

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA ĐÀO TẠO SAU ĐẠI HỌC

CƠ SỞ DỮ LIỆU NÂNG CAO



BÁO CÁO BÀI TẬP LỚN

XÂY DỰNG HỆ THỐNG QUẢN LÝ

MỘT HÃNG VẬN TẢI HÀNH KHÁCH ĐƯỜNG DÀI

Giảng viên hướng dẫn : T.S DƯƠNG TRẦN ĐỨC

Họ và tên học viên : Nguyễn Thế Anh

Mã sinh viên : B24CHKH001

Mã lớp : M24CQKH01-B

Hà Nội – 10/2024

MỤC LỤC

I. GIỚI THIỆU.....	4
1. Mục đích dự án.....	4
2. Phạm vi của dự án.....	4
II. CÔNG NGHỆ SỬ DỤNG.....	6
1. Node.js.....	6
2. MongoDB.....	7
3. Mongoose.....	8
4. Docker Desktop.....	10
5. Navicat Premium.....	12
III. THIẾT KẾ CƠ SỞ DỮ LIỆU.....	15
1. Giới thiệu về các bảng (collections)	15
2. Cấu trúc Schema	16
3. Mô hình liên kết các bảng.....	20
IV. CHỨC NĂNG CHÍNH.....	21
1. Tính lương tài xế và phụ xe	21
2. Tính doanh thu cho từng xe	21
3. Kiểm tra và hiển thị lịch bảo dưỡng.....	22
V. DỮ LIỆU MẪU	23
1. Dữ liệu mẫu cho Bus.....	23
2. Dữ liệu mẫu cho Driver	24
3. Dữ liệu mẫu cho Route	25

4.	<i>Dữ liệu mẫu cho Trip</i>	26
VI.	THỰC THI VÀ KẾT QUẢ	27
1.	<i>Kết quả tính lương tài xế</i>	27
2.	<i>Kết quả tính doanh thu</i>	27
3.	<i>Kết quả kiểm tra bảo dưỡng xe</i>	28
VII.	CÔNG CỤ VÀ MÔI TRƯỜNG PHÁT TRIỂN	29
1.	<i>Cấu hình Docker cho MongoDB</i>	29
2.	<i>Sử dụng Navicat Premium để truy cập dữ liệu</i>	30
VIII.	Kết luận	32
1.	<i>Kết quả đạt được</i>	32
2.	<i>Hướng phát triển tiếp theo</i>	32

I. GIỚI THIỆU

1. Mục đích dự án

Mục đích của dự án này là xây dựng một hệ thống quản lý vận tải hành khách đường dài cho một hãng xe. Hệ thống cho phép quản lý toàn diện các yếu tố liên quan đến vận hành xe khách, bao gồm quản lý thông tin xe, tài xế, tuyến đường và chuyến xe. Ngoài ra, hệ thống hỗ trợ các chức năng quan trọng như tính lương cho tài xế dựa trên số chuyến xe thực hiện và vai trò trên mỗi chuyến, tính doanh thu cho từng xe trong khoảng thời gian cụ thể, cũng như quản lý lịch bảo dưỡng xe nhằm đảm bảo an toàn vận hành.

Dự án còn giúp cung cấp dữ liệu chính xác và kịp thời, hỗ trợ doanh nghiệp ra quyết định nhanh chóng và hiệu quả hơn trong việc quản lý đội xe, tài xế, cũng như các tuyến đường.

2. Phạm vi của dự án

Phạm vi của dự án bao gồm việc xây dựng một hệ thống quản lý vận tải hành khách với các chức năng chính sau:

- **Quản lý xe khách:** Hệ thống lưu trữ thông tin về xe bao gồm biển số, màu sắc, hãng sản xuất, số năm sử dụng, số ghế, và các thông tin liên quan đến lịch bảo dưỡng và số km đã chạy.
- **Quản lý tài xế:** Lưu trữ thông tin tài xế, bao gồm tên, số chứng minh thư, mã số bằng lái, loại bằng lái, địa chỉ và số năm kinh nghiệm. Tài xế có thể đảm nhận các vai trò khác nhau như lái xe hoặc phụ xe.

- ***Quản lý tuyến đường***: Lưu trữ thông tin về các tuyến đường bao gồm điểm đầu, điểm cuối, độ dài tuyến và mức độ phức tạp của tuyến đường.
- ***Quản lý chuyến xe***: Theo dõi chi tiết từng chuyến xe bao gồm thông tin về xe, tuyến đường, tài xế, phụ xe, số lượng hành khách, và giá vé.
- ***Tính lương tài xế***: Dựa trên số chuyến xe mà tài xế đã thực hiện và vai trò trên chuyến xe đó (lái xe hoặc phụ xe). Mức lương còn phụ thuộc vào độ phức tạp của tuyến đường.
- ***Tính doanh thu***: Doanh thu của mỗi xe được tính dựa trên số chuyến xe đã thực hiện và số lượng hành khách trên từng chuyến trong khoảng thời gian xác định.
- ***Quản lý bảo dưỡng xe***: Tự động tính toán ngày bảo dưỡng kế tiếp cho mỗi xe dựa trên số km đã chạy và lịch sử bảo dưỡng. Các xe quá hạn bảo dưỡng sẽ được liệt kê vào danh sách cần kiểm tra ngay.

Phạm vi triển khai của dự án bao gồm việc cài đặt và cấu hình MongoDB bằng Docker Desktop để quản lý dữ liệu, sử dụng Node.js để xây dựng các chức năng xử lý, và Navicat Premium để theo dõi và quản lý dữ liệu trong cơ sở dữ liệu.

II. CÔNG NGHỆ SỬ DỤNG

1. Node.js

Node.js là nền tảng phía server được sử dụng để xây dựng toàn bộ logic xử lý của hệ thống quản lý vận tải hành khách đường dài trong dự án này. Node.js cung cấp một môi trường chạy JavaScript trên máy chủ, giúp ứng dụng xử lý nhiều tác vụ cùng lúc một cách hiệu quả nhờ mô hình bất đồng bộ và kiến trúc dựa trên event-driven.

