**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC NGOẠI NGỮ-TIN HỌC THÀNH PHỐ HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

A yellow oval with red text

Description automatically generated

**BÀI BÁO CÁO KẾT THÚC MÔN HỌC**

**DỮ LIỆU LỚN**

**ĐỀ TÀI**

**HỆ THỐNG THEO DÕI GIAO NHẬN TRỰC TUYẾN BẰNG KAFKA**

**SINH VIÊN THỰC HIỆN:**

Uông Thành Trung – 22DH113985

Nguyễn Quốc Minh – 22DH112165

Lê Phạm Hoàng Vũ – 22DH114826

**Giảng viên hướng dẫn:** ThS. Võ Thị Hồng Tuyết

**Thành phố Hồ Chí Minh, tháng 4 năm 2025**

MỤC LỤC

Contents

[MỤC LỤC 1](#_Toc1831804525)

[Chương 1. Giới thiệu 3](#_Toc992851746)

[1.1 Giới thiệu đề tài: 3](#_Toc390224403)

[1.2 Mục tiêu và nội dung đề tài: 4](#_Toc1458018074)

[1.3 Giới hạn đề tài: 5](#_Toc971849812)

[1.4 Cấu trúc báo cáo: 6](#_Toc1591954822)

[Chương 2. Cơ sở lý thuyết 7](#_Toc2052753044)

[2.1 Tổng quan về hệ thống theo dõi giao nhận hàng trực tuyến: 7](#_Toc478747884)

[2.2 Giới thiệu về Apache Kafka: 8](#_Toc1580968146)

[2.3 Các thành phần chính trong Kafka: 9](#_Toc1977571545)

[2.4 Cơ chế hoạt động của Kafka trong hệ thống theo dõi giao nhận hàng: 9](#_Toc927616024)

[2.5 Ưu điểm của Kafka trong hệ thống theo dõi giao nhận hàng: 10](#_Toc271692906)

[2.6 Các công nghệ bổ trợ: 10](#_Toc218145407)

[Chương 3. Phương pháp thực hiện 12](#_Toc794246547)

[3.1 Yêu cầu bài toán: 12](#_Toc1902114952)

[3.2 Giải thuật thực hiện: 13](#_Toc976339645)

[3.2.1 Kiến trúc tổng thể của hệ thống: 13](#_Toc150358573)

[3.2.2 Thiết kế luồng dữ liệu Kafka 14](#_Toc1315440919)

[3.3 Phương pháp đánh giá 17](#_Toc1455781568)

[3.3.1 Khả Năng Mở Rộng: 18](#_Toc1880115234)

[3.3.2 Tính Ổn Định: 18](#_Toc1862234266)

[3.3.3 Giao Diện Web: 18](#_Toc1298290848)

[3.3.4 Đăng Nhập & Phân Quyền: 19](#_Toc135305195)

[Chương 4. Hiện thực kết quả 19](#_Toc1811859936)

[4.1 Yêu cầu hệ thống và tập dữ liệu thực nghiệm: 19](#_Toc1688455138)

[4.1.1 Yêu cầu hệ thống: 19](#_Toc1948386166)

[4.1.2 Tập dữ liệu thực nghiệmCác đơn hàng được sinh ra trong hệ thống bởi người dùng khi nhập từ UI 20](#_Toc1084548620)

[4.2 Kết quả thực nghiệm: 20](#_Toc707758541)

[4.2.1 Chức năng Admin 21](#_Toc627372486)

[4.2.2 Chức năng Shipper: 24](#_Toc50358285)

[4.2.3 Chức năng Customer 25](#_Toc1825375835)

[Chương 5. Kết luận 28](#_Toc276148419)

[5.1 Kết quả đạt được: 28](#_Toc411704864)

[5.2 Ưu và nhược điểm: 28](#_Toc1758890884)

[5.2.1 Ưu điểm : 28](#_Toc2145890993)

[5.2.2 Nhược điểm: 29](#_Toc1120947431)

[5.3 Hướng mở rộng tương lai: 29](#_Toc109788868)

[5.4 Tổng kết: 30](#_Toc1132184587)

[Tài liệu tham khảo. 30](#_Toc1150060282)

[PHỤ LỤC CODE DEMO 31](#_Toc64245439)

1. Giới thiệu
   1. Giới thiệu đề tài:

Trong những năm gần đây, ngành thương mại điện tử và dịch vụ vận chuyển đã có sự phát triển vượt bậc. Sự bùng nổ của các nền tảng mua sắm trực tuyến, giao đồ ăn, và vận chuyển hàng hóa đòi hỏi một hệ thống quản lý đơn hàng thông minh, minh bạch và hiệu quả. Tuy nhiên, một trong những thách thức lớn mà nhiều doanh nghiệp vận tải và giao nhận gặp phải chính là khả năng theo dõi trạng thái đơn hàng theo thời gian thực và quản lý dữ liệu giao nhận một cách có hệ thống.

Hiện nay, nhiều hệ thống truyền thống vẫn sử dụng các phương thức quản lý đơn hàng thủ công hoặc sử dụng các nền tảng không tối ưu, gây ra một số vấn đề như:

* **Cập nhật trạng thái đơn hàng chậm**: Một số hệ thống yêu cầu shipper hoặc nhân viên nhập dữ liệu thủ công vào hệ thống, dẫn đến độ trễ trong quá trình cập nhật thông tin.
* **Khả năng mở rộng kém**: Khi lượng đơn hàng tăng cao, hệ thống dễ bị quá tải, gây ra tình trạng chậm trễ hoặc lỗi xử lý.
* **Thiếu tính nhất quán trong việc quản lý dữ liệu**: Dữ liệu về đơn hàng có thể bị phân tán, khó truy xuất hoặc dễ bị mất mát khi có sự cố xảy ra.
* **Giao diện không thân thiện**: Một số hệ thống cũ không tối ưu cho người dùng, gây khó khăn trong việc thao tác và quản lý đơn hàng.

Để giải quyết những vấn đề trên, đề tài “Xây dựng hệ thống theo dõi giao nhận hàng trực tuyến” ra đời nhằm cung cấp một giải pháp tối ưu giúp doanh nghiệp quản lý đơn hàng hiệu quả hơn.

Hệ thống này sử dụng Apache Kafka, một nền tảng xử lý dữ liệu theo thời gian thực, giúp cập nhật trạng thái đơn hàng ngay lập tức mà không bị gián đoạn. Điều này giúp khách hàng, shipper và quản lý dễ dàng theo dõi đơn hàng mà không phải chờ đợi quá lâu. Ngoài ra, hệ thống sử dụng Streamlit để xây dựng giao diện thân thiện, trực quan và dễ sử dụng, giúp tất cả các đối tượng tham gia vào hệ thống có thể thao tác một cách thuận tiện nhất.

Bên cạnh đó, hệ thống còn áp dụng cơ chế phân quyền chặt chẽ cho từng vai trò:

* **Khách hàng**: Đặt hàng, kiểm tra trạng thái đơn hàng.
* **Shipper**: Nhận đơn và cập nhật trạng thái giao hàng.
* **Admin**: Quản lý toàn bộ hệ thống, tìm kiếm, sửa/xóa đơn hàng.

Việc triển khai một hệ thống theo dõi đơn hàng hiện đại không chỉ giúp tối ưu quy trình vận hành của doanh nghiệp mà còn nâng cao trải nghiệm của khách hàng, đảm bảo tính minh bạch và hiệu quả trong quá trình giao nhận. Với sự kết hợp giữa Kafka và Streamlit, hệ thống hứa hẹn mang đến một giải pháp mạnh mẽ, linh hoạt và có khả năng mở rộng trong tương lai.

* 1. Mục tiêu và nội dung đề tài:

**Mục tiêu**:

Mục tiêu của đề tài là xây dựng một hệ thống theo dõi giao nhận hàng trực tuyến có khả năng:

* Theo dõi trạng thái đơn hàng theo thời gian thực sử dụng công nghệ Apache Kafka.
* Phân quyền rõ ràng giữa các vai trò: Khách hàng, Shipper, và Admin.
* Cung cấp một giao diện trực quan và dễ sử dụng trên nền tảng Streamlit.
* Ứng dụng MapReduce đơn giản để xử lý nhanh thống kê và tra cứu đơn hàng từ dòng sự kiện Kafka mà không cần truy vấn MySQL.
* Cho phép hệ thống mở rộng và chịu tải tốt hơn khi số lượng đơn hàng tăng lên theo thời gian.

**Nội dung**:

**Tìm hiểu và cài đặt Apache Kafka**

* Cấu hình Kafka và Zookeeper bằng Docker Compose.
* Khởi tạo các topic cần thiết, đặc biệt là topic orders.

**Thiết kế hệ thống theo dõi đơn hàng**

* Gồm 3 vai trò: **Admin**, **Shipper**, **Customer**.
* Mỗi vai trò có giao diện riêng và các chức năng khác nhau.

**Xây dựng giao diện bằng Streamlit**

* Khách hàng: Tạo và theo dõi đơn hàng.
* Shipper: Cập nhật trạng thái giao hàng.
* Admin: Quản lý người dùng, đơn hàng, thống kê tổng quan hệ thống.

**Phát triển Kafka Producer và Consumer**

* **Producer**: gửi đơn hàng và sự kiện thay đổi trạng thái lên Kafka.
* **Consumer**: lắng nghe và xử lý dữ liệu từ Kafka, cập nhật vào MySQL.

**Thống kê và tìm kiếm đơn hàng bằng MapReduce**

* Sử dụng Kafka Consumer như một bộ MapReduce đơn giản:
  + Map: duyệt qua từng bản ghi Kafka.
  + Reduce: thống kê số lượng theo trạng thái đơn hàng.
* Hỗ trợ tra cứu đơn hàng nhanh mà không truy vấn database.
  1. Giới hạn đề tài:

**Phạm vi triển khai:**

* Hệ thống hướng đến mô phỏng quy trình theo dõi và giao nhận hàng cơ bản trong môi trường học thuật.
* Giao diện người dùng được xây dựng bằng Streamlit, chỉ hoạt động trên nền web nội bộ.
* Kafka được sử dụng làm trung tâm xử lý dữ liệu theo thời gian thực (event streaming), nhưng chỉ mới ghi nhận và xử lý đơn hàng, chưa triển khai tính năng lưu dữ liệu dài hạn bằng hệ thống phân tán như HDFS.
* MapReduce được triển khai dưới dạng đơn giản (sử dụng vòng lặp và Counter) để thống kê và tìm kiếm từ các bản ghi Kafka không sử dụng Spark hay Hadoop Framework phức tạp.

