# Report 3: Core Feature Development Report (Development Phase)

## Development Progress Overview

A. Current Progress (Đã Hoàn Thành)

| **Module** | **Mô tả ngắn gọn** |
| --- | --- |
| Đăng nhập (Login) | Login.xaml hoạt động, AccountService xử lý xác thực |
| Quản lý Phòng | Full CRUD: tạo, sửa, xóa, xem phòng trong RoomManager.xaml |
| Quản lý Người Thuê | CRUD thông tin người thuê, thêm người ở ghép trong AddRoomate.xaml |
| Quản lý Hợp Đồng | Tạo và liên kết người thuê với phòng trong CreateContract.xaml, Contract.xaml |
| Quản lý Hóa Đơn | Tạo hóa đơn hàng tháng từ hợp đồng, chỉnh sửa, thanh toán |
| Thanh Toán | Ghi nhận thanh toán qua PaymentWindow.xaml |
| Báo cáo | Có ReportManager.xaml, ReportService – framework đã sẵn sàng |

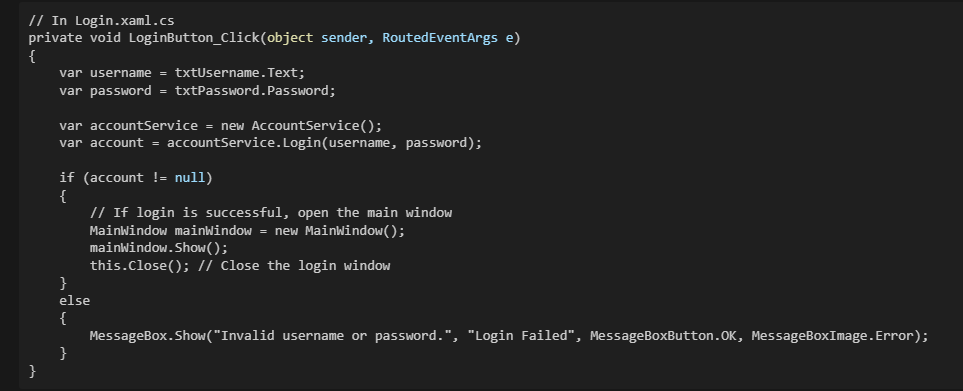
B. Các Hạng Mục Đang Phát Triển / Kế Hoạch

| **Tính năng** | **Trạng thái** | **Ghi chú** |
| --- | --- | --- |
| Chức năng cốt lõi | Hoàn tất | Đầy đủ các thao tác CRUD và logic chính |
| Giao diện người dùng | Hoàn tất | Giao diện đầy đủ, có thể cần tối ưu thêm về UX |
| Cơ sở dữ liệu | Hoàn tất | Đã có ERD rõ ràng, sơ đồ quan hệ chuẩn hóa |
| Unit Test | Một phần | Đã có test cho RoomService và TenantService, cần mở rộng toàn bộ hệ thống |
| Báo cáo nâng cao | Một phần | Mới là nền tảng cơ bản, cần thêm các báo cáo chi tiết hơn |
| Dashboard tổng quan | Chưa có | Chưa thấy MainWindow có thống kê tổng quan như số phòng trống, số tiền nợ, v.v. |
| Kiểm tra & Xử lý lỗi | Một phần | Cần cải thiện thông báo lỗi, xử lý exception thân thiện với người dùng |
| CI/CD | Kế hoạch | Dự kiến tích hợp GitHub Actions để build/test/deploy tự động |
| Bảo mật mật khẩu | Kế hoạch | Cần mã hóa mật khẩu (BCrypt) thay vì lưu plain-text trong CSDL |

