**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC MỎ - ĐỊA CHẤT**

**PHẠM TRỌNG NGHĨA**

**CHUYÊN NGÀNH MẠNG MÁY TÍNH**



**PHẠM TRỌNG NGHĨA**

ĐỒ ÁN TỐT NGHIỆP

**NGÀNH CÔNG NGHỆ THÔNG TIN**

**TÊN ĐỀ TÀI:**

**XÂY DỰNG PHẦN MỀM QUẢN LÝ SẢN XUẤT RƯỢU RUKISO**

**KHÓA 65**

HÀ NỘI - 2025

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC MỎ - ĐỊA CHẤT**

ĐỒ ÁN TỐT NGHIỆP

**CHUYÊN NGÀNH: MẠNG MÁY TÍNH**

GIẢNG VIÊN HƯỚNG DẪN SINH VIÊN THỰC HIỆN

**THS. NGUYỄN TUẤN ANH PHẠM TRỌNG NGHĨA**

HÀ NỘI - 2025

# Lời mở đầu

Trong bối cảnh cách mạng công nghiệp 4.0 và kinh tế nước nhà đang phát tiển như hiện nay, việc ứng dụng công nghệ thông tin vào các ngành nghề truyền thống đang ngày càng trở nên cần thiết để bắt kịp xu hướng phát triển đất nước cũng như phát triển kinh tế địa phương. Sản xuất rượu các sản phẩm thủ công truyền thống hay rượu làng nghề không chỉ là một nét văn hóa đặc trưng mà còn là một ngành nghề mang lại thu nhập cho người dân. Tại các vùng quê thuộc huyện Kim Sơn, tỉnh Ninh Bình công việc sản xuất sản phẩm rượu gạo này vẫn được duy trì, phát triển và là kế sinh nhai của nhiều hộ gia đình. Tuy nhiên, quá trình sản xuất và kinh doanh rượu tại các hộ gia đình đơn lẻ này còn đang sơ sài, hiện nay vẫn còn gặp nhiều khó khăn trong việc quản lý nguyên liệu, theo dõi quy trình sản xuất, thống kê tài chính, và kiểm soát kho hàng. Những vấn đề này gây ra không ít thách thức trong việc nâng cao hiệu quả hoạt động, duy trì chất lượng sản phẩm và giảm khả năng cạnh tranh trên thị trường. Việc áp dụng công nghệ thông tin vào quản lý sản xuất rượu giúp hiện đại hóa quy trình sản xuất, tăng hiệu quả cạnh tranh trên thị trường mà vẫn giữ lại được nét mộc mạc truyền thống của quê hương.

Đề tài "Xây dựng phần mềm quản lý sản xuất rượu RuKiSo" ra đời với mục tiêu giải quyết những khó khan kể trên. Phần mềm này được thiết kế nhằm hỗ trợ việc quản lý nhập xuất các sản phẩm cũng như nguyên liệu đầu vào, theo dõi các mẻ ủ rượu và thống kê tài chính chi tiết một cách có hệ thống. Với những tính năng này, RuKiSo không chỉ giúp tăng cường hiệu quả quản lý mà còn đóng góp vào việc chuẩn hóa và hiện đại hóa quy trình sản xuất rượu thủ công, rượu làng nghề.

Đối tượng nghiên cứu của đề tài là quy tình sản xuất và kinh doanh tại các hộ dân sản xuất rượu tại huyện Kim Sơn, tỉnh Ninh Bình với quy mô nhỏ lẻ và quy trình lạc hậu. Phạm vi nghiên cứu của đề tài bao gồm các nghiệp vụ quản lý từ các nguyên liệu đầu vào, quy trình sản xuất cho tới sản phẩm đầu ra và thống kê tài chính.

Với việc ứng dụng các công nghệ hiện đại như .NET API, SQL Server và .NET MAUI, phần mềm không chỉ mang lại hiệu quả quản lý cao mà còn dễ sử dụng, phù hợp với người dân địa phương và dễ dàng nâng cấp để chạy trên mọi nền tảng sau này.

Đề tài kỳ vọng góp phần hiện đại hóa quy trình sản xuất rượu truyền thống, nâng cao giá trị kinh tế và bảo tồn nét đẹp văn hóa của nghề làm rượu Kim Sơn. Kết quả đạt được bao gồm một phần mềm quản lý sản xuất rượu hoạt động ổn định, giao diện thân thiện, dễ sử dụng với người Việt Nam và tích hợp các tính năng cần thiết để hỗ trợ cho các cơ sở sản xuất. RuKiSo kỳ vọng trở thành một công cụ hữu ích, góp phần nâng cao hiệu quả sản xuất kinh doanh và bảo tồn giá trị văn hóa của sản phẩm rượu Kim Sơn.

# Chương 1: Mô tả bài toán

## Mục tiêu của bài toán

Mục tiêu chính của đề tài “Xây dựng phần mềm quản lý sản xuẩ rượu RuKiSo” là tạo ra một phần mềm giúp quản lý và tối ưu hóa quy trình sản xuất rượu thủ công, rượu làng nghề tại các hộ gia đình ở huyện Kim Sơn, tỉnh Ninh Bình.

Đối tượng người dùng được RuKiSo hướng tới là những người không quen sử dụng công nghê nên muc tiêu của phần mềm còn là cung cấp giao diện trực quan sinh động, dễ sử dụng và thân thiện. Phần mềm này giúp cho người dùng quản lý tốt hơn các nguyên liệu đầu vào cũng như sản phẩm đầu ra, theo dõi và nhắc nhở quá trình lên men, tạo ra hệ thống báo cáo, thống kê tài chính góp phần nâng cao hiệu quả sản xuất và kinh doanh, giúp đỡ người dân trong phát triển kinh tế vào bảo tồn truyền thống quy báu.

## Các yêu cầu đặt ra của bài toán

### Yêu cầu về dữ liệu

Để đáp ứng được mục tiêu quản lý toàn diện quy trình sản xuất và kinh doanh, phần mềm RuKiSo cần xử lý và quản lý một lượng dữ liệu không quá phức tạp nhưng mang tính hệ thống và rõ ràng. Dữ liệu này đóng vai trò cốt lõi trong việc hỗ trợ người dùng thực hiện các nghiệp vụ quản lý hàng ngày một cách hiệu quả và chính xác. Các loại dữ liệu cụ thể mà phần mềm cần xử lý bao gồm:

* **Quản lý danh mục nguyên liệu đầu vào**: Tên nguyên liệu, đơn vị tính, ngày nhập
* Quản lý danh mục sản phẩm: Tên sản phẩm, giá bán, số lượng
* **Ghi nhận các thông tin tài chính**: Số vốn, doanh thu từ bán hàng
* **Quản lý quy trình sản xuất**: Ngày bắt đầu ủ, ngày sản xuất, sản lượng
* **Thống kê kinh doanh:** Số lượng và giá trị của sản phẩm bán được hoặc các nguyên liệu đầu vào

### Yêu cầu về chức năng xử lý

Mục tiêu của RuKiSo là hiện đại hóa và tối ưu hóa quy trình sản xuất rượu, phần mềm RuKiSo cần cung cấp các chức năng xử lý không chỉ đáp ứng được nhu cầu quản lý cơ bản mà còn hỗ trợ người dùng cải thiện hiệu suất công việc. Các chức năng này phải được thiết kế một cách khoa học, tập trung vào sự đơn giản, dễ sử dụng và khả năng mang lại giá trị thực tế cao cho người dùng. Dưới đây là các yêu cầu chức năng chính mà phần mềm cần phải đáp ứng:

* Thống kê các sản phẩm bán chạy, nguyên liệu được sử dụng nhiều
* Quản lý nguyên liệu: Thêm, sửa, xóa nguyên liệu và theo dõi tồn kho
* Quản lý sản phẩm: Thêm, sửa, xóa sản phẩm và theo dõi tồn kho
* Quản lý quy trình sản xuất: Tạo vào theo dõi lịch sản xuất, nhắc nhờ ghi đến thời gian hoàn thành
* Thống kê tài chính: hiện thị doanh thu, chi phí theo tuần, tháng
* Tương tác người dùng: Giao diện thân thiện dễ sử dụng, biểu đồ trực quan

### Yêu cầu về quy trình xử lý

Một phần mềm quản lý hiệu quả không chỉ dừng lại ở việc cung cấp các chức năng cần thiết mà còn phải đảm bảo quy trình xử lý thông tin diễn ra mượt mà, chính xác và thuận tiện. Quy trình xử lý đóng vai trò quan trọng trong việc quyết định trải nghiệm người dùng, độ chính xác của dữ liệu và khả năng duy trì hoạt động ổn định của phần mềm. Vì vậy, RuKiSo cần thiết kế quy trình xử lý rõ ràng, chặt chẽ và thân thiện với người dùng, từ khâu nhập liệu đến khâu xử lý và xuất thông tin, nhằm đạt được hiệu quả tối ưu trong việc vận hành hệ thống. Các yêu cầu bao gồm:

* Quy trình nhập liệu đơn giản, dễ sử dụng và có thông báo nếu người dùng nhập chưa đúng
* Phần mềm phải chính xác và hoạt động ổn định

## Mô tả các nguồn tài liệu, dữ liệu, biểu mẫu liên quan

### Nguồn tài liệu

Các nguồn tài liệu phục vụ cho việc nghiên cứu và xây dựng phần mềm quản lý sản xuất rượu RuKiSo được thu thập từ thực tế sản xuất và kinh nghiệm trong ngành sản xuất rượu truyền thống của chính gia đình, ngoài ra các ý kiến đóng góp và kinh nghiệm tham khảo cũng như học hỏi từ các hộ - cơ sở sản xuất khác cũng là một nguồn tài liệu quý giá để phát triển phần mềm này:

* **Kinh nghiệm gia đình**: Gia đình em có nhiều năm kinh nghiệm và nhiều thế hệ hoạt động trong lĩnh việc sản xuất rượu truyền thống. Với việc sản xuất để tự sử dụng trong gia đình trong thời gian đầu sau đó là phát triển trở thành công việc kinh doanh đem lại thu nhập cũng như bảo tồn nét văn hóa truyền thống của làng nghề rượu Kim Sơn. Quá trình này mất nhiều năm nhưng cũng đem lại các kiến thức sâu sắc về các bước sản xuất, các nguyên liệu và quản lý một cách thủ công.
* **Học hỏi từ các hộ sản xuất khác**: Trong suốt quá trình hình thành và phát triển thương hiệu rượu Kim Sơn, việc học hỏi trao đổi và khảo sát giữa các hộ sản xuất là một hoạt động cần thiết để ngành nghề này được duy trì và phát triển. Các kiến thức và khó khan gặp phải cũng là nhưng nguồn tài liệu vô cùng quý giá để định hướng và phát triển phẩn mềm
* **Tài liệu hướng dẫn phát triển phần mềm**: Các tài liệu về lập trình về .NET MAUI .NET API và SQL Server cùng các kiến thức lập trình được giảng dạy trên trường giúp xây dựng và triển khai một cách nhanh chóng và hiệu quả

### Dữ liệu thực nghiệm

Dữ liệu thực nghiệm đóng vai trò quan trọng trong việc đảm bảo tính chính xác và phù hợp của phần mềm quản lý sản xuất rượu RuKiSo. Các dữ liệu này được thu thập từ thực tế sản xuất của gia đình và từ các hộ sản xuất khác trong khu vực, kết hợp với những bộ dữ liệu giả lập nhằm mô phỏng và kiểm tra các chức năng của phần mềm. Việc sử dụng dữ liệu thực nghiệm không chỉ phản ánh đúng thực trạng mà còn giúp phần mềm đáp ứng tốt nhu cầu quản lý thực tế của người dùng. Dưới đây là các nguồn dữ liệu cụ thể được sử dụng trong dự án.