**Giới hạn:**

* Không triển khai tính năng thanh toán hoặc xử lý giao dịch tài chính thực tế.
* Không tích hợp hệ thống định vị GPS hoặc bản đồ định tuyến thực tế.
* Không triển khai phân tán thật sự trên nhiều máy chủ hay sử dụng Kubernetes.
* Không xử lý bảo mật chuyên sâu (như SSL, xác thực đa yếu tố,…), chỉ sử dụng xác thực người dùng cơ bản với vai trò.
* Không hỗ trợ gửi thông báo thời gian thực qua email, SMS hoặc push notification.
  1. Cấu trúc báo cáo:

Báo cáo bao gồm các phần sau:

* **Chương 1**: Giới thiệu đề tài, mục tiêu, phạm vi nghiên cứu.
* **Chương 2**: Cơ sở lý thuyết về Apache Kafka, kiến trúc Microservices và hệ thống theo dõi giao nhận.
* **Chương 3**: Phương pháp thực hiện, từ yêu cầu bài toán, thuật toán, đến đánh giá kết quả.
* **Chương 4**: Kết quả thực nghiệm và đánh giá hệ thống.
* **Chương 5**: Kết luận và hướng phát triển trong tương lai.

1. Cơ sở lý thuyết
   1. Tổng quan về hệ thống theo dõi giao nhận hàng trực tuyến:

Hệ thống theo dõi giao nhận hàng trực tuyến là một nền tảng công nghệ hỗ trợ quản lý, giám sát và cập nhật trạng thái đơn hàng trong quá trình vận chuyển từ người gửi đến người nhận. Trong thời đại thương mại điện tử phát triển mạnh mẽ, nhu cầu theo dõi đơn hàng theo thời gian thực đã trở thành một yêu cầu thiết yếu nhằm nâng cao trải nghiệm khách hàng và tối ưu quy trình vận hành của doanh nghiệp.

Hệ thống này cho phép **khách hàng**, **shipper** và **quản trị viên** tương tác và theo dõi thông tin đơn hàng một cách liên tục. Các chức năng cơ bản bao gồm:

* **Khách hàng**: tạo đơn hàng, theo dõi trạng thái, cập nhật thông tin cá nhân.
* **Shipper**: nhận và cập nhật trạng thái đơn hàng được phân công.
* **Admin**: quản lý người dùng, phân phối đơn hàng, thống kê và giám sát toàn bộ hệ thống.

Đặc biệt, trong đồ án này, hệ thống được triển khai với sự hỗ trợ của **Apache Kafka** – một nền tảng truyền tải dữ liệu theo thời gian thực giúp **xử lý các sự kiện đơn hàng (event-driven)** hiệu quả. Kafka hoạt động như một trung tâm trung gian nhận và phát tán thông tin đơn hàng tới các thành phần liên quan.

Ngoài ra, hệ thống còn tích hợp các công cụ thống kê và tìm kiếm đơn hàng nhanh chóng dựa trên kỹ thuật **MapReduce** đơn giản, phục vụ cho nhu cầu truy vấn nhanh và trực quan.

* 1. Giới thiệu về Apache Kafka:

**Apache Kafka** là một nền tảng xử lý luồng dữ liệu (stream processing platform) mã nguồn mở, được phát triển ban đầu bởi LinkedIn và sau đó được duy trì bởi Apache Software Foundation. Kafka được thiết kế để xử lý dữ liệu theo thời gian thực, với khả năng **truyền tải, lưu trữ, và xử lý hàng triệu thông điệp mỗi giây** một cách hiệu quả, ổn định và có khả năng mở rộng cao.

**Ưu điểm nổi bật của Kafka:**

* **Hiệu suất cao**: Kafka có thể xử lý hàng triệu thông điệp mỗi giây với độ trễ thấp.
* **Tính mở rộng**: Có thể mở rộng dễ dàng bằng cách thêm broker.
* **Đảm bảo độ tin cậy**: Kafka hỗ trợ lưu trữ phân mảnh và sao lưu nhiều bản.
* **Tích hợp linh hoạt**: Dễ dàng tích hợp với các công nghệ BigData như Hadoop, Spark, HDFS.

Nhờ những đặc điểm này, Kafka trở thành một lựa chọn lý tưởng cho các hệ thống theo dõi thời gian thực, đặc biệt là trong lĩnh vực giao nhận và xử lý sự kiện đơn hàng.

* 1. Các thành phần chính trong Kafka:

**Cơ chế hoạt động cơ bản:**

Apache Kafka hoạt động dựa trên kiến trúc **publish-subscribe** (pub/sub), trong đó:

* **Producer** là thành phần gửi dữ liệu (messages) vào một **topic** trong Kafka.
* **Consumer** là thành phần đăng ký (subscribe) để nhận dữ liệu từ topic đó.
* **Kafka Broker** là trung tâm chịu trách nhiệm tiếp nhận, lưu trữ và phân phối dữ liệu giữa producer và consumer.
* **Topic** là kênh (channel) để các message được gửi và nhận.

Kafka lưu trữ các bản ghi (messages) theo dạng log, có thể truy xuất theo thứ tự thời gian và giữ lại theo cấu hình thời gian hoặc dung lượng.

* 1. Cơ chế hoạt động của Kafka trong hệ thống theo dõi giao nhận hàng:

**Lý do chọn Kafka trong đề tài**

Trong hệ thống theo dõi giao nhận hàng trực tuyến, Apache Kafka được sử dụng làm trung tâm truyền tải thông tin đơn hàng. Cụ thể:

**Bước 1: Tạo đơn hàng:**

Khi khách hàng tạo đơn hàng qua giao diện Streamlit, hệ thống sẽ đóng gói thông tin đơn hàng (order\_id, khách hàng, địa chỉ, hàng hóa, v.v.) và gửi vào Kafka qua topic orders bằng một Kafka producer.

**Bước 2: Lắng nghe và xử lý dữ liệu:**

Một Kafka consumer lắng nghe topic orders, nhận các bản ghi mới và lưu dữ liệu vào MySQL hoặc xử lý trực tiếp để hiển thị trong hệ thống.

**Bước 3: Cập nhật trạng thái đơn hàng:**

Khi shipper cập nhật trạng thái (ví dụ: "Đang giao", "Đã giao"), một producer khác gửi thông tin trạng thái mới lên Kafka, vẫn trên topic orders.

Consumer tiếp tục ghi nhận thay đổi và cập nhật lại trạng thái trong cơ sở dữ liệu.

**Bước 4: Truy vấn và thống kê:**

Kafka cũng hỗ trợ MapReduce trực tiếp trên các bản ghi từ topic orders, dùng để:

Thống kê số lượng đơn hàng theo trạng thái.

Tìm kiếm đơn hàng dựa trên order\_id mà không cần truy vấn vào MySQL.

* 1. Ưu điểm của Kafka trong hệ thống theo dõi giao nhận hàng:

1. **Xử lý dữ liệu theo thời gian thực:**

* Kafka hỗ trợ truyền tải dữ liệu theo mô hình **streaming**, giúp đơn hàng mới hoặc trạng thái cập nhật được đồng bộ ngay lập tức giữa các vai trò (khách hàng, shipper, admin). Điều này giúp:
* Cập nhật thông tin đơn hàng nhanh chóng.
* Hạn chế độ trễ trong quy trình giao nhận.

1. **Hỗ trợ thống kê bằng MapReduce:**

* Kafka giúp dễ dàng **thực hiện MapReduce** ngay trên dữ liệu streaming mà không cần phụ thuộc vào hệ quản trị cơ sở dữ liệu:
* Thống kê trạng thái đơn hàng theo thời gian thực.
* Tìm kiếm nhanh thông tin đơn hàng từ topic.

1. **Tăng khả năng mở rộng:** Hệ thống có thể dễ dàng thêm nhiều consumer xử lý dữ liệu mà không ảnh hưởng hiệu suất.
2. **Bảo đảm dữ liệu không mất mát:** Kafka hỗ trợ cơ chế lưu trữ message theo phân vùng, đảm bảo dữ liệu có thể được khôi phục nếu xảy ra lỗi hệ thống.
   1. Các công nghệ bổ trợ:

Ngoài Kafka, hệ thống sử dụng các công nghệ sau:

**Streamlit – Giao diện người dùng:**

Được sử dụng để xây dựng giao diện web trực quan cho người dùng như: Admin, Khách hàng và Shipper.

* **MySQL – Hệ quản trị cơ sở dữ liệu:**

Dùng để lưu trữ thông tin người dùng, bao gồm:

Tài khoản và mật khẩu.

Vai trò (admin, khách hàng, shipper).

Thông tin cá nhân như họ tên, số điện thoại, email.

Hệ thống không lưu đơn hàng trong MySQL mà Kafka đóng vai trò lưu và phân phối đơn hàng.

* **Docker & Docker Compose:**

Được sử dụng để triển khai môi trường Kafka:

Gồm Kafka Broker, Zookeeper.

Dễ cấu hình, dễ tái sử dụng và đảm bảo tính đồng nhất giữa các máy.

* **Kafka Tools (Offset Explorer):**

Công cụ theo dõi topic và offset trong Kafka.

Giúp kiểm tra dữ liệu gửi/nhận giữa producer – consumer có hoạt động đúng hay không.

Hữu ích trong quá trình debug và phát triển ứng dụng.

1. Phương pháp thực hiện
   1. Yêu cầu bài toán:
2. Giao diện Đăng nhập và Phân quyền

* **Mục tiêu:** Phân quyền người dùng dựa trên các vai trò như **Admin**, **Shipper**, và **Customer**.
* **Yêu cầu:** Đảm bảo rằng người dùng chỉ có thể truy cập các chức năng được phân quyền (ví dụ: Admin có thể quản lý tài khoản, Shipper có thể cập nhật trạng thái đơn hàng, Customer có thể tạo và theo dõi đơn hàng).

1. Giao diện Tài Xế (Dành cho Shipper)

* **Mục tiêu:** Tài xế (Shipper) có thể xem các đơn hàng đã được phân công và cập nhật trạng thái đơn hàng.
* **Yêu cầu:** Shipper chỉ có thể thấy các đơn hàng của họ và có quyền thay đổi trạng thái đơn hàng.

1. Giao diện Thống Kê (Dành cho Admin)

* **Mục tiêu:** Admin có thể theo dõi tổng quan hệ thống, bao gồm số lượng đơn hàng, tài xế, và khách hàng.
* **Yêu cầu:** Admin có thể xem các thông tin tổng hợp và có các biểu đồ thống kê từ Kafka.

1. Giao diện Quản Lý (Dành cho Admin)

* **Mục tiêu:** Admin có thể quản lý người dùng và đơn hàng, bao gồm chức năng thêm, sửa, xóa người dùng và phân công đơn hàng.
* **Yêu cầu:** Admin có thể thêm, xóa người dùng và phân công đơn hàng cho tài xế.

1. Giao diện Tìm Kiếm (Dành cho Admin)

* **Mục tiêu:** Admin có thể tìm kiếm tài xế và đơn hàng thông qua các thông tin như tên, số điện thoại, lịch sử chuyến đi.
* **Yêu cầu:** Admin có thể tìm kiếm tài xế, xe và trạng thái đơn hàng.
  1. Giải thuật thực hiện:
     1. Kiến trúc tổng thể của hệ thống:
* **Front end:** cung cấp giao diện người dùng thân thiện, hỗ trợ các nhóm người dùng khác nhau với các chức năng:
* Giao diện Đăng nhập: Xác thực người dùng theo vai trò (User, Driver, Dispatcher, Admin)
* Giao diện Đặt xe: Cho phép người dùng gửi yêu cầu đặt xe.
* Giao diện Tài xế: Hiển thị danh sách chuyến đi được gán cho tài xế.
* Giao diện Thống kê: Hiển thị các thống kê tổng quan về tài xế, xe và chuyến đi.
* Giao diện Tìm kiếm: Cho phép Admin tìm kiếm thông tin tài xế, xe, và chuyến đi.
* Giao diện Quản lý:Quản lý tài khoản người dùng (Thêm, sửa, xóa) và phân phối công việc (chuyến đi, tài xế).

