Architectural Drivers Document: Food Delivery System

ส่วนที่ 1: System Overview

1. System Description

ระบบ **Food Delivery System** (ในที่นี้อ้างอิงโมเดล GrabFood) เป็นแพลตฟอร์มกลางที่ทำหน้าที่เชื่อมโยงระหว่าง **ลูกค้า (Customer)**, **ร้านอาหาร (Restaurant)** และ **ผู้ขับขี่ (Rider)** เข้าด้วยกัน เพื่ออำนวยความสะดวกในการสั่งอาหารและจัดส่งอาหารแบบ On-demand โดยระบบจะจัดการตั้งแต่การค้นหาร้าน, การสั่งอาหาร, การชำระเงิน, การจับคู่ไรเดอร์, ไปจนถึงการติดตามสถานะการจัดส่งจนถึงมือลูกค้า

เป้าหมายหลักของระบบคือการสร้าง Marketplace ที่รวดเร็วและมีประสิทธิภาพ ช่วยให้ร้านอาหารขยายฐานลูกค้า ช่วยให้ไรเดอร์มีรายได้เสริม และช่วยให้ลูกค้าได้รับประทานอาหารที่ต้องการโดยไม่ต้องเดินทาง โดยระบบต้องรองรับผู้ใช้งานจำนวนมากพร้อมกัน (High Concurrency) โดยเฉพาะในช่วงเวลาเร่งด่วน (Lunch/Dinner time) และต้องมีความแม่นยำในเรื่องพิกัดและข้อมูลธุรกรรมการเงิน

2. Target Users

- **Consumer (ลูกค้า):** ผู้ใช้งานที่ต้องการสั่งอาหารผ่านแอปพลิเคชัน
- **Merchant (ร้านอาหาร):** เจ้าของร้านค้าหรือพนักงานที่รับออเดอร์และเตรียมอาหาร
- **Rider (ไรเดอร์):** พาร์ทเนอร์ผู้ขับขี่ที่รับงานจัดส่งอาหารจากร้านไปหาลูกค้า
- **Admin (ผู้ดูแลระบบ):** เจ้าหน้าที่ที่คอยจัดการปัญหา (Support), ตรวจสอบการสมัคร, และดูภาพรวมระบบ

3. Key Features (Top 10)

1. **Search & Discovery:** ค้นหาร้านอาหารตามพิกัด GPS, ประเภทอาหาร, และเรตติ้ง
2. **Order Placement:** ระบบตระกร้าสินค้า, การเลือกตัวเลือกอาหาร (Toppings), และยืนยันคำสั่งซื้อ
3. **Real-time Tracking:** แสดงตำแหน่งของไรเดอร์และสถานะออเดอร์แบบเรียลไทม์บนแผนที่
4. **Driver Matching:** อัลกอริทึมจับคู่ไรเดอร์ที่ใกล้ที่สุดและเหมาะสมที่สุดกับออเดอร์
5. **Digital Payment:** รองรับการตัดเงินผ่าน Credit Card, E-Wallet และ Mobile Banking
6. **Restaurant Management:** ระบบรับออเดอร์, จัดการเมนู, เปิด/ปิดร้านสำหรับฝั่งร้านค้า
7. **Promotion & Discount:** ระบบจัดการคูปองส่วนลดและการคำนวณราคาสุทธิ
8. **Review & Rating:** ให้คะแนนร้านค้าและไรเดอร์หลังจบงาน
9. **Chat System:** ระบบแชทระหว่างลูกค้า-ไรเดอร์ หรือ ลูกค้า-Support ภายในแอป
10. **Notification:** แจ้งเตือนสถานะต่างๆ (รับออเดอร์, อาหารเสร็จ, ไรเดอร์ใกล้ถึง) ผ่าน Push Notification

ส่วนที่ 2: Functional Requirements

User Management & Profile

- **FR-01:** ผู้ใช้ (Customer, Rider, Merchant) ต้องสามารถลงทะเบียนและเข้าสู่ระบบผ่านเบอร์โทรศัพท์ (OTP) หรือ Social Login ได้
- **FR-02:** ผู้ใช้สามารถจัดการข้อมูลส่วนตัว เช่น ที่อยู่จัดส่ง, เบอร์โทรติดต่อ, และวิธีการชำระเงินได้
- **FR-03:** Merchant ต้องสามารถอัปโหลดเอกสารยืนยันตัวตนร้านค้าและตั้งค่าเวลาเปิด-ปิดร้านได้

