

BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH

KHOA CƠ KHÍ CHẾ TẠO MÁY



HCMUTE

Hướng dẫn sử dụng trang Web MooseHome

Giảng viên hướng dẫn: ThS. Nguyễn Hữu Trung

Nhóm SVTH:

Hồ Xuân Huy - 21146465

Võ Duy Khải - 21146474

Trần Hoàng Duy - 21146073

Lê Quốc Cường - 21146070

Mã học phần: 241WEPR330479

TP. Thủ Đức, 12 tháng 12 năm 2024

MỤC LỤC

1. GIỚI THIỆU.....	1
2. ĐIỀU HƯỚNG	3
2.1. Administrator	3
2.1.1. Đăng nhập vào hệ thống (Login)	3
2.1.2. Quản lý thông tin của các người dùng cá nhân.....	4
2.1.3. Quản lý tất cả ngôi nhà đã được đăng ký	7
2.2. Personal User	11
2.2.1. Đăng nhập vào hệ thống (Login)	11
2.2.2. Đăng ký người dùng cá nhân (Sign Up).....	13
2.2.3. Quên mật khẩu (Forgot Password)	13
2.2.4. Xem và Sửa lại thông tin cá nhân (Edit & Update).....	16
2.2.5. Thao tác của với các ngôi nhà.....	16
2.2.6. Đăng xuất (Logout)	17
3. CHỨC NĂNG	18
3.1. Biểu đồ Quan hệ thực thể (Entity Relationship Diagram).....	18
3.2. Chức năng Security	20
3.3. Các chức năng xác thực người dùng	23
3.3.1. Chức năng đăng nhập.....	23
3.3.2. Chức năng đăng ký	26
3.3.3. Chức năng quên mật khẩu	28
3.3.4. Chức năng thay đổi mật khẩu.....	31
3.3.5. Chức năng đăng xuất.....	33
3.4. Administrator	34
3.4.1. Account.....	34
<i>a. Hiển thị danh sách người dùng</i>	<i>34</i>
<i>b. Chức năng thay đổi trạng thái người dùng người dùng cá nhân ...</i>	<i>35</i>
<i>c. Xóa người dùng.....</i>	<i>39</i>
<i>d. Tìm kiếm người dùng.....</i>	<i>41</i>
3.4.2. House Management.....	42

<i>a. Hiển thị danh sách các ngôi nhà</i>	<i>43</i>
<i>b. Thêm nhà.....</i>	<i>45</i>
<i>c. Sửa thông tin của ngôi nhà</i>	<i>47</i>
<i>d. Lưu thông tin sau khi thêm hoặc sửa</i>	<i>48</i>
<i>e. Xóa nhà.....</i>	<i>50</i>
<i>f. Tìm nhà</i>	<i>52</i>
<i>g. Hiển thị danh sách tất cả ngôi nhà</i>	<i>54</i>
3.4.3. Equipment.....	56
<i>a. Hiển thị danh sách thiết bị</i>	<i>56</i>
<i>b. Thêm thiết bị.....</i>	<i>58</i>
<i>c. Sửa thiết bị.....</i>	<i>60</i>
<i>d. Lưu thông tin thiết bị khi thêm hoặc sửa</i>	<i>61</i>
<i>e. Xóa thiết bị.....</i>	<i>63</i>
3.5. User	65
3.5.1. Edit Profile	65
3.5.2. Xem danh sách các ngôi nhà của user.....	68
3.5.3. Xem danh sách thiết bị của nhà:	69
3.6. Internet of Things (IoT).....	70
3.6.1. Giám sát thiết bị	71
3.6.2. Điều khiển thiết bị	73
4. ĐIỀU KHOẢN PHÁP LÝ VÀ TUÂN THỦ	78
4.1. Điều Khoản Dịch Vụ	78
4.2. Tiêu Chuẩn Tuân Thủ	78
4.3. Cam Kết Bảo Mật và Quyền Riêng Tư.....	78
5. KẾT LUẬN	80

PHỤ LỤC HÌNH ẢNH

Hình 1: Hình ảnh HomePage của trang web MooseHome.....	2
Hình 2: Hình ảnh trang Login của Administrator.....	3
Hình 3: Hình ảnh trang quản lý người dùng cá nhân của Administrator.....	4
Hình 4: Hình ảnh trang quản lý danh sách nhà của tất cả người dùng cá nhân....	4
Hình 5: Hình ảnh trang Edit User.	5
Hình 6: Hình ảnh trang Add House.....	6
Hình 7: Hình ảnh trang quản lý danh sách tất cả nhà đã đăng ký.....	7
Hình 8: Hình ảnh trang Edit House.....	8
Hình 9: Hình ảnh trang Add Equipment.	9
Hình 10: Hình ảnh trang Edit Equip.	10
Hình 11: Hình ảnh trang IOT Monitor.....	11
Hình 12: Hình ảnh trang Login cho Personal User.....	12
Hình 13: Hình ảnh trang quản lý người dùng cá nhân của Personal user.....	12
Hình 14: Hình ảnh trang Sign Up của người dùng cá nhân.	13
Hình 15: Hình ảnh xin cấp phép lấy lại mật khẩu.....	14
Hình 16: Hình ảnh Submit Forgot Password thành công.	14
Hình 17: Hình ảnh email cấp lại mật khẩu.....	15
Hình 18: Hình ảnh trang Reset Your Password.	15
Hình 19: Hình ảnh trang Edit Profile.	16
Hình 20: Entity Relationship Diagram (ERD).	18
Hình 21: Sequence Diagram trang Sign in.....	24
Hình 22: Sequence Diagram trang Sign up.....	26

Hình 23: Sequence Diagram trang Forgot Password.....	28
Hình 24: Sequence Diagram trang Change Password.	31
Hình 25: Sequence Diagram trang Sign Out.....	33
Hình 26: Sequence Diagram Quản lý danh sách người dùng.....	34
Hình 27: Sequence Diagram trang Edit User.....	36
Hình 28: Sequence Diagram chức năng xóa người dùng.	39
Hình 29: Sequence Diagram trang Tìm kiếm người dùng.....	41
Hình 30: Sequence Diagram trang Hiển thị danh sách các ngôi nhà.....	43
Hình 31: Sequence Diagram trang Add House.....	45
Hình 32: Sequence Diagram trang Edit House.	47
Hình 33: Sequence Diagram trang Delete House.	50
Hình 34: Sequence Diagram trang Search Houses.	52
Hình 35: Sequence Diagram trang Hiển thị danh sách các ngôi nhà.....	54
Hình 36: Sequence Diagram trang hiển thị danh sách các thiết bị.	56
Hình 37: Sequence Diagram chức năng thêm thiết bị.	58
Hình 38: Sequence Diagram trang Edit Equipment.....	60
Hình 39: Sequence Diagram chức năng Delete Equipment.....	63
Hình 40: Sequence Diagram chức năng Edit Profile.	65