Trong dự án này, Node.js được sử dụng để:

- ***Xây dựng API và xử lý yêu cầu***: Node.js cung cấp các API để tương tác với cơ sở dữ liệu MongoDB, cho phép hệ thống truy vấn, thêm mới, cập nhật và xóa dữ liệu liên quan đến các thực thể như xe, tài xế, tuyến đường, chuyến xe.
- ***Thực hiện các chức năng chính của hệ thống***: Các chức năng tính lương tài xế, tính doanh thu cho từng xe, và tính toán ngày bảo dưỡng tiếp theo đều được thực thi trên nền tảng Node.js. Sử dụng tính năng bất đồng bộ của Node.js (Promises và async/await) giúp hệ thống xử lý được các tác vụ phức tạp và truy vấn cơ sở dữ liệu một cách hiệu quả.
- ***Mongoose ORM***: Mongoose, một thư viện của Node.js, được sử dụng để quản lý kết nối và thực hiện các thao tác trên MongoDB. Nó giúp đơn giản hóa việc định nghĩa các schema cho các đối tượng dữ liệu (xe, tài xế, tuyến đường, chuyến xe) và cung cấp các phương thức dễ sử dụng để thao tác với dữ liệu.

- **Xử lý truy vấn bất đồng bộ:** Với sự hỗ trợ của Node.js, các truy vấn đến cơ sở dữ liệu MongoDB đều được thực thi theo cách bất đồng bộ, giúp cải thiện hiệu suất của hệ thống và giảm thiểu thời gian chờ đợi khi xử lý nhiều yêu cầu cùng lúc.

Nhờ vào sức mạnh và sự linh hoạt của Node.js, hệ thống quản lý vận tải hành khách trong dự án có thể xử lý các yêu cầu phức tạp một cách hiệu quả, đồng thời đảm bảo khả năng mở rộng và duy trì tính ổn định khi quy mô dữ liệu và số lượng yêu cầu tăng lên.

2. MongoDB

MongoDB là hệ quản trị cơ sở dữ liệu NoSQL được sử dụng trong dự án này để lưu trữ và quản lý toàn bộ dữ liệu của hệ thống vận tải hành khách đường dài. MongoDB được chọn vì khả năng lưu trữ dữ liệu linh hoạt dưới dạng tài liệu (document) với cấu trúc JSON, dễ dàng mở rộng và phù hợp với các hệ thống cần xử lý lượng dữ liệu lớn và phức tạp.

Trong dự án, MongoDB được sử dụng cho các mục đích sau:

- **Lưu trữ dữ liệu phi cấu trúc:** Các thông tin về xe, tài xế, tuyến đường, và chuyến xe được lưu trữ dưới dạng document trong các collection tương ứng. Điều này cho phép hệ thống dễ dàng quản lý các loại dữ liệu khác nhau mà không cần phải sử dụng các bảng quan hệ như trong cơ sở dữ liệu truyền thống.
- **Mongoose ORM:** Thư viện Mongoose được sử dụng để tương tác với MongoDB từ Node.js. Nó cung cấp các phương thức tiện lợi để định nghĩa schema cho các document và thực hiện các truy vấn, giúp quản lý dữ liệu trở nên đơn giản và hiệu quả hơn. Mỗi schema

(xe, tài xế, tuyến đường, chuyến xe) được ánh xạ thành một collection trong MongoDB.

- ***Quản lý các thực thể***: MongoDB giúp lưu trữ thông tin chi tiết về các thực thể như xe khách, tài xế, tuyến đường và chuyến xe một cách linh hoạt. Điều này cho phép hệ thống dễ dàng mở rộng hoặc thay đổi cấu trúc dữ liệu mà không cần thiết lập lại toàn bộ cơ sở dữ liệu.
- ***Truy vấn dữ liệu hiệu quả***: MongoDB hỗ trợ các truy vấn phức tạp để tìm kiếm, lọc và tính toán dữ liệu. Điều này được tận dụng trong các chức năng chính của hệ thống như tính lương tài xế, doanh thu của xe, và xác định ngày bảo dưỡng tiếp theo. Với cơ chế lưu trữ dữ liệu dưới dạng document, MongoDB giúp các truy vấn này được thực hiện nhanh chóng và tối ưu hơn.
- ***Quản lý dữ liệu phân tán và mở rộng dễ dàng***: MongoDB có khả năng mở rộng theo chiều ngang (horizontal scaling), giúp hệ thống dễ dàng mở rộng khi cần xử lý khối lượng dữ liệu lớn và lưu trữ trên nhiều node khác nhau mà vẫn đảm bảo hiệu suất tốt.

Với MongoDB, hệ thống quản lý vận tải hành khách của dự án có thể xử lý dữ liệu linh hoạt, dễ mở rộng, và hỗ trợ các truy vấn phức tạp một cách nhanh chóng, đảm bảo khả năng vận hành hiệu quả khi số lượng chuyến xe, tài xế và dữ liệu người dùng tăng lên.

3. Mongoose

Mongoose là một thư viện Object Data Modeling (ODM) cho MongoDB và Node.js, được sử dụng trong dự án để quản lý và tương tác với cơ sở dữ liệu MongoDB một cách hiệu quả. Mongoose cung cấp một cách tiếp cận dựa trên mô hình hướng đối tượng, giúp đơn giản hóa việc định nghĩa schema, thực hiện

các thao tác CRUD (Create, Read, Update, Delete) và các truy vấn phức tạp trên cơ sở dữ liệu MongoDB.

Trong dự án, Mongoose được sử dụng cho các mục đích sau:

- ***Định nghĩa schema dữ liệu:*** Mongoose cho phép định nghĩa các schema chi tiết cho các thực thể trong hệ thống như xe khách, tài xế, tuyến đường, và chuyến xe. Các schema này bao gồm cấu trúc dữ liệu và các ràng buộc như kiểu dữ liệu, thuộc tính bắt buộc, giá trị mặc định và các quy tắc ràng buộc. Điều này giúp đảm bảo dữ liệu trong MongoDB luôn được lưu trữ với cấu trúc nhất quán và hợp lệ.
 - Ví dụ, schema của tài xế bao gồm các trường như tên, số CMT, mã số bằng lái, và ngày sinh, đảm bảo mỗi tài xế đều có thông tin đầy đủ và chính xác trước khi được lưu trữ trong cơ sở dữ liệu.
- ***Quản lý quan hệ giữa các thực thể:*** Mặc dù MongoDB là cơ sở dữ liệu NoSQL và không có các ràng buộc quan hệ như cơ sở dữ liệu quan hệ, Mongoose hỗ trợ việc tạo liên kết giữa các schema khác nhau. Trong dự án, Mongoose được sử dụng để tạo mối quan hệ giữa các chuyến xe và các tài xế, tuyến đường và xe khách, thông qua việc tham chiếu các đối tượng (ObjectId) giữa các schema. Điều này giúp dễ dàng truy vấn và lấy dữ liệu liên quan giữa các thực thể.
 - Ví dụ, mỗi chuyến xe trong schema của chuyến xe có thể tham chiếu đến tài xế chính, phụ xe và tuyến đường tương ứng thông qua các ObjectId.