## Implemented Features

1. User Authentication and Authorization

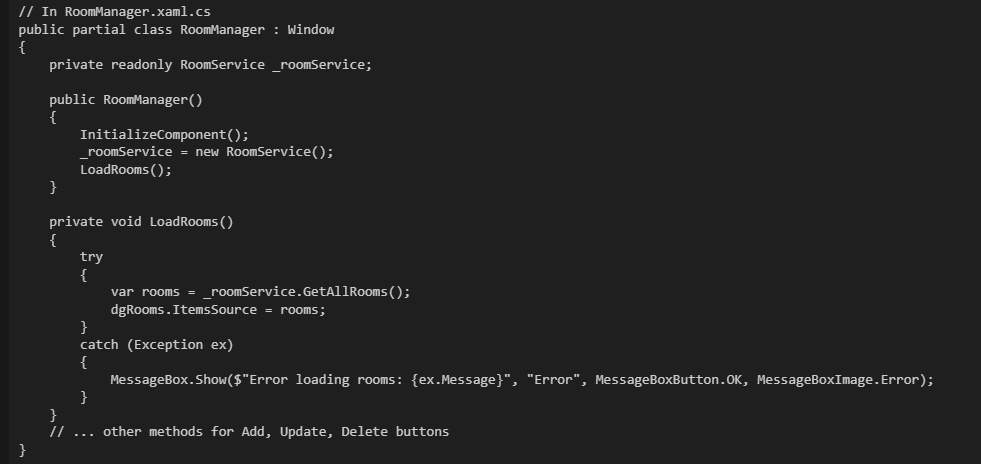
**Mô tả:** Hệ thống được bảo mật thông qua màn hình đăng nhập, yêu cầu người dùng (quản lý tài sản) xác thực trước khi truy cập ứng dụng chính. Chỉ những người đã được ủy quyền mới có thể quản lý dữ liệu. Giao diện chính chỉ mở sau khi đăng nhập thành công.

Đoạn mã minh họa (WPF/Login.xaml.cs):  


1. Room Management

**Mô tả:** Tính năng này cho phép quản lý thực hiện đầy đủ các thao tác CRUD (Tạo, Đọc, Cập nhật, Xóa) đối với phòng cho thuê. Người dùng có thể xem danh sách các phòng, trạng thái ("Trống", "Đã thuê"), thêm mới, chỉnh sửa thông tin (giá, diện tích), và xóa các phòng không còn hoạt động.

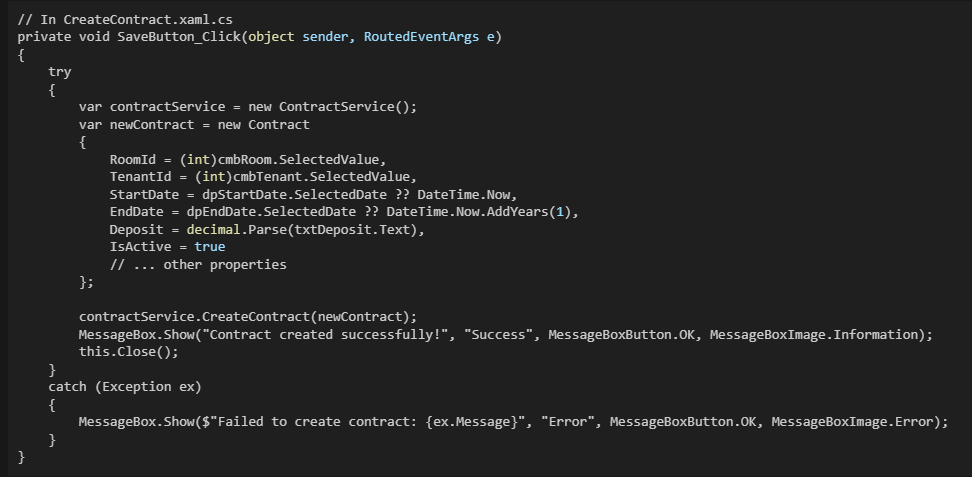
Đoạn mã minh họa (WPF/RoomManager.xaml.cs):



1. Tenant and Contract Management

**Mô tả:**Tính năng này quản lý thông tin người thuê và hợp đồng thuê nhà. Quản lý có thể thêm người thuê, lưu thông tin cá nhân và tạo hợp đồng gắn liền với một phòng cụ thể. Hợp đồng chứa thông tin ngày bắt đầu/kết thúc, tiền thuê, tiền cọc. Hệ thống cũng cho phép thêm nhiều người ở ghép vào cùng hợp đồng.

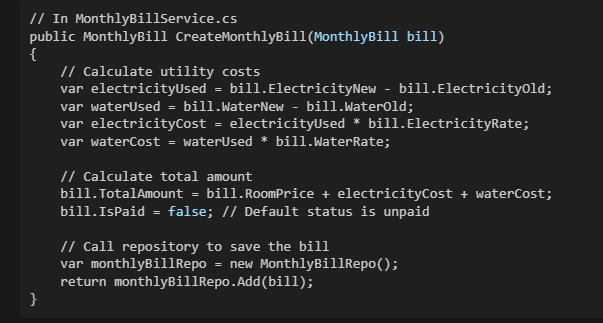
Đoạn mã minh họa (WPF/CreateContract.xaml.cs):



1. Monthly Billing and Payment Tracking

**Mô tả:** Hệ thống tự động tạo hóa đơn hàng tháng dựa trên hợp đồng đang hoạt động, bao gồm tiền phòng và chi phí điện nước dựa vào chỉ số đồng hồ. Sau khi tạo hóa đơn, quản lý có thể theo dõi tình trạng thanh toán (Đã/Chưa thanh toán) và cập nhật khi nhận tiền từ người thuê.