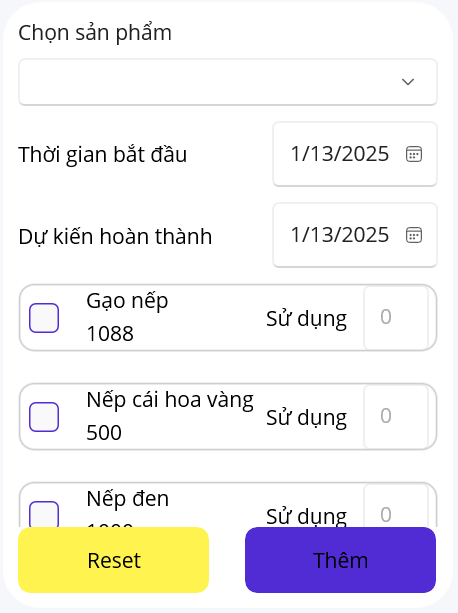
* **Dữ liệu từ sản xuất thực tế của gia đình**: Các ghi chép bao gồm các loại nguyên liệu (gạo nếp, men rượu), các giai đoạn sản xuất, sản lượng cũng như giá cả của các sản phẩm trong kho hang.
* **Dữ liệu tham khảo từ các hộ kinh doanh khác**: Cách họ quản lý, theo dõi quy trình sản xuất và tính toán chi phí trong kinh doanh
* **Dữ liệu giả lập để kiểm tra phần mềm**: Tạo ra bộ dữ liệu giả lập dựa trên kinh nghiệm thực tiển để kiểm tra các chức năng của phần mềm

### 1.3.3 Mẫu biểu

Mẫu biểu là yếu tố không thể thiếu để hỗ trợ việc thu thập, tổ chức và trình bày dữ liệu một cách trực quan, khoa học trong phần mềm quản lý sản xuất rượu RuKiSo. Các mẫu biểu này được thiết kế dựa trên thực tế hoạt động sản xuất và kinh doanh tại các hộ gia đình, nhằm đảm bảo tính tiện dụng và dễ hiểu cho người dùng. Các biểu mẫu cũng là cơ sở để phần mềm hiển thị thông tin và hỗ trợ người dùng trong việc theo dõi quy trình sản xuất, quản lý tồn kho và thống kê tài chính.

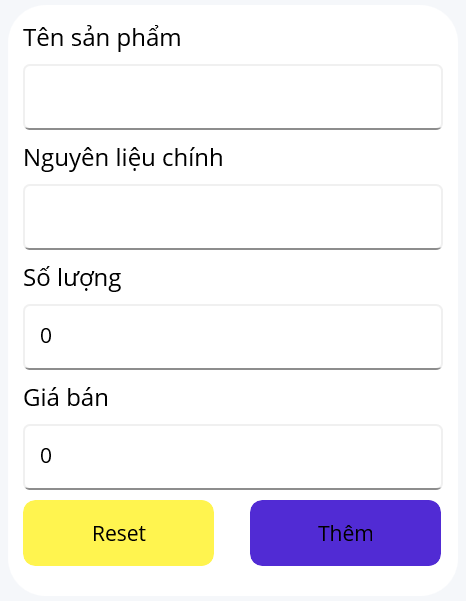
Dưới đây là một số loại mẫu biểu tiêu biểu được sử dụng trong dự án:

* Mẫu biểu tạo một mẻ rượu để theo dõi quy trình sản xuất. Mẫu biểu gồm có loại rượu thành phẩm, ngày tháng bắt đầu và dự kiến kết thúc ,các nguyên liệu có trong kho, được sử dụng bằng cách chọn và nhập số lượng



Hình 1-1 Mẫu biểu tạo mới mẻ rượu

* Mẫu biểu thêm mới sản phẩm bao gồm tên sản phẩm, các nguyên liệu chính, số lượng và giá bán



Hình 1-2 Mẫu biểu tạo mới sản phẩm

## Phân tích, đánh giá hiện trạng và những vấn đề còn tồn tại

### Hiện trạng sản xuất và kinh doanh rượu tại địa phương

Tại huyện Kim Sơn, tỉnh Ninh Bình, các hộ gia đình sản xuất rượu gạo truyền thống vẫn đang duy trì quy trình sản xuất lâu đời, mang đậm nét văn hóa làng nghề. Tuy nhiên, hoạt động sản xuất và kinh doanh chủ yếu diễn ra theo cách thủ công với các đặc điểm:

* Chủ yếu sử dụng sổ sách giấy hoặc trí nhớ để quản lý nguyên liệu đầu vào và theo dõi các mẻ ủ rượu.
* Việc ghi nhận và theo dõi thu chi chủ yếu dựa trên kinh nghiệm, không có hệ thống cụ thể

### Những vấn đề còn tồn tại

Các vấn đề nếu trên ở hiện trạng sản xuất và kinh doanh khiến cho việc phát triển lâu dài gặp nhiều khó khan, không bắt kịp xu hướng hiện tại. Việc quản lý thủ công như vậy dẫn đến nhiều sai sót, thiếu chính xác trong việc kiểm soát tồn kho và lịch trình sản xuất, khó khan trong việc đánh giá hiệu quả kinh doanh, giảm chất lượng và sự đồng đều của sản phẩm. Tóm lại vấn đề hiện tại của các hộ sản xuất rượu truyền thống đang phải đối mặt bao gồm:

* Thiếu công cụ quản lý sổ sách, kho
* Khó khăn trong việc thống kê tài chính
* Khả năng cạnh tranh thấp do sản phẩm chưa đồng đều về chất lượng

### Những vấn đề cần tập trung nghiên cứu giải quyết

Đề tài “Xây dựng phần mềm quản lý sản xuất rượu RuKiSo” tập chung giải quyết các vấn đề được nêu trên bằng cách:

* **Hỗ trợ quản lý nguyên liệu và sản phẩm**: Cung cấp hệ thống nhập, xuất, thêm, sửa, xóa và theo dõi tồn kho của cả nguyên liệu và sản phẩm
* **Quản lý quy trình sản xuất**: Tạo công cụ theo dõi lịch trình sản xuất và nhắc nhở mốc thời gian quan trọng giúp đồng bộ chất lượng sản phẩm đầu ra
* **Thông kê và báo cáo tài chính**: Xây dựng hệ thống báo cáo doanh thu, chi phí, lợi nhuận, hỗ trợ người dùng đánh giá hiệu quả kinh doanh
* **Tăng cường tương tác người dùng**: Thiết kế giao diện thuần Việt, trức quan và dễ sử dụng

# Chương 2: Cơ sở lý thuyết và cơ sở thực nghiệm về vấn đề cần giải quyết

## 2.1. Cơ sở lý thuyết

### 2.1.1. Kiến trúc Client-Server

Mô hình client-server là một kiến trúc phổ biến trong phát triển phần mềm, nơi:

* **Client:** Đóng vai trò giao diện người dùng, chịu trách nhiệm gửi yêu cầu (request) và hiển thị dữ liệu nhận được từ server.
* **Server:** Đóng vai trò xử lý logic nghiệp vụ, quản lý cơ sở dữ liệu và cung cấp API để client tương tác.

### 2.1.2. .NET API

.NET API là một nền tảng mạnh mẽ để xây dựng các dịch vụ backend:

ASP.NET Core Web API: Được sử dụng để triển khai server-side. Đây là nền tảng mã nguồn mở, hiệu suất cao, hỗ trợ phát triển API RESTful.

Các điểm nổi bật:

* **Dependency Injection (DI):** Quản lý sự phụ thuộc giữa các thành phần, giúp code dễ kiểm thử và bảo trì.
* **Routing linh hoạt:** Cung cấp cơ chế định tuyến URL rõ ràng và dễ cấu hình.
* **Entity Framework Core:** Dùng để tương tác với cơ sở dữ liệu.

### 2.1.3. .NET MAUI

.NET MAUI (Multi-platform App UI) là nền tảng phát triển ứng dụng đa nền tảng (cross-platform) cho phép chia sẻ code giữa các hệ điều hành (Windows, Android, iOS, macOS):

* **MVVM (Model-View-ViewModel):** Là một mẫu thiết kế giúp tách biệt giao diện người dùng (View) khỏi logic nghiệp vụ (Model), thông qua một lớp trung gian gọi là ViewModel. Điều này giúp ứng dụng dễ bảo trì, mở rộng và kiểm thử.
* **Model**: Đại diện cho dữ liệu và logic nghiệp vụ, có thể bao gồm các lớp tương tác với cơ sở dữ liệu, API hoặc các dịch vụ khác. Model không chứa bất kì tham chiếu nào đến giao diện người dùng.
* **ViewModel**: Cầu nối giữa View và Model chưa dữ liệu mà view cần hiển thị, ở đây dữ liệu được sử lý logic tương tác, sự kiện hoặc gọi các dịch vụ và cập nhật trên view
* **View:** Thành phần giao diện người dùng kết nối với viewmodel thông qua cơ chế binding data, không chưa logic nghiệp vụ.
* **Data binding:** Là một cơ chế giúp dữ liệu từ view được cập nhât đến viewmodel hoặc ngược lại. Binding gồm nhiều mode và kiểu binding khác nhau giúp đơn giản hóa công việc.
* **Command binding:** Dùng để xử lý các sự kiện ở viewmodel thay vì viết trong code-behind của view điều này giúp tách biệt logic và giảm sự phụ thuộc của các thành phần trong một chương trình.

### 2.1.4. Cơ chế giao tiếp giữa client và server

Client và server giao tiếp qua giao thức HTTP/HTTPS bằng cách sử dụng các phương thức của RESTful API:

**GET**: Lấy dữ liệu (ví dụ: danh sách sản phẩm, nguyên liệu).

**POST**: Gửi dữ liệu mới (ví dụ: thêm giao dịch).

**PUT/PATCH**: Cập nhật dữ liệu (ví dụ: sửa thông tin sản phẩm).

**DELETE**: Xóa dữ liệu (ví dụ: xóa giao dịch).

### 2.1.5. Quản lý trạng thái với MVVM

Trong client sử dụng .NET MAUI, mô hình MVVM giúp:

* **Tách biệt logic và giao diện**: Dữ liệu được quản lý trong ViewModel thông qua các cơ chế binding.
* **Command và Event**: Thao tác người dùng (như bấm nút) được xử lý qua ICommand.
* **ObservableCollection**: Danh sách dữ liệu được cập nhật tự động mỗi khi có thay đổi, giúp giao diện luôn phản ánh trạng thái mới nhất.

## 2.2. Cơ sở thực nghiệm

### 2.2.1. Kiến trúc hệ thống

Hệ thống được chia thành hai phần chính:

* **Server:** Xây dựng bằng ASP.NET Core Web API được kết nối với cơ sở dữ liệu qua Entity Framework Core, dữ liệu được seed để kiểm tra các tính năng ngoài ra các API được thiết kế theo chuẩn RESTful, bao gồm các endpoint như:
  + /api/products: Quản lý sản phẩm.
  + /api/ingredients: Quản lý nguyên liệu.
  + /api/transactions: Quản lý giao dịch.
  + /api/batches: Quản lý nguyên liệu.
* **Client**: Xây dựng bằng .NET MAUI ap dụng mẫu thiết kế MVVM để quản lý giao diện và logic. Sử dụng HttpClient để gửi và nhận dữ liệu từ API. Các giao diện chính bao gồm:
  + Thông tin và danh sách sản phẩm (Products).
  + Thông tin và danh sách nguyên liệu (Ingredients).
  + Thông tin và danh sách các mẻ rượu đang ủ (Batches)
  + Quản lý giao dịch (Transactions).

### 2.2.2. Tương tác giữa client và server

* **Truyền dữ liệu**: Client gửi yêu cầu HTTP với JSON payload đến server, server phản hồi với dữ liệu JSON tương ứng.
* **Xử lý lỗi**: Client hiển thị popup thông báo lỗi khi có các vấn đề khi thêm sửa xóa dữ liệu

### 2.2.3. Môi trường thực nghiệm

Công nghệ sử dụng:

* **Server**: ASP.NET Core Web API, Entity Framework Core, SQL Server.
* **Client**: .NET MAUI.
* **Công cụ phát triển**: Visual Studio 2022, Git, SourceTree
* **Cơ sở dữ liệu**: SQL Server

Kiểm thử chức năng:

* Thêm, sửa, xóa sản phẩm từ client và kiểm tra dữ liệu trên server.
* Thêm, sửa, xóa nguyên liệu từ client và kiểm tra dữ liệu trên server.
* Thêm, sửa, xóa mẻ ủ rượu từ client và kiểm tra dữ liệu trên server.
* Thêm, sửa, xóa giao dịch từ client và kiểm tra dữ liệu trên server.
* Hiện thị các dữ liệu thống kê dưới dạng biểu đồ

### 2.2.4. Kết quả thực nghiệm

Client có giao diện đẹp mắt, dễ dàng sử dụng và thân thiện với người dùng. Ngoài ra còn hoạt động mượt mà, hiển thị đúng thông tin được gửi lên từ server.