Ordering & Payment

- **FR-04:** Customer สามารถค้นหาร้านอาหารภายในรัศมีที่กำหนดและ Filter ตามหมวดหมู่ได้
- **FR-05:** Customer สามารถเพิ่มรายการอาหารลงตะกร้า ระบุรายละเอียดพิเศษ (เช่น ไม่ใส่ผัก) และกดสั่งซื้อได้
- **FR-06:** ระบบต้องสามารถคำนวณค่าอาหาร ค่าจัดส่ง (ตามระยะทาง) และหักส่วนลดได้ถูกต้อง
- **FR-07:** ระบบต้องตัดเงินผ่าน Payment Gateway ได้สำเร็จก่อนส่งออเดอร์ไปที่ร้าน (กรณี Cashless)

Order Fulfillment & Tracking

- **FR-08:** Merchant จะต้องได้รับแจ้งเตือนเมื่อมีออเดอร์ใหม่และสามารถกดรับ (Accept) หรือปฏิเสธ (Reject) ได้
- **FR-09:** ระบบต้องทำการค้นหาและมอบหมายงานให้ Rider ที่เหมาะสมที่สุดภายในเวลาที่กำหนด
- **FR-10:** Rider สามารถกดรับงาน, ดูเส้นทางไปร้านค้า, และเส้นทางไปบ้านลูกค้าผ่าน Map API ได้
- **FR-11:** Customer ต้องเห็นสถานะออเดอร์ (Preparing, Picked up, Arriving) และตำแหน่ง Rider บนแผนที่แบบ Real-time

Notification & Support

- **FR-12:** ระบบต้องส่ง Push Notification ไปยังผู้ที่เกี่ยวข้องทุกครั้งที่สถานะออเดอร์เปลี่ยน
- **FR-13:** Customer และ Rider ต้องสามารถส่งข้อความแชทหากันได้เมื่อออเดอร์ถูกรับแล้ว
- **FR-14:** Customer สามารถกดรายงานปัญหา (Report Issue) ของออเดอร์เพื่อให้ Admin ตรวจสอบได้
- **FR-15:** ระบบต้องบันทึกประวัติการสั่งซื้อ (Order History) ให้ลูกค้าดูย้อนหลังได้

ส่วนที่ 3: Quality Attributes & Scenarios

QA-1: Performance (Latency)

Scenario: ลูกค้ากดค้นหาร้านอาหารในช่วงพักเที่ยง (Peak Hour)

- **Source:** Customer
- **Stimulus:** กดค้นหาร้าน "ก๋วยเตี๋ยว"
- **Artifact:** Search Service / Database
- **Environment:** Normal operation during peak hours (12:00 PM)
- **Response:** ระบบแสดงรายชื่อร้านก๋วยเตี๋ยวที่ใกล้ที่สุดพร้อมรูปภาพและเรตติ้ง
- **Response Measure:** แสดงผลลัพธ์ภายใน **< 2 วินาที** (95th percentile)

QA-2: Scalability (Throughput)

Scenario: มีโปรโมชั่นใหญ่ (Mega Sale) ทำให้ยอดสั่งซื้อพุ่งสูงขึ้น

- **Source:** Customers จำนวนมาก
- **Stimulus:** ทำรายการสั่งซื้อพร้อมกัน 10,000 requests/วินาที
- **Artifact:** Order Processing Service
- **Environment:** Peak Load event
- **Response:** ระบบสามารถรับออเดอร์ได้โดยไม่ล่ม (Scale out อัตโนมัติ)
- **Response Measure:** ระบบรองรับ Concurrent Users ได้ **20,000 คน** โดย Error rate < 0.1%

QA-3: Availability

Scenario: เครื่อง Server หลักตัวหนึ่งในโซน Database เกิดพังลง (Hardware Failure)