PHỤ LỤC BẢNG

Bảng 1: Bảng các thao tác trong nhóm Action của Administrator. 4

Bảng 2: Bảng nhóm các thao tác trong trang quản lý nhà. 7

LỜI CẢM ƠN

Chúng tôi xin gửi lời cảm ơn chân thành đến ThS. Nguyễn Hữu Trung, giảng viên hướng dẫn, đã tận tình chỉ bảo và hỗ trợ chúng tôi trong suốt quá trình thực hiện dự án. Những ý kiến đóng góp và sự hỗ trợ của thầy là nguồn động lực và tri thức vô cùng quý giá giúp nhóm hoàn thành nhiệm vụ này. Chúng tôi cũng xin cảm ơn nhà trường, đặc biệt là khoa Cơ Khí Chế Tạo Máy, bộ môn Cơ Điện Tử của Trường Đại học Sư Phạm Kỹ Thuật TP.HCM, đã tạo điều kiện thuận lợi về cơ sở vật chất và môi trường học tập để chúng tôi có thể thực hiện dự án một cách hiệu quả. Cuối cùng, chúng tôi xin cảm ơn các bạn đồng hành và tất cả các nguồn tài liệu, bao gồm sách, bài báo, website và công cụ hỗ trợ, đã cung cấp thông tin và cảm hứng để hoàn thiện dự án này.

1. GIỚI THIỆU

Chào mừng bạn đến với cuốn Hướng dẫn sử dụng trang web MooseHome, nền tảng trực tuyến tối ưu giúp bạn dễ dàng quản lý và điều khiển tất cả các thiết bị thông minh trong ngôi nhà của mình. Với sự phát triển của công nghệ, việc sở hữu một ngôi nhà thông minh không còn là điều xa vời. Trang web MooseHome ra đời với mục tiêu giúp bạn đơn giản hóa việc điều khiển và giám sát các thiết bị điện tử, từ các thiết bị chiếu sáng, điều hòa không khí, đến các hệ thống an ninh, khóa cửa, thiết bị gia dụng và nhiều thứ khác (sẽ được tùy chỉnh để thỏa sức sáng tạo của bạn) – tất cả điều đó chỉ cần qua một giao diện duy nhất, dễ sử dụng và trực quan. Trang web sẽ mang đến những điều vượt xa ngoài mong đợi của bạn.

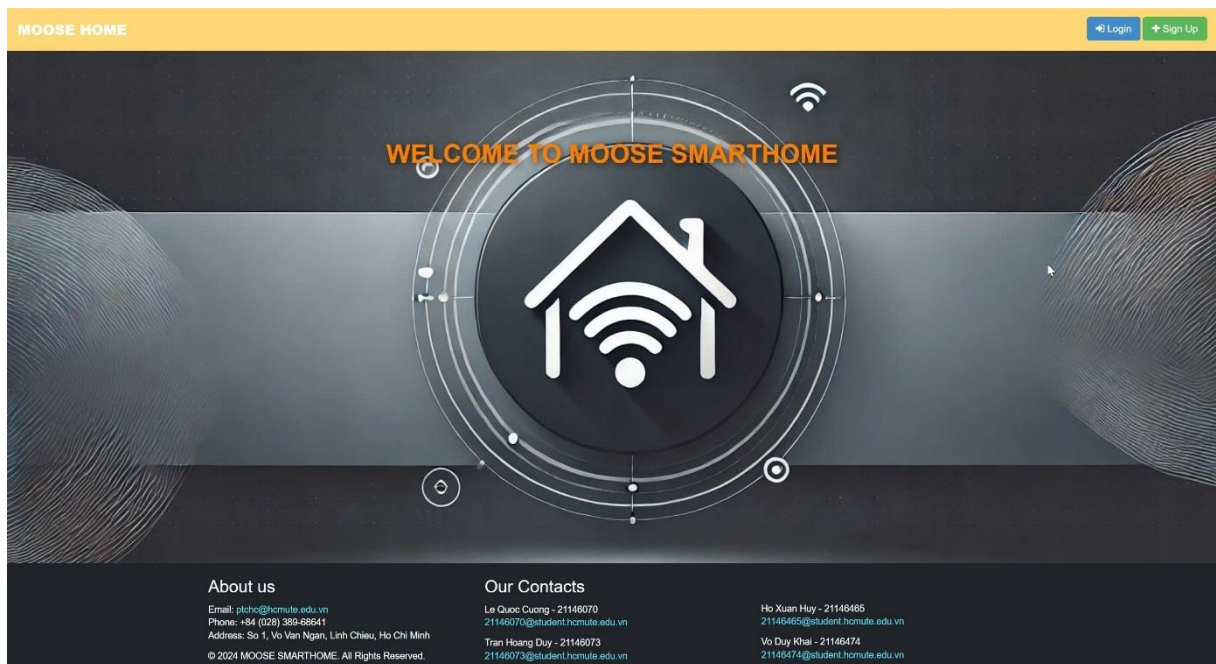
Trang web MooseHome được thiết kế với giao diện thân thiện, dễ hiểu và tương thích với nhiều loại thiết bị, từ điện thoại thông minh, máy tính bảng hoặc máy tính xách tay đến máy tính để bàn. Các tính năng của trang web được phát triển để đáp ứng nhu cầu đa dạng của người dùng, giúp bạn kiểm soát không chỉ các thiết bị trong nhà mà còn có thể thiết kế đưa ngôi nhà của bạn trở nên tự động hóa để nâng cao hiệu quả và tiện ích trong sinh hoạt hằng ngày. Bạn có thể dễ dàng bật/tắt đèn, điều chỉnh nhiệt độ phòng, theo dõi an ninh, hoặc thậm chí lập lịch tự động cho các thiết bị mà không cần phải ở gần chúng.

Mục tiêu chính của tài liệu này là cung cấp cho bạn một cái nhìn toàn diện và chi tiết về cách sử dụng và tận dụng tối đa các tính năng của trang web MooseHome. Cuốn sách được chia thành các phần rõ ràng, giúp bạn từng bước làm quen với giao diện và các chức năng cơ bản, từ việc đăng ký tài khoản cho đến các thao tác điều khiển thiết bị, thiết lập tự động hóa, tạo cảnh báo, và theo dõi hoạt động của các thiết bị thông minh trong ngôi nhà của bạn. Những hướng dẫn chi tiết kèm theo các hình ảnh minh họa sẽ giúp bạn dễ dàng tiếp cận và sử dụng hiệu quả trang web.

Bên cạnh đó, cuốn sách này cũng không quên dành một phần quan trọng cho các nhà phát triển và những người muốn mở rộng các tính năng của trang web. Chúng tôi sẽ cung cấp cái nhìn chi tiết về cấu trúc của hệ thống, các API và công cụ phát

triển mở rộng, giúp bạn hiểu rõ hơn về cách tích hợp trang web với các hệ thống khác, cũng như cách tạo ra các ứng dụng hoặc tính năng riêng biệt phù hợp với nhu cầu của bạn. Dù bạn là người mới bắt đầu hay một lập trình viên mới bắt đầu, phần này sẽ là tài liệu hữu ích để bạn có thể tùy chỉnh và phát triển các tính năng sáng tạo cho hệ thống MooseHome của mình.