Đoạn mã minh họa (Services/MonthlyBillService.cs):



## Technical Implementation

Hệ thống được xây dựng theo mô hình **3 lớp cổ điển (3-tier architecture)** – một ví dụ tiêu biểu cho nguyên lý **Separation of Concerns** trong lập trình hướng đối tượng (OOP). Mỗi lớp có trách nhiệm riêng biệt:

* **WPF (Lớp Giao Diện - Presentation Layer):** Xử lý giao diện và tương tác người dùng.
* **Services (Lớp Logic - Business Logic Layer):** Chứa logic xử lý chính, tính toán và điều phối dữ liệu.
* **Repositories (Lớp Truy cập dữ liệu - Data Access Layer):** Tương tác với cơ sở dữ liệu thông qua EF Core.

Cấu trúc này giúp hệ thống **mô-đun hóa**, **dễ bảo trì**, và **mở rộng linh hoạt**.

1. Nguyên Lý OOP

* **Tính đóng gói (Encapsulation):**

Lớp Room chứa các thuộc tính liên quan đến phòng như RoomId, RoomName, Price, và Status. Dữ liệu được bảo vệ và chỉ truy cập thông qua các thuộc tính công khai.

* **Tính trừu tượng (Abstraction):**  
  Lớp RoomService và RoomRepo ẩn đi chi tiết xử lý phía sau. Giao diện chỉ cần gọi các phương thức như \_roomService.AddNewRoom(room) mà không cần biết cách lưu phòng xuống cơ sở dữ liệu.

1. Triển Khai EF Core

* **DbContext và DbSet:**  
  Lớp RoomManagerContext kế thừa từ DbContext và cung cấp DbSet<Room> tương ứng với bảng Room trong cơ sở dữ liệu.
* **Mẫu Repository:**  
  Lớp RoomRepo sử dụng RoomManagerContext để thực hiện các thao tác CRUD (thêm, lấy, cập nhật, xóa). Điều này giúp tách biệt mã liên quan đến EF Core khỏi phần còn lại của ứng dụng.

1. WPF trong Giao Diện

* **Giao diện khai báo bằng XAML:**  
  Tệp RoomManager.xaml định nghĩa bố cục màn hình quản lý phòng bằng XAML, sử dụng DataGrid để hiển thị danh sách và các nút chức năng.
* **Ràng buộc dữ liệu (Data Binding):**  
  DataGrid được ràng buộc với danh sách các đối tượng Room trong code-behind. Ví dụ: Binding="{Binding RoomName}" sẽ tự động hiển thị tên phòng tương ứng.
* **Lập trình hướng sự kiện:**  
  Khi người dùng nhấn nút “Add New”, sự kiện AddButton\_Click được kích hoạt và gọi logic xử lý trong lớp dịch vụ.

## Challenges and Solutions

### Thách Thức 1: Quản Lý Trạng Thái & Đồng Bộ Dữ Liệu Trong Ứng Dụng Desktop

* **Mô tả:**  
  Trong WPF, một trong những khó khăn lớn là đảm bảo giao diện người dùng luôn phản ánh chính xác trạng thái dữ liệu trong backend. Ví dụ: sau khi thêm một phòng mới, danh sách phòng cần được cập nhật tự động mà không cần người dùng phải nhấn nút "Làm mới" hoặc khởi động lại ứng dụng.
* **Giải pháp:**
  1. **Sử dụng ObservableCollection<T>:**  
     Thay vì sử dụng List<T>, giao diện được liên kết với ObservableCollection<T>, một loại danh sách đặc biệt hỗ trợ tự động thông báo cho UI mỗi khi có thay đổi (thêm, xóa, cập nhật).
  2. **Tầng dịch vụ trung tâm (Service Layer):**  
     Tất cả thao tác thay đổi dữ liệu (Create, Update, Delete) đều được thực hiện thông qua tầng dịch vụ như RoomService, TenantService. Điều này giúp kiểm soát logic tập trung, giảm rủi ro sai lệch dữ liệu.
  3. **Tải lại dữ liệu sau thao tác:**  
     Sau khi thực hiện thao tác CUD thành công, một phương thức như LoadRooms() sẽ được gọi để truy vấn lại dữ liệu từ database, cập nhật ObservableCollection và đảm bảo UI luôn đồng bộ với dữ liệu thực.