- **Back end:** chịu trách nhiệm xử lý dữ liệu và lưu trữ thông tin hệ thống:

* SQL
* Bảng users: Chứa thông tin về người dùng (khách hàng, tài xế, admin).
* Bảng orders: Lưu trữ thông tin về các chuyến đi, bao gồm mã đơn, khách hàng, địa chỉ, trạng thái, tài xế được phân công và phí vận chuyển.
* File CSV
* Dùng để lưu trữ dữ liệu đầu vào về chuyến đi ban đầu từ các yêu cầu khách hàng.
* Sau khi các yêu cầu được xử lý, hệ thống sẽ ghi lại dữ liệu vào CSV để tiếp tục sử dụng trong các mô phỏng tiếp theo.

- Kafka: đóng vai trò xử lý dữ liệu phân tán:

Kafka đóng vai trò trong việc phân phối các sự kiện liên quan đến các chuyến đi, giúp các dịch vụ có thể nhận và xử lý dữ liệu theo thời gian thực:

* Producer (Gửi dữ liệu lên Kafka): Khi có một yêu cầu đặt đơn hàng hoặc cập nhật trạng thái đơn hàng, thông tin sẽ được gửi lên Kafka để admin có thể xử lý.
* Consumer (Tiêu thụ dữ liệu từ Kafka): Lắng nghe và xử lý dữ liệu từ các đơn hàng, ví dụ: cập nhật trạng thái chuyến đi, phân phối đơn hàng đến shiper, và gửi thông tin cho admin.
  + 1. Thiết kế luồng dữ liệu Kafka

Tóm tắt luồng dữ liệu:

1. Khởi tạo dữ liệu từ

2. Job Tracker 1:

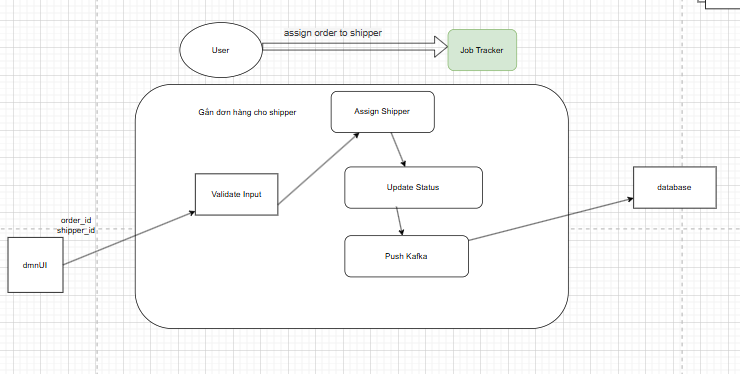
* Vai trò: Xử lý việc tạo đơn hàng từ phía người dùng (customer).
* Luồng hoạt động:
  + Customer nhập thông tin từ UI: tên, địa chỉ, sản phẩm,...
  + Hệ thống map đơn hàng → sinh order\_id làm key, value gồm customer, address, item.
  + Ghi đơn hàng vào DB (insert order record).
  + Gửi thông tin đơn hàng tới Kafka topic order\_status để các thành phần khác biết có đơn hàng mới.

Job Tracker Tạo đơn hàng


Job Tracker tạo đơn hàng

3. Job Tracker 2:

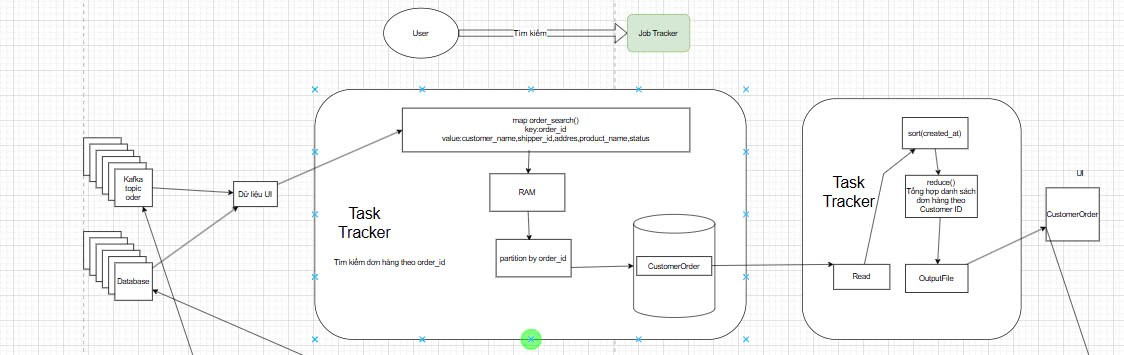
* Vai trò: Phân phối đơn hàng vừa tạo cho một shipper cụ thể.
* Luồng hoạt động:
* User gán đơn từ UI → nhập order\_id và shipper\_id.
* Hệ thống validate input.
* Gán shipper cho đơn hàng (Assign Shipper).
* Cập nhật trạng thái đơn hàng và push thông tin vào Kafka.
* Lưu thông tin phân phối vào database.



Job Tracker phân phối đon hàng cho shipper

Job Tracker 3:

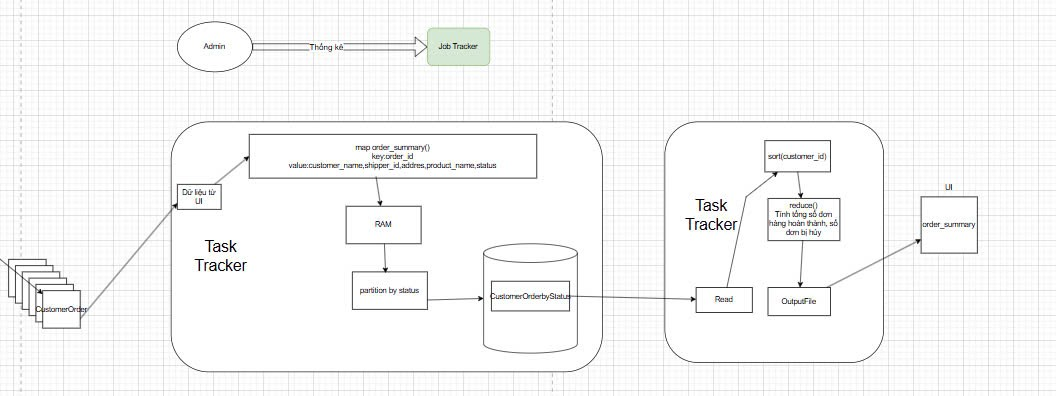
* **Vai trò:** Cho phép tìm kiếm nhanh đơn hàng theo order\_id.
* **Luồng hoạt động:**
* User nhập lệnh tìm kiếm → gửi tới Job Tracker.
* Dữ liệu được map thành order\_id → customer\_name, shipper, product, status,...
* Phân vùng theo order\_id, lưu vào tập dữ liệu CustomerOrder.
* Task Tracker đọc dữ liệu, reduce theo created\_at → kết quả hiển thị ra UI (giao diện khách hàng).



job Tracker tìm kiếm đơn hàng

Job Tracker 4:

* **Vai trò:** Hỗ trợ Admin tổng hợp thông tin đơn hàng (số đơn đã giao, huỷ,...).
* **Luồng hoạt động:**
* Admin yêu cầu thống kê → gửi truy vấn đến Job Tracker.
* Dữ liệu đơn hàng được map theo trạng thái (status) → chứa thông tin customer\_name, shipper, product,...
* Phân vùng theo trạng thái, lưu vào tập CustomerOrderByStatus.
* Task Tracker đọc file, reduce theo customer\_id → thống kê số lượng đơn hoàn thành, đơn huỷ.
* Kết quả gửi ra UI phần báo cáo (order\_summary).



Job Tracker thống kê báo cáo

* 1. Phương pháp đánh giá
     1. Khả Năng Mở Rộng:
* Khả năng mở rộng của hệ thống đề cập đến khả năng của hệ thống trong việc xử lý một lượng lớn người dùng, dữ liệu và yêu cầu mà không gặp phải sự cố hoặc suy giảm hiệu suất.
* **Khả năng mở rộng cao** nếu hệ thống sử dụng Kafka, Docker, Kubernetes và có thiết kế cơ sở dữ liệu phân tán.
* **Khả năng mở rộng hạn chế** nếu sử dụng SQLite làm cơ sở dữ liệu chính, vì SQLite không hỗ trợ tốt với quy mô lớn và không dễ dàng mở rộng theo chiều ngang.
  + 1. Tính Ổn Định:
* Tính ổn định của hệ thống đề cập đến khả năng duy trì hoạt động bình thường trong suốt thời gian vận hành, ngay cả khi có lỗi xảy ra. Hệ thống cần có các cơ chế đảm bảo rằng mọi dữ liệu sẽ được lưu trữ và xử lý một cách an toàn và không mất mát.
* Ổn định cao nếu hệ thống sử dụng Kafka với chế độ replication, cơ sở dữ liệu phân tán, và có giám sát và cảnh báo tích hợp.
* Ổn định thấp nếu không có cơ chế dự phòng dữ liệu (tình trạng dữ liệu bị mất khi gặp sự cố).
  + 1. Giao Diện Web:
* Giao diện web của hệ thống sẽ phục vụ nhiều nhóm người dùng khác nhau, từ khách hàng đến tài xế và admin. Đánh giá giao diện web sẽ tập trung vào tính thân thiện, dễ sử dụng, và đáp ứng các yêu cầu của từng nhóm người dùng.
* Giao diện của Streamlit rất đơn giản và dễ hiểu, giúp người dùng nhanh chóng tương tác với hệ thống mà không cần quá nhiều hướng dẫn.
* Cung cấp các chức năng rõ ràng cho từng vai trò (Khách hàng, Tài xế, Admin) với các nút bấm, thông tin hiển thị hợp lý.
* Giao diện Streamlit có thể không tối ưu trên các thiết bị di động, vì mặc dù Streamlit hỗ trợ responsive design, nhưng một số tính năng vẫn chưa thể tối ưu cho các màn hình nhỏ.
* Cần kiểm tra giao diện trên nhiều thiết bị và điều chỉnh các thành phần sao cho dễ sử dụng trên mobile.
* Giao diện tốt nếu hệ thống sử dụng thiết kế rõ ràng, dễ sử dụng, có các tính năng dành riêng cho từng vai trò, và hỗ trợ trên nhiều thiết bị.
* Giao diện chưa tối ưu nếu không hỗ trợ tốt trên thiết bị di động hoặc thiếu tính năng tìm kiếm và lọc.
  + 1. Đăng Nhập & Phân Quyền:
* **Xác thực qua username và password** cho phép người dùng đăng nhập vào hệ thống.
* Hệ thống cần phân quyền người dùng dựa trên vai trò (Admin, shipper, Customer), từ đó điều hướng người dùng đến các chức năng phù hợp.
* **Admin** có thể quản lý tài khoản, đơn hàng và người dùng.
* **Shipper** có thể cập nhật trạng thái đơn hàng và xem đơn hàng được phân công.
* **Customer** có thể tạo đơn hàng và theo dõi trạng thái đơn hàng của mình.
* **Đăng nhập và phân quyền tốt** nếu hệ thống sử dụng cơ chế xác thực an toàn, phân quyền rõ ràng cho các nhóm người dùng, và bảo vệ dữ liệu người dùng.
* **Đăng nhập và phân quyền yếu** nếu không có mã hóa mật khẩu hoặc phân quyền không rõ ràng, dễ dàng bị truy cập trái phép.