Chúng tôi hy vọng tài liệu này sẽ là người bạn đồng hành tuyệt vời, mang lại một không gian sống tiện nghi, an toàn và hiện đại. Chúc bạn có những trải nghiệm tuyệt vời và hiệu quả khi sử dụng trang web MooseHome!



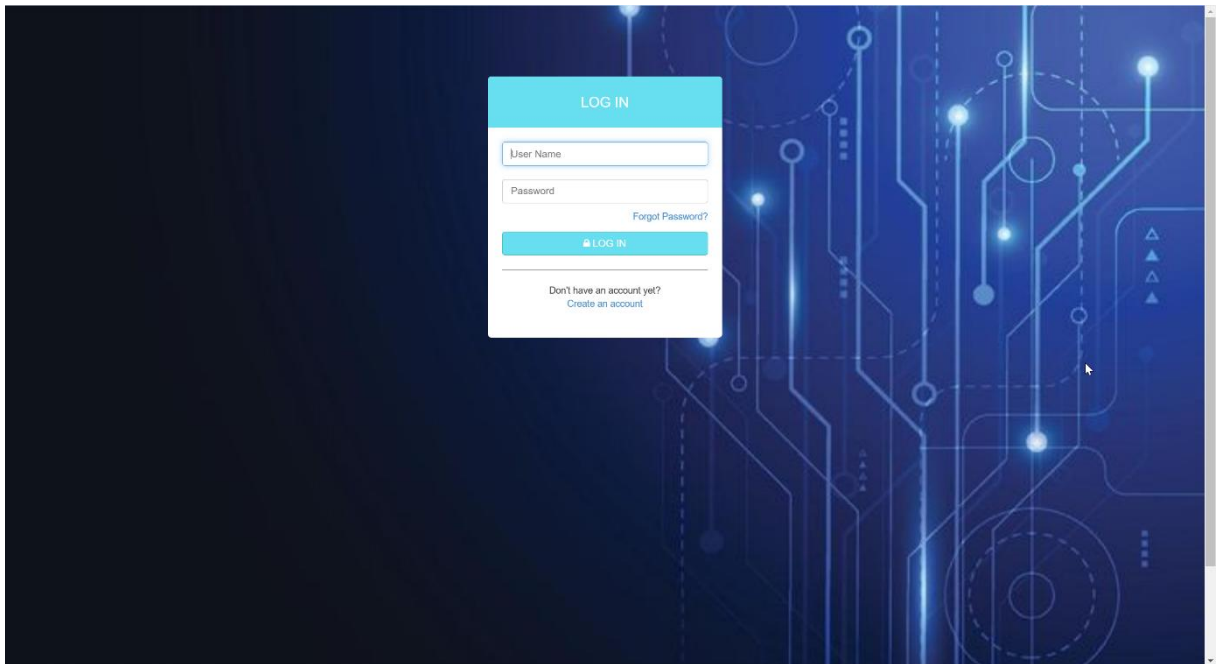
Hình 1: Hình ảnh HomePage của trang web MooseHome.

2. ĐIỀU HƯỚNG

2.1. Administrator

2.1.1. Đăng nhập vào hệ thống (Login)

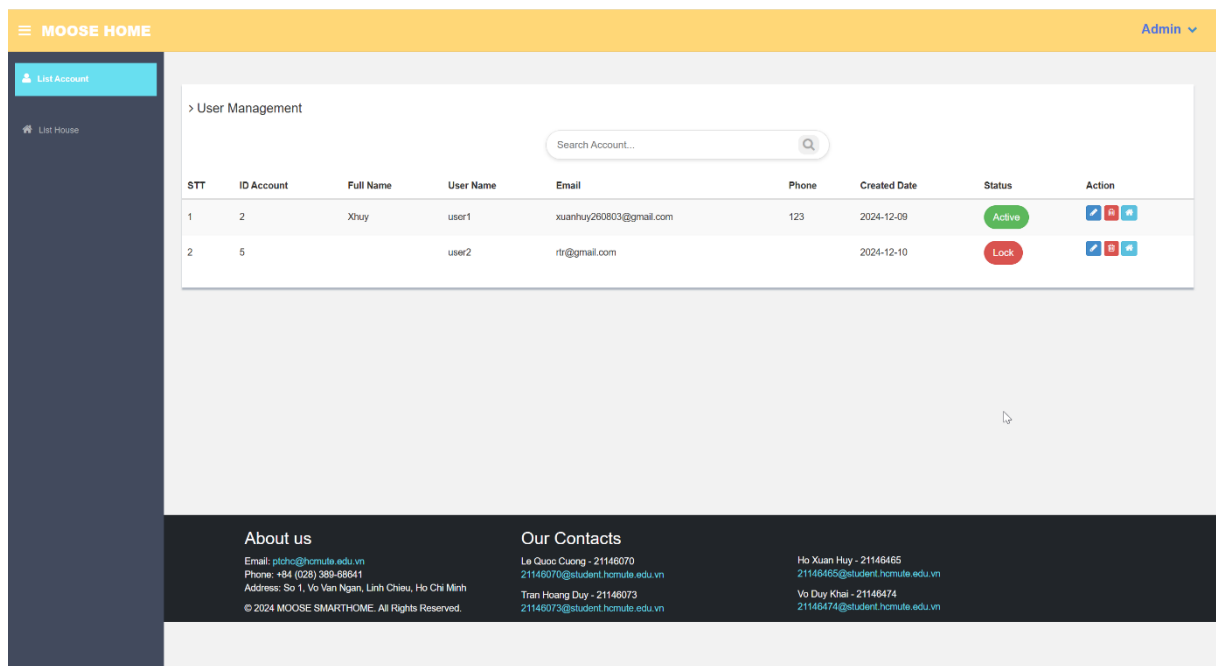
Bước 1: Tại vị trí trang chủ HomePage, hướng đến góc trên cùng bên trái người dùng sẽ thấy hai icon bao gồm “Login” và “Sign up”. Ở đây administrator cần chọn vào “Login”.