- **Thực hiện các truy vấn phức tạp:** Mongoose cung cấp các phương thức dễ sử dụng để thực hiện các truy vấn đến MongoDB, bao gồm các truy vấn lọc, sắp xếp, phân trang và các truy vấn phức tạp với nhiều điều kiện. Điều này giúp dễ dàng xử lý các yêu cầu như tính lương tài xế theo vai trò và số chuyên trong tháng, tính doanh thu của xe trong một khoảng thời gian, và kiểm tra ngày bảo dưỡng xe.
- **Xử lý dữ liệu bất đồng bộ:** Với sự hỗ trợ của cơ chế async/await và Promises, Mongoose cho phép các thao tác truy vấn và cập nhật dữ liệu từ MongoDB diễn ra một cách bất đồng bộ. Điều này giúp hệ thống có khả năng xử lý đồng thời nhiều yêu cầu mà không gây gián đoạn, tăng hiệu suất và khả năng mở rộng của ứng dụng.
- **Tích hợp với Node.js:** Mongoose tích hợp mạnh mẽ với Node.js, giúp đơn giản hóa quá trình phát triển ứng dụng bằng cách cung cấp API dễ sử dụng và hiệu quả để kết nối, tương tác với cơ sở dữ liệu MongoDB trực tiếp trong ứng dụng Node.js.

Nhờ vào Mongoose, dự án có thể quản lý dữ liệu một cách linh hoạt và dễ dàng, đồng thời đảm bảo các thao tác với cơ sở dữ liệu MongoDB được thực hiện nhanh chóng, chính xác và hiệu quả.

4. Docker Desktop

Docker Desktop là một nền tảng phát triển container dành cho các nhà phát triển, giúp tạo, quản lý và chạy các ứng dụng trong các container độc lập. Trong dự án này, Docker Desktop được sử dụng để triển khai môi trường MongoDB cục bộ trên máy tính của lập trình viên mà không cần cài đặt trực tiếp MongoDB trên hệ điều hành.

Docker Desktop hỗ trợ dự án với các lợi ích sau:

- ***Tạo môi trường phát triển độc lập và nhất quán:*** Docker Desktop cho phép đóng gói MongoDB vào một container riêng biệt, đảm bảo rằng môi trường cơ sở dữ liệu luôn đồng nhất giữa các lần phát triển hoặc triển khai. Điều này giúp giảm thiểu các lỗi do sự khác biệt về cấu hình hoặc phiên bản MongoDB giữa các máy phát triển.
 - Trong dự án, MongoDB được khởi chạy dưới dạng một container thông qua Docker, đảm bảo rằng cơ sở dữ liệu luôn chạy trong môi trường được kiểm soát và dễ dàng tái tạo khi cần.
- ***Dễ dàng thiết lập và cấu hình:*** Thay vì phải trải qua các bước cài đặt phức tạp trên hệ điều hành, Docker Desktop chỉ cần một file docker-compose.yml hoặc một câu lệnh Docker đơn giản để tạo và khởi chạy một container MongoDB. Việc cấu hình như cổng, volume để lưu trữ dữ liệu, và thiết lập mạng có thể được thực hiện dễ dàng trong Docker.
 - Ví dụ, trong dự án, chỉ cần chạy một lệnh Docker, container MongoDB đã được khởi tạo và kết nối tới ứng dụng Node.js, giúp tiết kiệm thời gian thiết lập môi trường.
- ***Tính linh hoạt và di động cao:*** Docker Desktop giúp việc chia sẻ môi trường phát triển giữa các thành viên trong nhóm trở nên dễ dàng. Các đồng nghiệp có thể tải xuống và chạy các container MongoDB giống hệt nhau mà không cần lo lắng về sự khác biệt về cấu hình hệ thống hoặc phiên bản phần mềm.
- ***Quản lý tài nguyên dễ dàng:*** Docker Desktop cung cấp giao diện đồ họa trực quan để quản lý các container, giúp lập trình viên dễ dàng kiểm tra trạng thái của MongoDB, khởi động, dừng hoặc xóa

container khi cần. Việc giám sát tài nguyên sử dụng như CPU, RAM của container cũng được thực hiện đơn giản.

- ***Tích hợp với các công cụ khác:*** Docker Desktop có thể dễ dàng tích hợp với các công cụ phát triển khác như Navicat Premium (dùng trong dự án) để truy cập và quản lý cơ sở dữ liệu MongoDB trong container. Điều này cho phép các lập trình viên làm việc với dữ liệu theo cách trực quan hơn, đồng thời vẫn giữ được lợi ích từ việc sử dụng container.

Nhờ có Docker Desktop, dự án có thể dễ dàng triển khai MongoDB cục bộ một cách nhanh chóng, ổn định, và nhất quán. Docker Desktop cung cấp một môi trường phát triển hiện đại, linh hoạt, và dễ bảo trì cho hệ thống quản lý vận tải hành khách.

5. Navicat Premium

Navicat Premium là một công cụ quản lý cơ sở dữ liệu mạnh mẽ và trực quan, hỗ trợ nhiều hệ quản trị cơ sở dữ liệu khác nhau, bao gồm MySQL, PostgreSQL, SQLite, Oracle, và MongoDB. Trong dự án này, Navicat Premium được sử dụng để truy cập, quản lý và thao tác dữ liệu trên MongoDB một cách trực quan, giúp giảm bớt các thao tác dòng lệnh phức tạp.

Navicat Premium cung cấp nhiều lợi ích quan trọng cho dự án:

- ***Giao diện trực quan và dễ sử dụng:*** Thay vì phải sử dụng các dòng lệnh để thao tác với MongoDB, Navicat Premium cung cấp giao diện đồ họa giúp việc quản lý cơ sở dữ liệu dễ dàng và nhanh chóng hơn. Người dùng có thể xem, thêm, sửa, và xóa dữ liệu trực tiếp trên giao diện mà không cần phải viết mã.