### Thách Thức 2: Xử Lý Quan Hệ Phức Tạp Trong Cơ Sở Dữ Liệu Với EF Core

* **Mô tả:**  
  Cơ sở dữ liệu có nhiều mối quan hệ phức tạp như Contract liên kết giữa Room và Tenant, đồng thời có nhiều MonthlyBill. Việc cấu hình các mối quan hệ này và viết truy vấn hiệu quả để lấy dữ liệu liên quan là một thách thức đáng kể.
* **Giải pháp:**
  1. **Thuộc tính điều hướng (Navigation Properties):**  
     Các entity như Room, Tenant, Contract được thiết kế với thuộc tính điều hướng để EF Core hiểu mối quan hệ và hỗ trợ truy vấn dễ dàng.
  2. **Tải dữ liệu sớm với Include():**  
     Để tránh vấn đề hiệu năng như “N+1 query problem”, các truy vấn LINQ trong repository sử dụng .Include() để tải trước các entity liên quan trong một truy vấn duy nhất.



* 1. **Mẫu Repository:**  
     Mọi thao tác truy vấn đều được đóng gói trong các lớp Repository, giúp tầng service không cần quan tâm đến chi tiết kỹ thuật EF Core, từ đó tăng khả năng tái sử dụng và bảo trì.

### Thách Thức 3: Đảm Bảo Tính Toàn Vẹn Dữ Liệu & Kiểm Tra Đầu Vào

* **Mô tả:**  
  Việc đảm bảo dữ liệu hợp lệ là rất quan trọng. Ví dụ: không cho phép tạo hợp đồng có ngày kết thúc nhỏ hơn ngày bắt đầu, tiền thuê âm, hoặc số điện thoại chứa ký tự không hợp lệ.
* **Giải pháp:**
  1. **Xác thực tại tầng dịch vụ:**  
     Trước khi gửi dữ liệu xuống repository, tầng dịch vụ như ContractService sẽ kiểm tra tính hợp lệ như ngày, tiền thuê, v.v. Điều này đảm bảo logic nghiệp vụ được kiểm soát tại một nơi duy nhất.
  2. **Lọc đầu vào từ giao diện người dùng:**  
     Các kiểm tra cơ bản được thực hiện ngay tại UI. Ví dụ: sự kiện PreviewTextInput của TextBox được dùng để từ chối ký tự không phải số trong trường nhập tiền.
  3. **Lập trình phòng thủ ở tầng repository:**  
     Repository vẫn được thiết kế cẩn thận để kiểm tra dữ liệu null, giá trị bất thường và sử dụng try-catch nhằm xử lý lỗi truy vấn database một cách an toàn.

## Git Commit History

<https://github.com/XuTruong/PRN212_Project.git>



## Code Quality and Documentation

### Duy trì chất lượng mã nguồn

Trong quá trình phát triển hệ thống, nhóm đã tuân thủ nghiêm ngặt các tiêu chuẩn lập trình để đảm bảo tính nhất quán, dễ hiểu và dễ bảo trì:

* **Quy ước đặt tên rõ ràng:**
  + Tên lớp, biến và phương thức tuân theo chuẩn PascalCase và camelCase phù hợp với quy ước C#.
  + Tên được đặt mang ý nghĩa rõ ràng, phản ánh đúng chức năng (ví dụ: AddNewRoom(), GetAllContracts()).
* **Tổ chức cấu trúc thư mục logic:**
  + Chia thành các lớp theo mô hình 3-tier: DataAccess, Repositories, Services, Views.
  + Mỗi chức năng chính được phân tách rõ ràng, tránh lẫn lộn giữa logic xử lý và giao diện.
* **Tuân thủ nguyên lý SOLID và Clean Code:**
  + Mỗi lớp, phương thức có trách nhiệm riêng.
  + Tránh lặp mã (DRY), tăng khả năng tái sử dụng.