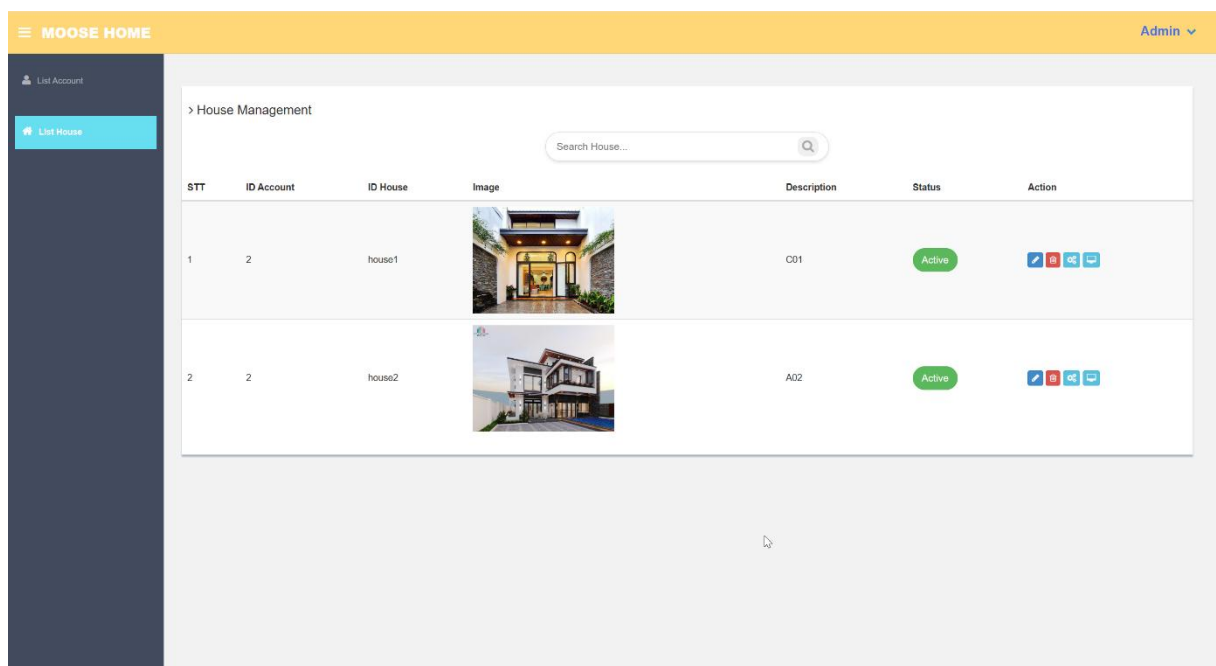
Hình 2: Hình ảnh trang Login của Administrator.

Bước 2: Điền thông tin đăng nhập của administrator đã được “Sign up” bao gồm: Username và Password. Sau đó chọn “Log In” để đăng nhập vào trang web với tư cách người quản lý.

Nếu bạn đăng nhập thành công, trang web sẽ được điều hướng đến trang quản lý cho các người dùng cá nhân như hình ảnh bên dưới.



Hình 3: Hình ảnh trang quản lý người dùng cá nhân của Administrator.





Hình 4: Hình ảnh trang quản lý danh sách nhà của tất cả người dùng cá nhân.

2.1.2. Quản lý thông tin của các người dùng cá nhân

Bước 1: Trong nhóm Action sẽ gồm các hành động:

Bảng 1: Bảng các thao tác trong nhóm Action của Administrator.

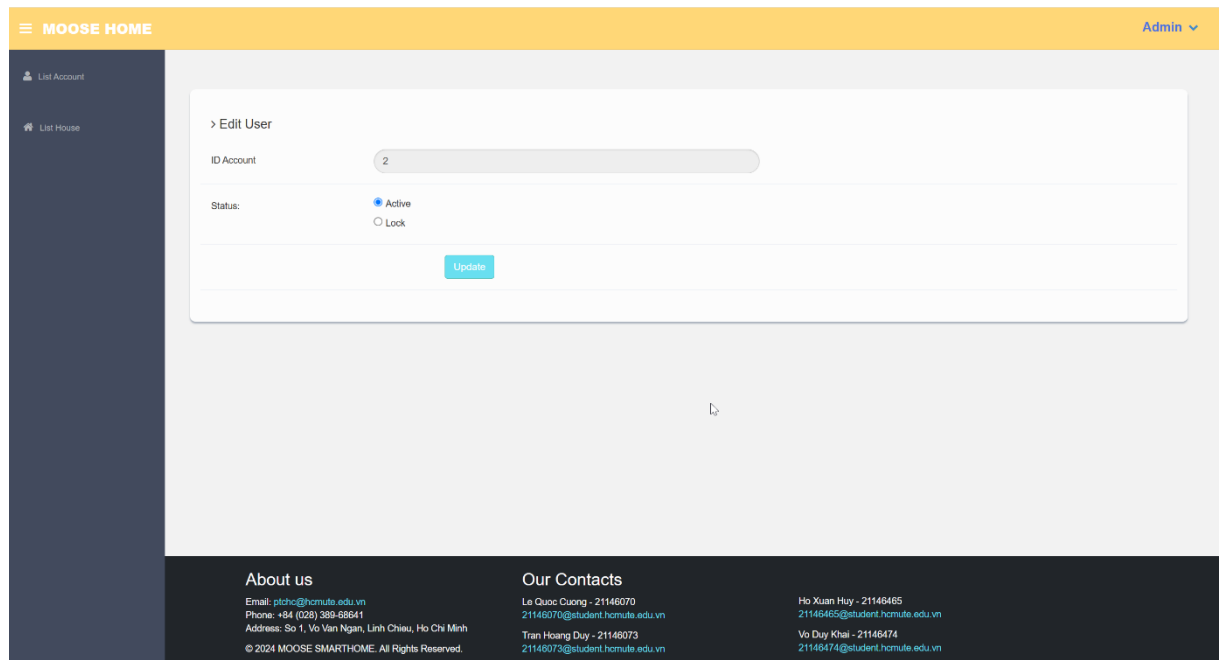
	Theo dõi ID Account.
--	----------------------

Edit User	Chỉnh sửa trạng thái đăng nhập tài khoản của người dùng cá nhân “Active” hoặc “Lock” .
 Remove User	Xóa tài khoản người dùng hiện tại.
 User House	Theo dõi được danh sách các ngôi nhà của người dùng đã đăng ký.

Tiếp theo chúng ta sẽ đến với từng thao tác trong nhóm Action của Administrator.

Bước 2: Đầu tiên là Edit User, Tại đây Admin có hai lựa chọn để thay đổi trạng thái của người dùng cá nhân:

- Active: Cho phép quyền đăng nhập cho tài khoản của người dùng cá nhân.
- Lock: Khóa quyền đăng nhập cho tài khoản của người dùng cá nhân.



The screenshot displays the 'Edit User' interface within the 'MOOSE HOME' application. The top navigation bar is orange with the 'MOOSE HOME' logo and an 'Admin' dropdown. The left sidebar is dark blue with links for 'List Account' and 'List House'. The main content area is light gray and contains the 'Edit User' form. The form has a title '> Edit User' and two input fields: 'ID Account' with the value '2' and 'Status' with radio buttons for 'Active' (selected) and 'Lock'. An 'Update' button is located at the bottom of the form. The footer is dark gray and contains contact information for MOOSE SMARTHOME, including email, phone, address, and a list of contacts with their names and phone numbers.

Hình 5: Hình ảnh trang Edit User.

Sau khi thay đổi trạng thái của người dùng cá nhân thì chọn **“Update”** để hoàn tất thay đổi.