Mô hình MVVM giúp dễ bảo trì và nâng cấp sau này

API phản hồi đúng chuẩn REST, cho phép client và server giao tiếp hiệu quả.

## 2.3 Lý do chọn công nghệ, framework

Là một trong nhưng tập đoàn công nghê lớn trong ngành công nghệ toàn cầu, Microsoft luôn luôn phát triển và hỗ trợ cho các công nghệ của mình một cách toàn diện nhất. Với hệ sinh thái rộng lớn và nền tảng công nghệ khổng lồ mà họ đang có, việc xây dựng phần mềm sử dụng các công nghệ do Microsoft phát triển và hỗ trợ trở nên dễ dàng hơn rất nhiều.

### 2.3.1 ASP.NET Core Web API cho server

ASP.NET Core Web API là một phần của ASP.NET Core, một framework mã nguồn mở, đa nền tảng được phát triển bởi Microsoft. ASP.NET Core Web API được sử dụng để xây dựng các dịch vụ HTTP RESTful, cho phép các ứng dụng khác nhau giao tiếp và trao đổi dữ liệu với nhau qua giao thức HTTP với các ưu điểm như:

* **Hiệu suất cao**: ASP.NET Core được tối ưu hóa để xử lý lượng lớn yêu cầu với thời gian phản hồi nhanh. Điều này quan trọng với các hệ thống client-server có khả năng mở rộng.
* **Đa nền tảng**: ASP.NET Core hoạt động trên Windows, Linux và macOS, cho phép triển khai server linh hoạt trong các môi trường khác nhau.
* **Hỗ trợ Dependency Injection (DI):** Đây là một tính năng tích hợp mạnh mẽ giúp giảm sự phụ thuộc giữa các thành phần, hỗ trợ tốt hơn cho các nguyên tắc SOLID và lập trình hướng đối tượng.
* **Tích hợp tốt với Entity Framework Core:** Cung cấp khả năng tương tác trực tiếp với cơ sở dữ liệu bằng cách sử dụng mô hình đối tượng (ORM), giúp giảm thiểu thời gian viết truy vấn SQL thủ công.
* **Hệ sinh thái phong phú:** ASP.NET Core hỗ trợ tích hợp các công cụ như Swagger (để tạo tài liệu API), Identity (quản lý xác thực người dùng), và SignalR (giao tiếp thời gian thực).
* **Bảo mật mạnh mẽ:** Tích hợp sẵn các cơ chế bảo mật như CORS, JWT, OAuth2, giúp bảo vệ API khỏi các lỗ hổng bảo mật phổ biến.

Ngoài các ưu điểm kể trên việc sử dụng ASP.NET Core Web API còn có một số lợi ích khác như có thể phát triển nhanh chóng nhờ các công cụ hỗ trợ và các thư việc mạnh mẽ đi kèm, có thể xây dựng các dịch vụ hỗ trợ tùy chỉnh với các yêu cầu khác nhau và cuối cùng nhưng quan trọng nhất là framework này là một sản phẩm của Microsoft nên có một cộng đồng hỗ trợ rất lớn và mạnh mẽ.

### 2.3.2 .NET MAUI và mẫu thiết kế MVVM cho client

**.NET MAUI (Multi-platform App UI)** là một nền tảng phát triển ứng dụng đa nền tảng hiện đại của Microsoft, được thiết kế để xây dựng các ứng dụng giao diện người dùng (UI) trên nhiều hệ điều hành khác nhau, bao gồm **Windows**, **Android**, **iOS**, và **macOS**. Là sự kế thừa của Xamarin.Forms, .NET MAUI không chỉ cải tiến mà còn tối ưu hóa khả năng phát triển ứng dụng, giúp lập trình viên dễ dàng chia sẻ code và tạo ra các trải nghiệm người dùng nhất quán trên mọi nền tảng. Được giới thiệu vào năm 2021 đây là một công nghệ rất mới nhưng đã được cộng đồng phát triển phần mềm đón nhận và yêu thích.

## 2.4 Vấn đề cần giải quyết

Trong quá trình nghiên cứu và phân tích hiện trạng, đề tài "Xây dựng phần mềm quản lý sản xuất rượu RuKiSo" hướng đến giải quyết những vấn đề cốt lõi sau đây:

### 2.4.1. Quản lý nguyên liệu và kho hàng

Việc quản lý nguyên liệu đầu vào như gạo nếp, men rượu, nước và các vật tư khác vẫn còn thực hiện theo cách thủ công, gây khó khăn trong việc theo dõi số lượng tồn kho để có hướng đối ứng cho công việc sản xuất. Vấn đề cần giải quyết là tự động theo dõi tồn kho, hệ thống cần cập nhật số lượng nguyên liệu mỗi khi có giao dịch nhập hoặc xuất điều này giúp cho quá trình sản xuất không bị gián đoạn vì thiếu nguyên liệu do yếu kém về mặt quản lý.

### 2.4.2. Quản lý quy trình sản xuất rượu

Việc sản xuất rượu truyền thống có các giai đoạn cần theo dõi như ủ men, lên men. Tuy nhiên, do không có công cụ hỗ trợ, nhiều hộ sản xuất không kiểm soát tốt các bước này, dẫn đến chất lượng sản phẩm không đồng đều ngoài ra còn xảy ra vấn đề nhớ sai ngày nấu rượu làm hỏng mẻ ủ rượu gây lãng phí tốn kém chi phí và công sức. Những vấn đề cần giải quyết:

* Tạo và theo dõi các mẻ rượu: Mỗi mẻ rượu có ngày bắt đầu, dự kiến hoàn thành và trạng thái hiện tại.
* Nhắc nhở sản xuất: Hệ thống gửi thông báo khi đến thời điểm chuyển bước từ ủ men đến lúc có thể chưng cất.

### 2.4.3. Quản lý giao dịch bán hàng và tài chính

Việc ghi nhận doanh thu, chi phí và lợi nhuận chủ yếu dựa trên ghi chép sổ sách hoặc trí nhớ, dễ dẫn đến sai sót gây khó khan hoặc sai sót trong việc tính toán lợi nhuận. Những vấn đề cần giải quyết:

* Ghi nhận doanh số bán hàng: Mỗi giao dịch được lưu trữ với thông tin chi tiết (ngày bán, số lượng, giá trị).
* Tính toán lợi nhuận: Tự động tính toán doanh thu, chi phí nguyên liệu và lợi nhuận theo từng tuần/tháng.
* Xuất báo cáo tài chính: Cung cấp báo cáo doanh thu, lợi nhuận dưới dạng biểu đồ trực quan.

### 2.4.4. Tích hợp với hệ thống lưu trữ dữ liệu SQL Server

Với lượng dữ liệu ngày càng lớn, cần có một cơ sở dữ liệu mạnh mẽ và ổn định giúp cho việc lưu trữ các thông tin trở nên dễ dàng và an toàn hơn. Những vấn đề cần giải quyết:

* Tối ưu truy vấn dữ liệu: Sử dụng Entity Framework để xử lý nhanh các truy vấn đến database liên quan đến kho hàng, giao dịch.
* Hỗ trợ mở rộng: Thiết kế hệ thống có thể dễ dàng mở rộng khi số lượng giao dịch tăng lên.

Tóm lại, các vấn đề cần giải quyết của đề tài bao gồm:

* Quản lý nguyên liệu và kho hàng – Theo dõi số lượng tồn kho, hạn sử dụng, dự báo nhập nguyên liệu.
* Quản lý quy trình sản xuất rượu – Ghi nhận từng giai đoạn sản xuất, nhắc nhở sản xuất.
* Quản lý giao dịch bán hàng và tài chính – Tự động ghi nhận doanh thu, chi phí, lợi nhuận và xuất báo cáo.
* Cải thiện trải nghiệm người dùng – Thiết kế UI trực quan, hỗ trợ tìm kiếm, truy cập đa nền tảng.
* Tích hợp với SQL Server để đảm bảo hiệu suất – Lưu trữ dữ liệu hiệu quả, hỗ trợ mở rộng hệ thống.

# Chương 3: Phương pháp, quy trình thực hiện giải quyết bài toán

## 3.1 Phương pháp tiếp cận

Phát triển phần mềm là quá trình thiết kế, lập trình, kiểm thử và bảo trì phần mềm để đáp ứng nhu cầu của người dùng. Dựa vào các vấn đề cần giải quyết và các yêu cầu đặt ra được đề cập ở các chương trước đó các giải đoạn phát triển RuKiSo bao gồm các giao đoạn:

* Khảo sát và phân tích yêu cầu
* Thiết kế hệ thống
* Phát triển
* Kiểm thử

## 3.2 Quy trình thực hiện

### 3.2.1 Khảo sát và phân tích yêu cầu

Dựa vào kinh nghiệm sản xuất rượu của gia đình ngoài ra các hộ sản xuất xung quanh cũng được phỏng vấn trực tiếp để thu tập nghiệp vụ quản lý hiện tại. Kết quả nhận được có đặc điểm chung là 100% được quản lý bằng kinh nghiệp, không qua giấy tờ hay tính toán thống kê. Điều này chỉ ra yếu điểm trong quy trình sản xuất hiện tại là thiếu hệ thống theo dõi tồn kho cho nguyên liệu và sản phẩm đầu ra, sai sót hoặc không có việc tính toán tài chính rõ rang, đây vừa là yếu điểm vừa là rào cản lớn trong việc giữ gìn và phát triển thương hiệu rượu Kim Sơn truyền thống này.

Từ kết quả khảo sát và các yếu điểm được phân tích trên, có thể rút ra được các yêu cầu của phần mềm để khắc phục được chia thành 2 loại yêu cầu chính:

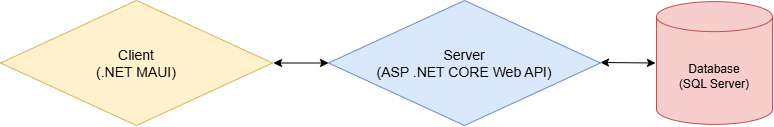
* Yêu cầu chức năng:
  + Quản lý kho bao gồm nguyên liệu và thành phẩm: CRUD (thêm/sửa/xóa/xem)

#### Theo dõi các mẻ ủ rượu: Tạo mẻ, nhắc nhở đến ngày chưng cất

* + Thống kê tài chính: Tạo biểu đồ doanh thu theo tuần/tháng
* Yêu cầu phi chức nănng
  + Hiệu suất: Phần mềm không được giật lác, hoạt động trơn tru mượt mà
  + Khả năng nâng cấp và mở rộng sau này: Có thể nâng cấp để hoạt động trên các thiết bị di động sau này

### 3.2.2 Thiết kế hệ thống

Phần mềm RuKiSo được thiết kế theo mô hình Client-Server với ưu điểm là quản lý tập trung và có khả năng mở rộng dễ dàng cho phép bổ sung các tính năng khác một cách dễ dàng trong tương lai. Kiến trúc tổng thể:

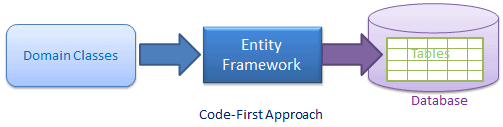


Hình 3-1 Mô hình phần mềm RuKiSo

Cụ thể, các thành phần bao gồm

#### ***3.2.2.1 Cơ sở dữ liệu***

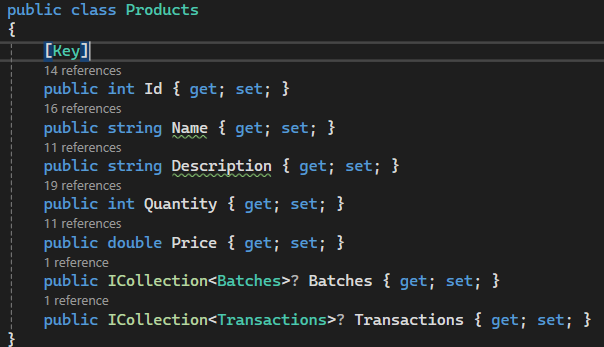
Cơ sở dữ liệu là thành phần quan trọng dùng để lưu trữ thông tin được gửi từ người dùng thông qua server, dữ liệu đã xử lí và được lưu trữ tập trung ở đây. Với RuKiSo Entity Framework Core (Code-First) là phương pháp được sử dụng để tạo các bảng

****

Hình 3-2 Mô hình hoạt động Entity Framework

Dựa vào yêu cầu bài toán đặt ra cơ sở dữ liệu RuKiSo bao gồm 5 bảng được ánh xạ với code c# thông qua Entity Framework bằng 5 lớp sau:

* Lớp Products biểu thị bảng sản phẩm lưu trữ các thông tin như: Id, tên sản phẩm, mô tả, số lượng, giá, cùng các khóa ngoại tới các bảng khác



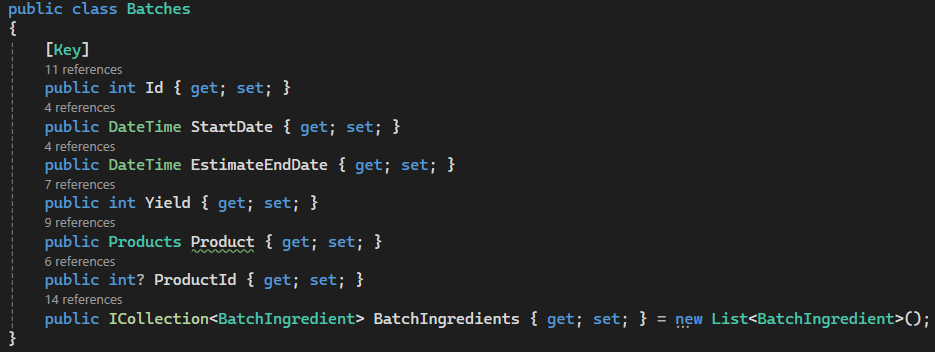
Hình 3-3 Lớp Products

* Lớp Ingredients biểu thị bảng nguyên liệu trong cơ sở dữ liệu lưu trữ các thông tin như Id, tên, đơn vị, số lượng và giá nhập.



