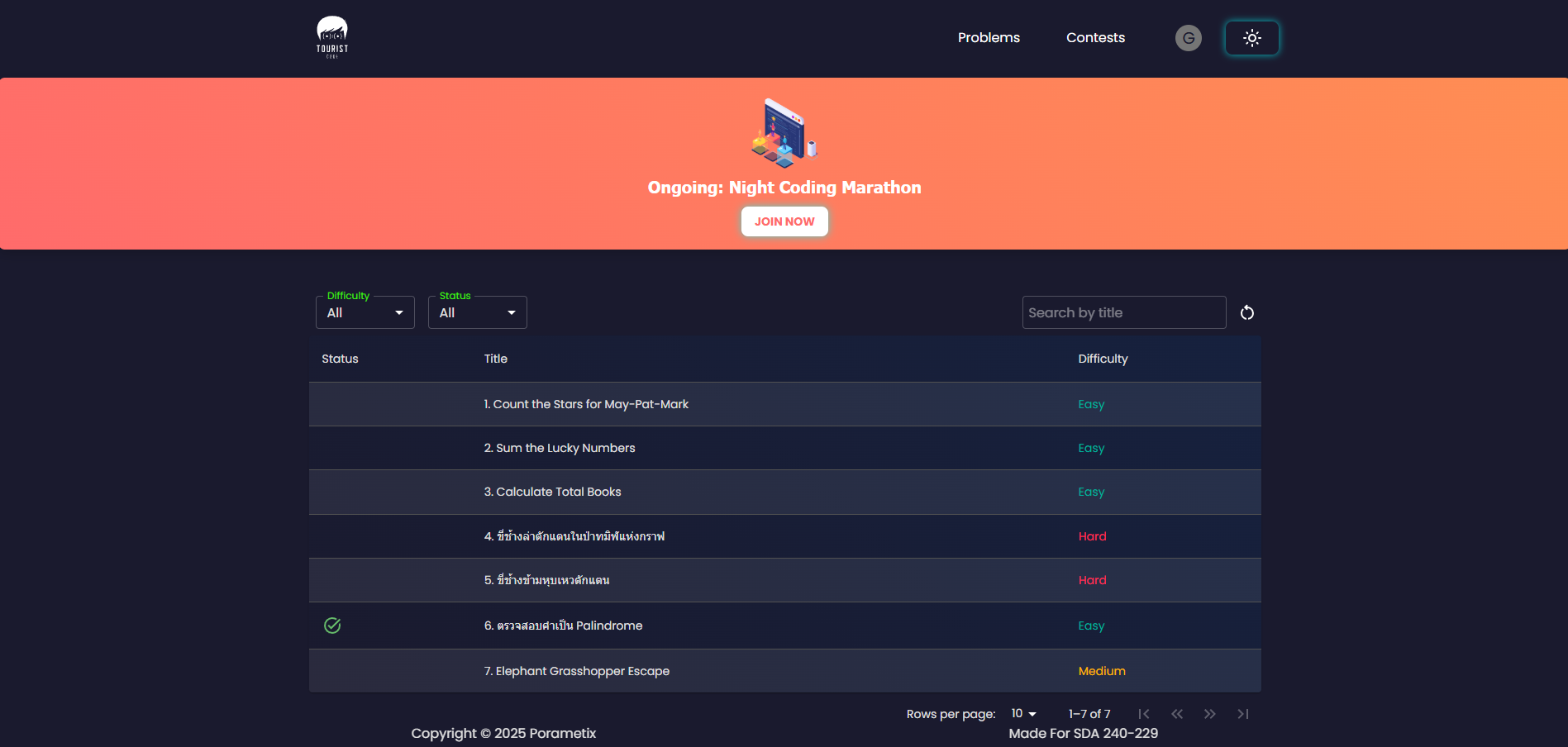
รูป 3.3.15 แสดงหน้าโปรไฟล์ หรือ หน้าดูข้อมูลส่วนตัว

อธิบายเพิ่มเติม: ทุกคนจะมี Point ของตัวเองเลยเรียกว่า “Distance” แล้วค่านี้เอาไว้เก็บเหมือนเพิ่ม Rank ให้กับตนเอง เมื่อค่านี้สูงๆ ก็เปรียบเหมือนประสบการณ์ที่ได้สะสมมา ก็จะทำให้ Rank เพิ่มขึ้นด้วย

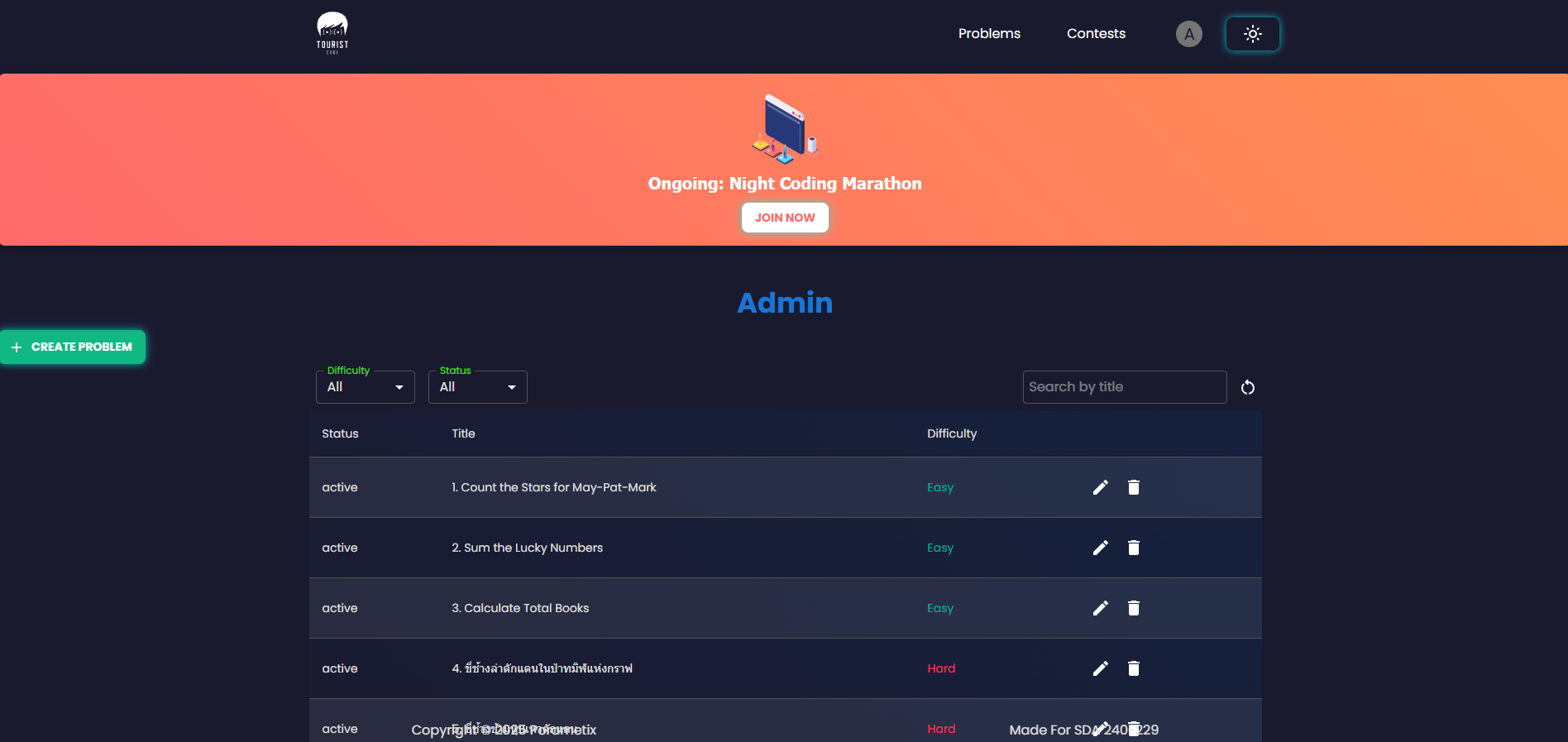


สถานะ(ผ่านหรือไม่ผ่าน)

รูป 3.3.16 แสดงหน้าดูประวัติการ Submit



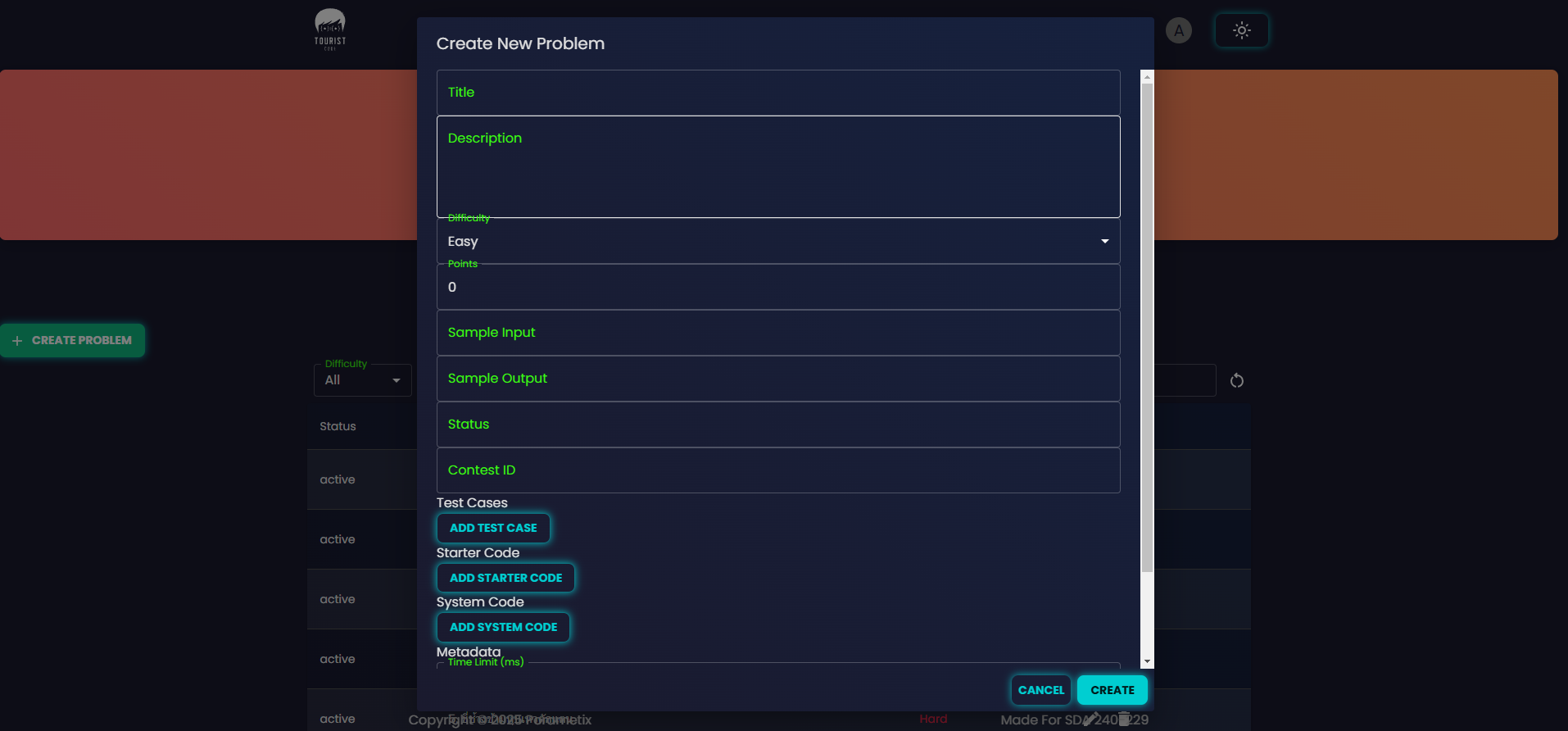
รูป 3.3.17 แสดงหน้าโจทย์ทั้งหมด (เมื่อเป็น Theme มืด)



เพิ่มโจทย์ปัญหาใหม่

แก้ไข/ลบ

รูป 3.3.18 แสดงหน้าโจทย์ทั้งหมดของระบบ Admin



รูป 3.3.18 แสดงระบบของการเพิ่มโจทย์ปัญหาของ Admin

**3.4 การทดสอบและการปรับปรุง**

การทดสอบการส่งโจทย์หรือการจำลองการแข่งขัน เพื่อให้เห็นว่าระบบเสถียรยังไง เช่น "ลองให้

คณะลองแข่งกันดู 10 คน พบว่าระบบแจ้งเตือนก่อนแข่งทำงานดี แต่บางครั้ง Monaco Editor โหลดช้า หรือกด Submit แล้ว Code แจ้งว่าผิดพลาด เลยมีการปรับปรุงให้ดียิ่งขึ้น"

**3.5 ความท้าทายในการพัฒนา**

การเชื่อม MongoDB กับ NestJS หรือการทำให้ TanStack Table อัปเดตแบบเรียลไทม์ แล้วแก้ยังไง เช่น "ตอนแรกตารางคะแนนมันไม่รีเฟรชเอง ต้องไปเพิ่ม WebSocket เข้ามาเสริม"

**3.6 ฟีเจอร์เด่นที่อยากเน้น**

การแจ้งเตือน 30 นาทีก่อนแข่ง หรือการใช้ Distance เป็นแต้มสะสม เช่น "ฟีเจอร์แจ้งเตือนนี่ช่วยให้คนไม่ลืมแข่ง ส่วน Distance ก็เหมือนให้รางวัลคนที่ขยันแก้โจทย์ หรือมีแรงจูงใจ ให้อยากมาแข่งขันมากขึ้น ได้ทั้งความรู้ใหม่ๆ และเรียนรู้เมื่อได้มีการแข่งขันกับคนอื่นๆ"

**4. เทคนิคที่ใช้**

ในการพัฒนาเราไม่ได้ใช้แค่ React กับ NestJS โดยเทคโนโลยีที่สำคัญอีกมากมาย และทำงานผสมผสาน เพื่อที่ทำให้เว็บไซต์สามารถทำงานได้อย่างมีประสิทธิภาพและปลอดภัย แบ่งออกเป็น 6 หมวดหมู่ ดังนี้:

**4.1 Category I**

**4.1.1 Concurrency**

ในการเขียนเว็บคือแนวคิดที่เกี่ยวกับการจัดการงานหลายๆ อย่างให้เกิดขึ้นหรือทำงานพร้อมกันได้ (หรือดูเหมือนพร้อมกัน) ในระบบ เพื่อให้เว็บไซต์สามารถตอบสนองผู้ใช้ได้อย่างมีประสิทธิภาพ โดยไม่ต้องรอให้งานหนึ่งเสร็จก่อนแล้วค่อยเริ่มงานต่อไป มันสำคัญมากในบริบทของเว็บ เพราะผู้ใช้มักจะเข้ามาใช้งานพร้อมกันเยอะๆ เช่น การส่งคำตอบโจทย์ ถ้ามีคนส่งโค้ดพร้อมกัน 10 คน 100 คน หรือมากกว่านั้น ระบบต้องจัดการให้ดี

**ตัวอย่างการทำงานของ Concurrency ในเว็บไซต์มีดังนี้:**

ถ้าไม่มี Concurrency:

* เซิร์ฟเวอร์จะตรวจโค้ดของคนแรก รอให้เสร็จ (สมมติ 2 วินาที) ค่อยไปตรวจคนต่อไป
* ถ้ามีผู้ใช้ 100 คน รวมแล้วคนสุดท้ายอาจรอถึง 200 วินาที (3 นาทีกว่าๆ)