- Trong dự án quản lý hãng vận tải, việc kiểm tra và thay đổi dữ liệu về xe, tài xế, tuyến đường hay chuyến đi trở nên đơn giản nhờ giao diện thân thiện của Navicat Premium.
- Quản lý nhiều cơ sở dữ liệu từ một công cụ: Với Navicat Premium, người dùng có thể kết nối và quản lý nhiều cơ sở dữ liệu khác nhau trong một môi trường duy nhất. Điều này rất tiện lợi nếu dự án yêu cầu tích hợp hoặc làm việc với nhiều hệ quản trị cơ sở dữ liệu cùng lúc.
 - Trong trường hợp này, Navicat Premium giúp lập trình viên dễ dàng truy cập MongoDB và kiểm tra dữ liệu trực tiếp từ máy tính cá nhân.
- **Tính năng nhập xuất dữ liệu dễ dàng:** Navicat Premium hỗ trợ nhiều định dạng dữ liệu khác nhau, cho phép người dùng dễ dàng nhập và xuất dữ liệu từ MongoDB. Điều này giúp việc sao lưu, khôi phục dữ liệu hoặc chuyển dữ liệu giữa các hệ thống trở nên đơn giản hơn.
 - Với dự án quản lý vận tải, tính năng này hữu ích khi cần nhập liệu ban đầu hoặc sao lưu dữ liệu chuyến đi, tài xế và xe khách.
- **Công cụ truy vấn mạnh mẽ:** Mặc dù MongoDB là cơ sở dữ liệu NoSQL, Navicat Premium vẫn cung cấp các công cụ truy vấn giúp người dùng thực hiện các thao tác tìm kiếm, lọc dữ liệu một cách nhanh chóng. Điều này giúp kiểm tra dữ liệu và xử lý các truy vấn phức tạp mà không cần viết quá nhiều mã.
- **Tích hợp bảo mật và kết nối linh hoạt:** Navicat Premium hỗ trợ nhiều loại kết nối bảo mật, bao gồm SSH và SSL, giúp đảm bảo rằng dữ liệu luôn được truyền tải một cách an toàn. Điều này đặc

biệt quan trọng khi làm việc với các cơ sở dữ liệu từ xa hoặc trên môi trường đám mây.

Trong dự án quản lý hãng vận tải, Navicat Premium đóng vai trò quan trọng trong việc hỗ trợ lập trình viên quản lý cơ sở dữ liệu MongoDB, truy cập dữ liệu nhanh chóng và trực quan, giảm thiểu các thao tác thủ công và tăng hiệu quả phát triển.

III. THIẾT KẾ CƠ SỞ DỮ LIỆU

1. Giới thiệu về các bảng (collections)

Trong dự án quản lý hãng vận tải hành khách đường dài, hệ thống cơ sở dữ liệu được thiết kế dựa trên MongoDB, một hệ quản trị cơ sở dữ liệu NoSQL. Thay vì sử dụng các bảng (tables) như trong cơ sở dữ liệu quan hệ, MongoDB sử dụng collections để lưu trữ các đối tượng dạng documents. Mỗi collection chứa các document với cấu trúc linh hoạt, phù hợp cho việc lưu trữ dữ liệu phi quan hệ.

Dưới đây là các collection chính trong hệ thống:

- ***Bus (xe khách)***: Collection này lưu trữ thông tin về các xe khách trong hãng. Mỗi document trong collection đại diện cho một chiếc xe với các thông tin như biển số xe, màu xe, hãng sản xuất, số ghế, năm sử dụng, ngày bảo dưỡng cuối cùng và tổng quãng đường đã di chuyển. Các dữ liệu này giúp hệ thống quản lý xe khách một cách chi tiết và hiệu quả.
- ***Driver (tài xế)***: Collection này lưu trữ thông tin chi tiết về các tài xế làm việc cho hãng vận tải, bao gồm tên, số chứng minh thư (CMT), mã số bằng lái, loại bằng lái, địa chỉ, ngày sinh và thâm niên làm việc. Mỗi document trong collection đại diện cho một tài xế và có thể được gán vai trò lái chính hoặc phụ xe cho các chuyến đi.
- ***Route (tuyến đường)***: Collection này lưu trữ các thông tin liên quan đến tuyến đường mà các chuyến xe sẽ di chuyển. Mỗi document đại diện cho một tuyến đường với điểm đầu, điểm cuối, độ dài và mức

độ phức tạp của tuyến. Độ phức tạp được chia thành ba cấp độ (1, 2, 3), giúp hệ thống tính toán lương và thời gian bảo dưỡng dựa trên độ khó của tuyến.

- ***Trip (chuyến xe)***: Collection này lưu trữ chi tiết từng chuyến xe đã hoặc sẽ được thực hiện, bao gồm thông tin về xe, tuyến, tài xế, phụ xe, số lượng hành khách và giá vé. Mỗi document trong collection đại diện cho một chuyến đi cụ thể. Đây là thông tin quan trọng cho việc tính lương tài xế, phụ xe, cũng như doanh thu của hãng từ mỗi chuyến đi.

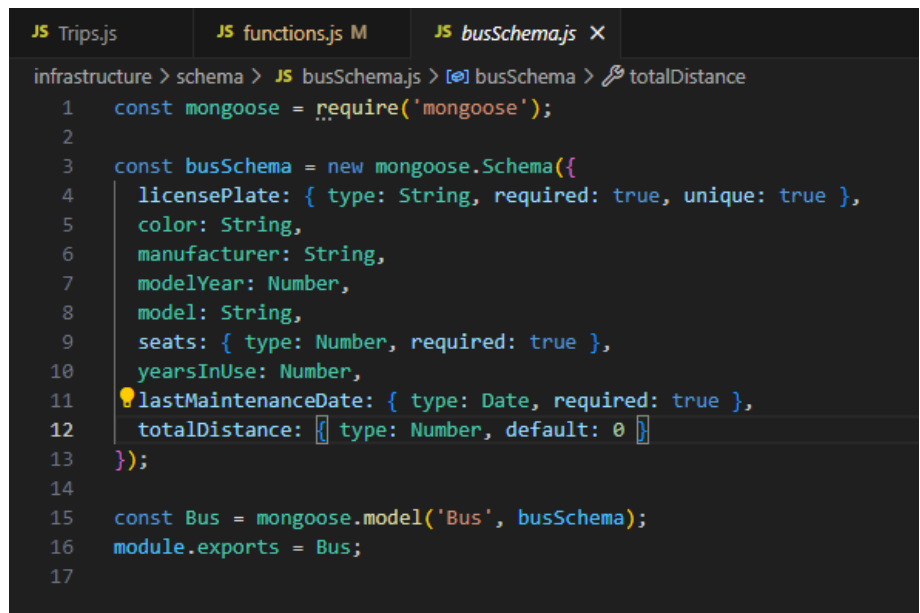
Các collections này giúp hệ thống quản lý một cách rõ ràng và mạch lạc thông tin liên quan đến xe, tài xế, tuyến đường, và chuyến đi. Mối quan hệ giữa các collections được thể hiện qua các trường tham chiếu (ObjectId), giúp kết nối dữ liệu giữa các đối tượng khác nhau trong hệ thống, tạo nên một cấu trúc cơ sở dữ liệu mạnh mẽ và linh hoạt.