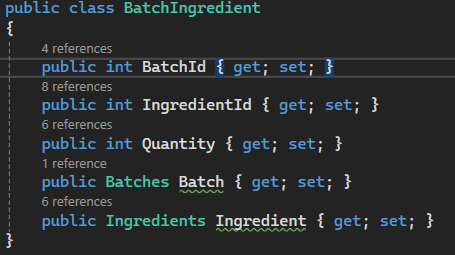
Hình 3-4 Lớp Ingredients

* Lớp Batches biểu thị thông tin của các mẻ rượu chờ lên men, lưu trữ các dữ liệu như: Id, ngày bắt đầu và dự kiến kết thúc, sản lượng sau khi ra thành phẩm, sản phẩm tạo ra và các nguyên liệu tạo thành.



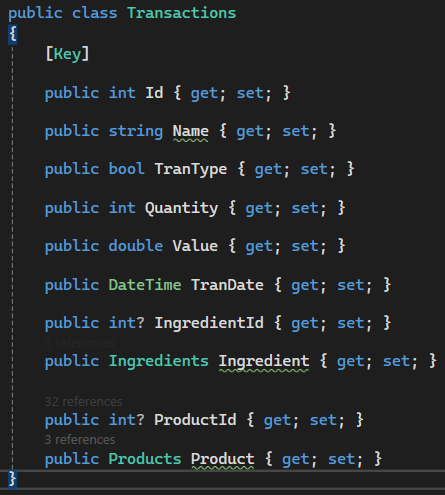
Hình 3- 5 Lớp Batches

* Lớp BatchIngredient lưu trữ thông tin gồm BatchId, Id của nguyên liệu, số lượng sử dụng.



Hình 3- 6 Lớp BatchIngredient

* Lớp Transactions lưu trữ các thông tin giao dịch bao gồm: Id, tên giao dịch, loại giao dịch (nhập/xuất) số lượng và giá trị,…



Hình 3-7 Lớp Transactions

Với các lớp trên ta cần thêm các câu lệnh dưới đây để tạo ra các mối quan hệ cho các bảng:

public partial class RuKiSoDataContext : DbContext

{

public RuKiSoDataContext(DbContextOptions contextOptions) : base(contextOptions) { }

public DbSet<Products> Products { get; set; }

public DbSet<Ingredients> Ingredients { get; set; }

public DbSet<Batches> Batches { get; set; }

public DbSet<Transactions> Transactions { get; set; }

public DbSet<BatchIngredient> BatchIngredients { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Transactions>(entity =>

{

entity.HasKey(t => t.Id);

entity.HasOne(t => t.Ingredient)

.WithMany(i => i.Transactions)

.HasForeignKey(t => t.IngredientId)

.IsRequired(false)

.OnDelete(DeleteBehavior.SetNull);

entity.HasOne(t => t.Product)

.WithMany(p => p.Transactions)

.HasForeignKey(t => t.ProductId)

.IsRequired(false)

.OnDelete(DeleteBehavior.SetNull);

});

modelBuilder.Entity<Products>(entity =>

{

entity.HasKey(p => p.Id);

entity.HasMany(p => p.Batches)

.WithOne(b => b.Product)

.HasForeignKey(b => b.ProductId)

.OnDelete(DeleteBehavior.SetNull);

});

modelBuilder.Entity<Ingredients>(entity =>

{

entity.HasKey(i => i.Id);

});

modelBuilder.Entity<Batches>(entity =>

{

entity.HasKey(b => b.Id);

});

modelBuilder.Entity<BatchIngredient>(entity =>

{

entity.HasKey(bi => new { bi.BatchId, bi.IngredientId });

entity.HasOne(bi => bi.Batch)

.WithMany(b => b.BatchIngredients)

.HasForeignKey(bi => bi.BatchId);

entity.HasOne(bi => bi.Ingredient)

.WithMany(i => i.BatchIngredients)

.HasForeignKey(bi => bi.IngredientId);

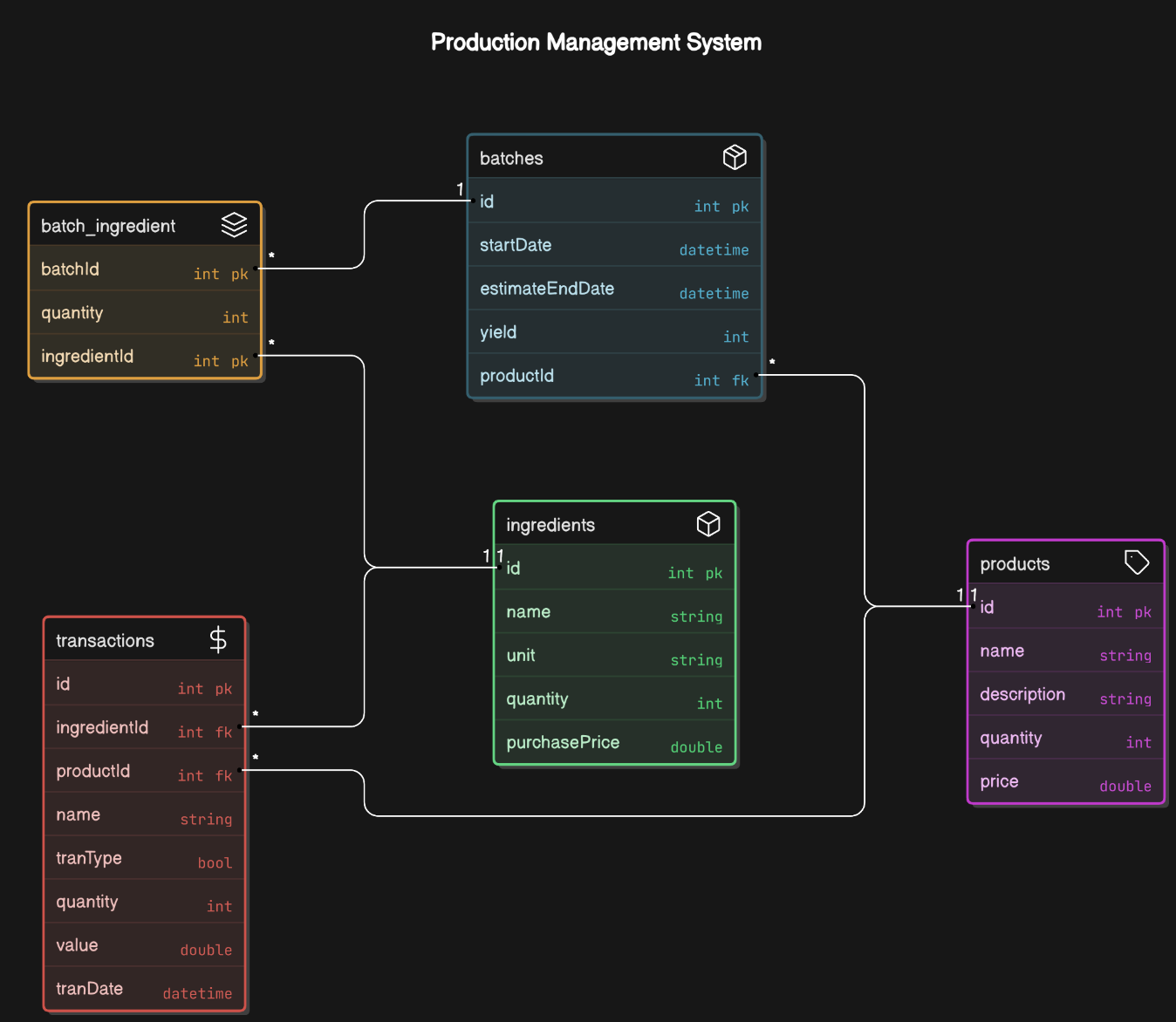
});

SeedData(modelBuilder);

}

}

Kết quả đạt được khi thực hiện migration từ code sang database có sơ đồ như sau:

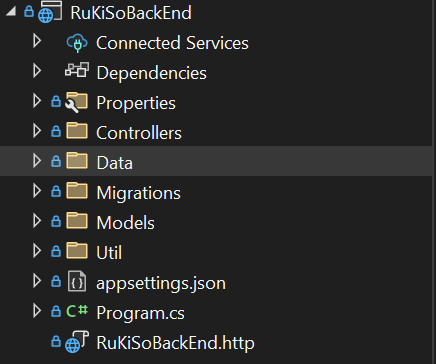


Hình 3-8 Sơ đồ cơ sở dữ liệu

#### 3.2.2.2 Các API endpoint

Các API endpoint của phần mềm RuKiSo được xây dựng dựa trên nền tảng ASP .NET CORE Web API, các dịch vụ HTTP được tạo ra có thể được truy cập từ bất kì ứng dụng nào từ client bao gồm cả trình duyệt, desktop app hay ứng dụng di động. ASP.NET Web API hỗ trợ tất cả các phương thức HTTP (GET, POST, PUT, DELETE) giúp giảm thời gian phát triển. Các thành phần của Server RuKiSo cũng được xây dựng theo kiến trúc bao gồm:

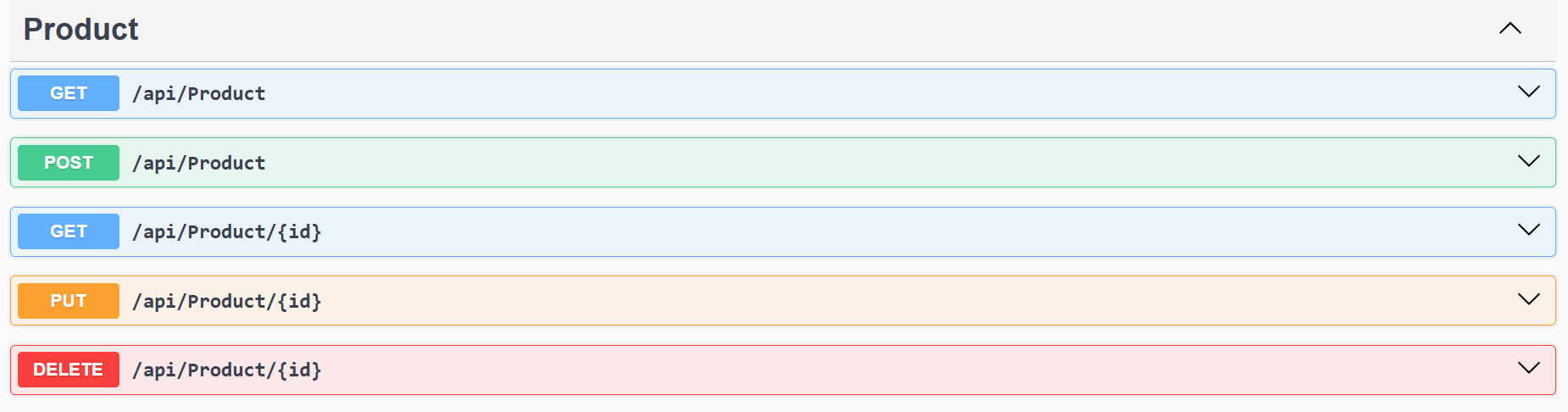
* **Model**: Đại diện cho dữ liệu của ứng dụng. Các mô hình này thường được sử dụng để chuyển đổi dữ liệu từ nguồn như cơ sở dữ liệu thành các đối tượng có thể sử dụng trong ứng dụng.
* **Controller**: Đây là thành phần trung tâm xử lý các yêu cầu HTTP. Mỗi phương thức trong Controller có thể xử lý một loại hành động HTTP như GET, POST, PUT hoặc DELETE. Controller tương tác với các Model để lấy dữ liệu, xử lý nó và trả về cho client dưới dạng JSON hoặc XML.
* **Data**: Lưu trữ class đại diện cho Database giúp tương tác với dữ liệu thông qua object được tạo.
* **Migration**: Thể hiện các trạng thái của database trong từng thời điểm nhất định, giúp backup thông database.
* **Routing**: ASP.NET Web API sử dụng cơ chế định tuyến tương tự như MVC để ánh xạ các URL tới các Controller. Định tuyến giúp xác định Controller nào sẽ xử lý yêu cầu dựa trên URL.
* **Dependency Injection (DI)**: ASP.NET Web API hỗ trợ tích hợp chặt chẽ với DI, giúp dễ dàng quản lý các dịch vụ bên ngoài và các lớp phụ thuộc.