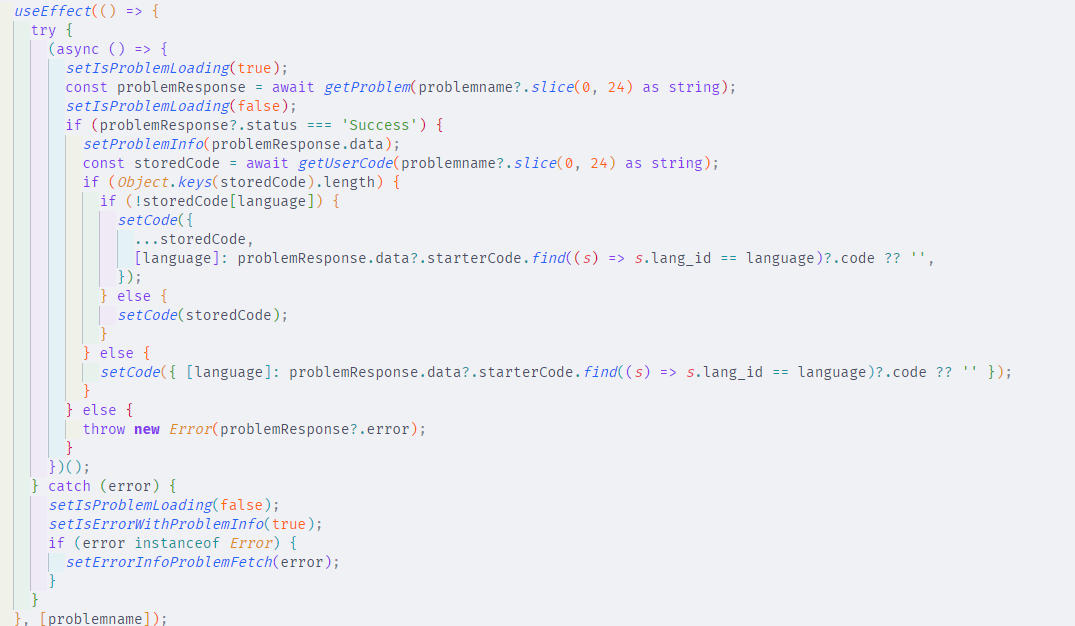
ถ้ามี Concurrency:

* เซิร์ฟเวอร์รับโค้ดทั้ง 100 อันมาแล้วจัดการพร้อมๆ กัน เช่น สลับไปตรวจโค้ดคนละอันทีละนิด หรือใช้ระบบแบบ asynchronous
* ทำให้ทุกคนได้ผลลัพธ์เร็วขึ้นมาก อาจแค่ 5-10 วินาที

เทคโนโลยีที่ช่วยเรื่อง Concurrency ในเว็บ เช่น Node.js, Promises และ Async/Await หรือ Web Workers ซึ่งสามารถนำมาใช้ เพื่อเพิ่มประสิทธิภาพในการจัดการคำขอจากผู้ใช้จำนวนมาก

**4.1.1.1 การดึงข้อมูลโจทย์ (getProblem) ใน useEffect**

การเรียก getProblem และ getUser เป็นการทำงานแบบ asynchronous โดยใช้ await ซึ่งหมายความว่าระหว่างที่รอข้อมูลจากเซิร์ฟเวอร์ ตัวแอปยังสามารถทำอย่างอื่นได้ (เช่น แสดง UI หรือตอบสนองผู้ใช้) ไม่ต้องหยุดรอแบบ synchronous ดังรูปที่ 4.1.1

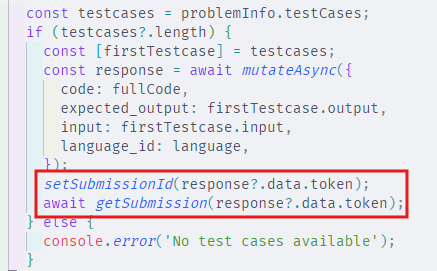


รูป 4.1.1 แสดงตัวอย่างโค้ดของการ getProblem และ getUser

อธิบายเพิ่มเติม: ถ้าผู้ใช้ 10 คนกดเข้าโจทย์พร้อมกัน การเรียก getProblem จะถูกส่งไปเซิร์ฟเวอร์พร้อมๆ กัน และเซิร์ฟเวอร์จะจัดการแบบไม่ต้องรอให้คนแรกเสร็จก่อน แอปฝั่งผู้ใช้ก็จะโหลด UI ไปก่อน ไม่ค้าง

**4.1.1.2 การส่งโค้ดไปตรวจ (onClickHandler)**

ถ้าผู้ใช้ 5 คนกด "Run" พร้อมกัน คำขอทั้ง 5 จะถูกส่งไปเซิร์ฟเวอร์พร้อมๆ กัน และเซิร์ฟเวอร์จะประมวลผลแต่ละอันแยกกัน ผู้ใช้แต่ละคนจะเห็นผลลัพธ์ของตัวเองโดยไม่ต้องรอคนอื่น



รูป 4.1.1 แสดงตัวอย่างโค้ดของการ Submit ใน Function onClickHandler

TanStack Query ช่วยจัดการ Concurrency โดยอัตโนมัติ เช่น ถ้าผู้ใช้กด "Run" หลายครั้งติดกัน มันจะควบคุมไม่ให้ส่งคำขอซ้ำซ้อนโดยไม่จำเป็น

**4.1.1.2 การส่งโค้ดแบบ Batch (onSubmitHandler)**

สมมติโจทย์มี 10 test cases ถ้าไม่มี Concurrency ระบบจะตรวจทีละตัว รอ 2 วินาทีต่อ test case รวม 20 วินาที แต่ด้วย Promise.all มันตรวจพร้อมกันได้เลย ใช้เวลาแค่ 2-3 วินาที (ขึ้นกับเซิร์ฟเวอร์)



รูป 4.1.2 แสดงตัวอย่างโค้ดของการส่งโค้ดแบบ Batch ใน Function onSubmitHandler

อธิบายเพิ่มเติม: การใช้ Promise.all ใน batchPromises เป็นตัวอย่างชัดๆ ของ Concurrency เพราะมันรันการตรวจสอบสถานะของทุก test case พร้อมกัน แทนที่จะรอให้ test case แรกเสร็จก่อนแล้วค่อยไปตัวต่อไป

อย่างไรก็ตาม ในระบบโดยรวมแล้วยังมีส่วนอื่นๆ อีกมากที่ใช้ Concurrency เพื่อเพิ่มประสิทธิภาพและความลื่นไหลในการทำงานของเว็บไซต์

**4.2 Category II**

**4.2.1 Docker**

Docker เป็นเครื่องมือที่ช่วย "แพ็กเกจ" โค้ดและสิ่งที่โค้ดต้องการไว้ใน container ซึ่งเหมือนกล่องที่พกพาได้กล่องนี้จะทำงานได้เหมือนกันทุกที่ ไม่ว่าจะบนเครื่อง, เซิร์ฟเวอร์, หรือคลาวด์ โดยใช้ Docker ในส่วนต่างๆ ดังนี้:

**4.2.1.1 สร้าง Container สำหรับ Backend และ Frontend**

โดยส่วนแรก ในส่วนของ Backend ได้ใช้ Docker ในการสร้าง 2 ขั้นตอน:

***Build Stage:*** ติดตั้ง dependencies ด้วย yarn install และ build โค้ดเป็น dist

***Production Stage****:* คัดเฉพาะไฟล์ที่จำเป็น (dist) มาใส่ container ใหม่ ลดขนาดโดยติดตั้งแค่ dependencies ที่ใช้จริง (--production) และรันด้วย node dist/main บนพอร์ต 3000

เปลี่ยน user เป็น node เพื่อความปลอดภัย (ไม่ใช้ root)



รูป 4.2.1 แสดงตัวอย่างโค้ดของการใช้งาน Docker ใน Backend

โดยส่วนต่อไป ในส่วนของ Frontend ได้ใช้ Docker ในการ:

* สร้าง build ของ frontend (React) ก่อน
* จากนั้นใช้ nginx:alpine (web server ขนาดเล็ก) เพื่อเสิร์ฟไฟล์ static จาก dist บนพอร์ต 80 คัดลอกไฟล์การตั้งค่า Nginx (nginx.conf) เพื่อกำหนดพฤติกรรมของเซิร์ฟเวอร์

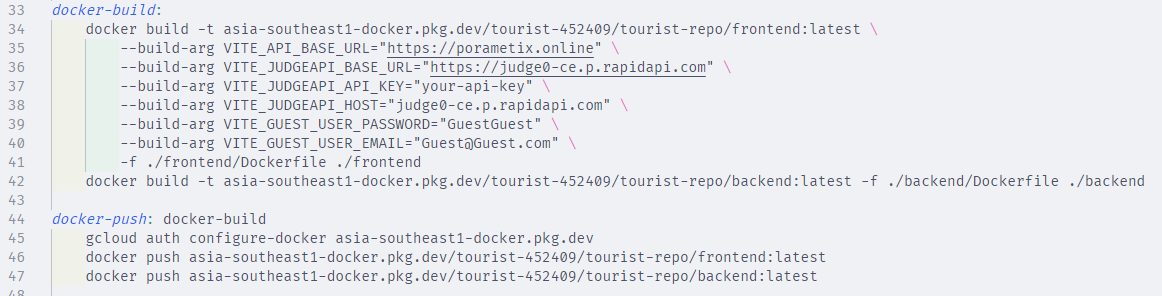
Docker ช่วยสร้าง container ที่แยก backend (NestJS) และ frontend (React + Nginx) ออกจากกัน โดยแต่ละอันมีสภาพแวดล้อมที่พร้อมใช้งานทันที ไม่ต้องติดตั้งอะไรเพิ่มบนเครื่องที่รัน



รูป 4.2.2 แสดงตัวอย่างโค้ดของการใช้งาน Docker ใน Frontend

**4.2.1.2 Build และ Push ไปยัง Container Registry**

***Makefile:***



รูป 4.2.3 แสดงตัวอย่างโค้ดของการใช้งาน Docker ใน Makefile

***GitHub Actions:***



รูป 4.2.4 แสดงตัวอย่างโค้ดของการใช้งาน Docker ใน GitHub Actions

โดยได้ใช้ Docker ร่วมกับเทคโนโลยีอื่นไม่ว่าจะเป็น Makefile และ Github Actions ดังนี้:

* ใช้สร้าง image ของ backend และ frontend ด้วยคำสั่ง docker build
* ใส่ build arguments (เช่น API URL, API Key) เพื่อกำหนดค่า environment variables ตอน build frontend
* Push image ไปยัง Google Artifact Registry (asia-southeast1-docker.pkg.dev) เพื่อเก็บไว้ใช้ในคลาวด์

Docker ทำให้รันแอปใน container บนคลาวด์ได้ง่าย ใช้ร่วมกับ Kubernetes (k8s) เพื่อ Deploy บน Cloud ซึ่งอยู่ในหัวข้อถัดๆ ไปใช้ container เหล่านี้เพื่อจัดการ scale และความเสถียร

**4.2.2 Cloud Function**

Google Cloud Functions ถูกใช้งานเพื่อดำเนินการส่งอีเมลแจ้งเตือนให้กับผู้เข้าร่วมการแข่งขัน (contest participants) โดยฟังก์ชัน send-reminder จะถูกเรียก (triggered) เมื่อมีข้อความใหม่จาก Google Cloud Pub/Sub ใน topic contest-notifications

การใช้งาน Cloud Functions ประกอบด้วยส่วนหลักดังนี้:

* Trigger: ฟังก์ชันถูกกำหนดให้ทำงานแบบ CloudEvent และรับข้อความจาก Pub/Sub
* Logic: ดึงข้อมูลการแข่งขันจากข้อความ และส่งอีเมลแจ้งเตือนผ่าน Nodemailer โดยใช้ Gmail SMTP

รูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร, จำนวน

เนื้อหาที่สร้างโดย AI อาจไม่ถูกต้อง

รูป 4.2.5 แสดงตัวอย่างโค้ดส่วนของ Google Cloud Function send-reminder

*อธิบายเพิ่มเติม*:

* Pub/Sub Trigger -> Cloud Function -> Email
* ฟังก์ชัน send-reminder ถูกกำหนดให้เป็น CloudEvent Trigger ซึ่งรับข้อมูลจาก Pub/Sub topic contest-notifications
* ข้อมูลที่ได้รับจะถูก decode จาก base64 และแปลงเป็น JSON เพื่อดึงรายละเอียดการแข่งขัน
* จากนั้นใช้ nodemailer สร้าง transporter เพื่อส่งอีเมลไปยังผู้เข้าร่วมทั้งหมดโดยใช้ Gmail SMTP และรอการส่งทั้งหมดด้วย Promise.all

รูปภาพประกอบด้วย ข้อความ, ตัวอักษร, อัลจีบรา, ภาพหน้าจอ

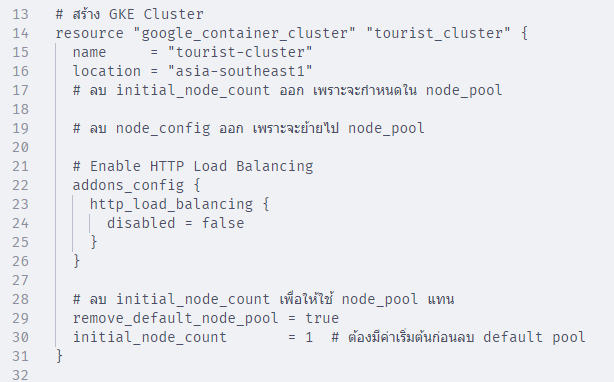
เนื้อหาที่สร้างโดย AI อาจไม่ถูกต้อง

รูป 4.2.6 แสดงตัวอย่าง email ที่ได้รับจาก Cloud Function

**4.2.3 Virtual Machine (VM)**

ในโปรเจกต์นี้ VM ไม่ได้ถูกใช้งานโดยตรงในลักษณะที่ต้องสร้างหรือจัดการ VM ขึ้นมาเอง เช่น ใช้ Google Compute Engine แบบ manual แต่ถูกใช้งานผ่าน Google Kubernetes Engine (GKE)ซึ่ง GKE ใช้ VM เป็นโครงสร้างพื้นฐานให้อัตโนมัติ ซึ่งมีหลายส่วนดังนี้:

**4.2.3.1 GKE Cluster และ Node Pool (Terraform)**

****

รูป 4.2.5 แสดงตัวอย่างโค้ดของการสร้าง GKE Cluster ใน Terraform

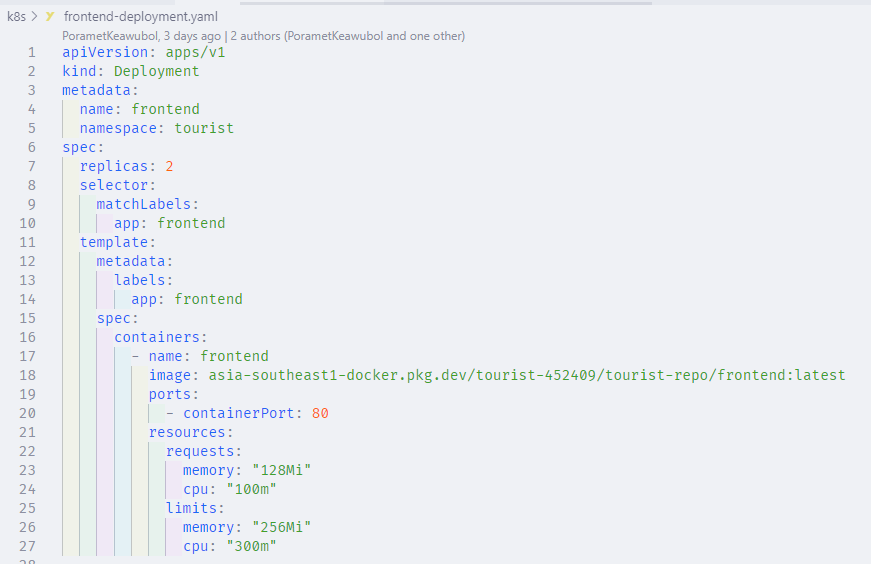


รูป 4.2.6 แสดงตัวอย่างโค้ดของการสร้าง Node Pool แยกพร้อม Auto-scaling ใน Terraform

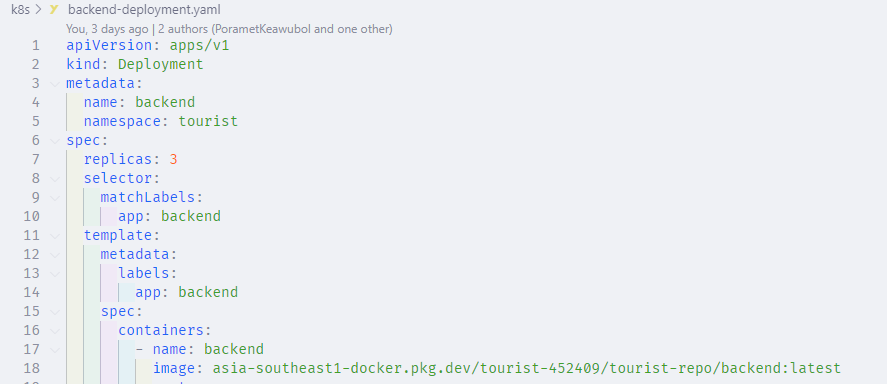
อธิบายเพิ่มเติม: จากโค้ดในรูป 4.2.5 และรูป 4.2.6 ที่อยู่ใน Terraform ซึ่งเป็นการใช้สร้าง GKE Cluster ซึ่งภายใน GKE จะมี VM เป็น "nodes" ที่รัน container

* GKE สร้าง cluster ชื่อ tourist-cluster และ node pool ชื่อ primary-node-pool
* แต่ละ node ใน pool เป็น VM ประเภท e2-medium (1 vCPU, 4GB RAM) ที่ Google จัดการให้
* Autoscaling ตั้งให้มี VM อย่างน้อย 1 เครื่อง สูงสุด 5 เครื่อง ตามโหลดของแอป

**4.2.3.2 Kubernetes Deployment**



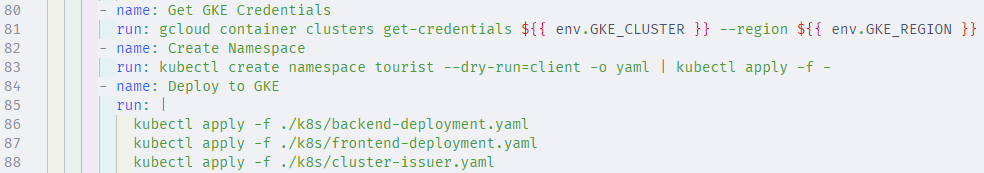
รูป 4.2.7 แสดงตัวอย่างโค้ด “k8s/backend-deployment.yaml”



รูป 4.2.8 แสดงตัวอย่างโค้ด “k8s/frontend-deployment.yaml”

*อธิบายเพิ่มเติม:* จากตัวอย่างโค้ดก่อนหน้ารูป 4.2.7 และรูป 4.2.8 เป็นส่วนของ Backend ที่รัน 3 replicas และ Frontend รัน 2 replicas บน VM ใน GKE โดยในส่วนนี้ VM เหล่านี้ถูกสร้างและจัดการโดย GKE ไม่ต้องแตะเอง

**4.2.3.3** CI/CD Pipeline (GitHub Actions)



รูป 4.2.9 แสดงตัวอย่างโค้ดใน GitHub Actions deploy ไป GKE

*อธิบายเพิ่มเติม:* จากรูป 4.2.9 เมื่อรัน kubectl apply เพื่อ deploy container ไปยัง GKE ซึ่งใช้ VM เป็นโครงสร้างพื้นฐาน สรุปสั้นๆคือ GitHub Actions ดึง credential ของ GKE แล้ว deploy container ไปรันบน VM ใน cluster

สรุปการใช้ VM ถูกใช้ผ่าน GKE เท่านั้น ไม่มีการใช้ VM แบบ standalone (เช่น Google Compute Engine) โดย

* Terraform: สร้าง GKE Cluster และกำหนด VM ใน node pool (e2-medium)
* Kubernetes: รัน Docker container ของ backend และ frontend บน VM
* CI/CD: Deploy ไป GKE ซึ่ง VM ทำงานอยู่เบื้องหลัง

โดยทั้งหมดนี้ ไม่มีการใช้ VM โดยตรงนอกเหนือจาก GKE ดังนั้น VM ในโปรเจกต์นี้ถูกฝังอยู่ใน GKE เป็นหลัก VM ใน Project นี้คือ "เครื่องคอมพิวเตอร์เสมือน" ที่ Google จัดให้ผ่าน GKE เพื่อรัน container ของ backend และ frontend ซึ่งไม่ต้องจัดการ VM เอง เพราะ Terraform และ Kubernetes ทำทุกอย่างให้อัตโนมัติ ซึ่งจะถูกพูดถึงในหัวข้อถัดๆไป

**4.3 Category III**

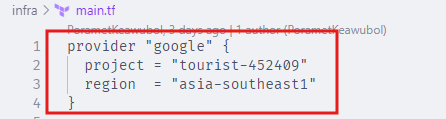
ซึ่งใน Category นี้เป็นเป็นส่วนหลักๆที่ถูกพูดถึงในหัวข้อก่อนๆหน้า ที่เกี่ยวกับการ Deploy เว็บไซต์ไปยัง Google Cloud อัตโนมัติ เช่น VM ที่ไม่ได้เกิดจากการสร้างแบบ manual บน Google Cloud และอีกหลายส่วนที่จัดการอัตโนมัติร่วมๆกันผ่าน เทคโนโลยีหลายๆส่วน ทำให้เกิดการ Deploy แบบอัตโนมัติ ดังนี้:

**4.3.1 Terraform**

Terraform เป็นเครื่องมือที่ช่วยสร้างและจัดการโครงสร้างพื้นฐานบนคลาวด์ (Google Cloud)

โดยอัตโนมัติ โดย Project นี้ใช้งาน ซึ่งแบ่งออกเป็น 3 ส่วน ดังนี้:

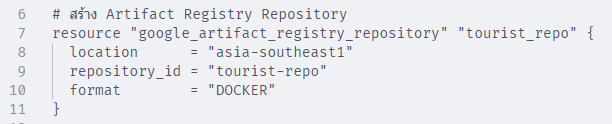
**4.3.1.1 กำหนดผู้ให้บริการ**

****

รูป 4.3.1 แสดงตัวอย่างโค้ดใน Terraform ที่กำหนด Provider

*อธิบายเพิ่มเติม:* เป็นการบอก Terraform ว่าจะใช้ Google Cloud ในโปรเจกต์ tourist-452409 และตั้ง region เป็น asia-southeast1 (สิงคโปร์)

**4.3.1.2 สร้าง Artifact Registry Repository**

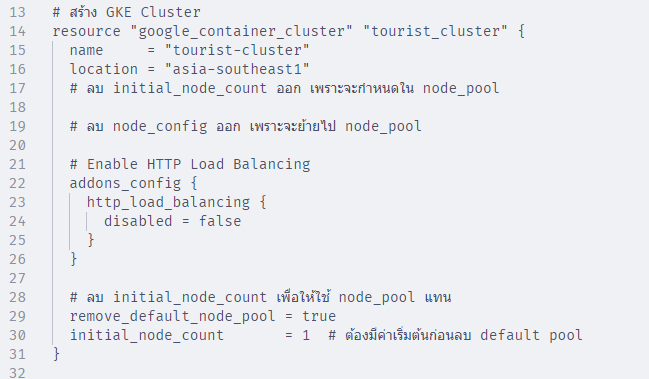
****

รูป 4.3.2 แสดงตัวอย่างโค้ดใน Terraform ที่สร้าง Artifact Registry Repository

*อธิบายเพิ่มเติม:* สร้างที่เก็บ Docker image ชื่อ tourist-repo ใน Google Artifact Registry เพื่อเก็บ image ของ backend และ frontend (เช่น frontend:latest, backend:latest) ที่ build จาก Docker

**4.3.1.3 สร้าง GKE Cluster และ Node Pool (เหมือนกันโค้ดในหัวข้อก่อนหน้าๆ)**

**โค้ด (Cluster):**

****

รูป 4.3.3 แสดงตัวอย่างโค้ดใน Terraform ที่สร้าง GKE Cluster

**โค้ด (Node Pool):**



รูป 4.3.4 แสดงตัวอย่างโค้ดใน Terraform ที่สร้าง Node Pool

*อธิบายเพิ่มเติม:* Terraform ในรูป 4.3.3 และรูป 4.3.4 แสดงให้เห็นถึงการทำงาน โดย:

สร้าง GKE Cluster ชื่อ tourist-cluster เพื่อรัน Kubernetes

ลบ default node pool แล้วสร้าง Node Pool ใหม่ชื่อ primary-node-pool โดย:

* ใช้ VM แบบ e2-medium (1 vCPU, 4GB RAM) ขนาดดิสก์ 20GB
* ตั้ง autoscaling ให้มี node อย่างน้อย 1 และสูงสุด 5 ตามปริมาณงาน
* เปิด auto-repair และ auto-upgrade เพื่อให้ GKE จัดการ VM อัตโนมัติ
* เปิด HTTP Load Balancing เพื่อกระจาย traffic ไปยัง container

**4.3.1.4 Terraform ถูกใช้ยังไงใน Project**

1. เขียนโค้ด Terraform นี้ในไฟล์ (เช่น main.tf)

2. รันคำสั่ง terraform init เพื่อโหลด provider

3. รัน terraform apply เพื่อสร้าง Artifact Registry และ GKE Cluster บน Google Cloud

**ผลลัพธ์:**

* ได้ที่เก็บ Docker image และ cluster พร้อมใช้งานทันที โดยไม่ต้องตั้งค่าผ่านหน้าเว็บ Google Cloud เอง
* เชื่อมกับ Docker และ Kubernetes ในขั้นตอนถัดไป (เช่น push image และ deploy)

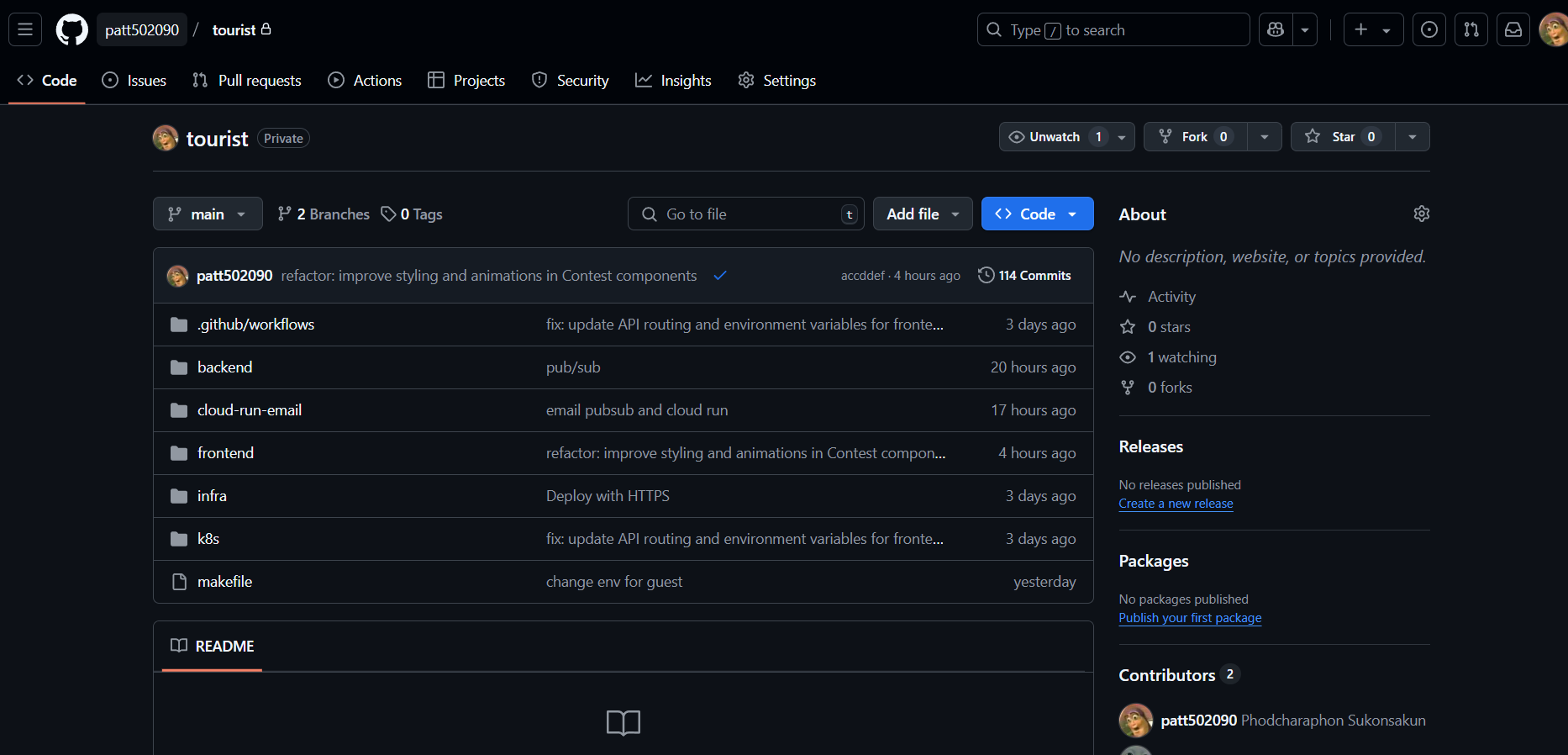
Terraform ถูกใช้เพื่อจัดการโครงสร้างพื้นฐานบน Google Cloud โดยกำหนด region ที่ asia-southeast1, สร้าง Artifact Registry ชื่อ tourist-repo สำหรับเก็บ Docker image, และตั้งค่า GKE Cluster ชื่อ tourist-cluster พร้อม node pool แบบ e2-medium ที่มี autoscaling 1-5 nodes และ HTTP load balancing เพื่อให้ระบบรัน container ได้อย่างมีประสิทธิภาพและขยายตามจำนวนผู้ใช้