1. Hiện thực kết quả
   1. Yêu cầu hệ thống và tập dữ liệu thực nghiệm:
      1. Yêu cầu hệ thống:
2. **Yêu cầu phần mềm:**

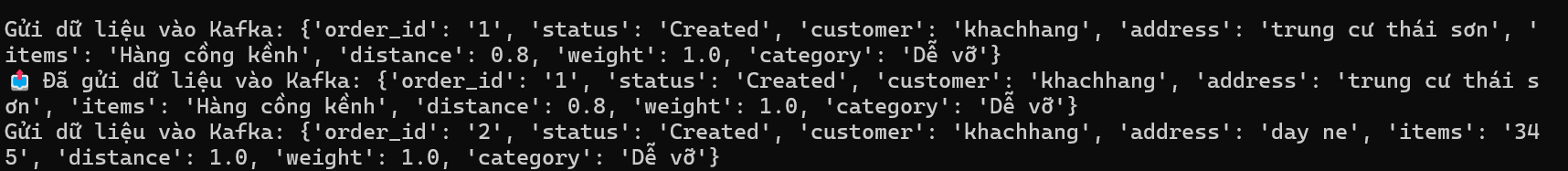
* **Python >= 3.8**: sử dụng để xây dựng backend, frontend, và các dịch vụ xử lý dữ liệu.
* **Apache Kafka >= 3.0**: hệ thống xử lý dòng dữ liệu theo thời gian thực.
* **Apache Zookeeper >= 3.7**: hỗ trợ quản lý Kafka cluster.
* **Docker & Docker Compose**: dùng để triển khai Kafka và Zookeeper một cách nhanh chóng và đồng bộ.
* **MySQL >= 5.7 hoặc 8.0**: hệ quản trị cơ sở dữ liệu lưu thông tin người dùng và đơn hàng.
* **Streamlit >= 1.20**: xây dựng giao diện người dùng.
* **Kafka-python**: thư viện Python để giao tiếp với Kafka.
* **SQLAlchemy**: thư viện ORM để thao tác với cơ sở dữ liệu MySQL.
* **bcrypt**: mã hóa mật khẩu người dùng để đảm bảo bảo mật.
* **Pandas, Matplotlib**: sử dụng để thống kê và trực quan hóa dữ liệu trong giao diện admin.

1. **Yêu cầu phần cứng:**

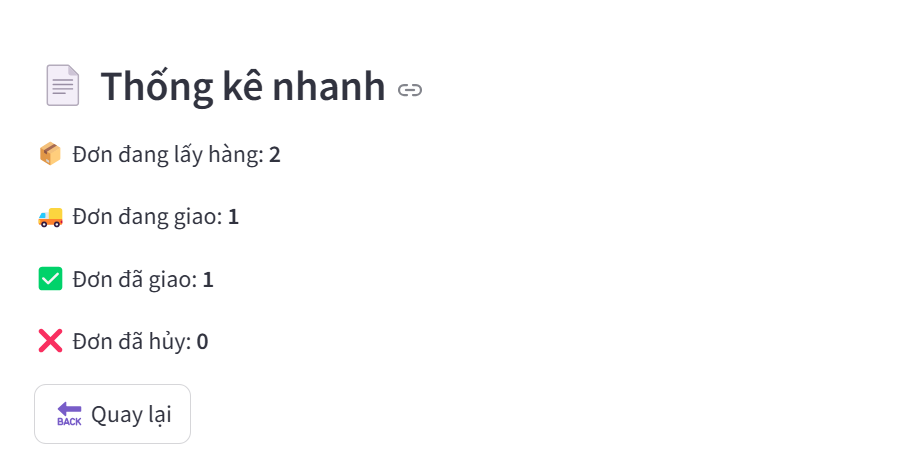
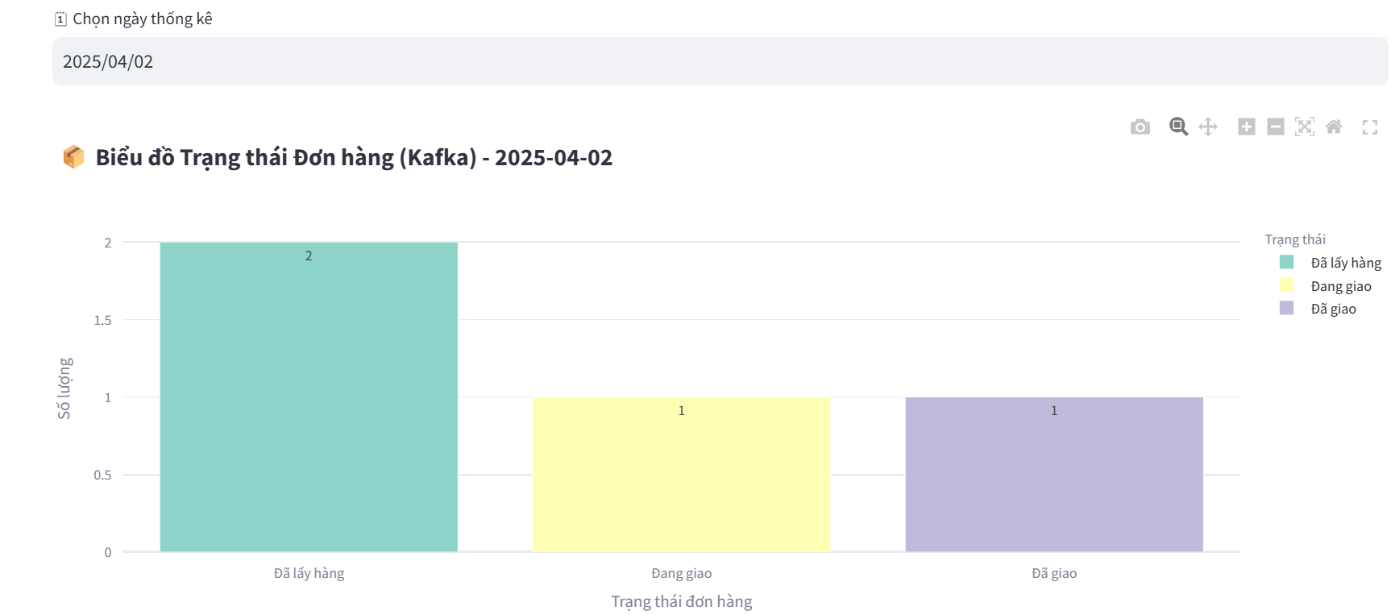
* CPU: Intel Core i5 hoặc tương đương trở lên.
* RAM: tối thiểu 8GB để đảm bảo Kafka và hệ thống xử lý mượt mà.
* Ổ cứng: ít nhất 10GB trống, ưu tiên ổ SSD để tăng tốc độ đọc/ghi.
* Hệ điều hành: Windows, Ubuntu hoặc macOS đều hỗ trợ tốt cho Python, Kafka và Docker.

1. **Kiến trúc triển khai hệ thống:**

* Kafka và Zookeeper được triển khai bằng Docker Compose, đặt trong thư mục D:\Docker.
* Frontend (Streamlit) và backend đặt tại thư mục C:\Users\hoang\BigData.
* Các dịch vụ giao tiếp nội bộ qua địa chỉ localhost.
  + 1. Tập dữ liệu thực nghiệm  
       Các đơn hàng được sinh ra trong hệ thống bởi người dùng khi nhập từ UI

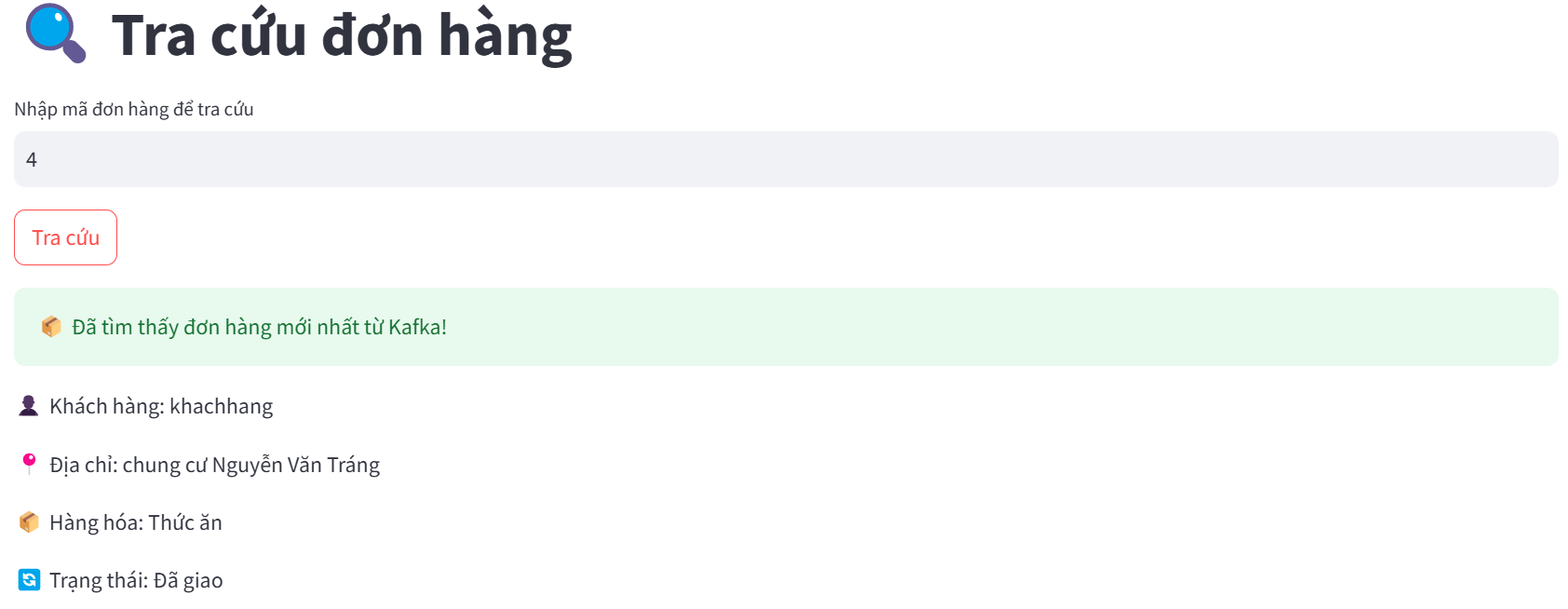


* 1. Kết quả thực nghiệm:
     1. Chức năng Admin
* Thống kê đơn hàng:



Hiển thị tổng số đơn hàng, phân loại theo trạng thái như "Created", "Đang giao", "Đã giao"... Dữ liệu được lấy từ Kafka bằng MapReduce để đảm bảo nhanh và không cần truy vấn MySQL

* Tìm kiếm đơn hàng:



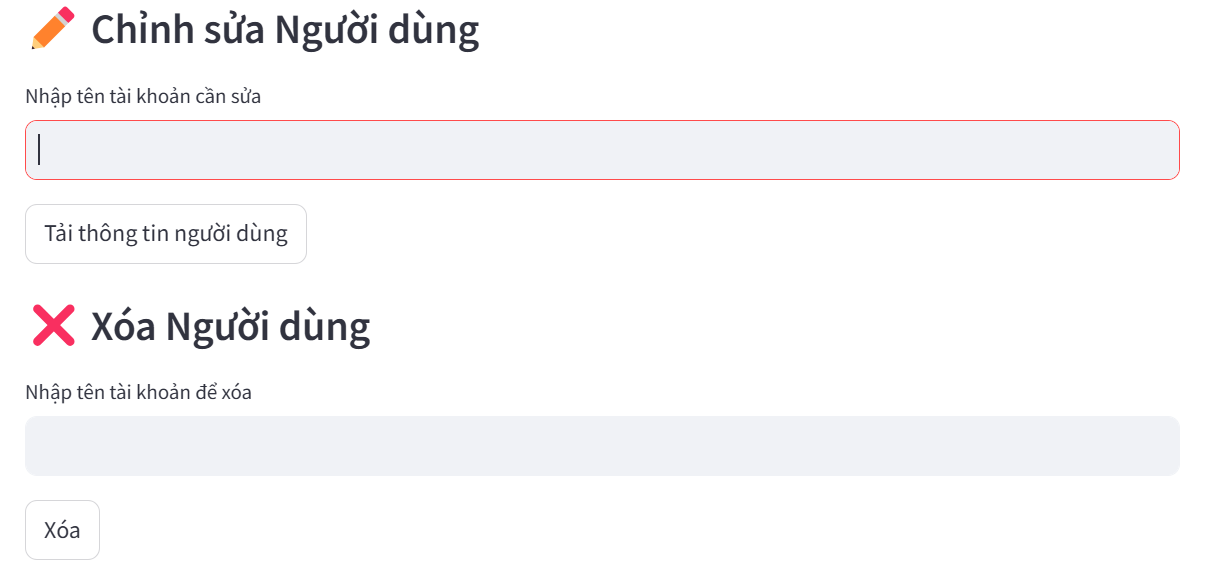
Cho phép tìm kiếm đơn hàng bằng order\_id hoặc các tiêu chí khác để hỗ trợ xử lý sự cố nhanh chóng.

* Phân phối đơn hàng:



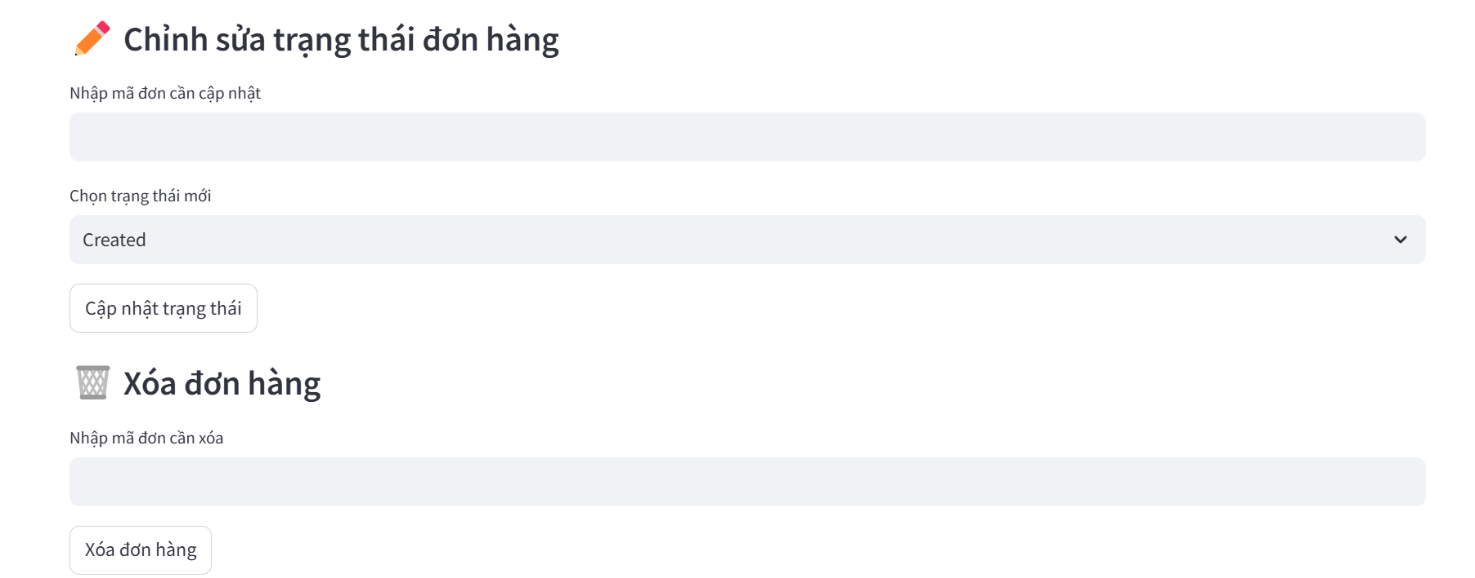
Tự động gán đơn hàng chưa có shipper cho các tài xế dựa trên danh sách tài xế có sẵn. Hệ thống sử dụng hàm chọn ngẫu nhiên hoặc theo quy tắc.

* Thêm, xóa, xửa người dùng

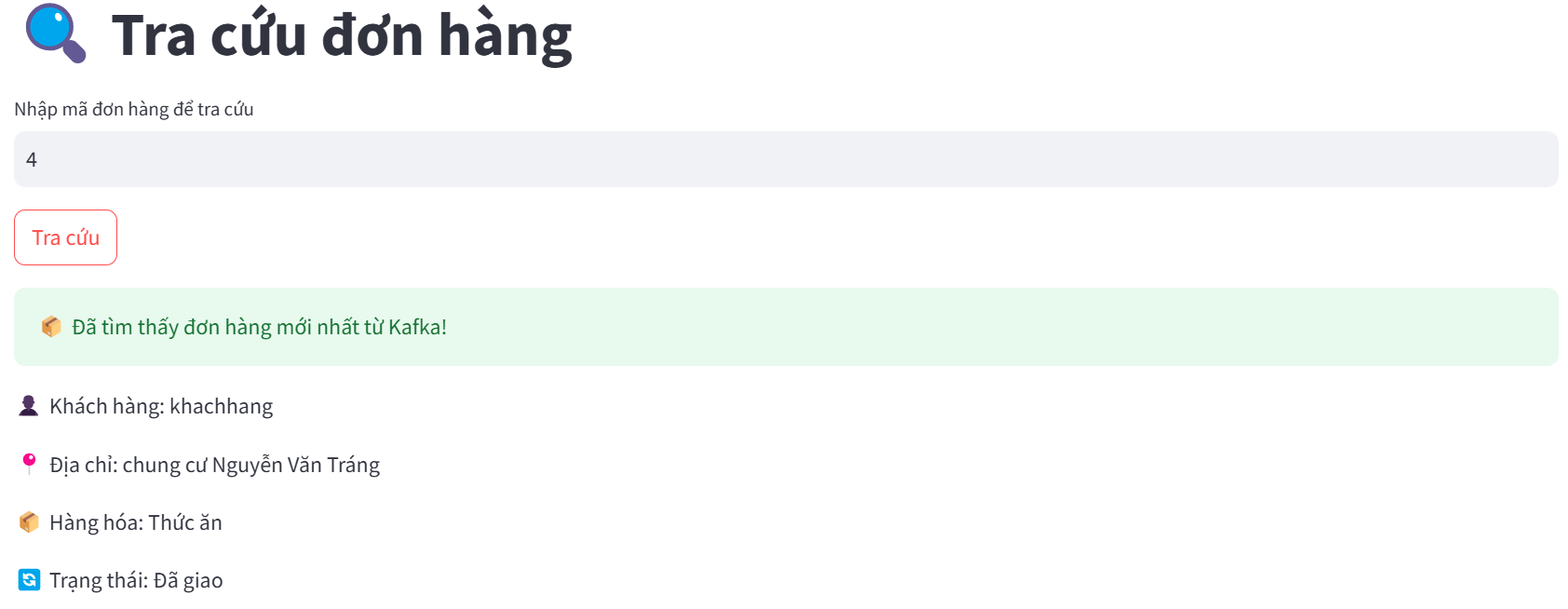


Admin có thể thêm tài khoản mới, chỉnh sửa thông tin hoặc xóa user (shipper, customer hoặc admin khác).

* Xóa, xửa đơn hàng:

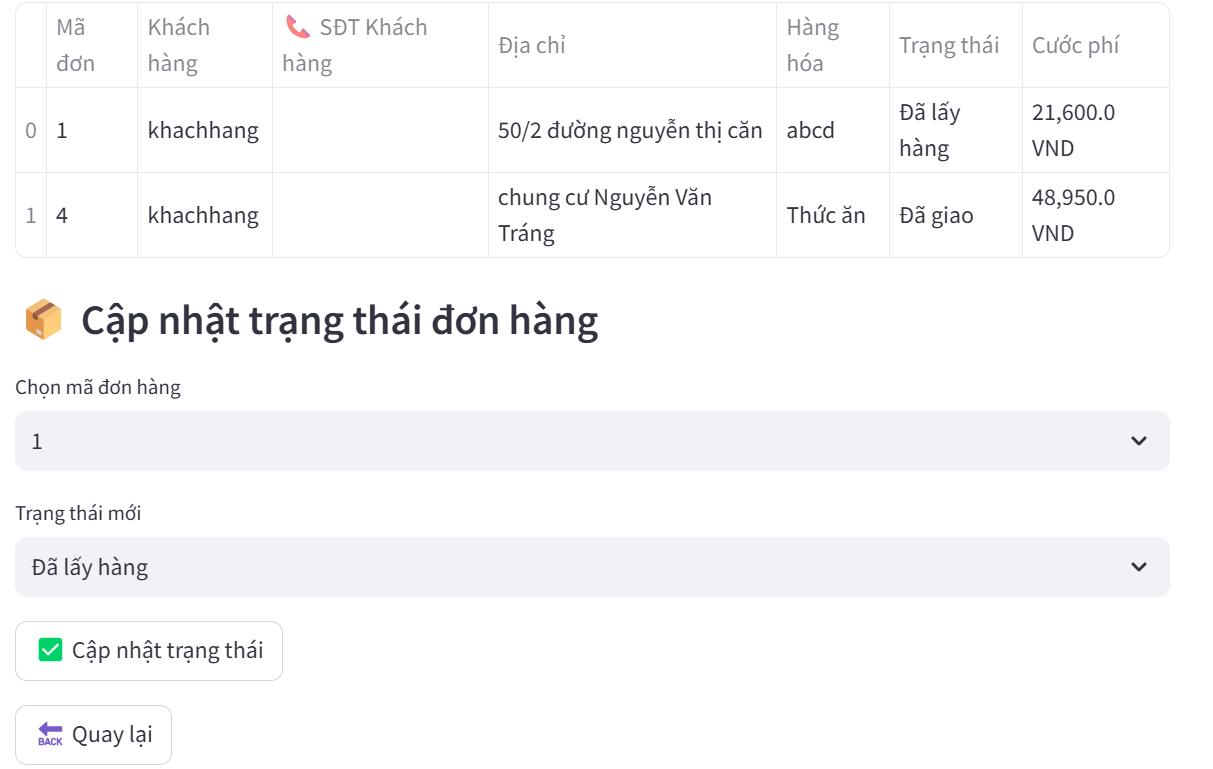
  
Quản trị viên có quyền chỉnh sửa thông tin đơn hàng như địa chỉ, hàng hóa hoặc trạng thái. Ngoài ra có thể xóa đơn hàng nếu cần.