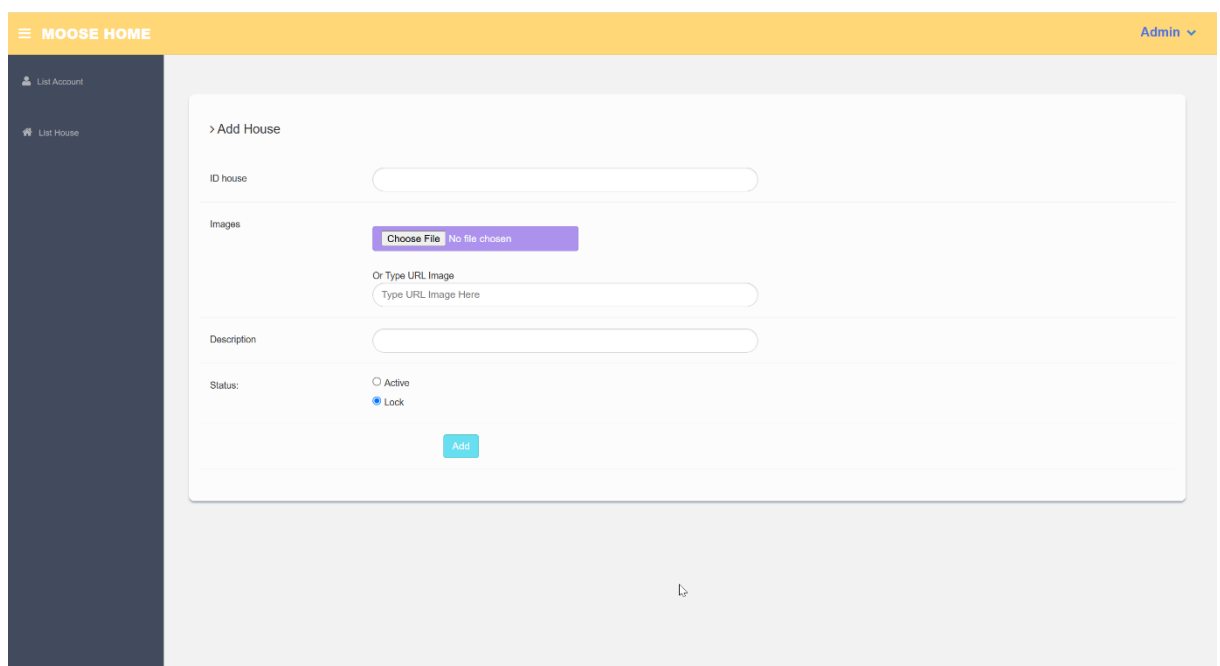
Bước 3: Tiếp theo là thao tác xóa tài khoản của người dùng cá nhân.

Tại thao tác này thì Admin chỉ cần chọn vào biểu tượng Remove User đã liệt kê trên *Bảng 1* để xóa tài khoản người dùng cá nhân ra khỏi cơ sở dữ liệu của trang web.

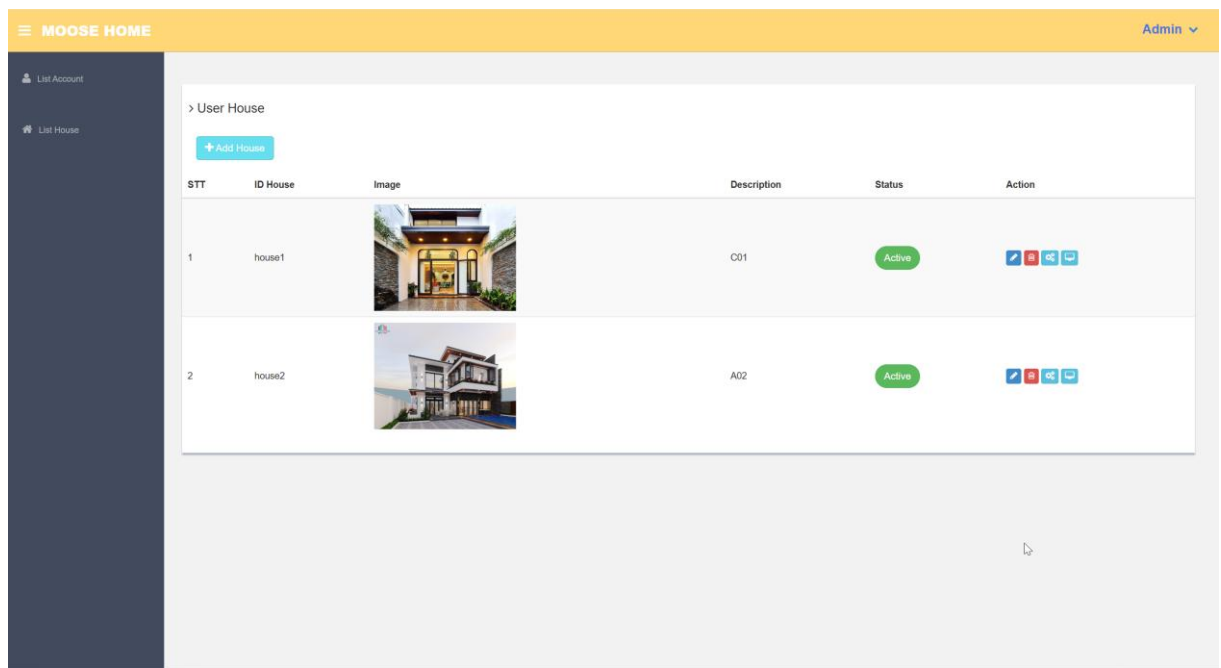
Bước 4: Quản lý danh sách nhà mà người dùng đã đăng ký.

Tại thao tác này người dùng cần chọn vào biểu tượng User House đã liệt kê trên *Bảng 1* để di chuyển vào trang User House như hình ảnh bên dưới.

Tại đây, trang User House sẽ có chức năng chính là thêm thêm ngôi nhà cho mỗi người dùng bằng cách chọn vào “Add House” phía trên danh sách ngôi nhà, sau đó Admin sẽ được truy cập vào trang Add House để thêm các ngôi nhà mới cho mỗi tài khoản cá nhân.

The screenshot shows the 'Add House' form within the 'MOOSE HOME' application. The interface has a yellow header bar with the text 'MOOSE HOME' and a user profile 'Admin' on the right. A dark blue sidebar on the left contains two menu items: 'List Account' and 'List House'. The main content area is titled '> Add House' and contains several input fields: 'ID house', 'Images' (with a 'Choose File' button and 'No file chosen' text), 'Or Type URL Image' (with a 'Type URL Image Here' input), 'Description', and 'Status' (with radio buttons for 'Active' and 'Lock', where 'Lock' is selected). A blue 'Add' button is at the bottom of the form.

Hình 6: Hình ảnh trang Add House.







Hình 7: Hình ảnh trang quản lý danh sách tất cả nhà đã đăng ký.