2. Cấu trúc Schema

Trong hệ thống quản lý vận tải hành khách đường dài, cấu trúc dữ liệu được thiết kế dựa trên mô hình Schema của Mongoose. Schema trong Mongoose đóng vai trò định nghĩa cấu trúc dữ liệu cho các document trong MongoDB, xác định các trường, kiểu dữ liệu và các ràng buộc cần thiết. Dưới đây là mô tả chi tiết về cấu trúc Schema cho từng collection:

- ***Bus (Schema xe khách)***:
 - licensePlate: Chuỗi (String), yêu cầu nhập, duy nhất.
 - color: Chuỗi (String), tùy chọn.
 - manufacturer: Chuỗi (String), tùy chọn.
 - modelYear: Số (Number), tùy chọn.

- model: Chuỗi (String), tùy chọn.
- seats: Số (Number), yêu cầu nhập.
- yearsInUse: Số (Number), tùy chọn.
- lastMaintenanceDate: Ngày (Date), yêu cầu nhập.
- totalDistance: Số (Number), mặc định là 0.
- Tạo scheme trong NodeJs sử dụng thư viện mongoose:



```

JS Trips.js  JS functions.js M  JS busSchema.js X
infrastructure > schema > JS busSchema.js > [?] busSchema > totalDistance
1  const mongoose = require('mongoose');
2
3  const busSchema = new mongoose.Schema({
4    licensePlate: { type: String, required: true, unique: true },
5    color: String,
6    manufacturer: String,
7    modelYear: Number,
8    model: String,
9    seats: { type: Number, required: true },
10   yearsInUse: Number,
11   lastMaintenanceDate: { type: Date, required: true },
12   totalDistance: { type: Number, default: 0 }
13 });
14
15 const Bus = mongoose.model('Bus', busSchema);
16 module.exports = Bus;
17

```

- **Driver (Schema tài xế):**

- name: Chuỗi (String), yêu cầu nhập.
- cmt: Chuỗi (String), yêu cầu nhập.
- licenseNumber: Chuỗi (String), yêu cầu nhập.
- licenseType: Chuỗi (String), tùy chọn.
- address: Chuỗi (String), tùy chọn.
- dateOfBirth: Ngày (Date), tùy chọn.
- experienceYears: Số (Number), tùy chọn.

- Tạo scheme trong NodeJs sử dụng thư viện mongoose:

```

JS Trips.js  JS functions.js M  JS driverSchema.js X
infrastructure > schema > JS driverSchema.js > ...
1  const mongoose = require('mongoose');
2  const driverSchema = new mongoose.Schema({
3      name: { type: String, required: true },
4      cmt: { type: String, required: true },
5      licenseNumber: { type: String, required: true },
6      licenseType: String,
7      address: String,
8      dateOfBirth: Date,
9      experienceYears: Number
10 });
11
12 const Driver = mongoose.model('Driver', driverSchema);
13 module.exports = Driver;

```

- **Route (Schema tuyến đường):**

- startPoint: Chuỗi (String), yêu cầu nhập.
- endPoint: Chuỗi (String), yêu cầu nhập.
- distance: Số (Number), yêu cầu nhập.
- complexity: Số (Number), yêu cầu nhập, với các giá trị 1, 2, 3 (enum).
- Tạo scheme trong NodeJs sử dụng thư viện mongoose:

```

JS Trips.js  JS functions.js M  JS routeSchema.js X
infrastructure > schema > JS routeSchema.js > ...
1  const mongoose = require('mongoose');
2  const routeSchema = new mongoose.Schema({
3      startPoint: { type: String, required: true },
4      endPoint: { type: String, required: true },
5      distance: { type: Number, required: true },
6      complexity: { type: Number, required: true, enum: [1, 2, 3] }
7  });
8
9  const Route = mongoose.model('Route', routeSchema);
10 module.exports = Route;