* + 1. Chức năng Shipper:
* Tìm kiếm đơn hàng



Shipper có thể tra cứu nhanh đơn hàng mình được giao để xem chi tiết thông tin và lịch sử trạng thái.

* Cập nhật trạng thái đơn hàng



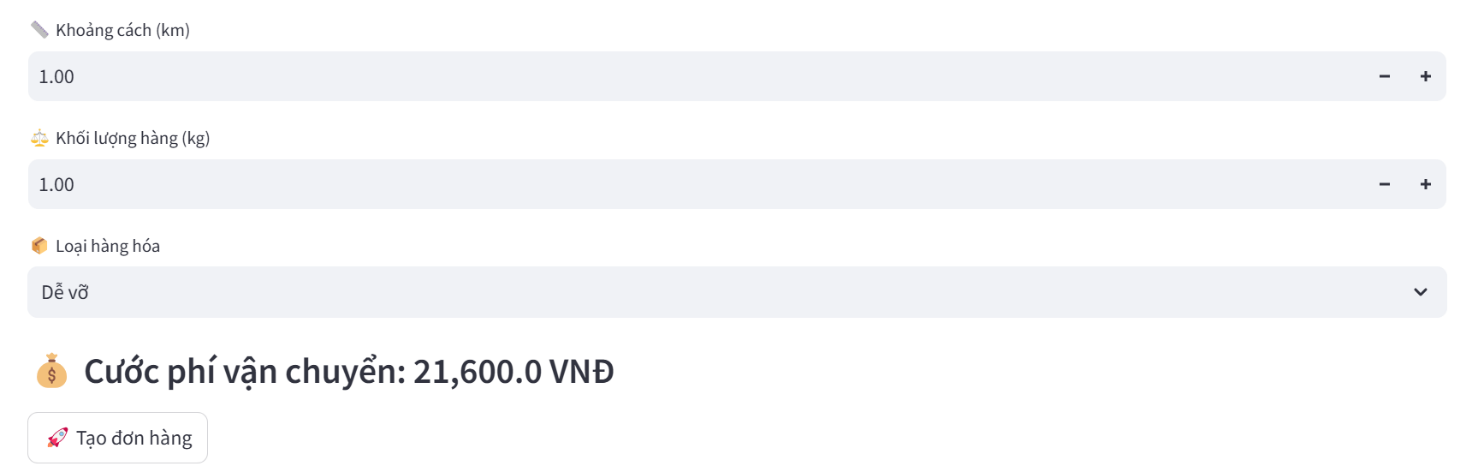
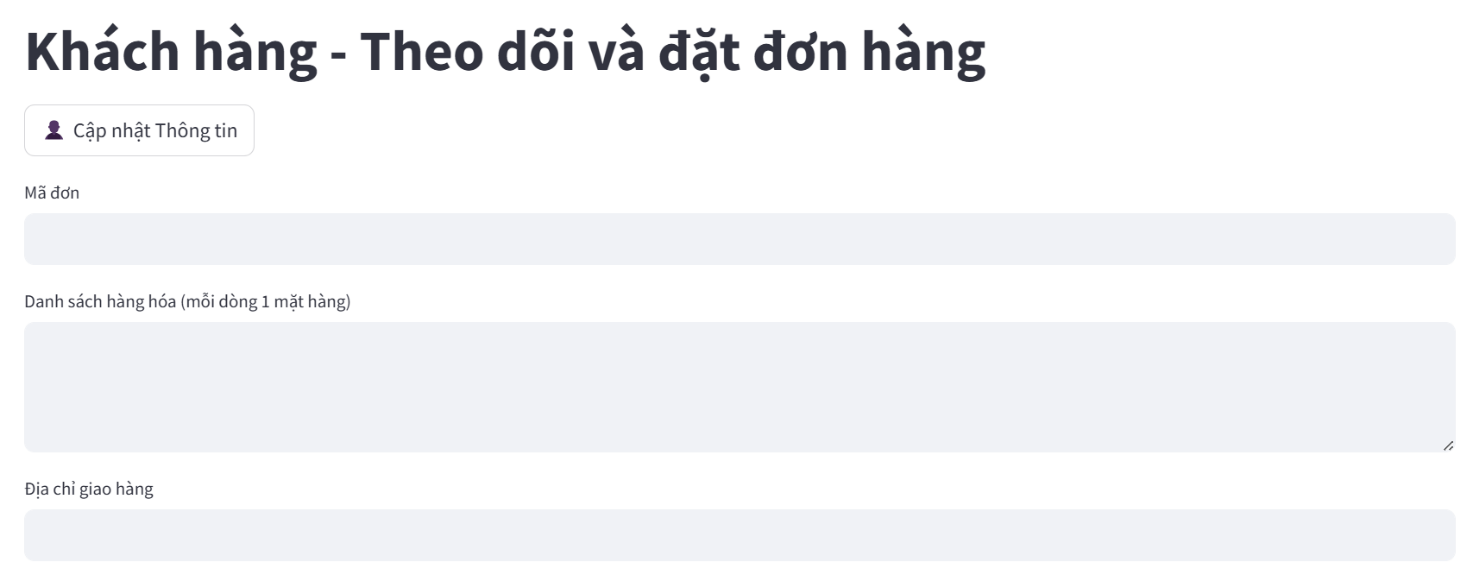
Sau khi nhận hàng, shipper cập nhật trạng thái như "Đã lấy hàng", "Đang giao", "Đã giao"... giúp khách hàng theo dõi thời gian thực.

* Xem danh sách đơn hàng được giao



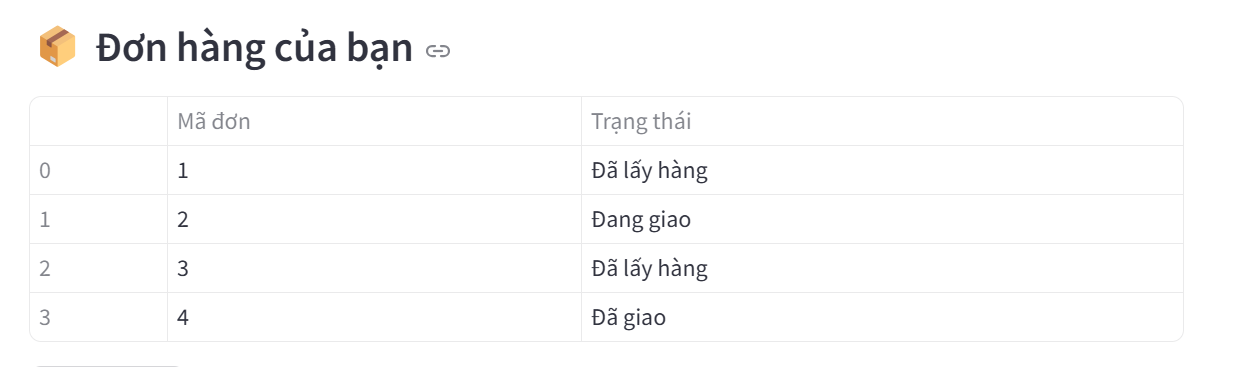
Chỉ hiển thị các đơn hàng đã được hệ thống phân công cho shipper đang đăng nhập.

* + 1. Chức năng Customer
* Đặt đơn hàng mới



Giao diện đơn giản cho phép khách hàng nhập thông tin đơn hàng (địa chỉ, hàng hóa, loại hàng...) và hệ thống sẽ gửi thông tin này lên Kafka để xử lý.

* Theo dõi trạng thái đơn hàng



Cho phép xem toàn bộ lịch sử và trạng thái mới nhất của từng đơn hàng. Trạng thái được lấy trực tiếp từ Kafka để đảm bảo cập nhật thời gian thực.

1. Kết luận
   1. Kết quả đạt được:

Sau thời gian nghiên cứu và triển khai, hệ thống **“Theo dõi giao nhận hàng trực tuyến sử dụng Apache Kafka”** đã đạt được một số kết quả tích cực như sau:

* **Xây dựng thành công hệ thống đa vai trò** gồm **Admin**, **Shipper** và **Khách hàng**, với mỗi vai trò có quyền truy cập và giao diện phù hợp.
* **Cập nhật trạng thái đơn hàng theo thời gian thực** thông qua Apache Kafka. Việc cập nhật từ phía shipper được ghi nhận và truyền tải ngay lập tức, đảm bảo thông tin luôn đồng bộ cho khách hàng và admin.
* **Giao diện người dùng trực quan**, được xây dựng bằng **Streamlit**, giúp thao tác nhanh chóng, thân thiện với người dùng và dễ triển khai trên nền tảng nội bộ.
* **Tích hợp MapReduce đơn giản** để thực hiện các thao tác thống kê trạng thái đơn hàng và tìm kiếm order nhanh từ Kafka topic orders, không cần truy vấn trực tiếp từ MySQL – giúp tăng hiệu năng và giảm tải cơ sở dữ liệu.
* **Triển khai thành công Kafka và Zookeeper bằng Docker Compose**, đảm bảo hệ thống dễ di động, dễ phục hồi và có thể mở rộng trong tương lai.
* **Cơ sở dữ liệu MySQL** lưu trữ thông tin người dùng và đơn hàng một cách an toàn, có mã hóa mật khẩu người dùng bằng bcrypt, đảm bảo tính bảo mật cơ bản.
  1. Ưu và nhược điểm:
     1. Ưu điểm :
* **Hiệu năng cao & thời gian thực**: Kafka xử lý và truyền dữ liệu gần như tức thời. Người dùng có thể theo dõi và cập nhật đơn hàng mà không cần tải lại trang hoặc chờ đợi backend xử lý lâu.
* **Phân quyền rõ ràng**:
* Khách hàng: tạo đơn, theo dõi trạng thái, cập nhật thông tin cá nhân.
* Shipper: xem đơn hàng được giao, cập nhật trạng thái đơn.
* Admin: quản lý người dùng, phân phối đơn, thống kê toàn hệ thống.
* **Thiết kế dễ mở rộng**: Các thành phần Kafka, Database, Streamlit frontend được tách riêng, dễ bảo trì và phát triển thêm.
* **Không phụ thuộc hoàn toàn vào database**: Nhờ Kafka, hệ thống có thể hoạt động ổn định cả khi MySQL bị gián đoạn tạm thời, hoặc dùng dữ liệu tạm từ message queue.
* **Đơn giản, dễ triển khai học thuật**: Phù hợp với mô phỏng, lab nhỏ hoặc làm nền tảng cho nghiên cứu chuyên sâu sau này.
  + 1. Nhược điểm:
* **Chưa hỗ trợ phân tán thực tế**: Mặc dù sử dụng Kafka nhưng hệ thống vẫn chạy trên 1 máy chủ. Chưa triển khai được trên nhiều node hoặc cloud.
* **Sử dụng MySQL cơ bản**: Chưa tích hợp các cơ sở dữ liệu NoSQL hoặc phân tán như Cassandra, MongoDB, v.v.
* **Chưa có tính năng cảnh báo (alert)** như gửi email/SMS khi đơn hàng thay đổi trạng thái hoặc khi xảy ra lỗi hệ thống.
* **Chưa có xác thực nâng cao** như OTP, 2FA, hoặc xác thực dựa trên token.
* **Streamlit không tối ưu trên thiết bị di động**, chỉ phù hợp cho desktop/laptop trình duyệt lớn.
  1. Hướng mở rộng tương lai:

1. **Cải tiến giao diện người dùng (UI/UX)**

* Tích hợp framework hiện đại hơn như **ReactJS**, **NextJS** kết hợp với **FastAPI** hoặc **Flask** để làm backend.
* Tối ưu cho thiết bị di động, tạo Progressive Web App (PWA) hoặc phiên bản mobile.

1. Bổ sung tính năng định vị & theo dõi lộ trình

* Tích hợp Google Maps API để shipper cập nhật lộ trình và khách hàng có thể xem vị trí đơn hàng theo thời gian thực.

1. Nâng cấp bảo mật

* Áp dụng **OAuth2.0**, **JWT** hoặc **2FA** để nâng cao bảo mật người dùng.
* Mã hóa thông tin nhạy cảm trong database.
* Dùng HTTPS thay cho HTTP khi triển khai thực tế.
  1. Tổng kết:

Đề tài đã hoàn thành các mục tiêu đề ra, bao gồm:

* Xây dựng hệ thống có phân quyền rõ ràng.
* Tích hợp Kafka để xử lý đơn hàng thời gian thực.
* Tạo giao diện thân thiện bằng Streamlit.
* Sử dụng MapReduce đơn giản để hỗ trợ thống kê và tra cứu.

Tài liệu tham khảo.

**Phụ lục:** Chứa mã nguồn demo của hệ thống.

# TÀI LIỆU THAM KHẢO

[1] “What is Apache Kafka,” Confluent, [Trực tuyến]. Available: <https://www.confluent.io/what-is-apache-kafka/>. [Đã truy cập 29-3-2025].

[2] Viblo, "Redis là gì?", \*Viblo\*, 19-Apr-2019. [Online]. Available: <https://viblo.asia/p/redis-la-gi-LzD5dN2OZjY>. [Truy cập: 30-3-2025].

[3] GeeksforGeeks, "FastAPI Introduction," \*GeeksforGeeks\*, 2025. [Online]. Available: <https://www.geeksforgeeks.org/fastapi-introduction/>. [Truy cập: 30-3-2025].

[4] Viblo, "WebSocket là gì? Hiểu rõ về WebSocket," \*Viblo\*, 2025. [Online]. Available: <https://viblo.asia/p/websocket-la-gi-hieu-ro-ve-websocket-1Je5E4BmlnL>. [Truy cập: 30-3-2025].

[5] Viblo, "Cùng tìm hiểu về Firebase Cloud Firestore," \*Viblo\*, 2025. [Online]. Available: <https://viblo.asia/p/cung-tim-hieu-ve-firebase-cloud-firestore-ByEZk32YZQ0>. [Truy cập: 30-3-2025].

[6] Viet Thanh, "Công thức tính khoảng cách giữa 2 điểm tọa độ," \*Viet-Thanh.vn\*, 2025. [Online]. Available: <https://viet-thanh.vn/cong-thuc-tinh-khoang-cach-giua-2-diem-toa-do/>. [Truy cập: 30-Mar-2025].

[7] Viblo, "Docker là gì? Kiến thức cơ bản về Docker," \*Viblo\*, 2025. [Online]. Available: <https://viblo.asia/p/docker-la-gi-kien-thuc-co-ban-ve-docker-maGK7qeelj2>. [Truy cập: 30-Mar-2025].

[8] Viblo, "Tổng quan về Apache Kafka – Hệ thống xử lý dữ liệu thời gian thực phân tán," \*Viblo\*, 2025. [Online]. Available: <https://viblo.asia/p/tong-quan-ve-apache-kafka-he-thong-xu-ly-du-lieu-thoi-gian-thuc-phan-tan-5OXLA5XkLGr>. [Truy cập: 30-Mar-2025].

[9] V. T. H. Tuyết, "MapReduce," Lecture Slides, [Trường đại học Ngoại Ngữ và Tin học], 2025.

PHỤ LỤC CODE DEMO

Code tạo Database và kết nối với MySQL:

DATABASE\_URL = "mysql+mysqlconnector://root:hoangvuzent9520%40@localhost:3306/tracking\_db"

engine = create\_engine(DATABASE\_URL)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative\_base()

class User(Base):

\_\_tablename\_\_ = "users"

id = Column(Integer, primary\_key=True, index=True)

full\_name = Column(String(100), nullable=True)

username = Column(String(50), unique=True, nullable=False)

password = Column(String(100), nullable=False)

role = Column(String(20), nullable=False) # Vai trò: "Khách hàng", "Shipper", "Admin"

phone\_number = Column(String(15), nullable=True) # Thêm số điện thoại

email = Column(String(100), nullable=True) # Thêm email

class Order(Base):

\_\_tablename\_\_ = "orders"

order\_id = Column(String(50), primary\_key=True, index=True)

customer = Column(String(50), nullable=False)

address = Column(String(255), nullable=False)

items = Column(String(500), nullable=False) # Lưu dạng chuỗi thay vì list

status = Column(String(50), nullable=False)

created\_at = Column(DateTime, default=datetime.datetime.utcnow)

assigned\_shipper = Column(String(50), nullable=True)

shipping\_fee = Column(Float, nullable=True)

MapReduce thống kê số lượng đơn hàng theo từng trạng thái trong một ngày cụ thể

def get\_order\_status\_counts\_by\_date(target\_date):

consumer = KafkaConsumer(

"orders",

bootstrap\_servers="localhost:9092",

value\_deserializer=lambda v: json.loads(v.decode("utf-8")),

group\_id="dashboard\_group\_by\_date",

auto\_offset\_reset="earliest",

enable\_auto\_commit=False

)

latest\_orders = {}

records = consumer.poll(timeout\_ms=3000, max\_records=1000)

for tp, messages in records.items():

for msg in messages:

order = msg.value

order\_id = order.get("order\_id")

if order\_id:

latest\_orders[order\_id] = (order, msg.timestamp) # Lưu bản ghi mới nhất cùng timestamp

consumer.close()

# Gom nhóm theo ngày và trạng thái

result = Counter()

for order, timestamp in latest\_orders.values():

msg\_date = datetime.fromtimestamp(timestamp / 1000.0).strftime('%Y-%m-%d')

if msg\_date == target\_date:

status = order.get("status", "Unknown")

result[status] += 1

return dict(result)

MapReduce dùng để tìm kiếm một đơn hàng:

def show():

st.title("🔍 Tra cứu đơn hàng")

order\_id = st.text\_input("Nhập mã đơn hàng để tra cứu")

if st.button("Tra cứu"):

consumer = KafkaConsumer(

"orders",

bootstrap\_servers="localhost:9092",

value\_deserializer=lambda v: json.loads(v.decode("utf-8")),

group\_id="tracking\_group",

auto\_offset\_reset="earliest",

enable\_auto\_commit=False

)

latest\_order = None # lưu bản ghi mới nhất

records = consumer.poll(timeout\_ms=2000, max\_records=500)

for tp, messages in records.items():

for msg in messages:

order = msg.value

if str(order.get("order\_id")) == order\_id:

latest\_order = order # luôn ghi đè để lấy bản cuối

consumer.close()

# Hiển thị kết quả nếu tìm thấy đơn hàng

if latest\_order:

st.success("📦 Đã tìm thấy đơn hàng mới nhất từ Kafka!")

st.write(f"👤 Khách hàng: {latest\_order.get('customer', 'Không có dữ liệu')}")

st.write(f"📍 Địa chỉ: {latest\_order.get('address', 'Không có dữ liệu')}")

st.write(f"📦 Hàng hóa: {latest\_order.get('items', 'Không có dữ liệu')}")

st.write(f"🔄 Trạng thái: {latest\_order.get('status', 'Không có dữ liệu')}")

else:

st.warning("❌ Không tìm thấy đơn hàng trong Kafka.")

Producer gửi trạng thái đơn hàng lên Kafka:

def send\_order\_event(order\_id, customer="", address="", items=None, distance=None, weight=None, category="", status="Created"):

"""

Gửi đơn hàng hoặc cập nhật trạng thái đơn hàng lên Kafka.

Nếu chỉ truyền order\_id và status thì tự động lấy dữ liệu đơn hàng từ DB để gửi đầy đủ.