2.1.3. Quản lý tất cả ngôi nhà đã được đăng ký

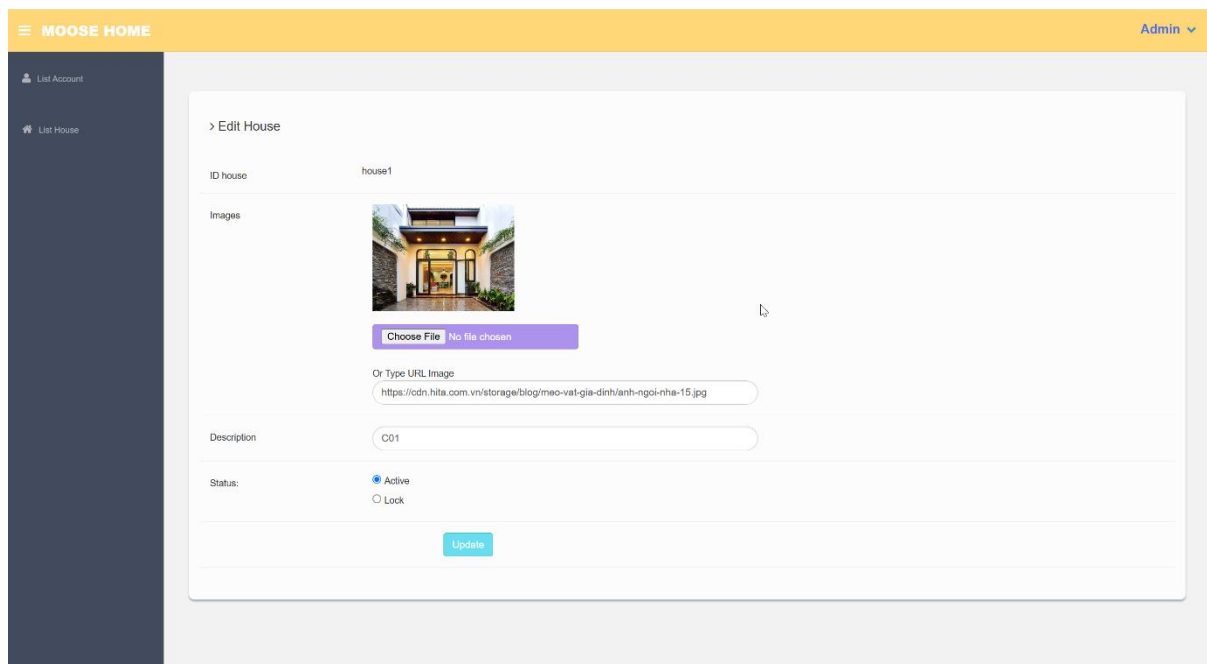
Sau khi vào được List House (Chọn ở Sidebar bên trái), chúng ta sẽ có các thao tác ở nhóm Action bao gồm:

Bảng 2: Bảng nhóm các thao tác trong trang quản lý nhà.

 Edit House	Theo dõi ID house của ngôi nhà. Chỉnh sửa nhà bao gồm: Image, Description, Status.
 Remove House	Xóa ngôi nhà ra khỏi danh sách đã đăng ký.
 Equipment	Xem thiết bị của ngôi nhà.
 IOT Monitor	Điều khiển và giám sát thiết bị IOT của ngôi nhà.

Bây giờ chúng ta sẽ đi vào từng chức năng của thể.

Bước 1: Sau khi thành công vào trang Edit House sẽ có hình ảnh giống bên dưới.



Hình 8: Hình ảnh trang Edit House.

Tại đây Admin có thể thay đổi các thông tin cơ bản của ngôi nhà về:

- Image: Admin có thể lấy hình ảnh trực tiếp được lưu trong thiết bị hoặc có thể lấy đường dẫn của hình ảnh trên trang web khác để cập nhật cho hình ảnh của ngôi nhà.
- Description: mô tả về ngôi nhà được đăng ký.
- Status: Thay đổi trạng thái của ngôi nhà. **“Active (cho phép người dùng cá nhân truy cập vào ngôi nhà của mình)”** hoặc **“Lock (khóa khả năng truy cập vào ngôi nhà của người dùng cá nhân)”**.

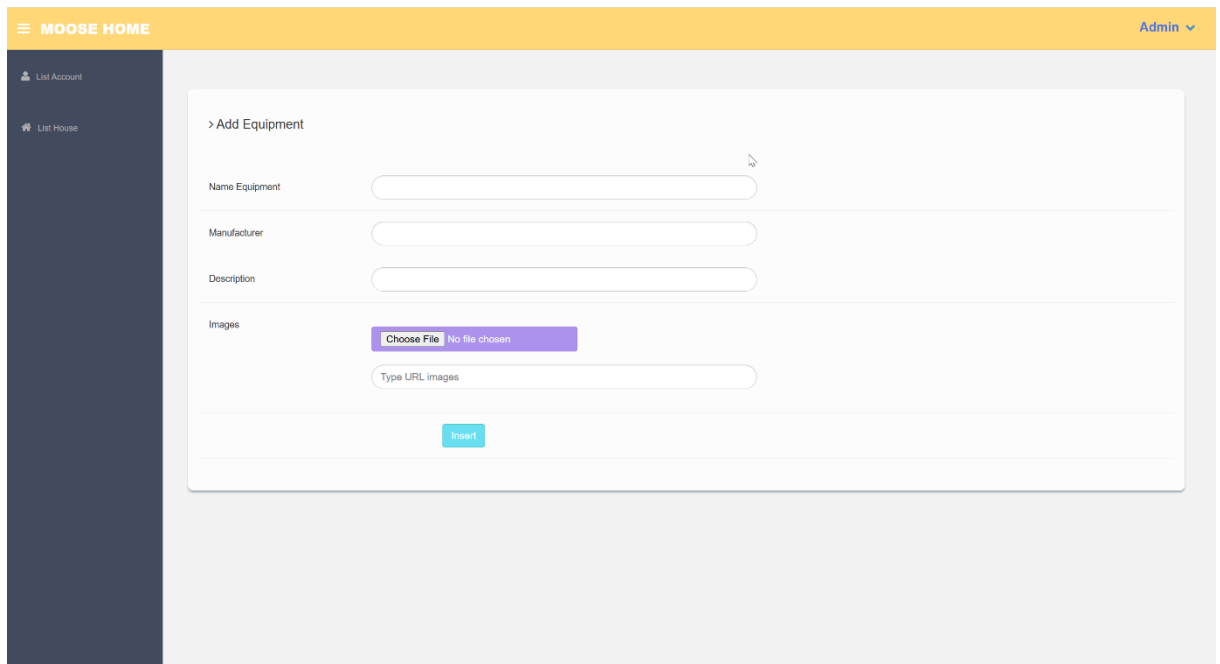
Cuối cùng cập nhật thông tin mới nhất mà Admin đã thay đổi bằng cách chọn vào “Update” ở cuối cùng của trang Edit House.

Bước 2: Tiếp theo là thao tác xóa ngôi nhà đã được đăng ký của tất cả người dùng cá nhân.

Tại thao tác này thì Admin chỉ cần chọn vào biểu tượng Remove House đã liệt kê trên *Bảng 2* để xóa ngôi nhà người dùng cá nhân ra khỏi cơ sở dữ liệu của trang web.

Bước 3: Tiếp theo là thao tác theo dõi danh sách các thiết bị IOT có trong ngôi nhà.

Tại đây, trang Equipment sẽ có chức năng chính là thêm các thiết bị IOT của ngôi nhà bằng cách chọn vào “Add Equipment” phía trên danh sách các thiết bị sau đó Admin sẽ được truy cập vào trang Add Equipment để thêm các thiết bị mới cho ngôi nhà.