Hình 3-9 Cấu trúc Server RuKiSo

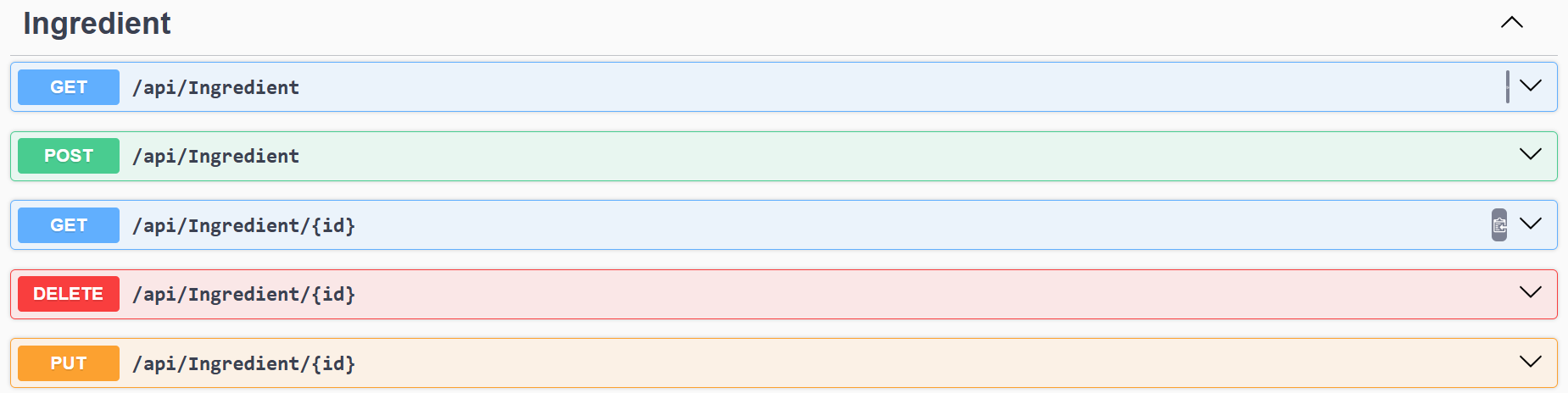
Dựa vào các yêu cầu đề ra trước đó các API enpoint bao gồm:

* /api/Product cung cấp các enpoint tương tác dữ liệu với sản phẩm



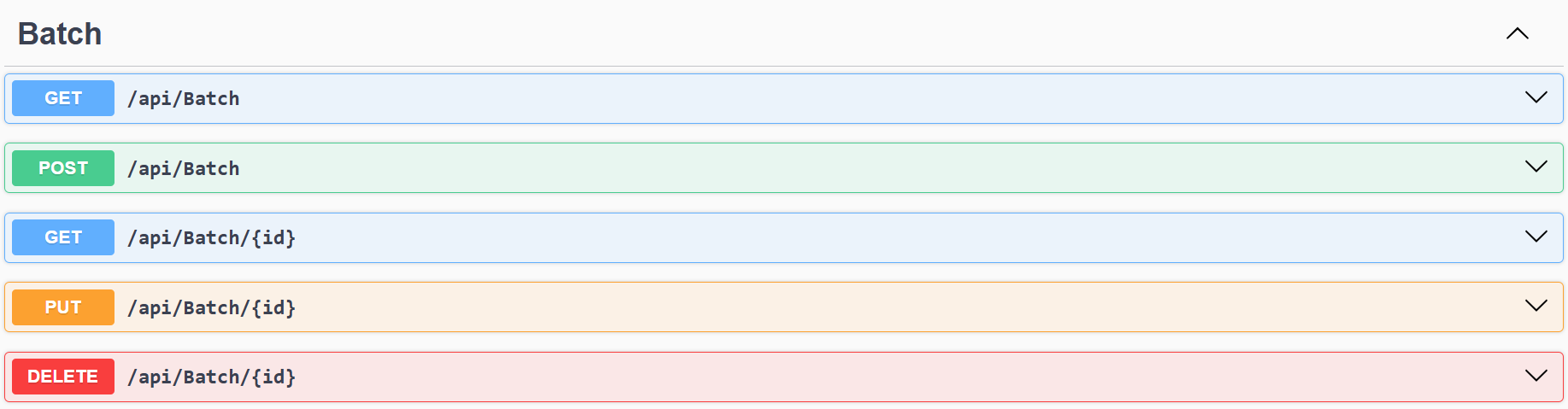
Hình 3-10 Các API của sản phẩm

* /api/Ingredient cung cấp các enpoint tương tác dữ liệu với nguyên liệu



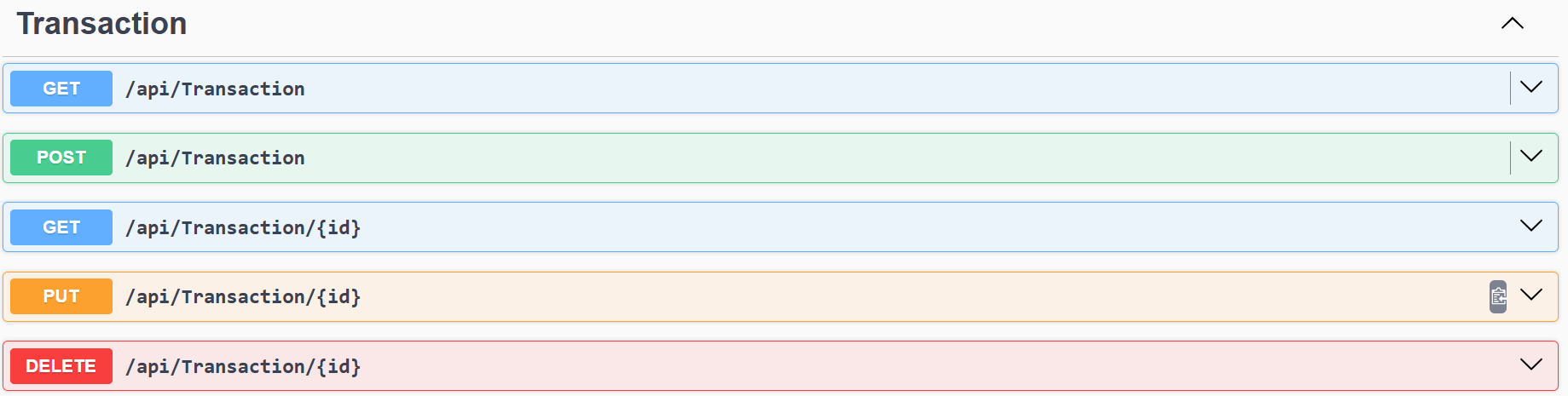
Hình 3-11 Các API của nguyên liệu

* /api/Batch cung cấp các enpoint tương tác dữ liệu với mẻ ủ



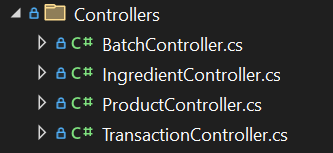
Hình 3-12 Các API của mẻ ủ rượu

* /api/Transaction cung cấp các enpoint tương tác dữ liệu với giao dịch



Hình 3-13 Các API của giao dịch

Cụ thể cấu trúc và source code của các api được xây dựng từ các lớp trong controllers như hình dưới đây:



Hình 3- 14 Cấu trúc controller

Souce code của Product:

namespace RuKiSoBackEnd.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class ProductController : ControllerBase

{

private readonly RuKiSoDataContext dbContext;

public ProductController(RuKiSoDataContext dbContext)

{

this.dbContext = dbContext;

}

[HttpGet]

public async Task<IActionResult> GetAll()

{

var domainProducts = await dbContext.Products.ToListAsync();

var products = domainProducts.Select(product => product.ToDTO()).ToList();

return Ok(products);

}

[HttpGet]

[Route("{id:int}")]

public async Task<IActionResult> GetById([FromRoute] int id)

{

var domainProduct = await dbContext.Products.FirstOrDefaultAsync(p => p.Id == id);

if (domainProduct == null)

return NotFound();

var product = domainProduct.ToDTO();

return Ok(product);

}

[HttpPost]

public async Task<IActionResult> Create([FromBody] ProductRequestAPI productRequest)

{

var domainProduct = productRequest.ToDomain();

await dbContext.Products.AddAsync(domainProduct);

await dbContext.SaveChangesAsync();

var productResponse = domainProduct.ToDTO();

return CreatedAtAction(nameof(GetById), new { id = productResponse.Id }, productResponse);

}

[HttpPut]

[Route("{id:int}")]

public async Task<IActionResult> Update([FromRoute] int id, [FromBody] ProductRequestAPI productRequest)

{

var domainProduct = await dbContext.Products.FirstOrDefaultAsync(p => p.Id == id);

if (domainProduct == null)

return NotFound();

domainProduct.Name = productRequest.Name;

domainProduct.Description = productRequest.Description;

domainProduct.Price = productRequest.Price;

domainProduct.Quantity = productRequest.Quantity;

await dbContext.SaveChangesAsync();

return Ok(domainProduct.ToDTO());

}

[HttpDelete]

[Route("{id:int}")]

public async Task<IActionResult> Delete([FromRoute] int id)

{

var domainProduct = await dbContext.Products.FirstOrDefaultAsync(p => p.Id == id);

if (domainProduct == null)

return NotFound();

dbContext.Products.Remove(domainProduct);

await dbContext.SaveChangesAsync();

return Ok();

}

}

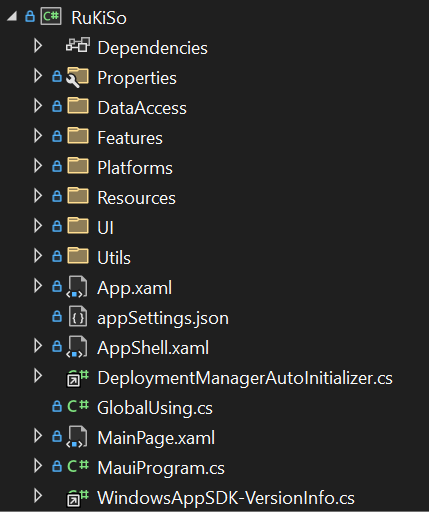
}

Các controller khác có cấu trúc và logic hoạt động tương tự.

#### 3.2.2.3 Client MAUI

.NET Multi-platform App UI (.NET MAUI) là một framework đa nền tảng để tạo ứng dụng di động và desktop gốc bằng ngôn ngữ lập trình C# và XAML. Là một dự án mã nguồn mở và là sự tiến hóa của Xamarin.Forms hiệu suất và tính mở rộng của MAUI đã được cải thiện rất nhiều. Ứng dụng này sử dụng XAML là viết tắt của Extensible Application Markup Language, một loại ngôn ngữ dùng để thiết kế giao diện (GUI) cho các ứng dụng dựa trên Windows, đặc biệt là của Microsoft Framework. XAML kết hợp với C# có thể xây dựng lên các logic UI hay chuyển động từ đơn giản tới phức tạp.