"""

order = {

"order\_id": order\_id,

"status": status

}

# Nếu chỉ có order\_id và status => Lấy dữ liệu từ DB

if not (customer or address or items or distance or weight or category):

db = SessionLocal()

existing\_order = db.query(Order).filter(Order.order\_id == order\_id).first()

db.close()

if existing\_order:

order["customer"] = existing\_order.customer

order["address"] = existing\_order.address

order["items"] = existing\_order.items

order["distance"] = getattr(existing\_order, "distance", 0)

order["weight"] = getattr(existing\_order, "weight", 0)

order["category"] = getattr(existing\_order, "category", "Không xác định")

else:

# Trường hợp tạo đơn mới hoặc cập nhật có đủ thông tin

if customer: order["customer"] = customer

if address: order["address"] = address

if items: order["items"] = items

if distance is not None: order["distance"] = distance

if weight is not None: order["weight"] = weight

if category: order["category"] = category

# Kiểm tra trước khi gửi

print(f"Gửi dữ liệu vào Kafka: {order}")

producer.send("orders", value=order)

producer.flush()

print(f"📤 Đã gửi dữ liệu vào Kafka: {order}")

Consumer đọc dữ liệu khi producer gửi lên:

consumer = KafkaConsumer(

"orders",

bootstrap\_servers="localhost:9092",

value\_deserializer=lambda v: json.loads(v.decode("utf-8")),

group\_id="orders\_group",

auto\_offset\_reset="earliest",

enable\_auto\_commit=True,

consumer\_timeout\_ms=1000

)

# Xử lý dữ liệu Kafka như bình thường

print("🟢 Kafka consumer đang lắng nghe topic 'orders'...")

for msg in consumer:

order = msg.value

print("\n📦 Nhận đơn hàng:")

print(f"- Mã đơn: {order.get('order\_id')}")

print(f"- Khách hàng: {order.get('customer')}")

print(f"- Địa chỉ: {order.get('address')}")

print(f"- Hàng hoá: {order.get('items')}")

print(f"- Khoảng cách: {order.get('distance')} km")

print(f"- Khối lượng: {order.get('weight')} kg")

print(f"- Loại hàng: {order.get('category')}")

print(f"📦 Đơn hàng nhận: {order}")

order\_id = order.get("order\_id")

status = order.get("status")

# Lấy lại thông tin cũ nếu cần

if status == "Đang giao":

# Thực hiện cập nhật hoặc xử lý thông tin cũ nếu cần

print(f"Đơn hàng {order\_id} đang giao")

elif status == "Đã giao":

# Xử lý khi trạng thái đã giao

print(f"Đơn hàng {order\_id} đã giao")

else:

# Trạng thái mặc định

print(f"Đơn hàng {order\_id} có trạng thái khác")

print("✅ Kết thúc lắng nghe Kafka.")

Phân phối đơn hàng cho shipper:

def assign\_orders():

""" Admin phân phối đơn hàng ngẫu nhiên cho shipper """

st.title("📦 Phân phối đơn hàng cho Shipper")

db = SessionLocal()

# Lấy danh sách đơn hàng chưa có shipper

unassigned\_orders = db.query(Order).filter(Order.assigned\_shipper == None).all()

shippers = db.query(User).filter(User.role == "Shipper").all()

# Hiển thị danh sách đơn hàng đã phân phối

st.subheader("📋 Danh sách đơn hàng đã phân phối")

assigned\_orders = db.query(Order).filter(Order.assigned\_shipper != None).all()

if assigned\_orders:

st.table([

{

"Mã đơn": o.order\_id,

"Khách hàng": o.customer,

"Shipper": o.assigned\_shipper,

"Trạng thái": o.status

} for o in assigned\_orders

])

else:

st.write("🚫 Hiện chưa có đơn hàng nào được giao cho shipper.")

# ✅ Hiển thị nút phân phối đơn hàng ngay cả khi đã có đơn hàng được phân phối trước đó

if not shippers:

st.error("🚨 Không có shipper nào trong hệ thống!")

db.close()

return

if st.button("🚀 Phân phối đơn hàng"):

if not unassigned\_orders:

st.success("✅ Tất cả đơn hàng đã được phân công. Phân phối lại toàn bộ đơn hàng.")

all\_orders = db.query(Order).all()

for order in all\_orders:

assigned\_shipper = random.choice(shippers)

order.assigned\_shipper = assigned\_shipper.username

db.commit()

st.rerun()

else:

for order in unassigned\_orders:

assigned\_shipper = random.choice(shippers)

order.assigned\_shipper = assigned\_shipper.username

db.commit()

st.success("✅ Đã phân phối đơn hàng thành công!")

st.rerun()

db.close()

Code admin sửa và xóa đơn hàng (quản lý đơn hàng):

def manage\_orders():

st.subheader("📌 Quản lý Đơn hàng")

db = SessionLocal()

orders = db.query(Order).all()

show\_orders(orders)

# Tìm kiếm đơn hàng

search\_order = st.text\_input("🔍 Nhập mã đơn để tìm kiếm")

if st.button("Tìm kiếm đơn hàng"):

order = db.query(Order).filter(Order.order\_id == search\_order).first()

if order:

st.success(f"📦 Tìm thấy đơn hàng: {order.order\_id} - Trạng thái: {order.status}")

else:

st.error("🚫 Không tìm thấy đơn hàng!")

# Sửa trạng thái đơn hàng

st.subheader("✏️ Chỉnh sửa trạng thái đơn hàng")

order\_id = st.text\_input("Nhập mã đơn cần cập nhật")

new\_status = st.selectbox("Chọn trạng thái mới", ["Created", "Đã lấy hàng", "Đang giao", "Đã giao", "Hủy đơn"])

if st.button("Cập nhật trạng thái"):

order = db.query(Order).filter(Order.order\_id == order\_id).first()

if order:

order.status = new\_status

db.commit()

# ✅ Gửi cập nhật trạng thái lên Kafka

send\_order\_event(order\_id=order\_id, status=new\_status)

st.success(f"✅ Đã cập nhật trạng thái đơn hàng {order\_id} thành {new\_status}")

st.rerun()

else:

st.error("🚫 Không tìm thấy đơn hàng!")

# Xóa đơn hàng

st.subheader("🗑️ Xóa đơn hàng")

delete\_order = st.text\_input("Nhập mã đơn cần xóa")

if st.button("Xóa đơn hàng"):

order = db.query(Order).filter(Order.order\_id == delete\_order).first()

if order:

db.delete(order)

db.commit()

st.success(f"✅ Đã xóa đơn hàng {delete\_order}")

else:

st.error("🚫 Không tìm thấy đơn hàng!")

db.close()

Code Admin quản lý người dùng:

def manage\_users():

st.subheader("📌 Quản lý Người dùng")

db = SessionLocal()

users = db.query(User).all()

show\_users(users)

# 🔍 Tìm kiếm người dùng

search\_user = st.text\_input("🔍 Nhập tên tài khoản để tìm kiếm")

if st.button("Tìm kiếm"):

user = db.query(User).filter(User.username == search\_user).first()

if user:

st.success(f"👤 Tìm thấy: {user.username} - Vai trò: {user.role}")

else:

st.error("🚫 Không tìm thấy tài khoản!")

# ➕ Thêm người dùng mới

st.subheader("➕ Thêm Người dùng")

new\_username = st.text\_input("Tên tài khoản mới")

new\_password = st.text\_input("Mật khẩu", type="password")

role = st.selectbox("Chọn vai trò", ["Khách hàng", "Shipper", "Admin"])

if st.button("Thêm Người dùng"):

try:

new\_user = User(username=new\_username, password=new\_password, role=role)

db.add(new\_user)

db.commit()

st.success(f"✅ Đã thêm {new\_username} ({role}) thành công!")

except IntegrityError:

db.rollback()

st.error("🚫 Tên tài khoản đã tồn tại!")

# ✏️ Chỉnh sửa người dùng

st.subheader("✏️ Chỉnh sửa Người dùng")

edit\_username = st.text\_input("Nhập tên tài khoản cần sửa")

if st.button("Tải thông tin người dùng"):

user = db.query(User).filter(User.username == edit\_username).first()

if user:

new\_role = st.selectbox("🛠️ Vai trò mới", ["Khách hàng", "Shipper", "Admin"], index=["Khách hàng", "Shipper", "Admin"].index(user.role))

new\_password = st.text\_input("🔒 Mật khẩu mới (để trống nếu không đổi)", type="password")

new\_email = st.text\_input("✉️ Email", value=user.email if user.email else "")

new\_phone = st.text\_input("📞 Số điện thoại", value=user.phone\_number if user.phone\_number else "")

if st.button("Lưu thay đổi"):

user.role = new\_role

if new\_password.strip():

user.password = new\_password

user.email = new\_email

user.phone\_number = new\_phone

db.commit()

st.success(f"✅ Đã cập nhật thông tin tài khoản {edit\_username}")

else:

st.error("🚫 Không tìm thấy tài khoản!")

# ❌ Xóa người dùng

st.subheader("❌ Xóa Người dùng")

delete\_user = st.text\_input("Nhập tên tài khoản để xóa")

if st.button("Xóa"):

user = db.query(User).filter(User.username == delete\_user).first()

if user:

db.delete(user)

db.commit()

st.success(f"✅ Đã xóa tài khoản {delete\_user}")

else:

st.error("🚫 Không tìm thấy tài khoản!")

db.close()

Code Shipper dùng để cập nhật đơn hàng theo phân phối:

# Lấy danh sách đơn hàng được phân cho shipper này

orders = db.query(Order).filter(Order.assigned\_shipper == username).all()

order\_data = []

for o in orders:

customer = db.query(User).filter(User.username == o.customer).first()

customer\_phone = customer.phone\_number if customer else "Không có"

order\_data.append({

"Mã đơn": o.order\_id,

"Khách hàng": o.customer,

"📞 SĐT Khách hàng": customer\_phone, # Thêm số điện thoại khách hàng

"Địa chỉ": o.address,

"Hàng hóa": o.items,

"Trạng thái": o.status,

"Cước phí": f"{o.shipping\_fee:,} VND"

})

db.close()

if order\_data:

st.table(order\_data)

else:

st.write("🚫 Bạn chưa có đơn hàng nào.")

# ✅ Chọn mã đơn hàng từ danh sách có sẵn

st.subheader("📦 Cập nhật trạng thái đơn hàng")

order\_id = st.selectbox("Chọn mã đơn hàng", [o.order\_id for o in orders])

new\_status = st.selectbox("Trạng thái mới", ["Đã lấy hàng", "Đang giao", "Đã giao"])

if st.button("✅ Cập nhật trạng thái"):

order = db.query(Order).filter(Order.order\_id == order\_id, Order.assigned\_shipper == username).first()

if order:

order.status = new\_status

db.commit()

db.close()

# ✅ Gửi sự kiện cập nhật trạng thái lên Kafka

send\_order\_event(order\_id=order\_id, status=new\_status)

st.success(f"✅ Đã cập nhật trạng thái đơn hàng {order\_id} thành {new\_status}")

st.rerun() # 🚀 Làm mới trang sau khi cập nhật

else:

st.error("🚫 Không tìm thấy đơn hàng hoặc bạn không được giao đơn hàng này.")

Code Customer tạo mới một đơn hàng:

# 🔹 Nút tạo đơn hàng

if st.button("🚀 Tạo đơn hàng"):

db = SessionLocal()

existing\_order = db.query(Order).filter(Order.order\_id == order\_id).first()

if existing\_order:

st.error(f"🚫 Đơn hàng {order\_id} đã tồn tại trong hệ thống!")

else:

order = Order(

order\_id=order\_id,

customer=st.session\_state["username"], # ✅ Lấy username của khách hàng đã đăng nhập

address=address,

items=", ".join(items.split("\n")), # ✅ Chuyển danh sách hàng hóa thành chuỗi

status="created",

shipping\_fee=shipping\_fee # ✅ Lưu cước phí vào DB

)

db.add(order)

db.commit()

db.close()

send\_order\_event(

order\_id=order\_id,

customer=st.session\_state["username"],

address=address,

items=", ".join(items.split("\n")),

distance=distance,

weight=weight,

category=category,

status="Created" # hoặc "Đang giao", "Đã giao" nếu cập nhật

)

st.success(f"✅ Đã tạo đơn hàng {order\_id} thành công với cước phí \*\*{shipping\_fee:,} VNĐ\*\*!")