The screenshot shows a web interface for 'MOOSE HOME'. On the left is a dark sidebar with 'List Account' and 'List House' links. The main area has a yellow header with 'MOOSE HOME' and 'Admin' on the right. Below the header is a white box titled '> Add Equipment'. Inside this box are four input fields: 'Name Equipment', 'Manufacturer', 'Description', and 'Images'. The 'Images' field has a 'Choose File' button (labeled 'No file chosen') and a 'Type URL Images' input field. At the bottom of the form is a blue 'Insert' button.

Hình 9: Hình ảnh trang Add Equipment.

Tại đây, Admin sẽ có các thao tác bao gồm:

- Name Equipment: Điền tên của thiết bị.
- Manufacturer: Nơi sản xuất.
- Description: Mô tả về thiết bị.
- Images: Hình ảnh của thiết bị, tại đây Admin có thể lấy hình ảnh trực tiếp lưu trong thiết bị hoặc đường dẫn của hình ảnh từ trang web khác.

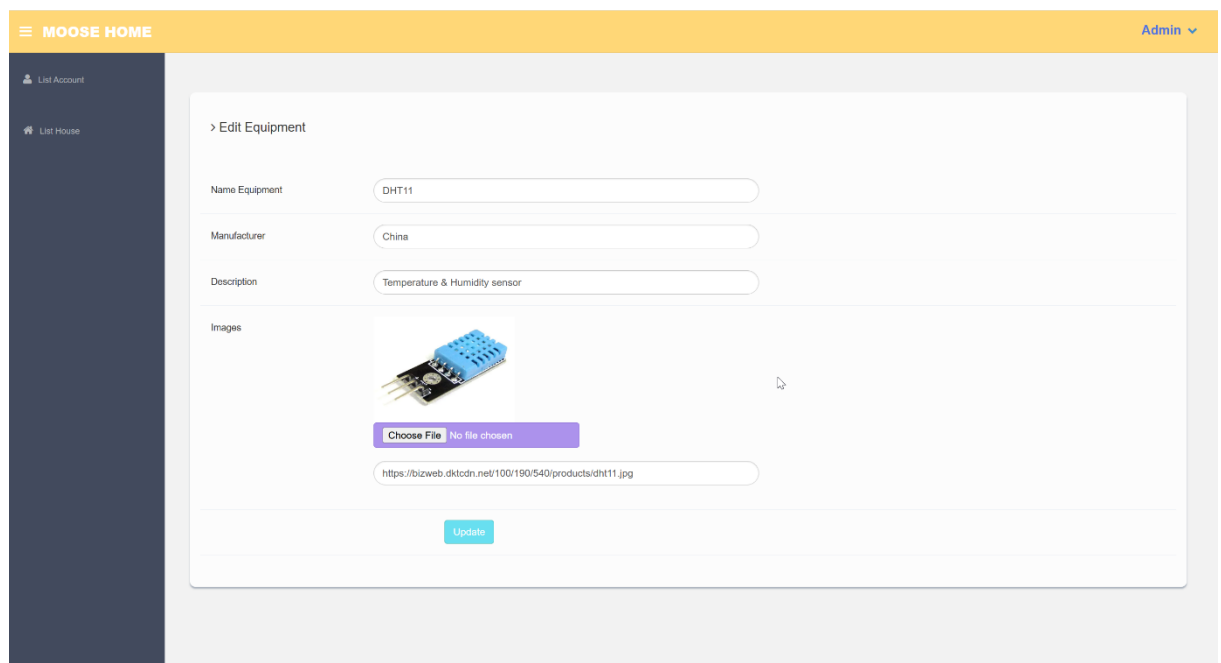
Sau khi đã xác nhận chính xác các thông tin thì Admin chỉ cần chọn “Insert” để thêm một thiết bị mới vào ngôi nhà được chọn.

Tiếp theo là nhóm Action mà Admin có thể thao tác với các thiết bị IOT của ngôi nhà. Tương tự với các trang trước đó, tại đây người dùng có thể chỉnh sửa thông tin của thiết bị hoặc xóa đi thiết bị bên trong ngôi nhà.

Bên dưới là hình ảnh khi thành công vào trang web có thể chỉnh sửa thông tin của thiết bị có trong ngôi nhà bao gồm:

- Name Equipment: Tên thiết bị.
- Manufacturer: Nơi sản xuất.
- Description: Mô tả về thiết bị.
- Images: Hình ảnh của thiết bị, tại đây Admin có thể lấy hình ảnh trực tiếp lưu trong thiết bị hoặc đường dẫn của hình ảnh từ trang web khác.

Cuối cùng, sau khi hoàn tất thay đổi thì Admin chỉ cần chọn “*Update*” để cập nhật thông tin mới nhất về thiết bị của ngôi nhà.

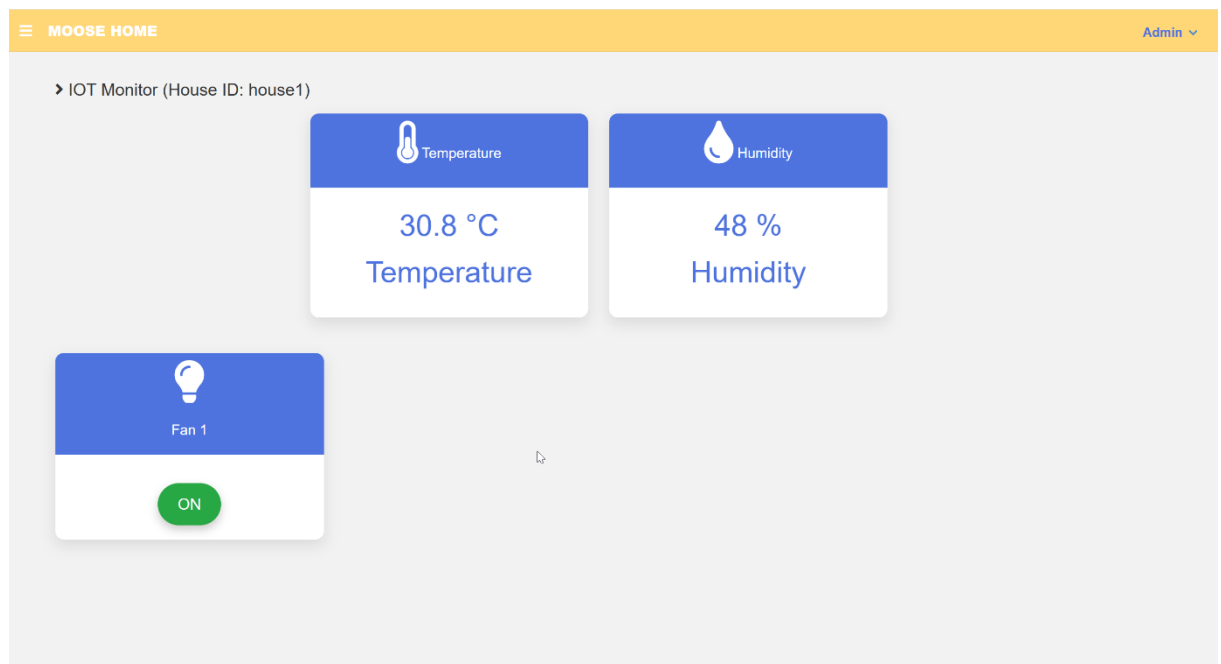


The screenshot displays the 'Edit Equipment' interface within the 'MOOSE HOME' application. The top navigation bar is orange with 'MOOSE HOME' on the left and 'Admin' on the right. A dark sidebar on the left contains 'List Account' and 'List House'. The main content area is titled '> Edit Equipment' and includes three text input fields: 'Name Equipment' with the value 'DHT11', 'Manufacturer' with 'China', and 'Description' with 'Temperature & Humidity sensor'. Below these is an 'Images' section featuring a small image of a blue DHT11 sensor module, a 'Choose File' button, and a text input field with the URL 'https://bizweb.dktcdn.net/100/190/540/products/dht11.jpg'. An 'Update' button is positioned at the bottom of the form.