### Tài liệu hóa nội bộ và chú thích mã nguồn

* **Chú thích trong mã:**
  + Các lớp và phương thức quan trọng đều có comment giải thích chức năng, đặc biệt với các logic xử lý nghiệp vụ trong tầng Service và Repository.



* **Tài liệu nội bộ:**
  + Nhóm đã duy trì tài liệu nội bộ mô tả kiến trúc hệ thống, quy trình xử lý từng chức năng chính, cách khởi chạy dự án và các yêu cầu về môi trường (CSDL, thư viện dùng).
  + Một số tài liệu được lưu dưới dạng .md hoặc .docx, giúp thành viên mới dễ dàng nắm bắt khi tham gia dự án.

## Testing Activities

Quá trình kiểm thử được thực hiện thông qua **kết hợp giữa kiểm thử thủ công và tự động**, nhằm đảm bảo ứng dụng hoạt động ổn định, đáng tin cậy và đáp ứng đầy đủ yêu cầu nghiệp vụ.

### A. Các hoạt động kiểm thử đã thực hiện

#### 1. ****Kiểm thử thủ công (Phương pháp chính):****

* **Quy trình:** Phương pháp kiểm thử chính là chạy ứng dụng và thực hiện trực tiếp tất cả các thao tác của người dùng để mô phỏng cách sử dụng thực tế bởi người quản lý phòng trọ.
* **Phạm vi kiểm thử:**
  + **Kiểm thử chức năng:** Xác minh rằng tất cả các tính năng cốt lõi đều hoạt động như mong đợi. Bao gồm:
    - Đăng nhập hệ thống
    - Tạo, xem, cập nhật, và xóa phòng
    - Quản lý người thuê và hợp đồng thuê trọ
  + **Kiểm thử giao diện người dùng (UI/UX):** Kiểm tra bố cục, khả năng phản hồi của các cửa sổ và điều khiển. Đảm bảo các biểu mẫu dễ sử dụng và điều hướng giữa các màn hình quản lý diễn ra mượt mà.
  + **Kiểm thử quy trình tổng thể (End-to-End Workflow):** Kiểm tra toàn bộ chuỗi nghiệp vụ, ví dụ:
    - Thêm người thuê mới.
    - Tạo hợp đồng thuê cho người thuê trong một phòng còn trống.
    - Sinh hóa đơn hàng tháng cho hợp đồng đó.
    - Ghi nhận thanh toán và xác minh trạng thái hóa đơn được cập nhật đúng.

#### 2. ****Kiểm thử đơn vị tự động (Giai đoạn đầu):****

* **Quy trình:** Một project UnitTests riêng biệt được tạo ra để kiểm thử tự động lớp xử lý nghiệp vụ. Các bài kiểm thử được viết bằng framework tiêu chuẩn (ví dụ: MSTest hoặc xUnit) để xác minh rằng các phương thức dịch vụ hoạt động chính xác một cách độc lập.
* **Phạm vi kiểm thử:**
  + Tập trung vào các dịch vụ cốt lõi. Đã triển khai kiểm thử đơn vị thành công cho RoomService và TenantService.
  + Các bài test kiểm tra logic chính như thêm mới một đối tượng hoặc lấy danh sách, đảm bảo rằng các quy tắc nghiệp vụ được áp dụng đúng mà không cần chạy toàn bộ ứng dụng.
* **Ví dụ mã kiểm thử (UnitTests/RoomServiceTests.cs):**



**B. Các vấn đề đã phát hiện và kế hoạch khắc phục**

| **STT** | **Vấn đề đã biết** | **Mô tả** | **Kế hoạch khắc phục** |
| --- | --- | --- | --- |
| 1 | **Phạm vi kiểm thử tự động còn hạn chế** | Hiện tại chỉ mới có kiểm thử đơn vị cho RoomService và TenantService. Các dịch vụ khác, lớp repository và giao diện người dùng chưa được kiểm thử tự động. | **Giai đoạn tiếp theo:** Mở rộng phạm vi kiểm thử đơn vị cho toàn bộ các phương thức trong project Services. Tích hợp framework giả lập (như Moq) để cô lập các dịch vụ khi kiểm thử. Bắt đầu xây dựng kiểm thử giao diện người dùng sử dụng công cụ như FlaUI. |
| 2 | **Bảo mật mật khẩu chưa đảm bảo** | Mật khẩu đang được lưu và xử lý dưới dạng văn bản thuần (plain text) trong mã nguồn và cơ sở dữ liệu. Đây là lỗ hổng bảo mật nghiêm trọng được phát hiện trong quá trình rà soát mã. | **Giai đoạn tiếp theo:** Thay thế việc lưu mật khẩu dạng plain text bằng giải pháp mã hóa mạnh. Sử dụng thư viện BCrypt.Net để mã hóa mật khẩu khi tạo/cập nhật tài khoản và xác thực mật khẩu khi đăng nhập. |
| 3 | **Thiếu kiểm tra đầu vào từ người dùng** | Kiểm thử thủ công cho thấy ứng dụng có thể bị lỗi hoặc hoạt động không đúng khi người dùng nhập dữ liệu không hợp lệ (ví dụ: chữ trong trường số, định dạng ngày sai). | **Giai đoạn tiếp theo:** Triển khai kiểm tra đầu vào toàn diện, bao gồm: xác thực phía client trong giao diện WPF để hiển thị thông báo lỗi tức thì, và xác thực phía server ở tầng dịch vụ nhằm đảm bảo toàn vẹn dữ liệu. |