```

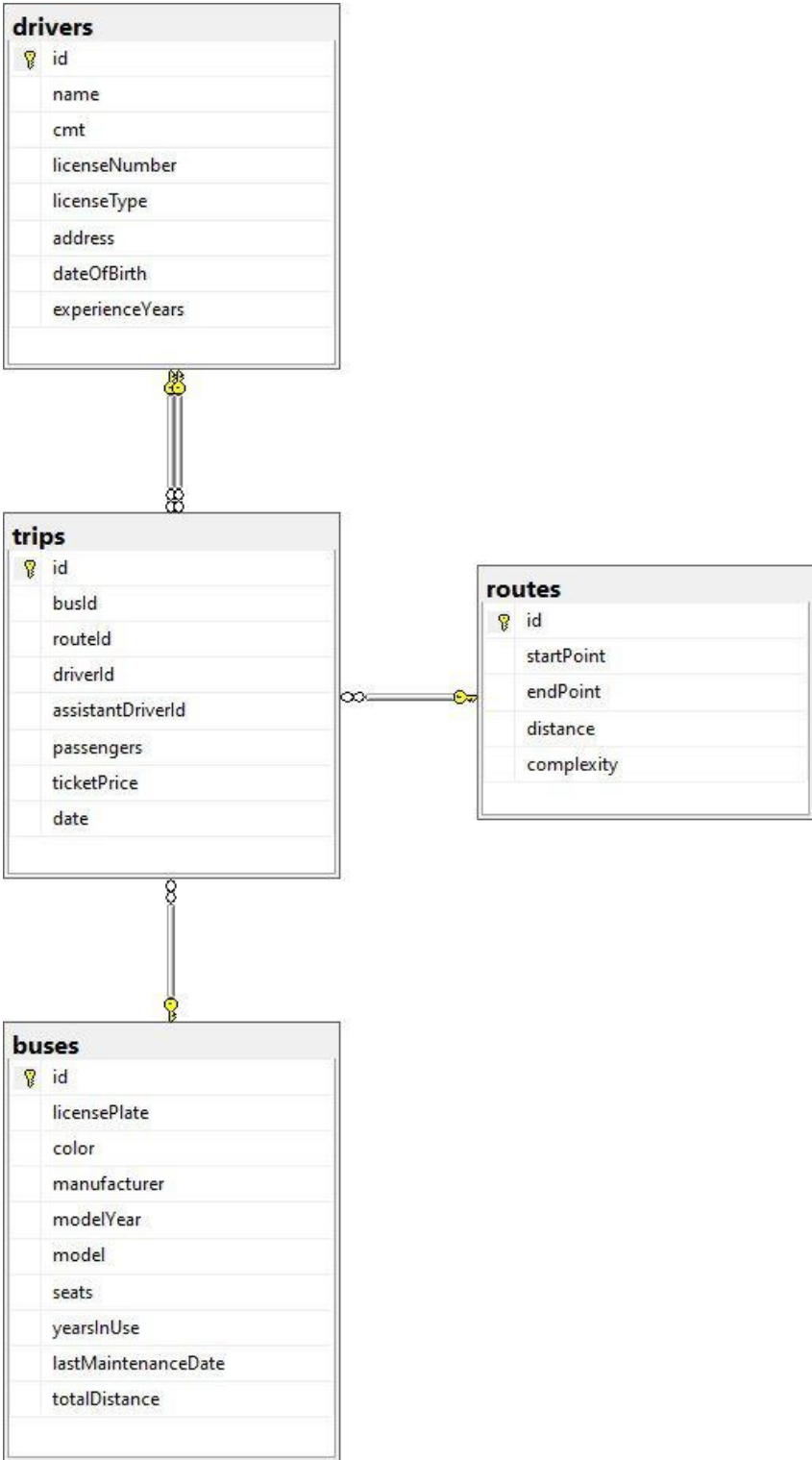
- **Trip (Schema chuyến xe):**

- bus: ObjectId tham chiếu tới Bus, yêu cầu nhập.
- route: ObjectId tham chiếu tới Route, yêu cầu nhập.

- driver: ObjectId tham chiếu tới Driver, yêu cầu nhập.
- assistantDriver: ObjectId tham chiếu tới Driver, yêu cầu nhập.
- passengers: Số lượng hành khách (Number), yêu cầu nhập.
- ticketPrice: Giá vé (Number), yêu cầu nhập.
- date: Ngày thực hiện chuyến đi (Date), yêu cầu nhập.
- Tạo scheme trong NodeJs sử dụng thư viện mongoose:

```
JS Trips.js JS functions.js M JS tripSchema.js X
infrastructure > schema > JS tripSchema.js > ...
1  const mongoose = require('mongoose');
2  const tripSchema = new mongoose.Schema({
3    bus: { type: mongoose.Schema.Types.ObjectId, ref: 'Bus', required: true },
4    route: { type: mongoose.Schema.Types.ObjectId, ref: 'Route', required: true },
5    driver: { type: mongoose.Schema.Types.ObjectId, ref: 'Driver', required: true },
6    assistantDriver: { type: mongoose.Schema.Types.ObjectId, ref: 'Driver', required: true },
7    passengers: { type: Number, required: true },
8    ticketPrice: { type: Number, required: true },
9    date: { type: Date, required: true }
10  });
11
12  const Trip = mongoose.model('Trip', tripSchema);
13  module.exports = Trip;
```

3. Mô hình liên kết các bảng



IV. CHỨC NĂNG CHÍNH

Hệ thống quản lý vận tải hành khách đường dài được xây dựng nhằm hỗ trợ việc quản lý và theo dõi các hoạt động vận tải, bao gồm việc tính lương tài xế, quản lý doanh thu và bảo dưỡng xe. Dưới đây là các chức năng chính của hệ thống:

1. Tính lương tài xế và phụ xe

Chức năng này cho phép hệ thống tự động tính toán lương cho tài xế và phụ xe dựa trên các chuyến đi mà họ thực hiện trong một tháng nhất định. Hệ thống sẽ xác định tài xế chính và phụ xe cho mỗi chuyến, từ đó áp dụng mức lương khác nhau cho từng người. Lương của tài xế chính được tính gấp đôi so với lương của phụ xe, và mức lương này phụ thuộc vào độ phức tạp của tuyến đường.

- **Dữ liệu cần thiết:** thông tin các chuyến đi (trips), tuyến đường (route) và tài xế (driver).
- **Kết quả:** Hiển thị danh sách lương của tất cả tài xế và phụ xe cho tháng hiện tại.

2. Tính doanh thu cho từng xe

Chức năng này giúp tính toán doanh thu cho mỗi xe dựa trên số lượng hành khách và giá vé cho từng chuyến đi trong một khoảng thời gian nhất định. Hệ thống sẽ lấy dữ liệu từ các chuyến đi, tính tổng doanh thu cho từng xe và hiển thị thông tin.

- **Dữ liệu cần thiết:** thông tin về chuyến đi (trips), xe khách (bus), và giá vé (ticket price).

- **Kết quả:** Hiện thị doanh thu của từng xe cho khoảng thời gian được yêu cầu.

3. Kiểm tra và hiển thị lịch bảo dưỡng

Chức năng này giúp theo dõi và quản lý thời gian bảo dưỡng xe dựa trên quãng đường mà xe đã di chuyển. Hệ thống sẽ kiểm tra các chuyến đi đã thực hiện và tính toán khoảng thời gian bảo dưỡng dựa trên độ phức tạp của tuyến đường và quãng đường di chuyển. Từ đó, hệ thống sẽ hiển thị các xe cần bảo dưỡng trong tương lai gần, cũng như các xe đã quá hạn bảo dưỡng.

- **Dữ liệu cần thiết:** thông tin các chuyến đi (trips), tuyến đường (route), và thời gian bảo dưỡng gần nhất của xe (last maintenance date).
- **Kết quả:** Hiện thị danh sách xe cần bảo dưỡng sắp tới và xe đã quá hạn bảo dưỡng.

V. DỮ LIỆU MẪU

1. Dữ liệu mẫu cho Bus

```
infrastructure > seedData > JS Busesjs > ...
1  const Bus = require('../schema/busSchema');
2
3  async function seedBuses() {
4      const buses = [
5          {
6              "licensePlate": "29A-12345",
7              "color": "Đỏ",
8              "manufacturer": "Hãng A",
9              "modelYear": 2020,
10             "model": "Model X",
11             "seats": 45,
12             "yearsInUse": 3,
13             "lastMaintenanceDate": "2024-01-07",
14             "totalDistance": 50000
15         },
16         {
17             "licensePlate": "30B-67890",
18             "color": "Xanh",
19             "manufacturer": "Hãng B",
20             "modelYear": 2018,
21             "model": "Model Y",
22             "seats": 40,
23             "yearsInUse": 5,
24             "lastMaintenanceDate": "2024-03-02",
25             "totalDistance": 75000
26         },
27         {
28             "licensePlate": "30B-67999",
29             "color": "Vàng",
30             "manufacturer": "Hãng C",
31             "modelYear": 2015,
32             "model": "Model Z",
33             "seats": 35,
34             "yearsInUse": 8,
35             "lastMaintenanceDate": "2024-05-04",
36             "totalDistance": 120000
37         }
38     ];
39
40     await Bus.insertMany(buses);
41     console.log('Buses seeded');
42 }
43 module.exports = seedBuses;
```