**4.3.2 Git**

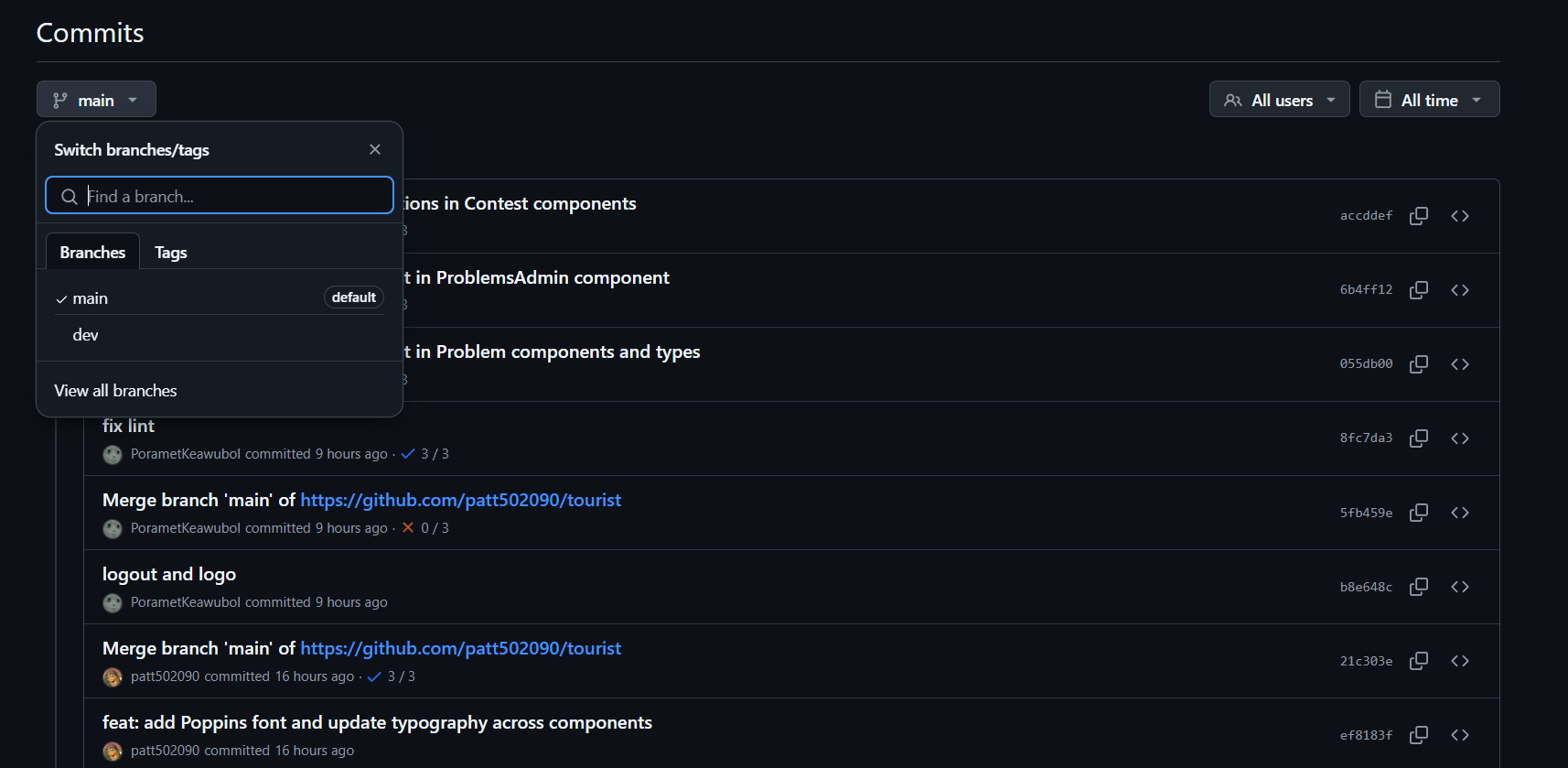
ในส่วนของเทคโนโลยีนี้เป็นส่วนหลักๆ สำคัญอีกส่วนหนึ่งที่ทำให้สามารถทำงานได้ง่ายมากขึ้น เกี่ยวกับการ Deploy ซึ่งหมายถึงเมื่อมีการเปลี่ยนแปลง Project แล้วจะทำการแก้ไข Project ที่อยู่ Production ได้อย่างรวดเร็วเมื่อมีการพัฒนาหรือแก้ไขฟีเจอร์ใหม่ ซึ่งเป็นส่วนสำคัญที่ช่วยให้กระบวนการพัฒนามีประสิทธิภาพและลดข้อผิดพลาดจากการทำงานร่วมกัน และทำงานร่วมกับเพื่อนร่วมทีมได้แบบราบรื่น

**4.3.2.1 การทำงานร่วมกับทีม**

ในโปรเจกต์นี้ Git ถูกใช้งานผ่าน GitHub ซึ่งช่วยให้เห็นประวัติการ push โค้ดของสมาชิกแต่ละคนได้อย่างชัดเจน ซึ่งหมายถึงการแก้ไข Project หรือพัฒนาฟีเจอร์ใหม่ๆ จะเห็นประวัติการทำงานและทำให้ Project มีความคืบหน้าและไม่ขัดแย้งกัน พัฒนาไปในทิศทางเดียวกัน



รูป 4.3.5 แสดงหน้าเว็บไซต์ของ Github ที่ใช้งาน

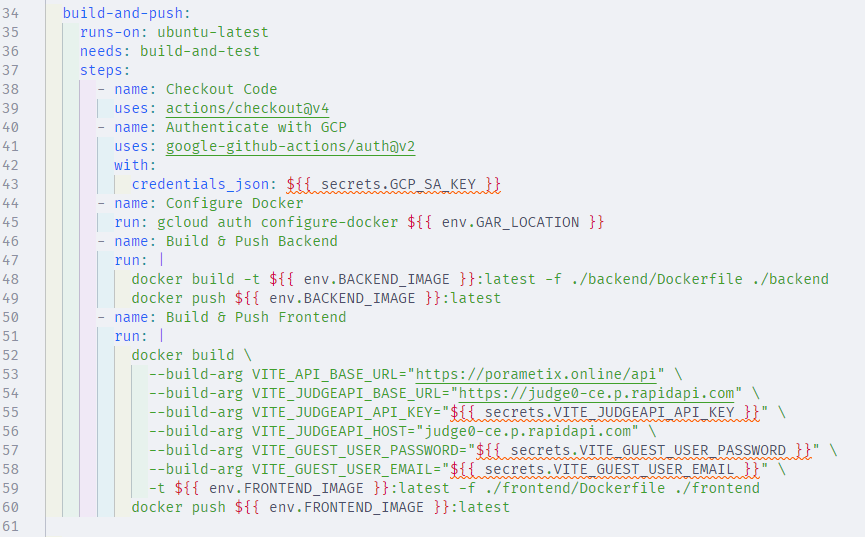


รูป 4.3.6 แสดง Flow พัฒนา Project บน Github

*อธิบายเพิ่มเติม:* แสดงหน้า Commit History บน GitHub ที่ระบุชื่อผู้ push โค้ด เช่น "นายกรวิทย์ กอหลัง" หรือ "นายพชรพล ศุกลสกุล" พร้อมวันที่และข้อความอธิบายการเปลี่ยนแปลง ทำให้ทีมสามารถตรวจสอบได้ว่าใครรับผิดชอบส่วนไหน และช่วยให้การทำงานร่วมกันเป็นไปอย่างโปร่งใสและมีระเบียบ

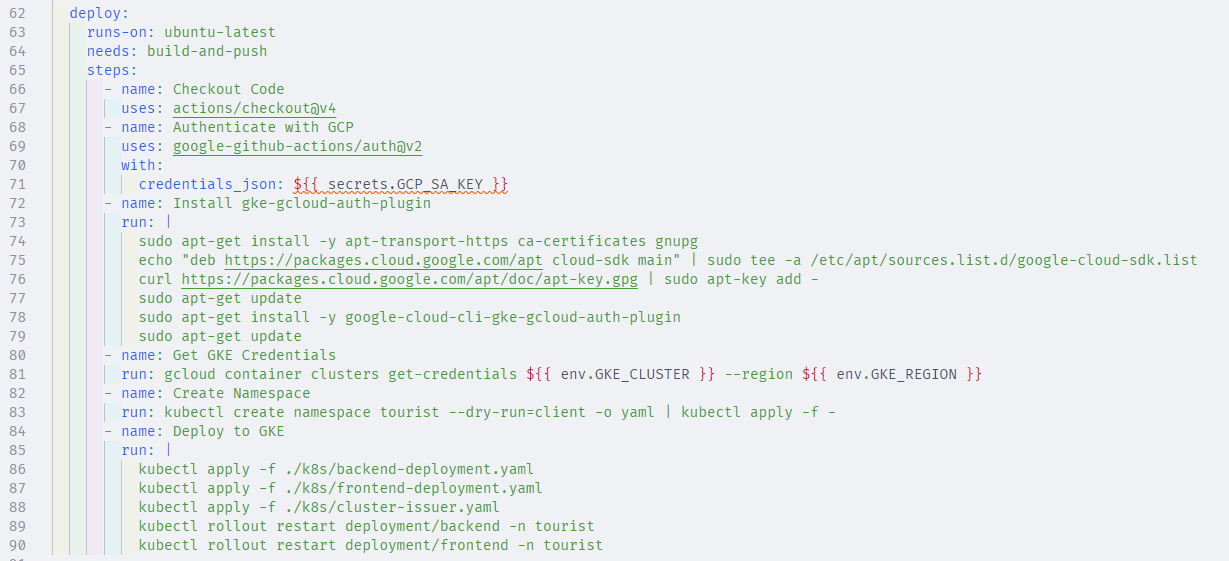
**4.3.2.3 การผสาน Git กับ CI/CD Pipeline**

Git ยังทำงานร่วมกับ CI/CD Pipeline ที่กำหนดใน GitHub Actions เพื่อทำให้การ deploy เป็นอัตโนมัติ เมื่อสมาชิกในทีม push โค้ดไปยัง branch main ระบบจะรันกระบวนการ build, test และ deploy ไปยัง Google Kubernetes Engine (GKE) โดยอัตโนมัติ ดังตัวอย่างโค้ดบางส่วนที่ใช้งาน:



รูป 4.3.7 แสดงโค้ดส่วนของ build-and-push บน workflows

*อธิบายเพิ่มเติม*: จากรูป 4.3.7 แสดงให้เห็น flow การทำงานของ CI/CD Pipeline ในส่วนของ build-and-push ที่จะทำงานโดย รันหลัง build-and-test เสร็จ จะดึงโค้ดใหม่แล้ว ล็อกอิน Google Cloud ด้วย key จาก secrets ที่เก็บไว้บน repository ของ Github แล้วไปตั้งค่า Docker ให้ push ไป Artifact Registry Build และ push Docker image ของ backend และ frontend (ใส่ build arguments เช่น API URL, API Key)



รูป 4.3.8 แสดงโค้ดส่วนของ deploy บน workflows

*อธิบายเพิ่มเติม*: จากรูป 4.3.8 แสดงให้เห็น flow การทำงานของ CI/CD Pipeline ในส่วน deploy ที่จะทำงานโดย รันหลัง build-and-push เสร็จ ดึงโค้ดใหม่ แล้วก็ล็อกอิน Google Cloud เมื่อเสร็จสิ้นก็ ติดตั้ง plugin สำหรับ GKE แล้วดึง credential ของ GKE cluster สร้าง namespace tourist แล้วในขึ้นสุดท้ายก็ Deploy backend, frontend, และ cluster issuer ไป GKE แล้ว restart deployment

โดยขั้นตอนทั้งหมดของ CI/CD Pipeline สรุปได้ดังนี้:

build-and-test: เตรียมโค้ด (test และ build)

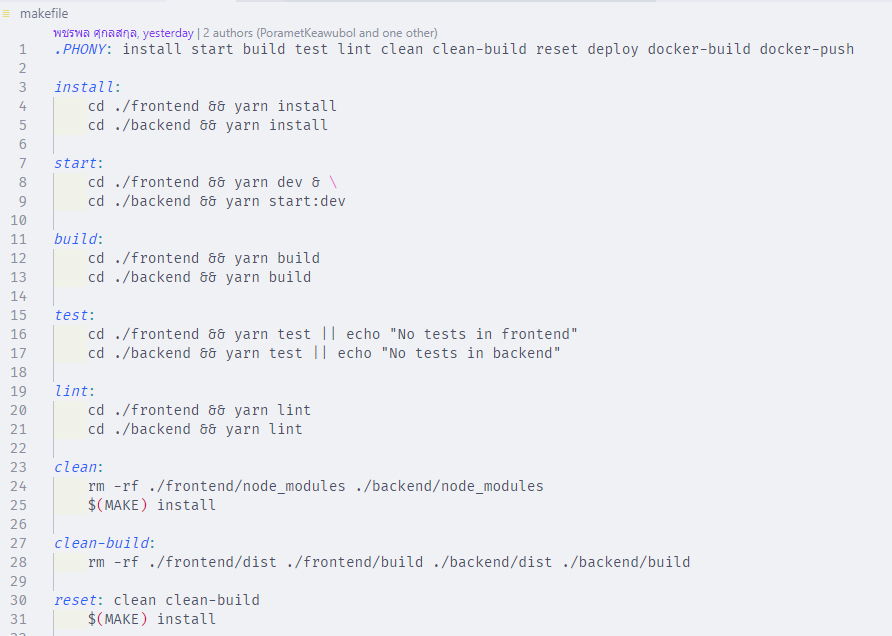
build-and-push: สร้างและ push Docker image ไป Artifact Registry

deploy: นำ image ไปรันบน GKE

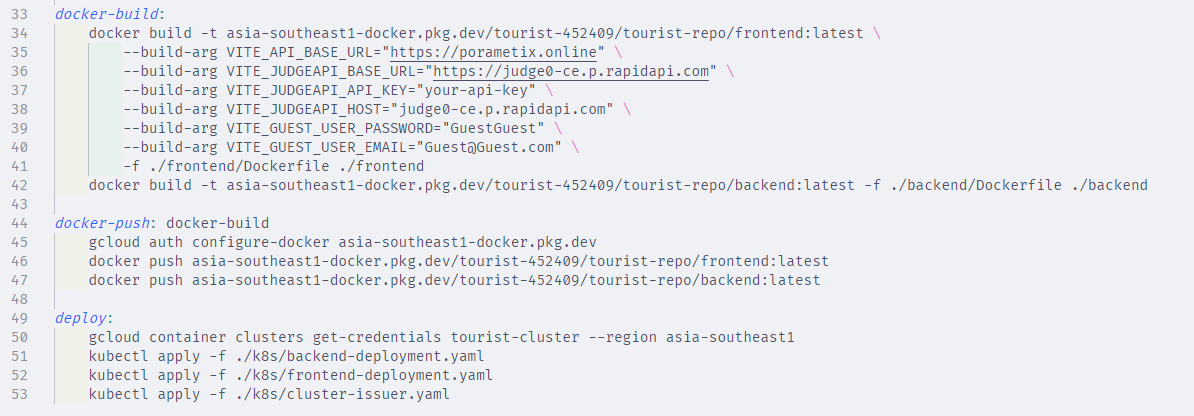
Link: <https://github.com/patt502090/tourist>

**4.3.3 Makefile**

Makefile เป็นเครื่องมือที่ใช้จัดการคำสั่งใน Project เพื่อทำให้การพัฒนา การ build และการ deploy เป็นไปอย่างสะดวกและเป็นระบบ โดยแบ่งการทำงานออกเป็นส่วนต่างๆ ดังนี้:



รูป 4.3.9 แสดงตัวอย่างการเขียนโค้ดส่วนแรกบน Makefile



รูป 4.3.10 แสดงตัวอย่างการเขียนโค้ดส่วนที่สองบน Makefile

*อธิบายเพิ่มเติม:* จากรูป 4.3.9 และ 4.3.10 แสดงถึงเห็นถึงการเขียนโค้ดใช้งาน Makefile ที่จะช่วยให้การทำงานเป็นอัตโนมัติ และจัดการ Cloud Deployment ได้ง่ายขึ้น

|  |  |
| --- | --- |
| คำสั่ง | ทำอะไรในโปรเจค Cloud นี้? |
| make install | ติดตั้ง Dependencies ทั้ง Frontend & Backend |
| make start | รันโปรเจคทั้งสองส่วนพร้อมกัน |
| make build | สร้าง Production Build |
| make test | รัน Unit Tests |
| make clean | ลบ node\_modules และติดตั้งใหม่ |
| make docker-build | สร้าง Docker Image ของแอป |
| make docker-push | Push Docker Image ขึ้น Google Cloud |
| make deploy | Deploy ระบบขึ้น Google Kubernetes Engine (GKE) |

Makefile ใน Tourist Code ใช้จัดการคำสั่งต่างๆ เช่น การติดตั้ง dependencies (install), รันโหมดพัฒนา (start), build โค้ด (build), ทดสอบ (test), ตรวจสอบโค้ด (lint), ล้างข้อมูล (clean, reset), และ deploy ด้วย Docker (docker-build, docker-push) และ Kubernetes (deploy) เพื่อให้กระบวนการพัฒนาและการนำไปใช้งานจริงเป็นไปอย่างราบรื่นและมีประสิทธิภาพ

**4.4 Category IV**

**4.4.1 Pub/Sub**

ในโปรเจคนี้ Google Cloud Pub/Sub ถูกใช้งานเพื่อจัดการระบบแจ้งเตือน (notification system) สำหรับการแข่งขัน (contests) โดยทำหน้าที่เผยแพร่ข้อความ (messages) ไปยังผู้เข้าร่วมเมื่อการแข่งขันใกล้เริ่มต้น ซึ่งช่วยให้ระบบสามารถส่งการแจ้งเตือนได้อย่างมีประสิทธิภาพ รองรับการขยายขนาด (scalable) เมื่อมีผู้ใช้จำนวนมาก

การใช้งาน Pub/Sub ในโปรเจคนี้ประกอบด้วย 2 ส่วนหลักดังนี้:

* **Publisher**: ฝั่ง backend (ContestsService) สร้างและส่งข้อความแจ้งเตือนไปยัง topic ชื่อ contest-notifications เมื่อการแข่งขันอยู่ห่างจากเวลาเริ่มต้นไม่เกิน 30 นาที
* **Subscriber**: Cloud Function (send-reminder) รับข้อความจาก topic และส่งอีเมลแจ้งเตือนไปยังผู้เข้าร่วมผ่าน Nodemailer

**รูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร, จำนวน

เนื้อหาที่สร้างโดย AI อาจไม่ถูกต้อง**

รูป 4.4.1 แสดงตัวอย่างโค้ดส่วนของการเผยแพร่ข้อความใน Google Cloud Pub/Sub

*อธิบายเพิ่มเติม*: ฟังก์ชัน sendNotification ถูกเรียกโดย Cron Job ใน ContestsService ทุก 30 นาที เพื่อตรวจสอบการแข่งขันที่ใกล้เริ่ม (checkUpcomingContests) ข้อมูลการแข่งขันถูกแปลงเป็น JSON และ encode เป็น Buffer ก่อนส่งไปยัง topic contest-notifications ผ่าน pubSubClient

รูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร, จำนวน

เนื้อหาที่สร้างโดย AI อาจไม่ถูกต้อง

รูป 4.4.2 แสดงตัว Trigger ของการรับข้อความจาก Pub/Sub และส่งอีเมลแจ้งเตือน

Cloud Function send-reminder ถูก trigger โดย Pub/Sub เมื่อมีข้อความใหม่ใน topic contest-notification โดย flow ของ Pub/Sub

จะเริ่มโดย Backend -> Pub/Sub -> Cloud Function -> Email

**4.4.2 Load Balancer**

ใน Project นี้ ได้มีการใช้ Load Balancer ถูกใช้งานเพื่อกระจายการรับส่งข้อมูล (traffic) ไปยัง container ของ frontend และ backend ที่รันบน Google Kubernetes Engine (GKE) ซึ่งช่วยให้ระบบสามารถรองรับผู้ใช้จำนวนมากได้อย่างมีประสิทธิภาพและป้องกันการ overload ของ container เดียว

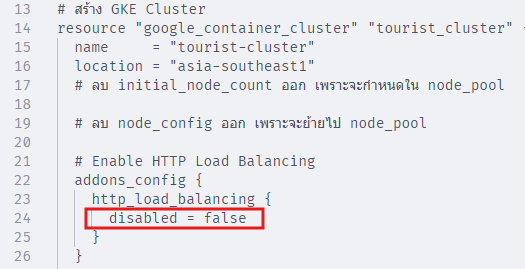
ซึ่งใน Project นี้การใช้งานประกอบด้วย 2 ส่วนดังนี้:

Frontend Service: เมื่อกำหนด type: LoadBalancer ใน frontend-service เพื่อให้ผู้ใช้เข้าถึงเว็บไซต์ผ่าน domain เช่น porametix.online

HTTP Load Balancing ใน GKE: เปิดใช้งานผ่าน Terraform เพื่อกระจาย traffic ภายใน cluster

โค้ดตัวอย่างเพิ่มเติมที่เกี่ยวข้อง:

Terraform (GKE Cluster):



# เปิดใช้งาน HTTP Load Balancing

รูป 4.4.3 แสดงตัวอย่างโค้ดส่วนของกำหนด LoadBlancer ใน Terraform

*อธิบายเพิ่มเติม:* เปิดฟีเจอร์ HTTP Load Balancing ใน GKE เพื่อให้ Kubernetes กระจาย traffic ไปยัง node และ pod ต่างๆ

Kubernetes (Frontend Service):



รูป 4.4.4 แสดงตัวอย่างโค้ดส่วนของกำหนด LoadBlancer ใน “frontend-deployment.yaml”

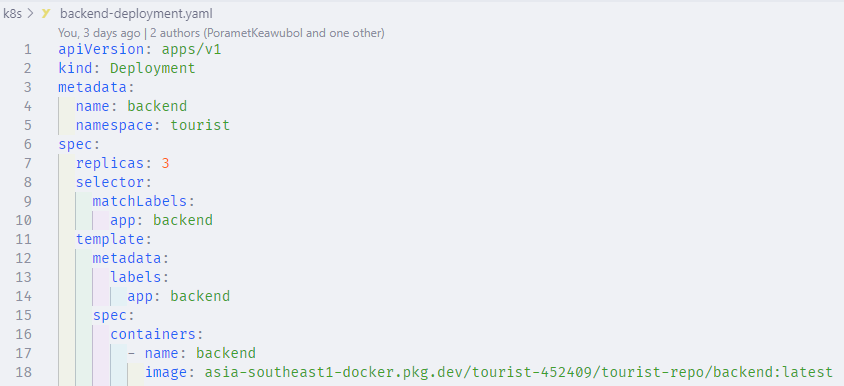
*อธิบายเพิ่มเติม:* กำหนด LoadBalancer เพื่อให้ GKE สร้าง external Load Balancer สำหรับ frontend โดยรับ traffic ที่พอร์ต 80

**4.4.3 Kubernetes (k8s)**

ในเทคโนโลยีนี้เป็นเทคโนโลยีที่สำคัญมากเป็นอันดับต้นๆของ Project เลยก็ได้ที่จะช่วยให้ ทีมผู้พัฒนาออกแบบ Project แล้วทำงานได้ง่ายมากขึ้น โดย Kubernetes ใน Project ถูกใช้เพื่อจัดการและรัน container ของ backend และ frontend บน Google Kubernetes Engine (GKE) โดยมีจุดประสงค์หลักดังนี้:

**4.4.3.1 จัดการ Container (Deployment)**

รันและควบคุม container ของ backend และ frontend ให้ทำงานอย่างเสถียร

****

รูป 4.4.5 แสดงตัวอย่างโค้ดส่วนของการจัดการ Container ใน “backend-deployment.yaml”



รูป 4.4.6 แสดงตัวอย่างโค้ดส่วนของการจัดการ Container ใน “frontend-deployment.yaml”

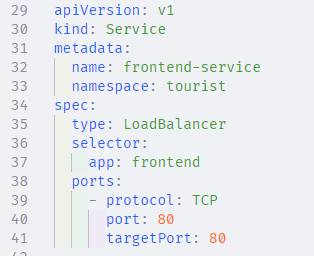
*อธิบายเพิ่มเติม:*

* Backend รัน 3 replicas (3 container) และ Frontend รัน 2 replicas เพื่อให้มีสำเนาสำรอง
* Kubernetes จัดการให้ container เหล่านี้รันจาก Docker image ที่เก็บใน Artifact Registry

**4.4.3.2 กระจาย Traffic (Load Balancer และ Cluster IP)**

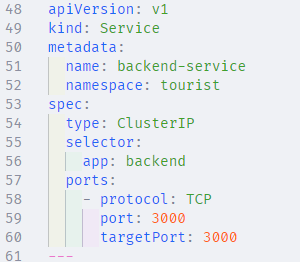
ใช้ Load Balancer และ Cluster IP เพื่อกระจาย traffic ไปยัง container โดย Load Balancer ทำงานในระดับภายนอกเพื่อให้ผู้ใช้เข้าถึงระบบได้ ส่วน Cluster IP ใช้จัดการการสื่อสารภายใน cluster ระหว่าง container

Load Balancer: ถูกกำหนดใน frontend-service เพื่อให้ผู้ใช้เข้าถึง frontend ผ่านพอร์ต 80 จากภายนอก (เช่น porametix.online)



รูป 4.4.7 แสดงตัวอย่างโค้ดส่วนของใช้ Load Balancer ใน “frontend-deployment.yaml”

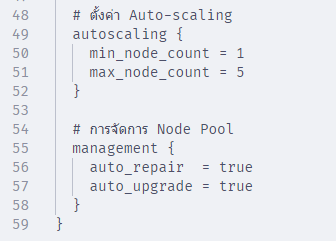
Cluster IP: ถูกใช้ใน backend-service เพื่อให้ container อื่นใน cluster (เช่น frontend) สามารถติดต่อ backend ได้ที่พอร์ต 3000 โดยไม่เปิดออกสู่ภายนอก



รูป 4.4.8 แสดงตัวอย่างโค้ดส่วนของกำหนด Load Balancer ใน “frontend-deployment.yaml”

**4.4.3.3 ตั้งค่า Auto-scaling และความเสถียร**

Terraform กำหนด autoscaling ใน GKE node pool (1-5 nodes) และ Kubernetes จัดการ replicas ปรับจำนวน container และซ่อมแซมอัตโนมัติเมื่อมีปัญหา



รูป 4.4.9 แสดงตัวอย่างโค้ดส่วนของการตั้งค่า Auto-scaling ใน “Terraform”

**4.4.3.4 Deploy อัตโนมัติผ่าน CI/CD**

รันคำสั่ง kubectl apply เพื่อ deploy และอัปเดต container จาก GitHub Actions

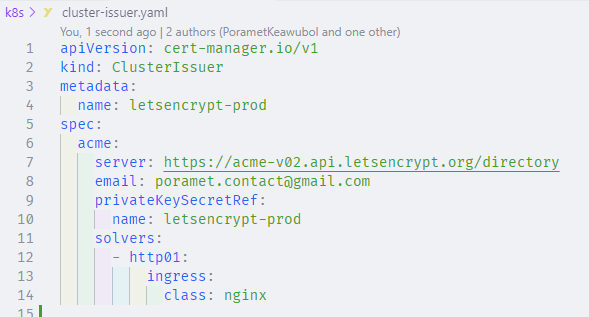


รูป 4.4.10 แสดงตัวอย่างโค้ดส่วนของการ Deploy ใน “GitHub Actions”

*อธิบายเพิ่มเติม:* เมื่อ push โค้ด GitHub Actions deploy container ใหม่ไป Kubernetes และ restart เพื่อใช้เวอร์ชันล่าสุด

**4.4.3.5 จัดการ SSL/TLS**

ใช้ Cert-Manager เพื่อออกใบรับรอง HTTPS อัตโนมัติ



รูป 4.4.11 แสดงตัวอย่างโค้ดส่วนของ Cert-Manager ใน “cluster-issuer.yaml”

*อธิบายเพิ่มเติม:* Kubernetes ทำงานกับ Ingress และ Cert-Manager เพื่อให้ porametix.online มี HTTPS

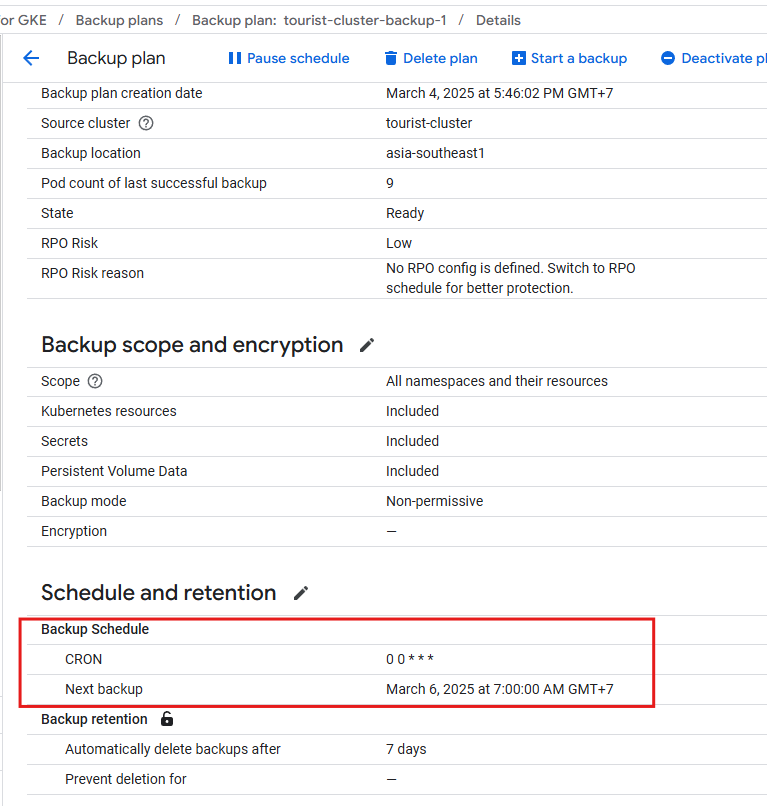
Kubernetes ใช้จัดการ container ของ backend (3 replicas) และ frontend (2 replicas) บน GKE โดยควบคุม deployment, กระจาย traffic ผ่าน Service และ Ingress, ตั้งค่า auto-scaling และความเสถียร, deploy อัตโนมัติผ่าน GitHub Actions, และจัดการ HTTPS ด้วย Cert-Manager ช่วยให้ระบบรองรับผู้ใช้จำนวนมากได้อย่างมีประสิทธิภาพและปลอดภัย"

**4.4.4 Scheduler**

ในส่วนของเทคโนโลยีนี้ได้มีการใช้ร่วมกับเทคโนโลยีอื่นๆ หลากหลายเทคโนโลยีเลย ยกตัวอย่างการงานดังนี้:

**4.4.4.1 ใช้กำหนดเวลา Backup ข้อมูลอัตโนมัติทุกวัน**

จะช่วยให้ ช่วยให้ข้อมูลของ tourist-cluster มีการสำรองอย่างสม่ำเสมอ โดยไม่ต้องสั่ง manual แค่ตั้งเวลาไว้กับระบบอย่างชัดเจน



รูป 4.4.11 แสดงตัวอย่างการตั้งค่า Schedule ให้ Backup อัตโนมัติ

**4.4.4.2 ใช้กำหนดเวลา ในการแจ้งเตือนการแข่งขันอัตโนมัติ** **ก่อนจะมีการเริ่มแข่งขัน**



รูป 4.4.11 แสดงตัวอย่างโค้ดการกำหนดค่าใน Backend เพื่อแจ้งเตือนการแข่งขัน

Schedule ใช้ CRON Job เพื่อสำรองข้อมูลของ tourist-cluster บน GKE ด้วย Google Cloud Backup for GKE โดยตั้งให้รันอัตโนมัติทุกวันเวลา 00:00 UTC (07:00 GMT+7) และกำหนดลบ Backup เกิน 7 วัน เพื่อจัดการพื้นที่และรักษาความสม่ำเสมอของการสำรองข้อมูล และ ระบบการส่งแจ้งเตือนไปยังผู้เข้าร่วมแข่งขันก่อน 30 นาที ยังมีระบบอีกมากมายที่ใช้เทคนิคนี้ซึ่งเป็นเทคนิคที่มีประโยชน์มากมาย

**4.5 Category V**

**4.5.1 Snapshot**

Snapshot ที่สร้างขึ้นทั้ง 3 ตัว ใช้สำหรับ **สำรองข้อมูลของ Kubernetes Node Pool**

**บทบาทของ Snapshot ร่วมกับ VM ในโปรเจคนี้**

**สำรองข้อมูลของ VM ก่อนอัปเดตหรือเปลี่ยนแปลงระบบ**

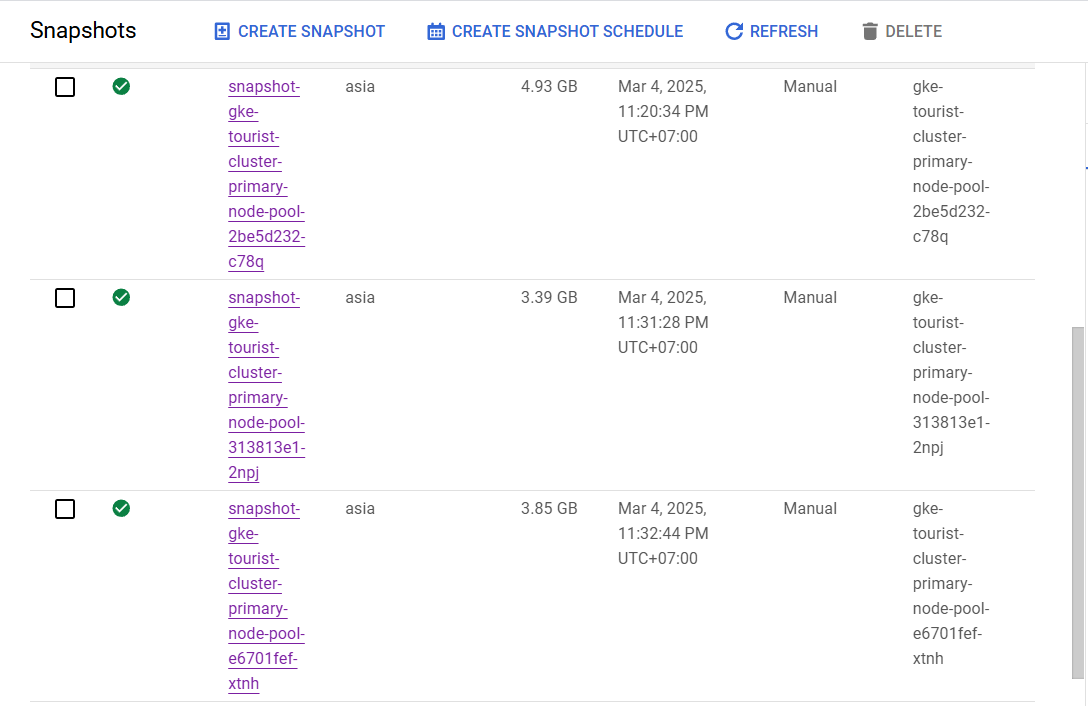
ใช้ Snapshot เพื่อป้องกันการสูญหายของข้อมูลใน VM หากมีการอัปเกรดหรือแก้ไขระบบ

**กู้คืน VM จาก Snapshot ในกรณีฉุกเฉิน**

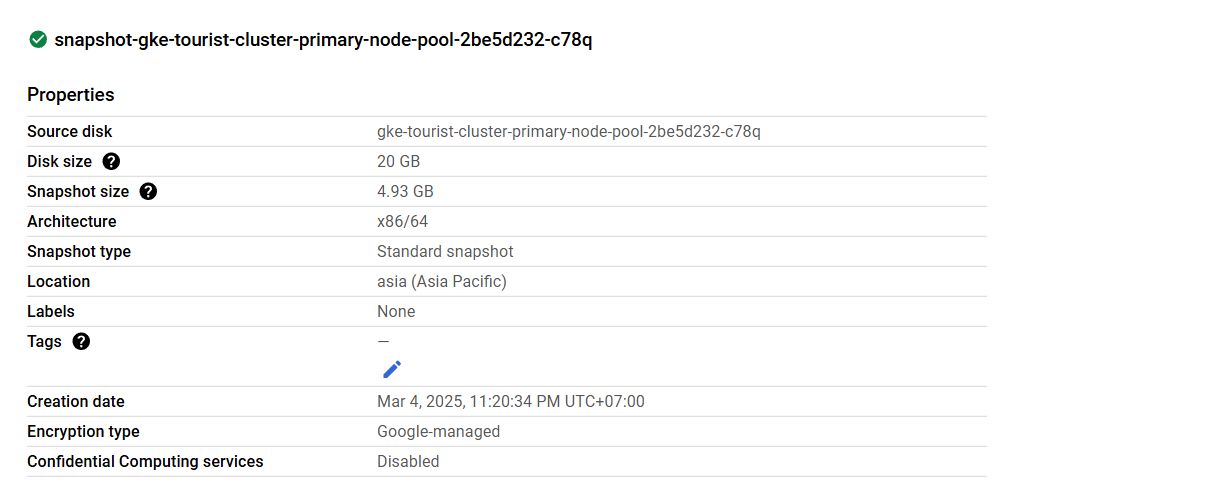
หาก VM มีปัญหา (เช่น ข้อมูลเสียหาย, ระบบล่ม) สามารถใช้ Snapshot เพื่อคืนค่า VM กลับสู่สถานะก่อนหน้า

**สร้าง VM ใหม่จาก Snapshot**

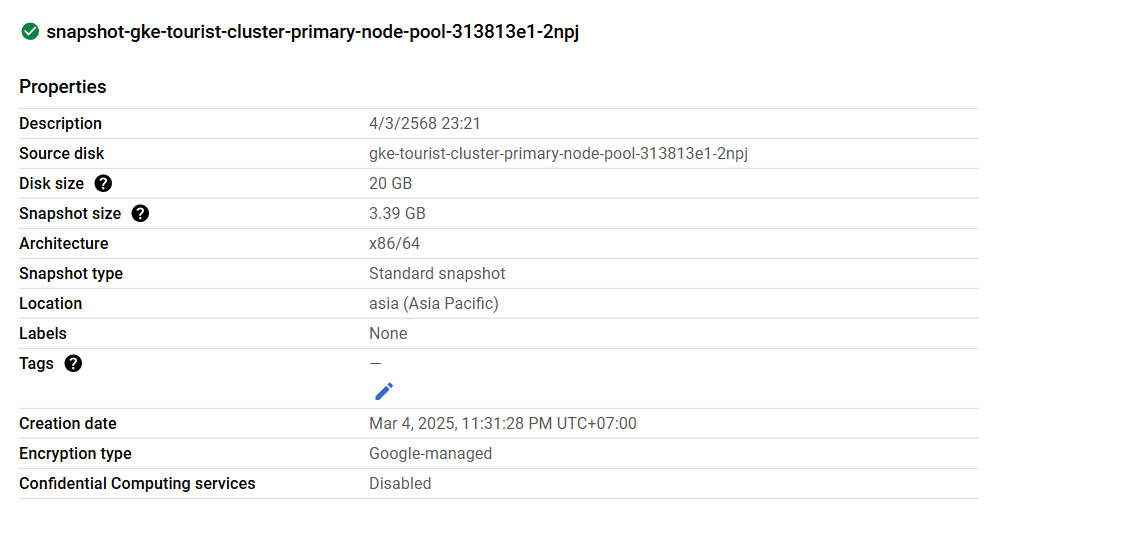
อาจมีการใช้ Snapshot เพื่อ **Deploy VM ใหม่** โดยไม่ต้องตั้งค่าระบบใหม่ทั้งหมด

****

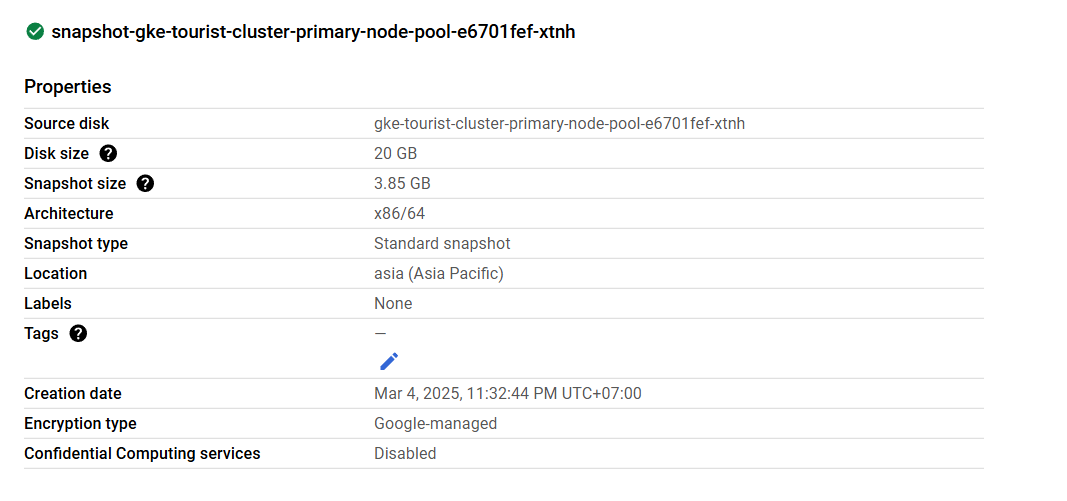
รูป 4.5.1 แสดง Snapshot ทั้ง 3 ตัว



รูป 4.5.2 แสดงรายละเอียด Snapshot ตัวที่หนึ่ง



รูป 4.5.3 แสดงรายละเอียด Snapshot ตัวที่สอง



รูป 4.5.4 แสดงรายละเอียด Snapshot ตัวที่สาม

โดย Snapshot ช่วยให้ระบบ cloud มีความมั่นคง ป้องกันการสูญหายของข้อมูล และรองรับการกู้คืนระบบได้อย่างรวดเร็ว โดยเป็นสิ่งสำคัญในการบริหารจัดการ Kubernetes Cluster (tourist-cluster) ซึ่งสามารถ Restore Cluster หรือสร้าง Disk ใหม่จาก Snapshot ได้ อีกทั้งยังถูกตั้งค่าให้สำรองข้อมูลอัตโนมัติทุกวัน และลบ Snapshot ที่ไม่จำเป็นเพื่อเพิ่มประสิทธิภาพการใช้งานทรัพยากร การใช้ Snapshot เป็นแนวทางที่ช่วยป้องกันปัญหาของระบบ และทำให้มั่นใจว่า Cluster สามารถทำงานได้อย่างราบรื่น

**4.5.2 Backup**

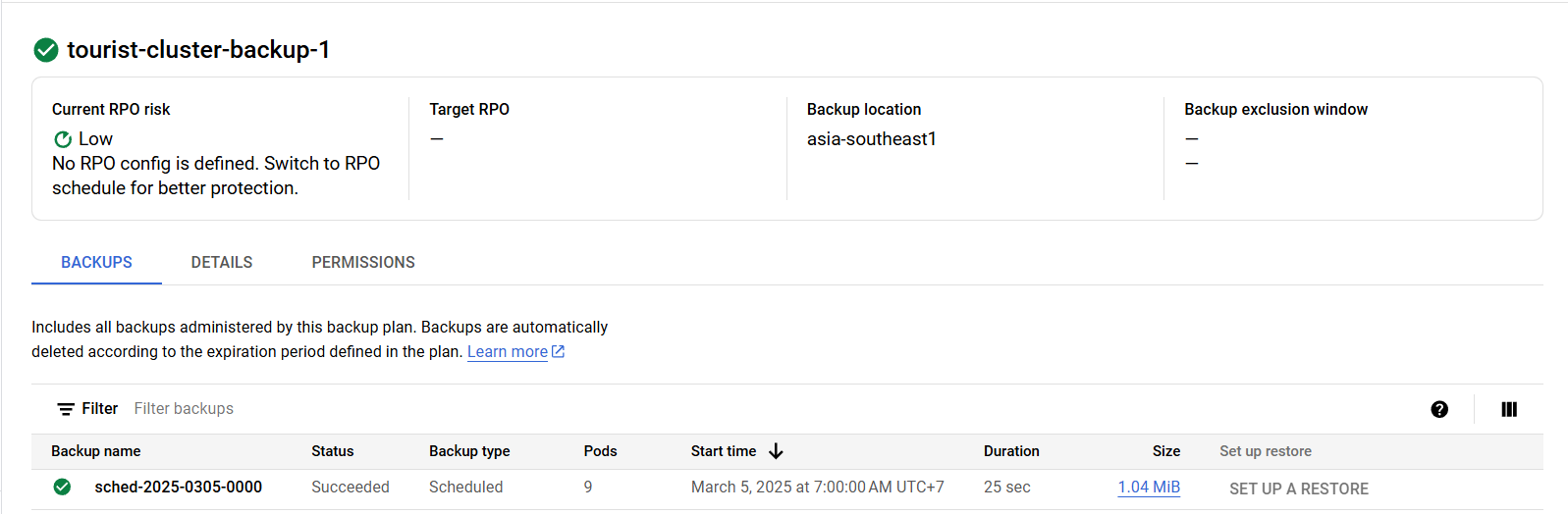
Backup นี้เป็นการสำรองข้อมูลของ Kubernetes Cluster (tourist-cluster) ใน Google Kubernetes Engine (GKE) โดยใช้ Google Cloud Backup for GKE เพื่อให้สามารถกู้คืนข้อมูลของ Pods, Kubernetes Resources, Secrets, และ Persistent Volume Data ได้เมื่อจำเป็น

**Backup นี้สำรองข้อมูล ดังนี้**

* Kubernetes Resources (รวม Deployment, Services, ConfigMaps)
* Secrets (รวม API Keys, Database Credentials)
* Persistent Volume Data (PV) (รวมไฟล์และฐานข้อมูลที่เก็บอยู่ใน PV)