Để đảm bảo khả năng bảo trì và nâng cấp sau này, Client MAUI được xây dựng dựa trên MVVM Desgin pattern là một mẫu thiết kế đặc biệt cho các ứng dụng sử dụng ngôn ngữ XAML trên giao diện. Lợi ích của MVVM la tách biệt logic và việc thiết kế giao diện nâng cao khả năng mở rộng, cấu trúc của client có dạng như hình dưới đây:



Hình 3-15 Cấu trúc Client MAUI

Các thành phần chính trong ứng dụng RuKiSo bao gồm:

* **View:**  là thành phần giao diện của ứng dụng. Tương tự như trong mô hình MVC, View là thành phần duy nhất mà người dùng tương tác trực tiếp. Tuy nhiên, trong MVVM, View chủ động hơn, thực hiện các hành vi và phản hồi lại người dùng thông qua tính năng binding và command. RuKiSo bao gồm 5 trang chính và các thông báo hiển thị dưới dạng popup. Các trang được tạo bởi ContentPage bao gồm các thành phần control như CollectionView để tạo ra danh sách hiển thị các item, Label để hiển thị các nhãn dán chưa text, Button cho các nút bấm,… Các page có dạng và cấu trúc tương tự như trang sản phẩm dưới đây:

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage

x:Class="RuKiSo.Views.ProductPage"

xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:controls="clr-namespace:RuKiSo.UI.Views"

xmlns:resource="clr-namespace:RuKiSo.Resources"

xmlns:ui="clr-namespace:RuKiSo.UI"

xmlns:vm="clr-namespace:RuKiSo.ViewModels"

x:DataType="{x:Type vm:ProductViewModel}">

<Grid

BackgroundColor="{x:Static ui:AppColors.PrimaryBackGround}"

ColumnDefinitions="\*"

RowDefinitions="auto, \*">

<controls:Header

Title="{x:Static resource:AppTexts.Products}"

HorizontalOptions="Fill"

ReminderViewModel="{Binding ReminderViewModel}" />

<Grid

Grid.Row="1"

Padding="20,8,20,8"

RowDefinitions="120,\*">

<Grid ColumnDefinitions="300,\*,300,\*,300">

<controls:OverViewCard

Title="{x:Static resource:AppTexts.TotalProduct}"

CustomWidthRequest="{x:Static ui:AppDimens.Height300}"

DesciptionText="{Binding TotalProduct}"

ImageSource="{x:Static ui:StaticImage.TotalProduct}" />

<controls:OverViewCard

Title="{x:Static resource:AppTexts.TotalValue}"

Grid.Column="2"

CustomWidthRequest="{x:Static ui:AppDimens.Height300}"

DesciptionText="{Binding TotalValue, StringFormat={x:Static resource:AppTexts.MoneyFormat}}"

ImageSource="{x:Static ui:StaticImage.TotalValue}" />

<controls:OverViewCard

Title="{x:Static resource:AppTexts.EstimatedProfit}"

Grid.Column="4"

CustomWidthRequest="{x:Static ui:AppDimens.Height300}"

DesciptionText="{Binding EstimatedProfit, StringFormat={x:Static resource:AppTexts.MoneyFormat}}"

ImageSource="{x:Static ui:StaticImage.EstimatedProfit}" />

</Grid>

<Grid

Grid.Row="1"

Margin="0,10,0,0"

ColumnDefinitions="300,\*,670">

<Grid RowDefinitions="40,\*">

<Label

Style="{StaticResource PrimaryLabelStyle}"

Text="{Binding SelectedProduct, Converter={StaticResource UpSertProductConverter}}"

VerticalOptions="Center" />

<Border Grid.Row="1" Style="{StaticResource Border25BackGroundWhite}">

<VerticalStackLayout Padding="10,10,10,10" Spacing="{x:Static ui:AppDimens.Space7}">

<Label FontSize="{x:Static ui:AppDimens.FontSize16}" Text="{x:Static resource:AppTexts.ProductName}" />

<Entry

HorizontalOptions="Start"

Text="{Binding Name}"

WidthRequest="{x:Static ui:AppDimens.Height280}" />

<Label FontSize="{x:Static ui:AppDimens.FontSize16}" Text="{x:Static resource:AppTexts.MainIngredients}" />

<Entry

HorizontalOptions="Start"

Text="{Binding Description}"

WidthRequest="{x:Static ui:AppDimens.Height280}" />

<Label

Margin="0,0,10,0"

FontSize="{x:Static ui:AppDimens.FontSize16}"

Text="{x:Static resource:AppTexts.Quantity}" />

<Entry

HorizontalOptions="Start"

Text="{Binding Quantity}"

WidthRequest="{x:Static ui:AppDimens.Height280}" />

<Label

Margin="0,0,10,0"

FontSize="{x:Static ui:AppDimens.FontSize16}"

Text="{x:Static resource:AppTexts.Price}" />

<Entry

HorizontalOptions="Start"

Text="{Binding Price}"

WidthRequest="{x:Static ui:AppDimens.Height280}" />

<HorizontalStackLayout>

<Button

BackgroundColor="{x:Static ui:AppColors.Yellow}"

Command="{Binding ResetCommand}"

HorizontalOptions="Center"

Text="{x:Static resource:AppTexts.Reset}"

TextColor="{x:Static ui:AppColors.Black}"

WidthRequest="{x:Static ui:AppDimens.Height127}" />

<Button

Margin="24,0,0,0"

Command="{Binding UpsertProductCommand}"

HorizontalOptions="Center"

Text="{Binding SelectedProduct, Converter={StaticResource UpSertButtonConverter}}"

TextColor="{x:Static ui:AppColors.Black}"

WidthRequest="{x:Static ui:AppDimens.Height127}" />

</HorizontalStackLayout>

</VerticalStackLayout>

</Border>

</Grid>

<Grid Grid.Column="2" RowDefinitions="40,\*">

<Label

Style="{StaticResource PrimaryLabelStyle}"

Text="{x:Static resource:AppTexts.ProductList}"

VerticalOptions="Center" />

<HorizontalStackLayout HorizontalOptions="End">

<Label Text="{x:Static resource:AppTexts.Quantity}" VerticalOptions="Center" />

<Button

BackgroundColor="Transparent"

Command="{Binding QuantityFilterCommand}"

FontFamily="{x:Static ui:FontNames.FaSolid}"

FontSize="{x:Static ui:AppDimens.FontSize15}"

Text="{x:Static resource:AwesomeIcon.Filter}"

TextColor="{x:Static ui:AppColors.Black}" />

<Label

Margin="18,0,0,0"

Text="{x:Static resource:AppTexts.Price}"

VerticalOptions="Center" />

<Button

Margin="0,0,170,0"

BackgroundColor="Transparent"

Command="{Binding PriceFilterCommand}"

FontFamily="{x:Static ui:FontNames.FaSolid}"

FontSize="{x:Static ui:AppDimens.FontSize15}"

Text="{x:Static resource:AwesomeIcon.Filter}"

TextColor="{x:Static ui:AppColors.Black}" />

</HorizontalStackLayout>

<CollectionView

Grid.Row="1"

ItemTemplate="{StaticResource ProductItemDataTemplate}"

ItemsSource="{Binding Products}">

<CollectionView.ItemsLayout>

<LinearItemsLayout ItemSpacing="{x:Static ui:AppDimens.Space10}" Orientation="Vertical" />

</CollectionView.ItemsLayout>

</CollectionView>

</Grid>

</Grid>

</Grid>

</Grid>

</ContentPage>

Ngoài các control có sẵn được MAUI cung cấp, các trang trong RuKiSo còn sử dụng các control tự tạo phục vụ cho các mục đích riêng biệt ví dụ như: OverViewCard: Là một thẻ thông tin bao gồm hình ảnh và các thông tin truyền vào được sử dụng để tổng hợp các thông tin kho hàng



Hình 3-16 OverViewCard

Để tái sử dụng mã nguồn các control có giao diện tương tự nhau được định nghĩa các style trong một file riêng tương tự như các class trong các file css ở ứng dụng web, ví dụ các style được viết cho control Label được sử dụng trong ứng dụng

<Style x:Key="PrimaryLabelStyle" TargetType="Label">

<Setter Property="FontSize" Value="{x:Static ui:AppDimens.FontSize18}" />

<Setter Property="TextColor" Value="{x:Static ui:AppColors.SecondaryColor}" />

<Setter Property="FontAttributes" Value="Bold" />

</Style>

<Style x:Key="PageTitleLabelStyle" TargetType="Label">

<Setter Property="FontSize" Value="{x:Static ui:AppDimens.FontSize28}" />

<Setter Property="TextColor" Value="{x:Static ui:AppColors.SecondaryColor}" />

<Setter Property="HorizontalOptions" Value="Center" />

<Setter Property="VerticalOptions" Value="Center" />

<Setter Property="FontAttributes" Value="Bold" />

<Setter Property="Margin" Value="10, 0, 0, 0" />

</Style>

<Style x:Key="Size16ColorBlue" TargetType="Label">

<Setter Property="FontSize" Value="{x:Static ui:AppDimens.FontSize16}" />

<Setter Property="TextColor" Value="{x:Static ui:AppColors.LightSteelBlue}" />

</Style>

<Style x:Key="Size16ColorBlack" TargetType="Label">

<Setter Property="FontSize" Value="{x:Static ui:AppDimens.FontSize16}" />

<Setter Property="TextColor" Value="{x:Static ui:AppColors.Black}" />

</Style>

<Style x:Key="Size15ColorBlue" TargetType="Label">

<Setter Property="FontSize" Value="{x:Static ui:AppDimens.FontSize16}" />

<Setter Property="TextColor" Value="{x:Static ui:AppColors.LightSteelBlue}" />

</Style>

* **ViewModel**: Đảm nhận công việc đồng bộ dữ liệu từ model lên View. Mối quan hệ giữa View và View-Model là View sẽ được ánh xạ tới View-Model nhưng View-Model lại không biết thông tin gì về View. Nó được ẩn dấu qua cách sử dụng Data-binding và cơ chế của mô hình Observer. Một View-Model có thể được ánh xạ từ nhiều View. Nói cách khác ViewModel tương dồng về chức năng và nhiệm vụ giống Controller trong mô hình MVC. Dưới đây là một source của BatchViewModel trong dự án, dữ liệu được lấy từ model và trả lại cho client sau khi xử lý:

using RuKiSo.Features.Models;

using RuKiSo.Features.Services;

using RuKiSo.Resources.Text;

using RuKiSo.Utils.MVVM;

using System.Collections.ObjectModel;

namespace RuKiSo.ViewModels

{

public partial class DashBoardViewModel : BaseViewModel

{

#region Fields

private readonly IGenericService<ProductRespone, ProductRequest> \_productService;

private readonly IGenericService<IngredientRespone, IngredientRequest> \_ingredientService;

private readonly IGenericService<TransactionResponse, TransactionRequest> \_transactionService;

private readonly IGenericService<BatchResponse, BatchRequest> \_batchService;

private ObservableCollection<WeeklyHistoryDTO> weeklyHistories;

private ObservableCollection<TopSellerDTO> topSellers;

private ObservableCollection<MostUsedIngredient> mostUsedIngredients;

private ObservableCollection<ProfitDTO> monthlyProfit;

#endregion

public DashBoardViewModel(

IGenericService<ProductRespone, ProductRequest> productService,

IGenericService<IngredientRespone, IngredientRequest> ingredientService,

IGenericService<TransactionResponse, TransactionRequest> transactionService,

IGenericService<BatchResponse, BatchRequest> batchService,

BatchReminderViewModel reminderViewModel,

IErrorHandlingService errorHandlingService) : base(errorHandlingService)

{

\_productService = productService;

\_ingredientService = ingredientService;

\_transactionService = transactionService;

\_batchService = batchService;

ReminderViewModel = reminderViewModel;

InitializeCollections();

}

#region Properties

private void InitializeCollections()

{

WeeklyHistories = new();

TopSellers = new();

MostUsedIngredients = new();

MonthlyProfit = new();

}

public ObservableCollection<WeeklyHistoryDTO> WeeklyHistories

{

get => weeklyHistories;

set

{

weeklyHistories = value;

OnPropertyChanged();

}

}

public ObservableCollection<TopSellerDTO> TopSellers

{

get => topSellers;

set

{

topSellers = value;

OnPropertyChanged();

}

}

public ObservableCollection<MostUsedIngredient> MostUsedIngredients

{

get => mostUsedIngredients;

set

{

mostUsedIngredients = value;

OnPropertyChanged();