2. Dữ liệu mẫu cho Driver

```
infrastructure > seedData > JS Drivers.js > seedDrivers > drivers
1  const Driver = require('../schema/driverSchema');
2
3  async function seedDrivers() {
4      const drivers = [
5          {
6              "name": "Nguyễn Văn A",
7              "cmt": "123456789",
8              "licenseNumber": "DL123456",
9              "licenseType": "B2",
10             "address": "Hà Nội",
11             "dateOfBirth": "1985-05-15",
12             "experienceYears": 10
13         },
14         {
15             "name": "Trần Thị B",
16             "cmt": "987654321",
17             "licenseNumber": "DL987654",
18             "licenseType": "D",
19             "address": "Hà Nội",
20             "dateOfBirth": "1990-03-20",
21             "experienceYears": 5
22         },
23         {
24             "name": "Lê Văn C",
25             "cmt": "456123789",
26             "licenseNumber": "DL456123",
27             "licenseType": "B2",
28             "address": "Hà Nội",
29             "dateOfBirth": "1980-08-30",
30             "experienceYears": 15
31         }
32     ];
33
34     await Driver.insertMany(drivers);
35     console.log('Drivers seeded');
36 }
37 module.exports = seedDrivers;
38
```


3. Dữ liệu mẫu cho Route

```
infrastructure > seedData > JS Routes.js > seedRoutes > routes
1  const Route = require('../schema/routeSchema');
2
3  async function seedRoutes() {
4    const routes = [
5      {
6        "startPoint": "Hà Nội",
7        "endPoint": "Đà Nẵng",
8        "distance": 780,
9        "complexity": 2
10     },
11     {
12       "startPoint": "Hồ Chí Minh",
13       "endPoint": "Nha Trang",
14       "distance": 450,
15       "complexity": 1
16     },
17     {
18       "startPoint": "Hải Phòng",
19       "endPoint": "Nam Định",
20       "distance": 120,
21       "complexity": 3
22     }
23   ];
24
25   await Route.insertMany(routes);
26   console.log('Routes seeded');
27 }
28 module.exports = seedRoutes;
29
30
```

4. Dữ liệu mẫu cho Trip

```
infrastructure > seedData > JS Trips.js > ...
1  const Trip = require('../schema/tripSchema');
2  const Bus = require('../schema/busSchema');
3  const Driver = require('../schema/driverSchema');
4  const Route = require('../schema/routeSchema');
5
6  async function seedTrips() {
7    const bus1 = await Bus.findOne({ licensePlate: '29A-12345' });
8    const driver1 = await Driver.findOne({ name: 'Nguyễn Văn A' });
9    const assistantDriver1 = await Driver.findOne({ name: 'Trần Thị B' });
10
11    const bus2 = await Bus.findOne({ licensePlate: '30B-67890' });
12    const driver2 = await Driver.findOne({ name: 'Trần Thị B' });
13    const assistantDriver2 = await Driver.findOne({ name: 'Nguyễn Văn A' });
14
15    const bus3 = await Bus.findOne({ licensePlate: '30B-67999' });
16    const driver3 = await Driver.findOne({ name: 'Lê Văn C' });
17    const assistantDriver3 = await Driver.findOne({ name: 'Nguyễn Văn A' });
18
19    const route1 = await Route.findOne({ startPoint: 'Hà Nội', endPoint: 'Đà Nẵng' });
20    const route2 = await Route.findOne({ startPoint: 'Hồ Chí Minh', endPoint: 'Nha Trang' });
21    const route3 = await Route.findOne({ startPoint: 'Hải Phòng', endPoint: 'Nam Định' });
22
23    const trips = [
24      {
25        "bus": bus1.id,
26        "route": route1.id,
27        "driver": driver1.id,
28        "assistantDriver": assistantDriver1.id,
29        "passengers": 40,
30        "ticketPrice": 150000,
31        "date": "2024-09-05"
32      },
33      {
34        "bus": bus2.id,
35        "route": route2.id,
36        "driver": driver2.id,
37        "assistantDriver": assistantDriver2.id,
38        "passengers": 30,
39        "ticketPrice": 200000,
40        "date": "2024-09-12"
41      },
42      {
43        "bus": bus3.id,
44        "route": route3.id,
45        "driver": driver3.id,
46        "assistantDriver": assistantDriver3.id,
47        "passengers": 25,
48        "ticketPrice": 80000,
49        "date": "2024-09-15"
50      }
51    ];
52
53    await Trip.insertMany(trips);
54    console.log('Trips seeded');
55  }
56  module.exports = seedTrips;
```

VI. THỰC THI VÀ KẾT QUẢ

Trong phần này, chúng tôi sẽ trình bày kết quả thực thi các chức năng chính của hệ thống, bao gồm tính lương tài xế, tính doanh thu từ các chuyến đi và kiểm tra tình trạng bảo dưỡng của xe.

1. Kết quả tính lương tài xế

Kết quả tính lương cho các tài xế và phụ xe trong tháng hiện tại đã được tính toán dựa trên số chuyến đi và mức lương theo độ khó của từng tuyến đường. Dưới đây là kết quả chi tiết:

Lương của tài xế, phụ xe cho tháng hiện tại:	
Tên: Nguyễn Văn A,	Lương: 3000000
Tên: Trần Thị B,	Lương: 1500000
Tên: Lê Văn C,	Lương: 5000000

Lưu ý: Mức lương cho tài xế là gấp đôi mức lương của phụ xe. Tùy thuộc vào số lượng chuyến đi và độ khó của các tuyến đường mà lương sẽ khác nhau.

2. Kết quả tính doanh thu

Doanh thu từ các chuyến đi trong khoảng thời gian từ ngày 01/09/2024 đến ngày 30/09/2024 đã được tính toán như sau:

Doanh thu từ 2024-09-01 đến 2024-09-30:	
BKS: 29A-12345,	Doanh thu: 5400000
BKS: 30B-67890,	Doanh thu: 8100000
BKS: 30B-67999,	Doanh thu: 2000000

Giải thích: Doanh thu được tính dựa trên số lượng hành khách và giá vé cho mỗi chuyến đi.

3. Kết quả kiểm tra bảo dưỡng xe

Hệ thống đã kiểm tra tình trạng bảo dưỡng của các xe. Kết quả như sau:

Xe bảo dưỡng sắp tới:	
BKS: 29A-12345	Ngày bảo dưỡng sắp tới: 2025-01-07
BKS: 30B-67890	Ngày bảo dưỡng sắp tới: 2025-03-02
Xe quá hạn bảo dưỡng:	
BKS: 30B-67999	Ngày bảo dưỡng kế tiếp: 2024-05-04

Lưu ý: Hệ thống sẽ cảnh báo về những xe cần bảo dưỡng sớm và những xe đã quá hạn bảo dưỡng, giúp quản lý bảo trì xe một cách hiệu quả.