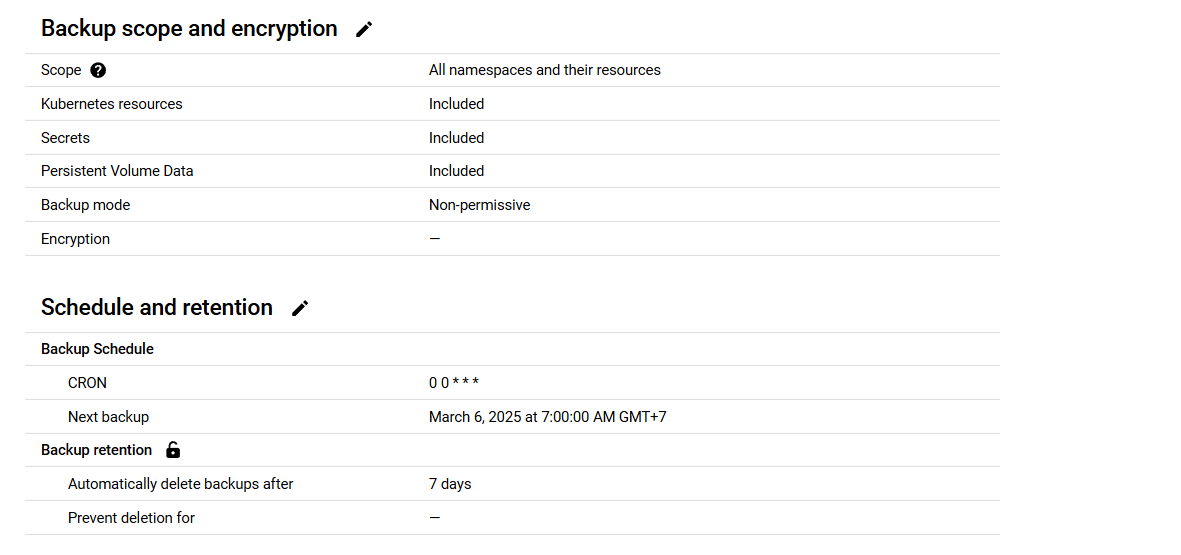
หมายความว่า:

* สามารถกู้คืน (Restore) ข้อมูลทั้งหมดของ Cluster ได้ หากระบบมีปัญหา
* รองรับการกู้คืนแบบเต็มระบบ รวมถึง Pods, Workloads, และ Persistent Data

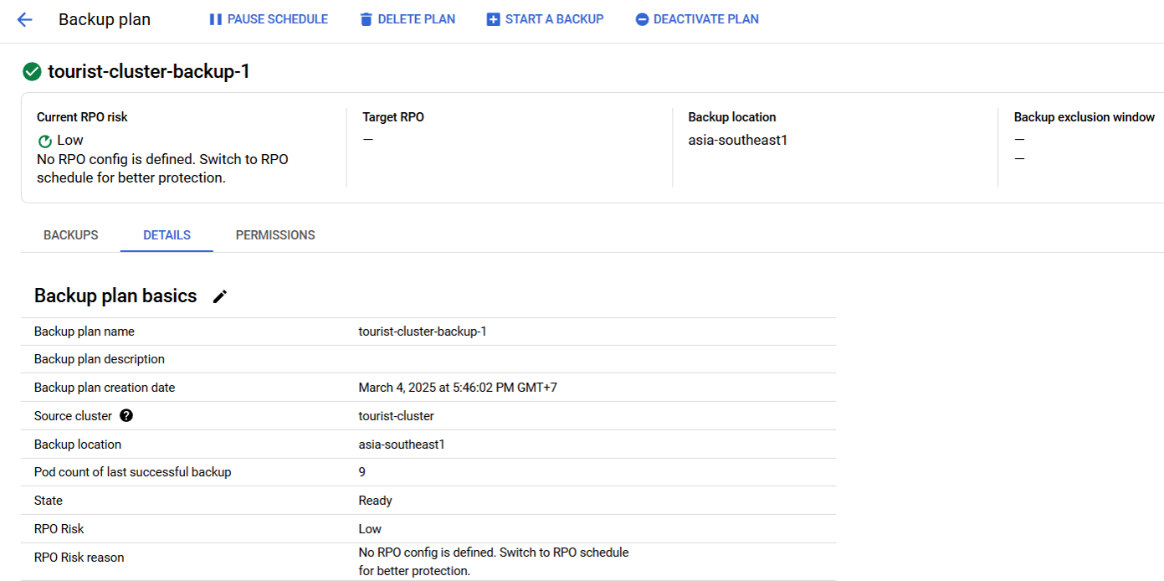


รูป 4.5.5 แสดง tourist-cluster-backup-1 เป็น Backup Plan สำหรับ Kubernetes Cluster (tourist-cluster) ซึ่งมี Backup ชื่อ sched-2025-0305-0000

Schedule and Retention ใช้ CRON Job สำรองข้อมูลทุกวัน 00:00 UTC (07:00 GMT+7) และลบ Backup อัตโนมัติหลัง 7 วันเพื่อประหยัดพื้นที่ โดย Backup ครั้งต่อไปกำหนดวันที่ 6 มี.ค. 2025 เวลา 07:00 GMT+7



รูป 4.5.6 แสดงรายละเอียด Backup scope and encryption และ Schedule and retention

****

รูป 4.5.6 แสดงรายละเอียด Backup plan basics

Backup ใน "Tourist Code" ใช้สำรองข้อมูลของ tourist-cluster บน GKE ด้วย Google Cloud Backup for GKE โดยตั้งสำรองอัตโนมัติทุกวัน 00:00 UTC (07:00 GMT+7) และลบ Backup เกิน 7 วัน ช่วยกู้คืนข้อมูลได้เร็ว ลดความเสี่ยงข้อมูลสูญหาย และรักษาเสถียรภาพของ cluster

**4.5.3 Security**

ในส่วนนี้ทาง Project ได้มีการใช้เทคโนโลยีมากมายเพื่อ ป้องกันความปลอดภัยจากผู้ไม่หวังดี และใช้เทคโนโลยีหลากหลายชนิด ซึ่งจะสามารถช่วยได้ระดับนึงแต่ ก็ยังไม่บางจุดเป็น จุดที่ไม่ปลอดภัย แต่ทางระบบได้มีการป้องกันหลายชนิด ดังนี้:

JWT (JSON Web Token)

* ใช้ยืนยันตัวตนผู้ใช้ และควบคุมการเข้าถึง API
* สร้าง token เมื่อล็อกอิน (ใน AuthService.signIn) และตรวจสอบใน AuthGuard
* ประโยชน์: ป้องกันการเข้าถึง API โดยไม่ได้รับอนุญาติ



รูป 4.5.7 แสดงตัวอย่างโค้ดของระบบการจัดการ Token ในส่วนของ SignIN

Cookies ด้วย HttpOnly และ Secure

* เก็บ token (access, refresh, session) อย่างปลอดภัย
* ตั้ง httpOnly: true ป้องกันการเข้าถึงจาก JavaScript และ secure: true ใน production (ใช้ HTTPS เท่านั้น)
* ประโยชน์: ลดความเสี่ยงจากการโจมตี XSS (Cross-Site Scripting)



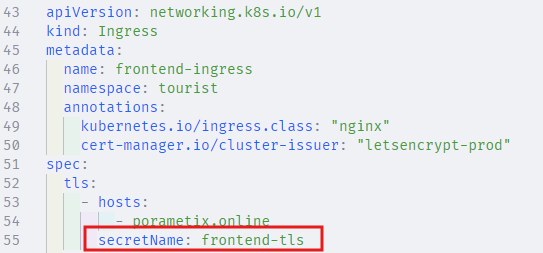
รูป 4.5.8 แสดงตัวอย่างโค้ดของระบบการเก็บ token อย่างปลอดภัย และ ป้องกันการโจมตี CSRF

SameSite Cookie

* ป้องกันการโจมตี CSRF (Cross-Site Request Forgery)
* ตั้ง sameSite: 'none' ใน production หรือ 'lax' ใน local เพื่อควบคุมการส่ง cookie ข้าม domain
* ประโยชน์: ป้องกันการปลอมแปลงคำขอจากเว็บอื่น

HTTPS/TLS (via Cert-Manager)

* เข้ารหัสการสื่อสารระหว่างผู้ใช้และเซิร์ฟเวอร์
* ใช้ Cert-Manager ใน Kubernetes ออกใบรับรอง SSL จาก Let's Encrypt สำหรับ porametix.online
* ประโยชน์: ป้องกันการดักฟังข้อมูล (man-in-the-middle attack)



รูป 4.5.9 แสดงตัวอย่างโค้ดการใช้ Cert-Manager ใน Kubernetes

Secrets ใน Kubernetes

* เก็บข้อมูลลับ เช่น MongoDB connection string และ JWT secret
* ประโยชน์: ป้องกันข้อมูลลับรั่วไหลในโค้ดหรือ config



รูป 4.5.10 แสดงตัวอย่างโค้ดการใช้ ใช้ Secret object เข้ารหัส base64 และดึงผ่าน secretKeyRef

AuthGuard (Authentication Guard)

* ตรวจสอบ token ก่อนอนุญาตให้เข้าถึง API
* ใช้ JwtService ตรวจสอบ access-token ใน cookie
* ถ้าไม่มี token หรือหมดอายุจะ throw UnauthorizedException
* ประโยชน์: ป้องกันผู้ใช้ที่ไม่ล็อกอินเข้าถึง endpoint ที่ต้องการการยืนยันตัวตน



รูป 4.5.11 แสดงตัวอย่างโค้ดระบบเมื่อ token หมดอายุ

SessionGuard

* ใช้ทำอะไร: ตรวจสอบ session token เพื่อยืนยันว่าผู้ใช้ยัง active
* รายละเอียด: ใช้ session-token จาก cookie และ validate ผ่าน SessiontokenService
* โค้ด: SessionGuard.canActivate ถ้า session หมดอายุ throw HttpException
* ประโยชน์: ป้องกันการใช้ session เก่าหรือไม่ถูกต้อง



รูป 4.5.12 แสดงตัวอย่างโค้ดระบบเมื่อ session หมดอายุ

Refresh Token

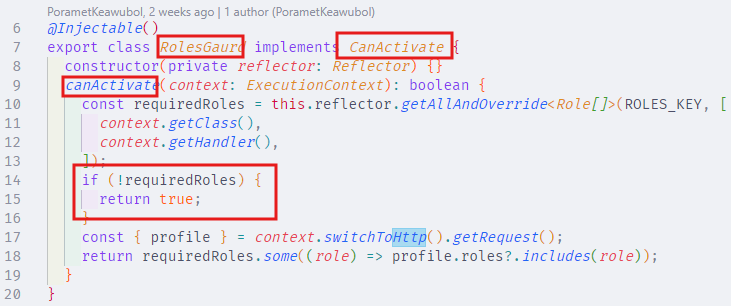
* ออก access token ใหม่เมื่อหมดอายุ โดยไม่ต้องล็อกอินซ้ำ
* ใช้ refresh-token เก็บใน cookie และตรวจสอบในAuthService.verifyAndGenerateNewToken
* POST /auth/refresh สร้าง token ใหม่ถ้า refresh token ยังไม่หมดอายุ
* เพิ่มความสะดวกและรักษาความปลอดภัยโดยจำกัดอายุ access token



รูป 4.5.13 แสดงตัวอย่างโค้ดระบบการออก access token ใหม่เมื่อหมดอายุ

Roles Guard

* ควบคุมการเข้าถึงตามบทบาท (role) เช่น admin, user
* รายละเอียด: ใช้ RolesGaurd ตรวจสอบว่า user มี role ที่กำหนดหรือไม่
* canActivate ตรวจ profile.roles กับ requiredRoles
* ประโยชน์: ป้องกันผู้ใช้ที่ไม่มีสิทธิ์เข้าถึงฟีเจอร์บางอย่าง



รูป 4.5.14 แสดงตัวอย่างโค้ดระบบการตรวจ Role เพื่อป้องการการเข้าถึง

Password Hashing

* เข้ารหัส password ก่อนเก็บในฐานข้อมูล
* ใช้ใน UsersService (เช่น authenticate method) เพื่อเปรียบเทียบ password\
* ประโยชน์: ป้องกันการรั่วไหลของ password แบบ plain text



รูป 4.5.14 แสดงตัวอย่างฐานข้อมูลของ User

**4.5.3.1 DNS Proxy**

DNS Records ใช้จัดการการตั้งค่า DNS สำหรับโดเมน porametix.online เพื่อให้สามารถเข้าถึงเว็บไซต์และบริการต่างๆ ได้อย่างถูกต้อง Cloudflare ทำหน้าที่เป็น authoritative DNS provider โดยกำหนด nameservers (dora.ns.cloudflare.com และ lakas.ns.cloudflare.com)  
การใช้ Proxy status ช่วยซ่อน IP จริงของเซิร์ฟเวอร์จากผู้โจมตี และเพิ่มชั้นความปลอดภัยผ่าน Cloudflare

รูปภาพประกอบด้วย ข้อความ, ตัวอักษร, จำนวน, ไลน์

เนื้อหาที่สร้างโดย AI อาจไม่ถูกต้อง

รูป 4.5.15 แสดงตาราง DNS Records

**4.5.3.2 SSL/TLS**  
 ส่วน SSL/TLS ใช้เพื่อเข้ารหัสการสื่อสารระหว่างผู้ใช้และเซิร์ฟเวอร์ของ porametix.online โดยใช้ใบรับรอง SSL จาก Cloudflare เพื่อป้องกันการดักฟังข้อมูล (man-in-the-middle attacks)

- Edge Certificates: Cloudflare จัดหา Universal SSL certificate ให้อัตโนมัติสำหรับ porametix.online และ wildcard (\*.porametix.online) ซึ่งครอบคลุมทุก subdomain ใบรับรองนี้มีวันหมดอายุ 3 มิถุนายน 2025 และมีการสำรอง (Backup) เพื่อความต่อเนื่อง  
การใช้ SSL/TLS ช่วยให้ข้อมูล เช่น รหัสผ่านใน SignUpForm หรือข้อมูลการ submit โจทย์ ถูกส่งอย่างปลอดภัย

รูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร, ซอฟต์แวร์

เนื้อหาที่สร้างโดย AI อาจไม่ถูกต้อง

รูป 4.5.16 แสดงตาราง Edge Certificates

**4.5.4 DDoS Protection**

DDoS Protection ใช้ Cloudflare เพื่อป้องกันการโจมตีแบบ Distributed Denial of Service (DDoS) ที่อาจทำให้เว็บไซต์หยุด โดยเฉพาะในช่วงที่มีการแข่งขันที่มีผู้ใช้จำนวนมากหรือมีการโจมตี

* HTTP DDoS Protection: เปิดใช้งานอัตโนมัติโดย Cloudflare เพื่อตรวจจับและลดการโจมตี HTTP-based DDoS เช่น HTTP floods โดยใช้ ruleset ที่กำหนดค่าเริ่มต้นเป็น "Challenge (CAPTCHA)"
* การทำงาน: Cloudflare วิเคราะห์ traffic และสร้าง real-time signatures เพื่อกรองการโจมตี โดยไม่กระทบต่อผู้ใช้จริง  
  ส่วนนี้สำคัญสำหรับเว็บ tourist ที่อาจถูกโจมตีเพื่อขัดขวางการแข่งขัน

รูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร, ซอฟต์แวร์

เนื้อหาที่สร้างโดย AI อาจไม่ถูกต้อง

รูป 4.5.17 แสดงการตั้งค่า HTTP DDoS Protection

**4.5.5 Web Application Firewall (WAF)**  
 Web Application Firewall (WAF) ใช้ Cloudflare เพื่อตรวจจับและป้องกันการร้องขอที่เป็นอันตราย (malicious requests) ที่อาจโจมตีเว็บไซต์ เช่น การพยายามแทรก SQL หรือ XSS

- Custom Rules: มีการกำหนดกฎ "Check human" ที่ใช้ "Challenge (CAPTCHA)" เพื่อกรอง traffic ตาม Threat Score

รูปภาพประกอบด้วย ข้อความ, ตัวอักษร, ไลน์, จำนวน

เนื้อหาที่สร้างโดย AI อาจไม่ถูกต้อง

รูป 4.5.18 แสดงตาราง Custom Rules

**4.5.6 DNSSEC**  
DNSSEC (DNS Security Extensions) ใช้เพื่อเพิ่มความปลอดภัยให้ DNS โดยใช้ cryptographic signatures เพื่อป้องกันการปลอมแปลง DNS records (DNS spoofing)เปิดเพื่อป้องกันการโจมตีแบบ man-in-the-middle

- การทำงาน: Cloudflare สามารถเปิดใช้งาน DNSSEC ได้ในหน้า DNS Settings เพื่อให้ DNS records ของ porametix.online ถูกเซ็นต์ด้วยลายเซ็นดิจิทัล ช่วยให้ผู้ใช้มั่นใจว่าเข้าถึงเซิร์ฟเวอร์ที่ถูกต้อง  
DNSSEC มีความสำคัญสำหรับเว็บที่ต้องการความน่าเชื่อถือ การใช้งานเว็บต้องมั่นใจว่า traffic ไปถึงเซิร์ฟเวอร์จริง

รูปภาพประกอบด้วย ข้อความ, ตัวอักษร, ภาพหน้าจอ

เนื้อหาที่สร้างโดย AI อาจไม่ถูกต้อง

รูป 4.5.19 แสดงการตั้งค่า DNSSEC

User -> Cloudflare (DNSSEC,DNS Proxy, SSL, DDoS, WAF) -> Origin Server

**4.6 Category VI**

**4.6.1 Multiusers**

โดยเว็บไซต์ ถูกออกแบบให้รองรับผู้ใช้หลายคน (Multiusers) เข้าใช้งานพร้อมกันได้อย่างมีประสิทธิภาพ โดยระบบจัดการการยืนยันตัวตน (authentication) และการทำงานแบบ asynchronous เพื่อให้ผู้ใช้แต่ละคนมีประสบการณ์ที่ราบรื่น โดยมีระบบมากมายดังนี้:

**4.6.1.1การจัดการ Multiusers**

ระบบใช้ JWT และ Session Token เพื่อแยกผู้ใช้แต่ละคน และรันบน Kubernetes ซึ่งสามารถ scale container เพื่อรองรับผู้ใช้จำนวนมากได้

โค้ดที่เกี่ยวข้อง (AuthService.signIn)



รูป 4.6.1 แสดงระบบใช้ JWT และ Session Token เพื่อแยกผู้ใช้แต่ละคน

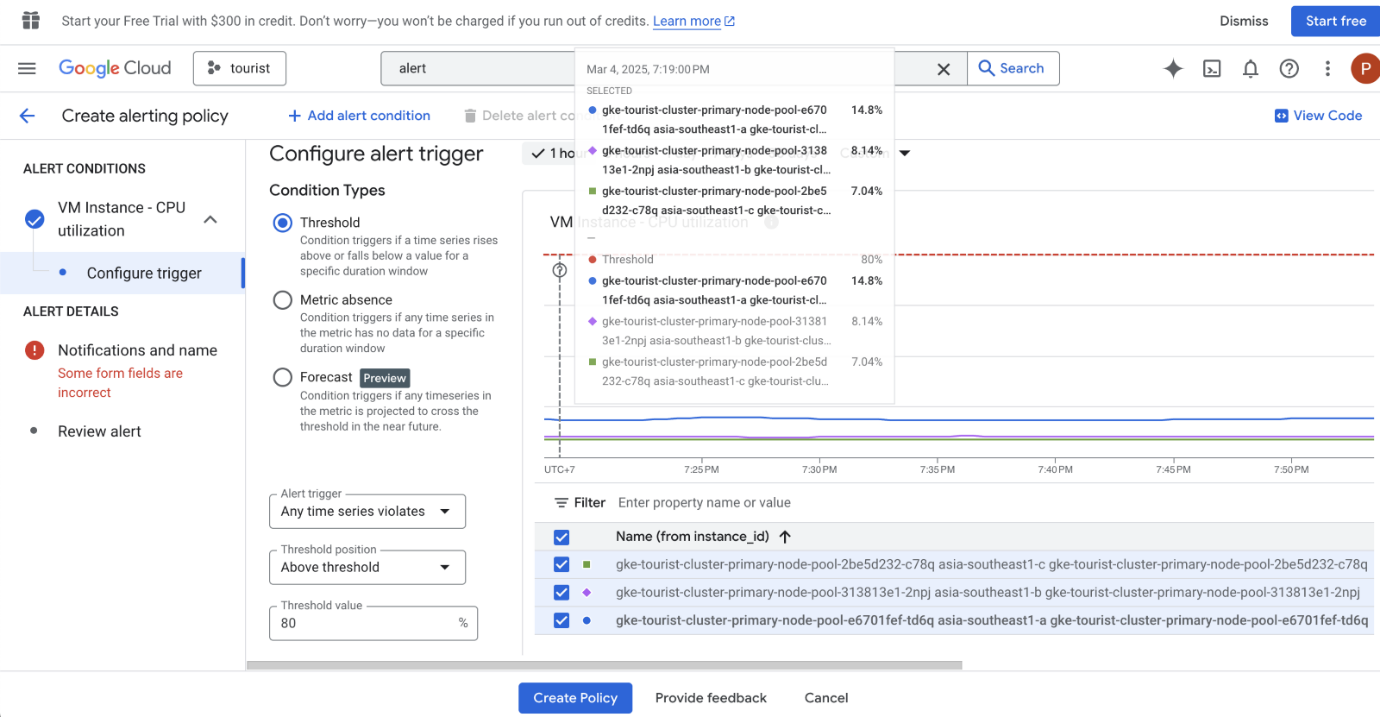
*อธิบายเพิ่มเติม:* เมื่อผู้ใช้ล็อกอิน ระบบสร้าง JWT และ session token เฉพาะสำหรับผู้ใช้แต่ละคน (ใช้ user.\_id) เพื่อแยกการทำงานของผู้ใช้แต่ละคน และรองรับการล็อกอินพร้อมกันจากหลายคน

**5. เทคนิคอื่นๆที่ใช้ใน Project**

**5.1 Monitoring**

**5.1.1 Google Cloud Monitoring**

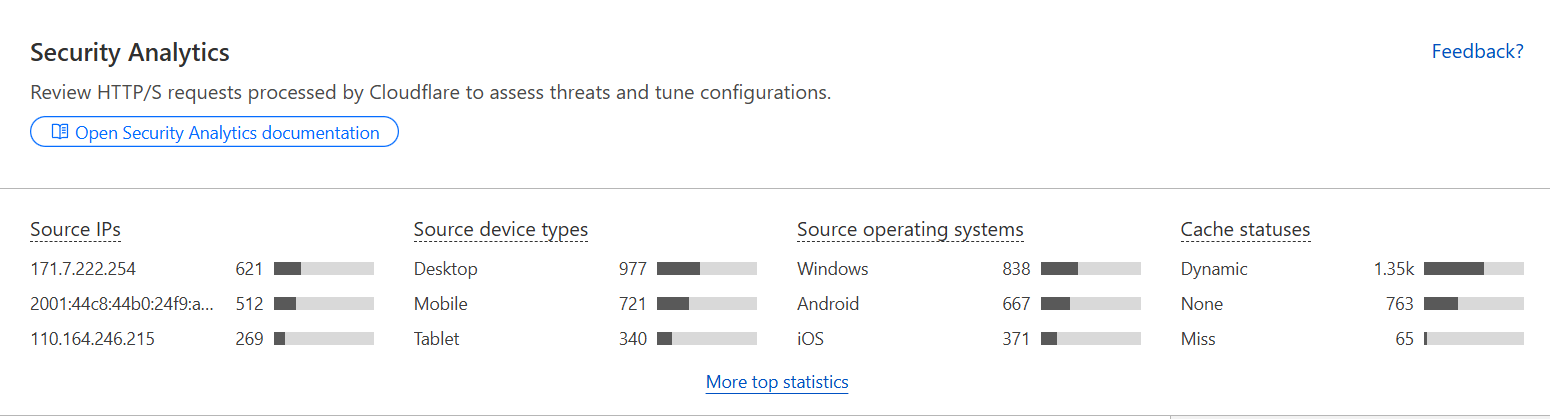
ถูกใช้เพื่อติดตามและวิเคราะห์ประสิทธิภาพของระบบที่รันบน Google Kubernetes Engine (GKE) โดยใช้ Google Cloud Observability เพื่อให้มั่นใจว่าระบบทำงานได้ดีและแก้ปัญหาได้ทันท่วงที

****

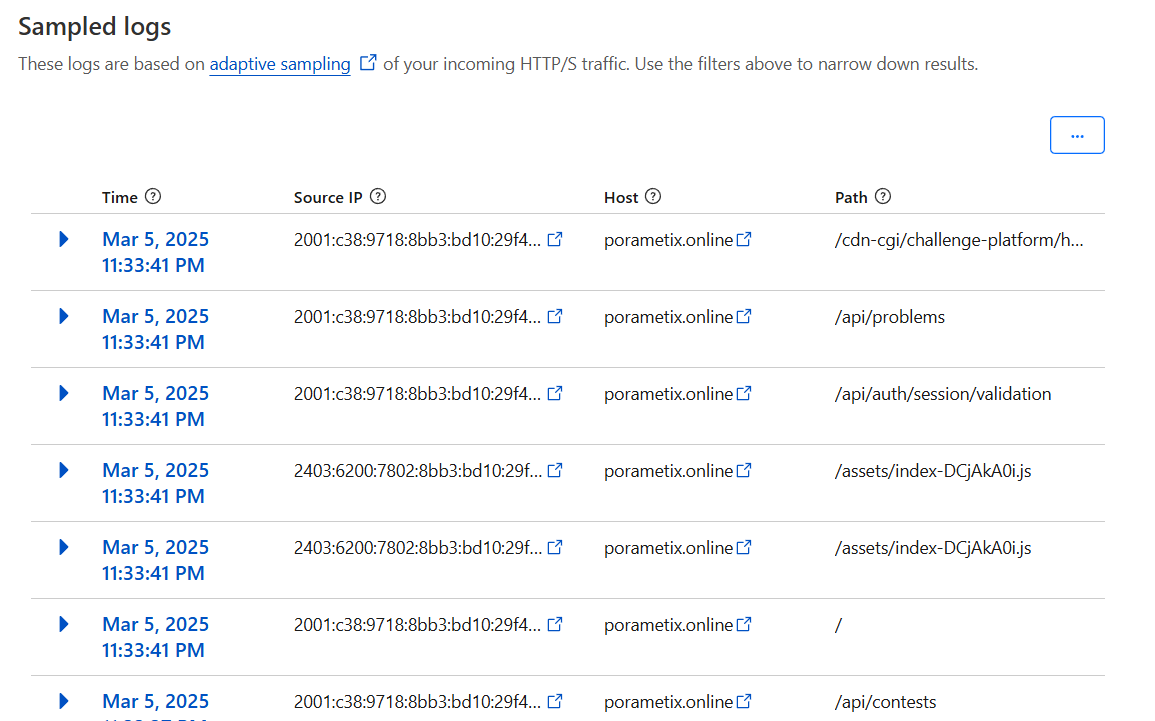
รูป 5.1 แสดงการใช้ระบบตรวจสอบการทำงานของ CPU

*อธิบายเพิ่มเติม:* เป็นการตรวจสอบระยะยาวเมื่อ มี CPU ของ VM ตัวไหนใช้ความสามารถของ CPU เกิน 80% เป็นเวลา 5 นาที จะมีการแจ้งเตือนไปที่ EMAIl ของคนในกลุ่มตามที่ได้ตั้งค่าเอาไว้

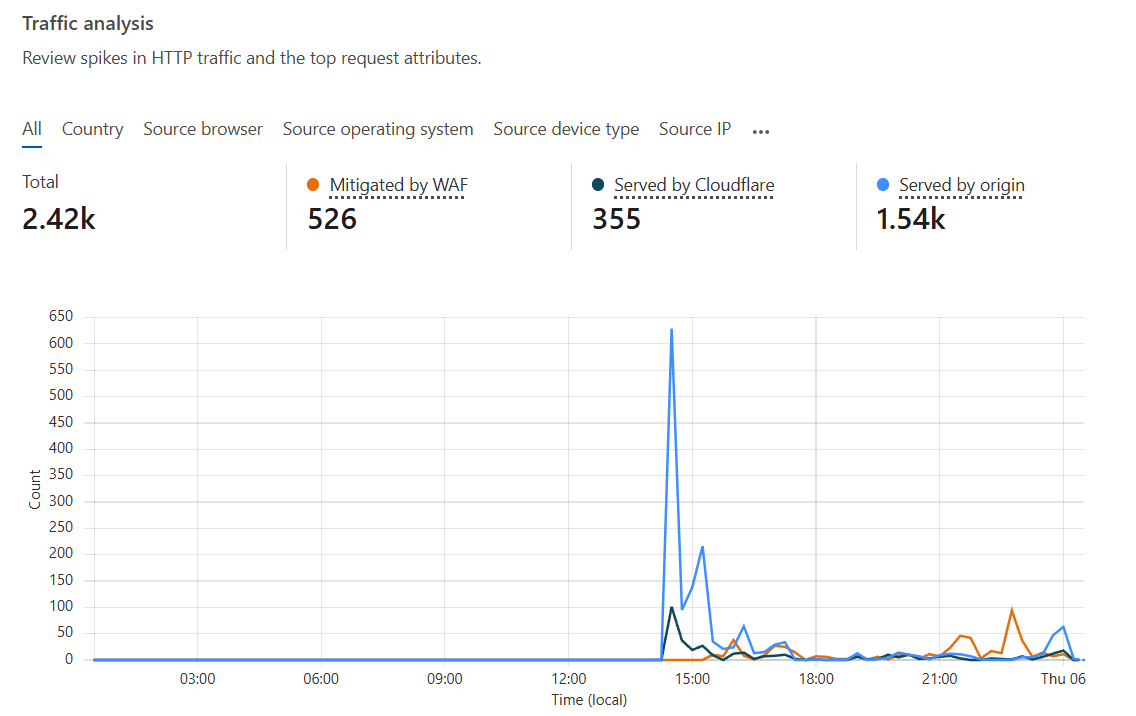
**5.1.2 Web Application Availability | Monitoring & Metrics ของ Cloudflare**



รูป 5.2 แสดงอุปกรณ์ส่วนใหญ่ของบุคคลเข้ามาในเว็บไซต์



รูป 5.2 แสดง IP ของแต่ละบุคคลและแสดง Path ที่เข้าถึง

  
 รูป 5.3 แสดง Traffic analysis ของคนที่เชื่อมต่อเข้ามาในเว็บไซต์