- **Source:** Internal Hardware Failure
- **Stimulus:** Database node crashes
- **Artifact:** Database Cluster
- **Environment:** 24/7 Operations
- **Response:** ระบบสลับไปใช้ Database node สำรองอัตโนมัติ (Failover)
- **Response Measure:** ระบบกลับมาใช้งานได้ภายใน **< 1 นาที** (Downtime < 1 min)

QA-4: Security (Data Privacy)

Scenario: Hacker พยายามดักจับข้อมูลบัตรเครดิตระหว่างการส่งข้อมูล

- **Source:** Malicious User (Hacker)
- **Stimulus:** Sniffing network traffic
- **Artifact:** Communication Channel / Payment Service
- **Environment:** Public Internet
- **Response:** ข้อมูลถูกเข้ารหัส (Encrypted) ทำให้อ่านไม่รู้เรื่อง
- **Response Measure:** ข้อมูลทั้งหมดต้องถูกเข้ารหัสด้วย **TLS 1.2+** และไม่มีข้อมูลบัตรหลุดออกไป

QA-5: Usability

Scenario: ไรเดอร์มือใหม่ใช้งานแอปครั้งแรกเพื่อรับงาน

- **Source:** New Rider
- **Stimulus:** พยายามกดรับงานและดูแผนที่นำทาง
- **Artifact:** Driver App UI
- **Environment:** On the road (Outdoors)
- **Response:** ไรเดอร์สามารถกดรับงานและเปิดแผนที่ได้โดยไม่ต้องถาม Support
- **Response Measure:** ไรเดอร์ทำงานสำเร็จได้ใน **< 3 คลิก** และใช้เวลารวม **< 30 วินาที**

QA-6: Modifiability

Scenario: Developer ต้องการเพิ่มช่องทางการจ่ายเงินใหม่ (เช่น Wallet เจ้าใหม่)

- **Source:** Developer
- **Stimulus:** Implement New Payment Gateway
- **Artifact:** Payment Service Codebase
- **Environment:** Development Phase
- **Response:** สามารถเพิ่ม Module ใหม่โดยไม่กระทบโค้ดเดิม (Loose Coupling)
- **Response Measure:** ใช้เวลา Implement และ Test จนเสร็จภายใน **3 วันทำการ**

ส่วนที่ 4: Constraints

Technical Constraints

- **TC-01:** แอปพลิเคชันฝั่ง Client ต้องรองรับทั้ง **iOS** และ **Android**
- **TC-02:** Backend ต้องพัฒนาด้วย **Microservices Architecture** เพื่อรองรับการขยายตัว (ตามข้อกำหนดวิชาเรียน/Project Requirement)
- **TC-03:** ต้องใช้ **Google Maps API** สำหรับบริการแผนที่ (เนื่องจากเป็นมาตรฐาน)

Time Constraints

- **TiC-01:** ต้องส่งตัวต้นแบบ (MVP) ภายใน **สิ้นภาคการศึกษา** (ประมาณ 14 สัปดาห์)
- **TiC-02:** ต้องนำเสนอ Architectural Design ภายใน **สัปดาห์ที่ 6**

Budget Constraints

- **BC-01:** งบประมาณสำหรับ Cloud Server (AWS/Google Cloud) ต้องไม่เกิน **$100 ต่อเดือน** (สำหรับเฟส Development)
- **BC-02:** ต้องใช้ Open Source Software เป็นหลักเพื่อลดค่าใช้จ่าย License

Legal/Policy Constraints

- **LC-01:** ต้องปฏิบัติตาม **PDPA (พ.ร.บ. คุ้มครองข้อมูลส่วนบุคคล)** อย่างเคร่งครัดในการเก็บข้อมูลลูกค้า
- **LC-02:** การเก็บข้อมูลธุรกรรมการเงินต้องสอดคล้องกับมาตรฐานความปลอดภัยพื้นฐาน