VII. CÔNG CỤ VÀ MÔI TRƯỜNG PHÁT TRIỂN

Trong phần này, chúng tôi sẽ trình bày về các công cụ và môi trường phát triển được sử dụng để xây dựng và triển khai hệ thống quản lý hàng vận tải hành khách đường dài.

1. Cấu hình Docker cho MongoDB

Docker được sử dụng để tạo môi trường ảo hóa cho cơ sở dữ liệu MongoDB. Việc sử dụng Docker giúp chúng tôi dễ dàng triển khai và quản lý cơ sở dữ liệu mà không cần phải cài đặt trực tiếp trên hệ thống. Các bước cấu hình MongoDB trong Docker như sau:

- 1.1. *Cài đặt Docker Desktop*: Trước tiên, Docker Desktop cần được cài đặt trên máy tính của chúng tôi. Docker Desktop hỗ trợ cả Windows và macOS.
- 1.2. *Tạo một file Docker Compose*: File này sẽ cấu hình cho MongoDB. Dưới đây là nội dung cơ bản của file docker-compose.yml:

```
version: '3.8'
services:
  mongodb:
    image: mongo:latest
    container_name: mongodb
    ports:
      - "27017:27017"
    volumes:
      - mongodb_data:/data/db
volumes:
  mongodb_data:
```

- 1.3. *Chạy Docker Compose*: Chạy lệnh sau trong terminal để khởi động MongoDB:

```
docker-compose up -d
```

- 1.4. *Kiểm tra trạng thái:* Sử dụng lệnh sau để kiểm tra xem container đã chạy thành công hay chưa:

```
docker ps
```

Bằng cách này, chúng tôi đã cấu hình thành công MongoDB chạy trong một container Docker, giúp dễ dàng quản lý và khôi phục cơ sở dữ liệu khi cần thiết.

2. Sử dụng Navicat Premium để truy cập dữ liệu

Navicat Premium là một công cụ quản lý cơ sở dữ liệu mạnh mẽ, cho phép chúng tôi dễ dàng truy cập và quản lý dữ liệu trong MongoDB. Dưới đây là các bước để sử dụng Navicat Premium:

- 2.1. ***Cài đặt Navicat Premium:*** Tải và cài đặt Navicat Premium từ trang chính thức của nhà phát triển.
- 2.2. ***Tạo kết nối đến MongoDB:***
 - Mở Navicat Premium và chọn "Connection".
 - Chọn "MongoDB" từ danh sách các loại kết nối.
 - Nhập các thông tin cần thiết như địa chỉ IP (localhost nếu chạy trên máy local), cổng (thường là 27017), và các thông tin xác thực nếu cần.
- 2.3. ***Kết nối đến cơ sở dữ liệu:*** Sau khi tạo kết nối, nhấn "Connect" để truy cập vào cơ sở dữ liệu MongoDB. Chúng tôi có thể xem và quản lý các collections, thực hiện truy vấn, cũng như nhập hoặc xuất dữ liệu một cách dễ dàng.

2.4. ***Quản lý dữ liệu:*** Navicat Premium cho phép chúng tôi thực hiện các thao tác như thêm, sửa, xóa bản ghi, cũng như thực hiện các truy vấn phức tạp để kiểm tra và phân tích dữ liệu trong hệ thống.

Sử dụng Navicat Premium giúp chúng tôi tiết kiệm thời gian và công sức trong việc quản lý và giám sát cơ sở dữ liệu, đồng thời cải thiện hiệu quả công việc trong quá trình phát triển ứng dụng.

VIII. KẾT LUẬN

1. Kết quả đạt được

Dự án đã hoàn thành với những kết quả quan trọng sau:

- ***Xây dựng hệ thống cơ sở dữ liệu:*** Chúng tôi đã thiết kế và triển khai một cơ sở dữ liệu MongoDB với các collections đáp ứng đầy đủ yêu cầu của hệ thống. Các schema được xây dựng rõ ràng, giúp quản lý thông tin về xe, tài xế, tuyến đường, và chuyến xe một cách hiệu quả.
- ***Phát triển ứng dụng Node.js:*** Ứng dụng được phát triển bằng Node.js đã hoàn thiện các chức năng chính, bao gồm tính toán lương tài xế, doanh thu của các xe trong một khoảng thời gian nhất định, và kiểm tra tình trạng bảo dưỡng của xe. Điều này không chỉ đáp ứng yêu cầu chức năng mà còn đảm bảo hiệu suất và khả năng mở rộng cho hệ thống.
- ***Quản lý và giám sát dữ liệu hiệu quả:*** Việc sử dụng Docker để triển khai MongoDB và Navicat Premium để quản lý cơ sở dữ liệu đã giúp chúng tôi dễ dàng giám sát, bảo trì và nâng cấp hệ thống mà không gặp khó khăn. Điều này tạo điều kiện thuận lợi cho việc phát triển và mở rộng trong tương lai.

2. Hướng phát triển tiếp theo

Mặc dù dự án đã hoàn thành các yêu cầu ban đầu, chúng tôi nhận thấy một số hướng phát triển có thể được thực hiện để nâng cao giá trị của hệ thống:

- **Mở rộng chức năng:** Có thể phát triển thêm các chức năng như quản lý vé điện tử, theo dõi lịch trình chuyển xe theo thời gian thực, và tích hợp các phương thức thanh toán trực tuyến để cải thiện trải nghiệm của khách hàng.
- **Tối ưu hóa hiệu suất:** Cần xem xét tối ưu hóa các truy vấn đến cơ sở dữ liệu để nâng cao tốc độ và hiệu suất của ứng dụng, đặc biệt là trong trường hợp số lượng người dùng và dữ liệu tăng cao.
- **Phát triển giao diện người dùng:** Việc phát triển một giao diện người dùng thân thiện và dễ sử dụng sẽ giúp người dùng tương tác với hệ thống một cách hiệu quả hơn. Có thể xây dựng một ứng dụng web hoặc ứng dụng di động để quản lý và theo dõi các chuyến xe dễ dàng.
- **Phân tích dữ liệu:** Triển khai các công cụ phân tích dữ liệu để có cái nhìn sâu sắc hơn về hiệu suất hoạt động của hãng vận tải, từ đó đưa ra các quyết định chiến lược tốt hơn.

Tổng kết lại, dự án đã đạt được những kết quả đáng khích lệ và có nhiều cơ hội để phát triển trong tương lai, nhằm mang lại giá trị cao hơn cho khách hàng và cải thiện hoạt động của hệ thống quản lý hãng vận tải hành khách.