}

}

public ObservableCollection<ProfitDTO> MonthlyProfit

{

get => monthlyProfit;

set

{

monthlyProfit = value;

OnPropertyChanged();

}

}

public BatchReminderViewModel ReminderViewModel { get; }

#endregion

protected override async Task LoadDataAsync()

{

try

{

await Task.WhenAll(

LoadWeeklyHistoryAsync(),

LoadTopSellersAsync(),

LoadMostUsedIngredientsAsync(),

LoadMonthlyProfitAsync()

);

}

catch (Exception ex)

{

HandleException(ErrorMessages.LOADING\_DASHBOARD, ex);

}

}

private async Task LoadWeeklyHistoryAsync()

{

try

{

var transactions = await \_transactionService.GetAllAsync();

if (transactions == null) return;

var endDate = DateTime.Today;

var startDate = endDate.AddDays(-6);

var weeklyData = transactions

.Where(t => t.TranDate >= startDate && t.TranDate <= endDate)

.GroupBy(t => new {

Date = t.TranDate.ToString("ddd"),

t.TranType

})

.Select(g => new WeeklyTransactionData

{

Date = g.Key.Date,

TranType = g.Key.TranType,

Total = g.Sum(t => t.Quantity)

})

.ToList();

var histories = GenerateWeeklyHistories(weeklyData);

WeeklyHistories = new(histories);

}

catch (Exception ex)

{

HandleException(ErrorMessages.WEEKLY\_HISTORY, ex);

}

}

private static List<WeeklyHistoryDTO> GenerateWeeklyHistories(List<WeeklyTransactionData> weeklyData)

{

var histories = new List<WeeklyHistoryDTO>();

var daysOfWeek = new[] { "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun" };

foreach (var day in daysOfWeek)

{

var sells = weeklyData.FirstOrDefault(x => x.Date == day && x.TranType)?.Total ?? 0;

var purchases = weeklyData.FirstOrDefault(x => x.Date == day && !x.TranType)?.Total ?? 0;

histories.Add(new WeeklyHistoryDTO

{

Date = day,

Sell = sells,

Purchase = purchases

});

}

return histories;

}

private async Task LoadTopSellersAsync()

{

try

{

var transactions = await \_transactionService.GetAllAsync();

var products = await \_productService.GetAllAsync();

if (transactions == null || products == null) return;

var topProducts = GetTopSellingProducts(transactions, products);

TopSellers = new(topProducts);

}

catch (Exception ex)

{

HandleException(ErrorMessages.TOP\_SELLERS, ex);

}

}

private async Task LoadMostUsedIngredientsAsync()

{

try

{

var batches = await \_batchService.GetAllAsync();

var ingredients = await \_ingredientService.GetAllAsync();

if (batches == null || ingredients == null) return;

var topIngredients = GetMostUsedIngredients(batches);

MostUsedIngredients = new(topIngredients);

}

catch (Exception ex)

{

HandleException(ErrorMessages.MOST\_USED, ex);

}

}

private async Task LoadMonthlyProfitAsync()

{

try

{

var transactions = await \_transactionService.GetAllAsync();

if (transactions == null) return;

var monthlyData = CalculateMonthlyProfit(transactions);

MonthlyProfit = new(monthlyData);

}

catch (Exception ex)

{

HandleException(ErrorMessages.MONTHLY\_PROFIT, ex);

}

}

private static List<TopSellerDTO> GetTopSellingProducts(

IEnumerable<TransactionResponse> transactions,

IEnumerable<ProductRespone> products)

{

return transactions

.Where(t => t.TranType && t.ProductId.HasValue)

.GroupBy(t => t.ProductId)

.Select(g => new { ProductId = g.Key, TotalQuantity = g.Sum(t => t.Quantity) })

.OrderByDescending(x => x.TotalQuantity)

.Take(3)

.Join(

products,

t => t.ProductId,

p => p.Id,

(t, p) => new TopSellerDTO

{

Name = p.Name,

Quantity = t.TotalQuantity

}

)

.ToList();

}

private static List<MostUsedIngredient> GetMostUsedIngredients(IEnumerable<BatchResponse> batches)

{

return batches

.SelectMany(b => b.Ingredients)

.GroupBy(bi => bi.IngredientName)

.Select(g => new

{

IngredientName = g.Key,

TotalQuantity = g.Sum(bi => bi.UsedQuantity)

})

.OrderByDescending(x => x.TotalQuantity)

.Take(5)

.Select(x => new MostUsedIngredient

{

Name = x.IngredientName,

Quantity = (int)x.TotalQuantity

})

.ToList();

}

private static List<ProfitDTO> CalculateMonthlyProfit(IEnumerable<TransactionResponse> transactions)

{

return transactions

.GroupBy(t => new DateTime(t.TranDate.Year, t.TranDate.Month, 1))

.Select(g => new ProfitDTO

{

Date = g.Key,

Profit = g.Where(t => t.TranType).Sum(t => t.Value \* 0.20)

})

.OrderBy(x => x.Date)

.Take(10)

.ToList();

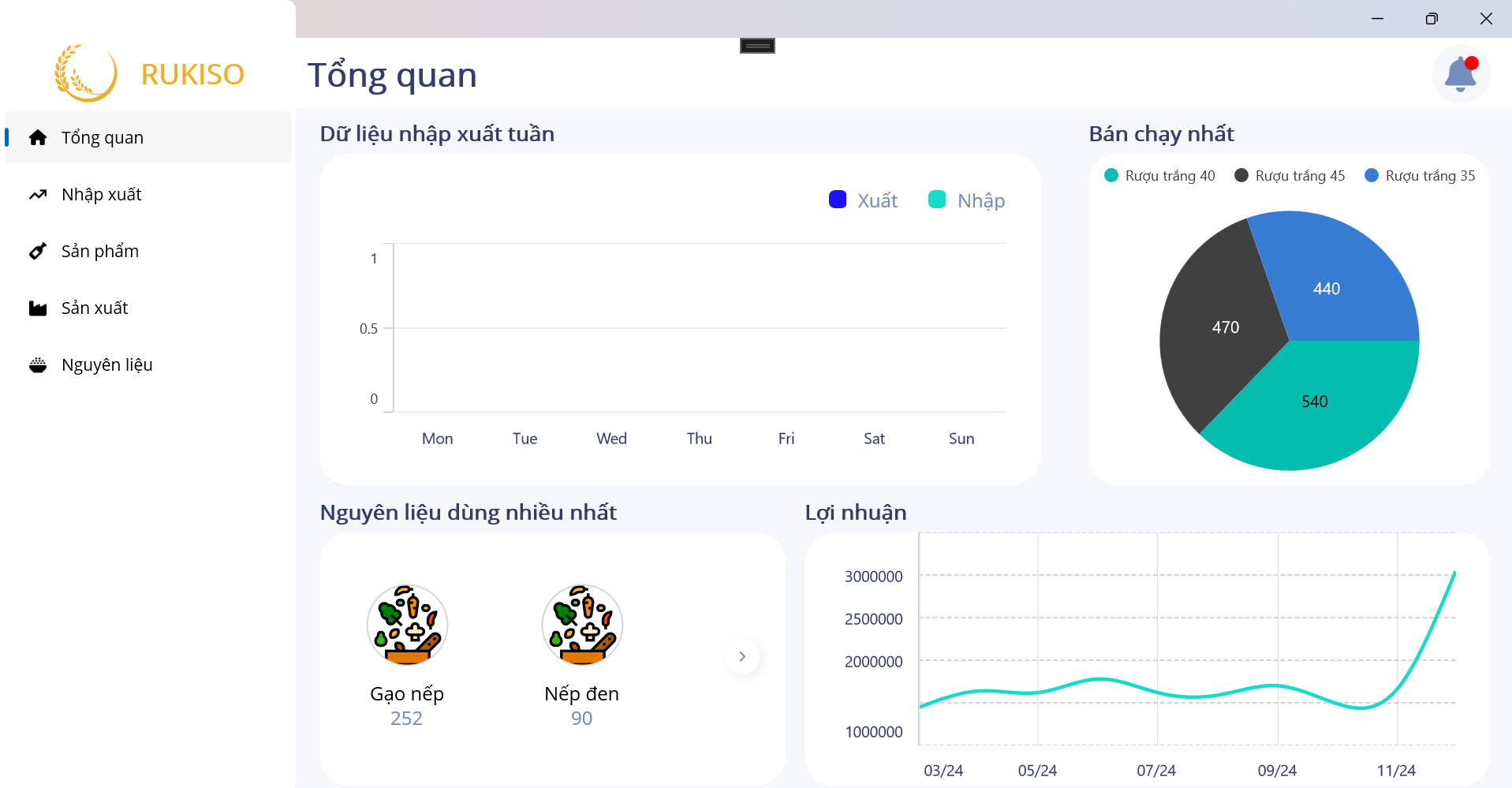
}

}

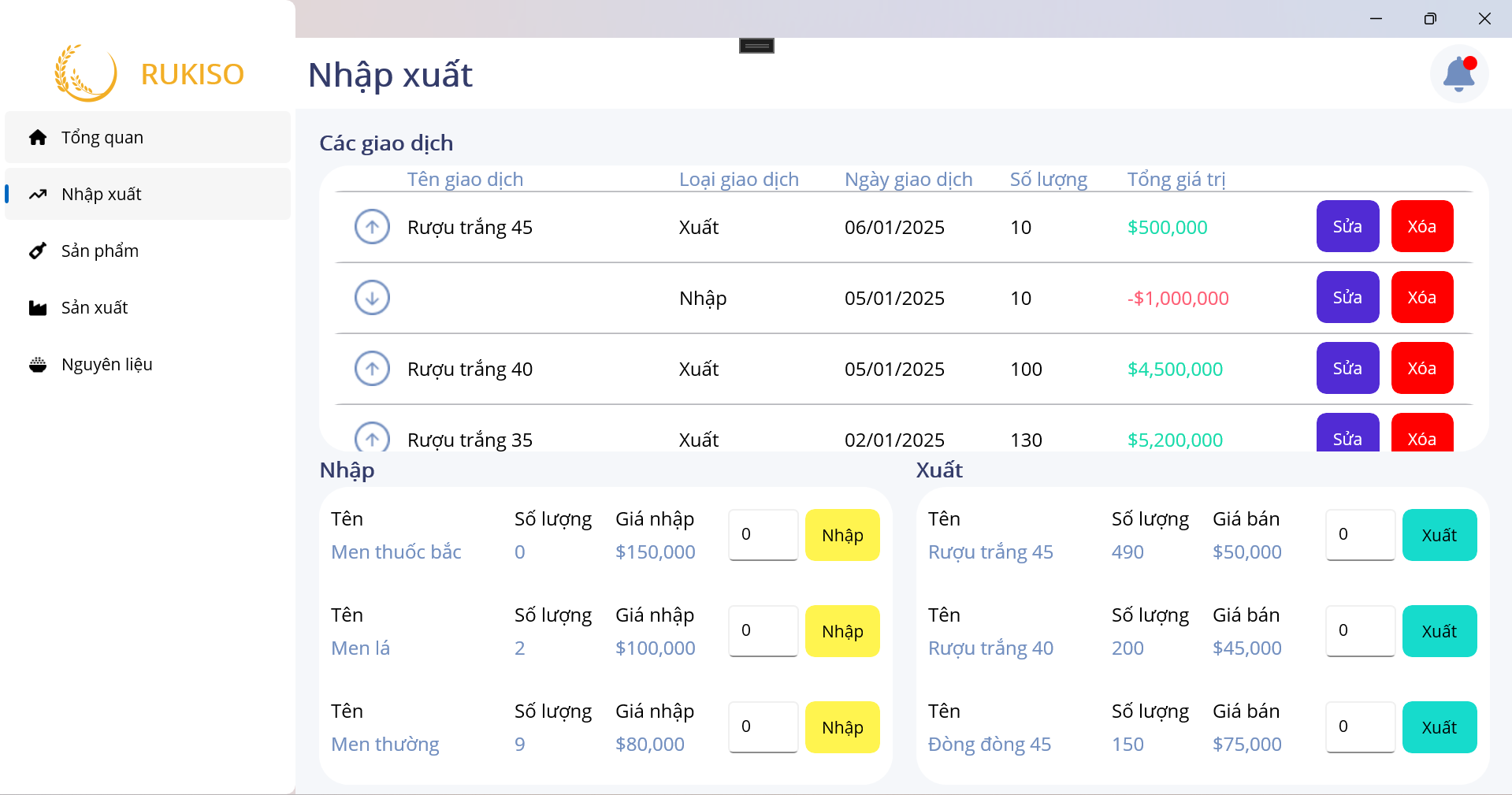
}

* **Model**: Là các đối tượng giúp truy xuất và thao tác trên dữ liệu thực sự, giống như trong mô hình MVC. Trong phần mềm này Model được chia thành hai loại là model thông thường và DTO để tách biệt class mà API trả về và các dữ liệu mà ViewModel và View sử dụng.