## Next Steps

## Các hoạt động dự kiến thực hiện

1. **Triển khai tự động hóa CI/CD:**
   * **Hành động:** Thiết lập các quy trình GitHub Actions đã đề xuất (ci.yml và cd.yml).
   * **Mục tiêu:** Tạo pipeline tự động hoàn toàn, thực hiện các bước build, kiểm thử và đóng gói ứng dụng mỗi khi có thay đổi được đẩy lên nhánh main. Điều này giúp mã nguồn tích hợp luôn được kiểm tra liên tục và kịp thời phát hiện lỗi.
2. **Mở rộng phạm vi kiểm thử tự động:**
   * **Hành động:** Viết bổ sung các bài kiểm thử đơn vị (unit test) cho toàn bộ các dịch vụ và repository còn lại (ví dụ: ContractService, MonthlyBillService, ...). Áp dụng framework mô phỏng như **Moq** để tách biệt dịch vụ khỏi cơ sở dữ liệu trong quá trình kiểm thử.
   * **Mục tiêu:** Nâng cao độ tin cậy cho logic nghiệp vụ và phát hiện lỗi sớm trước khi đưa vào môi trường thật.
3. **Tăng cường bảo mật hệ thống:**
   * **Hành động:** Thay thế cơ chế lưu trữ mật khẩu hiện tại (dạng plain-text) bằng thuật toán mã hóa mạnh. Tích hợp thư viện **BCrypt.Net** vào AccountService để mã hóa mật khẩu khi tạo/cập nhật tài khoản và xác thực khi đăng nhập.
   * **Mục tiêu:** Bảo vệ thông tin người dùng và loại bỏ các lỗ hổng bảo mật nghiêm trọng.
4. **Cải thiện kiểm tra dữ liệu đầu vào:**
   * **Hành động:** Triển khai logic kiểm tra dữ liệu đầu vào chặt chẽ ở cả giao diện người dùng (UI) và tầng dịch vụ. Bao gồm:
     + Cảnh báo ngay lập tức khi người dùng nhập sai (ở các form trong WPF).
     + Kiểm tra lại dữ liệu ở tầng service để bảo vệ toàn vẹn dữ liệu trước khi lưu vào cơ sở dữ liệu.
   * **Mục tiêu:** Ngăn chặn lỗi và sự cố do dữ liệu không hợp lệ, đồng thời đảm bảo độ ổn định và chính xác của hệ thống.
5. **Tạo trình cài đặt ứng dụng (Application Installer):**
   * **Hành động:** Sử dụng công cụ đóng gói **MSIX cho .NET** để tạo bộ cài đặt hiện đại và chuyên nghiệp cho ứng dụng WPF. Trình cài đặt này sẽ được tích hợp vào quy trình CD.
   * **Mục tiêu:** Giúp việc triển khai và phân phối phần mềm cho người dùng cuối (nhân viên quản lý phòng trọ) trở nên dễ dàng và thuận tiện hơn.
6. **Thực hiện kiểm thử chấp nhận người dùng cuối (UAT):**
   * **Hành động:** Sau khi hoàn thành các bước trên, sẽ tiến hành một đợt kiểm thử tổng thể (end-to-end) thủ công trên phiên bản đã đóng gói.
   * **Mục tiêu:** Đảm bảo tất cả các thành phần trong hệ thống hoạt động liền mạch với nhau và xác nhận rằng ứng dụng sẵn sàng để phát hành chính thức lần đầu.