ส่วนที่ 5: Assumptions

1. **Internet Connectivity:** สมมติว่าผู้ใช้งาน (ลูกค้า, ร้าน, ไรเดอร์) มีอินเทอร์เน็ต 4G/5G ใช้งานตลอดเวลาที่มีความเสถียรพอสมควร
2. **Map Accuracy:** สมมติว่าพิกัด GPS จาก Google Maps API มีความแม่นยำเพียงพอสำหรับการระบุตำแหน่งร้านและบ้านลูกค้า
3. **Payment API Reliability:** สมมติว่าระบบ Payment Gateway ของธนาคาร (Third-party) มีความเสถียร (Uptime 99.9%)
4. **Rider Availability:** สมมติว่ามีจำนวนไรเดอร์เพียงพอต่อความต้องการในพื้นที่ให้บริการ (ไม่ขาดแคลนไรเดอร์)
5. **Smartphone Capability:** สมมติว่าผู้ใช้งานทุกคนใช้สมาร์ทโฟนรุ่นที่ไม่เก่าเกิน 5 ปี ซึ่งรองรับฟีเจอร์แผนที่และ GPS ได้ดี

ส่วนที่ 6: Priority & Trade-offs

Quality Attributes Priority

Rank	Quality Attribute	เหตุผล
1	**Performance**	ระบบสั่งอาหารต้องเร็ว ลูกค้าหิวรอไม่ได้ ถ้าช้าลูกค้าจะเปลี่ยนไปใช้แอปคู่แข่งทันที
2	**Availability**	ระบบต้องพร้อมใช้งานตลอดเวลา โดยเฉพาะช่วงเที่ยง/เย็น ถ้าระบบล่ม = เสียรายได้มหาศาล
3	**Scalability**	ต้องรองรับคนจำนวนมากได้ในช่วง Peak ถ้า Scale ไม่ได้ ระบบจะช้าและล่มตามมา
4	**Security**	มีเรื่องเงินและข้อมูลส่วนตัว แต่ถ้าเทียบกับ Availability แล้ว ลูกค้ายอมรับความเสี่ยงได้บ้างแลกกับความสะดวก (แต่ยังไงก็สำคัญ)
5	**Usability**	แอปต้องใช้ง่าย แต่ถ้าลูกค้าหิวมาก UI ไม่สวยนิดหน่อยเขาก็พยายามกดสั่งได้
6	**Modifiability**	เป็นเรื่องภายใน ทีม Dev อาจจะยอมลำบากหน่อยเพื่อให้ระบบเร็วและเสถียรสำหรับลูกค้า

Trade-offs Analysis

Trade-off #1: Performance vs. Security

- **คำอธิบาย:** การเพิ่มมาตรการความปลอดภัยเข้มข้น เช่น การเข้ารหัสข้อมูลทุกขั้นตอน (End-to-End Encryption) หรือการตรวจสอบสิทธิ์หลายชั้น (MFA) ทุกครั้ง อาจทำให้ระบบทำงานช้าลง (Latency สูงขึ้น) และผู้ใช้รู้สึกลำบาก
- **Decision:** เลือกเน้น **Performance** ใน Flow ทั่วไป (เช่น การค้นหาร้าน, ดูเมนู) เพื่อให้ผู้ใช้รู้สึกลื่นไหล แต่จะบังคับใช้ **Security** เข้มงวดเฉพาะจุดที่เสี่ยงจริงๆ เช่น ตอนจ่ายเงิน หรือเปลี่ยนรหัสผ่าน เพื่อรักษาสมดุล

Trade-off #2: Availability vs. Consistency (CAP Theorem)

- **คำอธิบาย:** ในระบบกระจายศูนย์ (Distributed System) เราต้องเลือกระหว่างความพร้อมใช้งาน (Availability) หรือ ข้อมูลที่ตรงกันเป๊ะๆ ทุกจุด (Consistency) หากเลือก Consistency เวลา Node ใด Node หนึ่งตาย ระบบอาจต้องหยุดให้บริการชั่วคราวเพื่อรอ Sync ข้อมูล
- **Decision:** เลือกเน้น **Availability (Eventual Consistency)** สำหรับข้อมูลทั่วไป เช่น จำนวนไลค์, รีวิว, หรือแม้แต่สถานะร้านค้า (อาจจะดีเลย์นิดหน่อยรับได้) เพื่อให้ระบบไม่ล่ม แต่สำหรับ "สถานะการตัดเงิน" และ "สต็อกของ" จะต้องใช้ Strong Consistency เพื่อไม่ให้ตัดเงินผิดพลาด