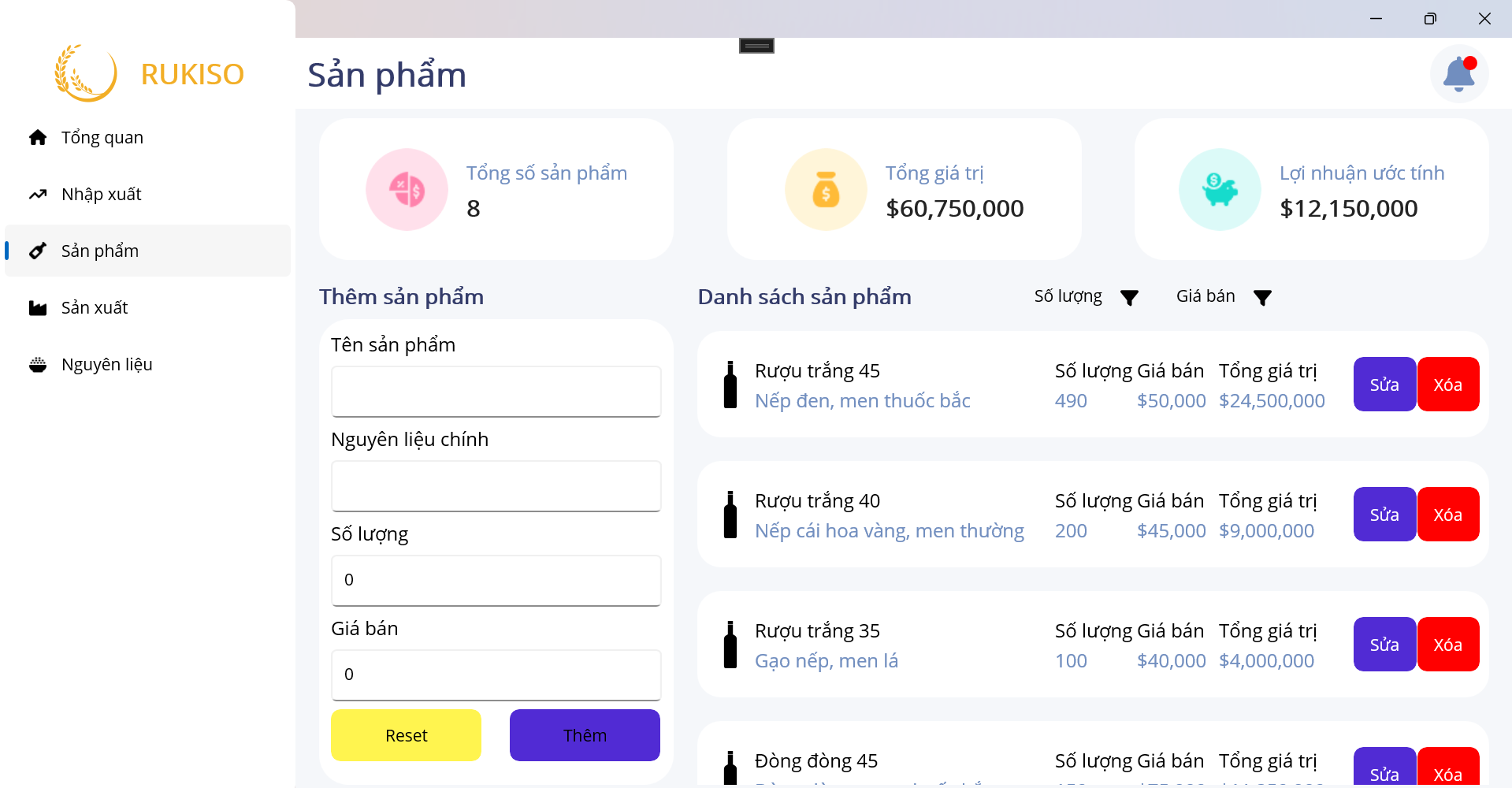
Với việc sử dụng .NET MAUI và mẫu thiết kế MVVM, client của RuKiSo đã đáp ứng được yêu cầu đề ra, Dưới đây là một số hình ảnh của Client.



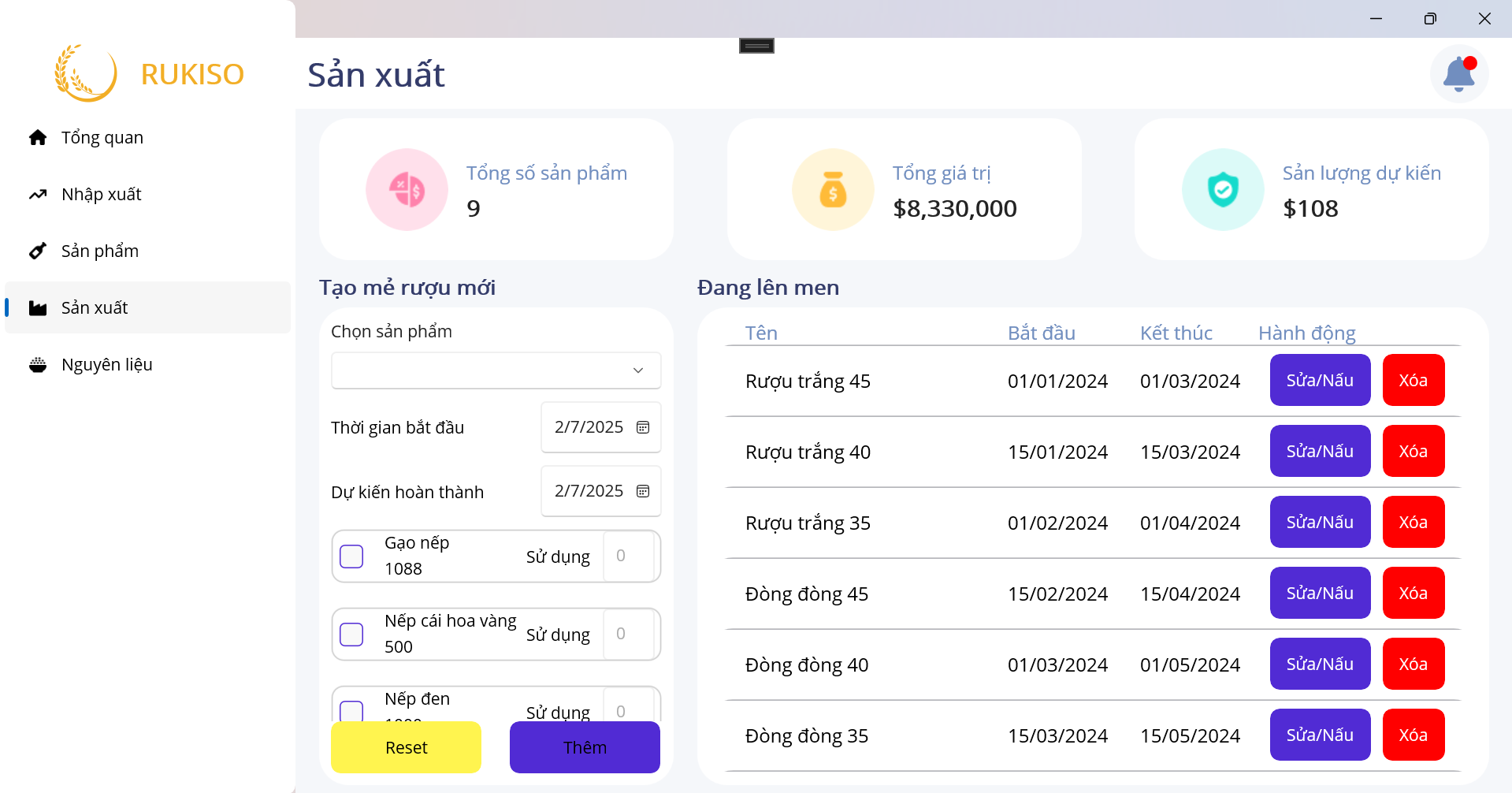
Hình 3-17 DashBoardPage



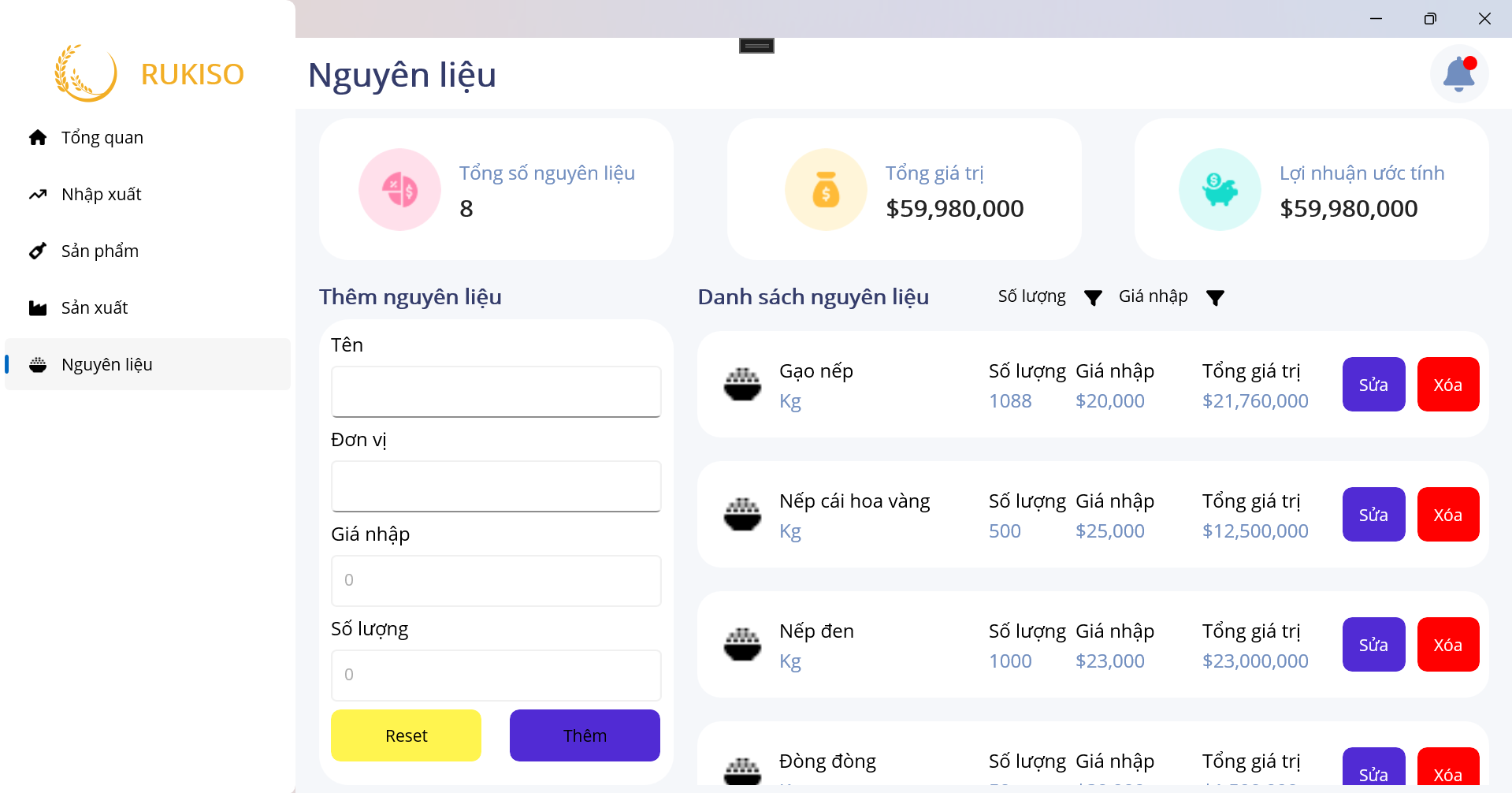
Hình 3-18 TransactionPage



Hình 3-19 ProductPage



Hình 3-20 BatchPage



Hình 3-21 IngredientPage