Hình 10: Hình ảnh trang Edit Equip.

Bên cạnh đó còn có thể xóa thiết bị khỏi danh sách bằng cách chọn vào biểu tượng Remove để xóa ra khỏi cơ sở dữ liệu của trang web.

Bước 4: Giám sát và điều khiển các thiết bị IOT của ngôi nhà bằng cách chọn vào biểu tượng “*IOT Monitor*” đã được liệt kê ở *Bảng 2*.

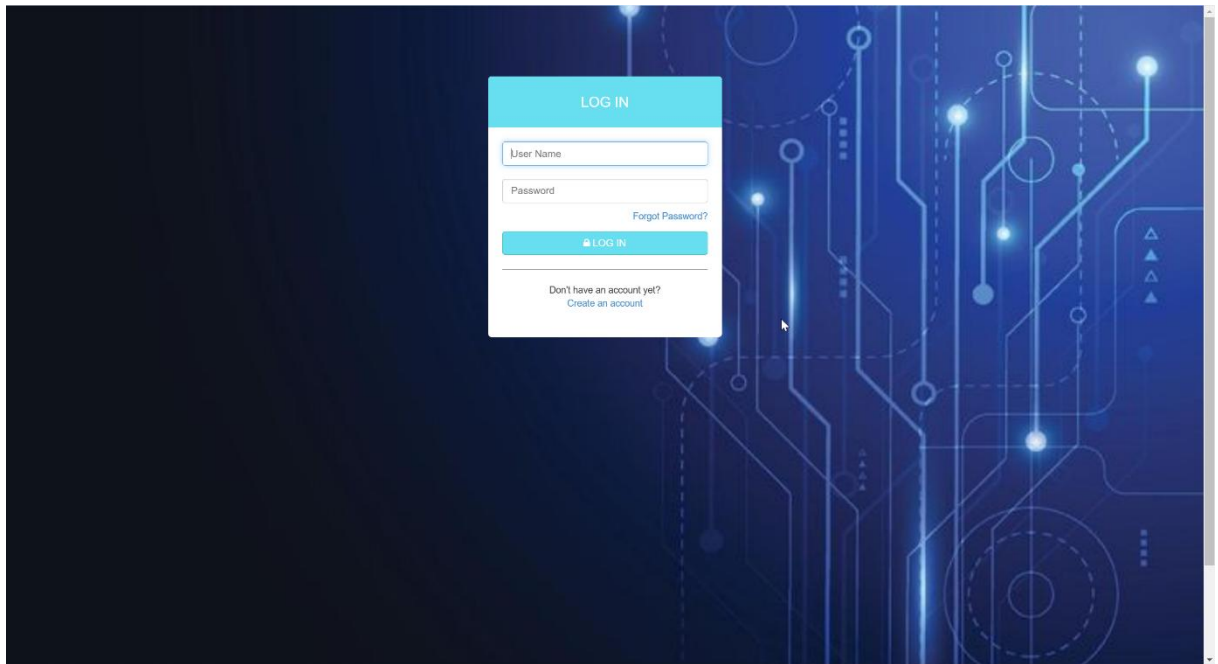


Hình 11: Hình ảnh trang IOT Monitor.

2.2. Personal User

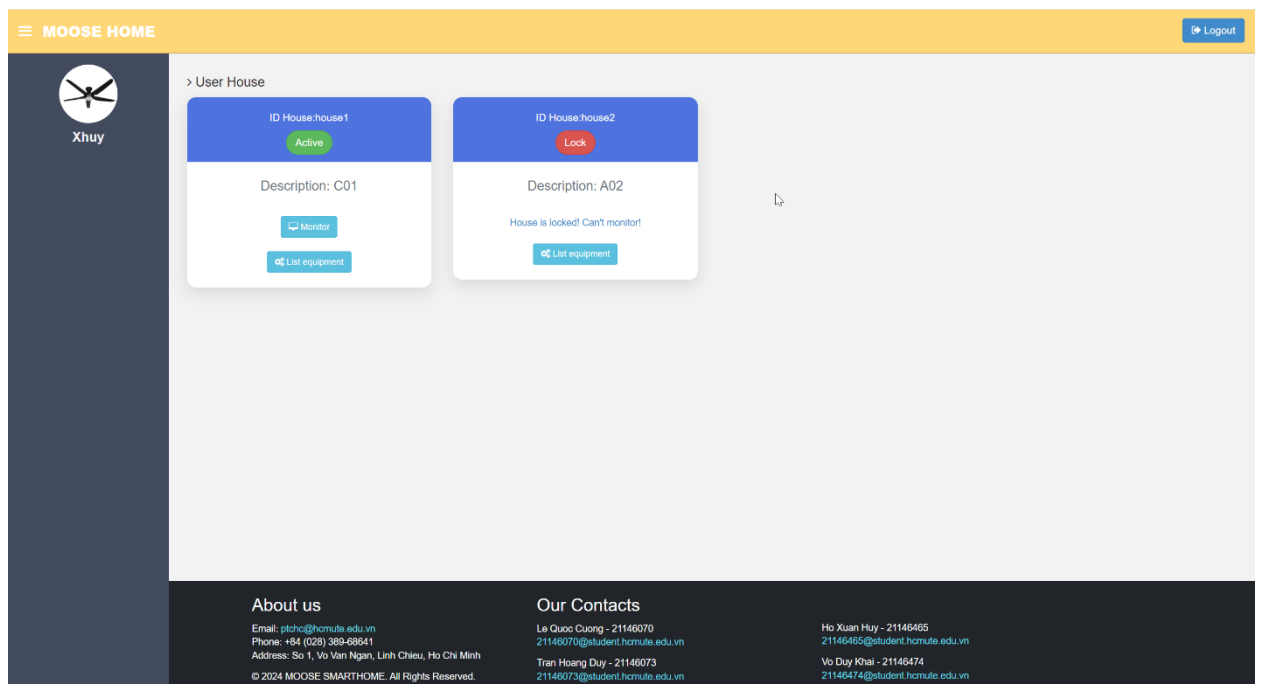
2.2.1. Đăng nhập vào hệ thống (Login)

Bước 1: Tại vị trí trang chủ HomePage, hướng đến góc trên cùng bên trái người dùng sẽ thấy hai icon bao gồm “*Login*” và “*Sign up*”. Ở đây personal user cần chọn vào “*Login*”.



Hình 12: Hình ảnh trang Login cho Personal User.

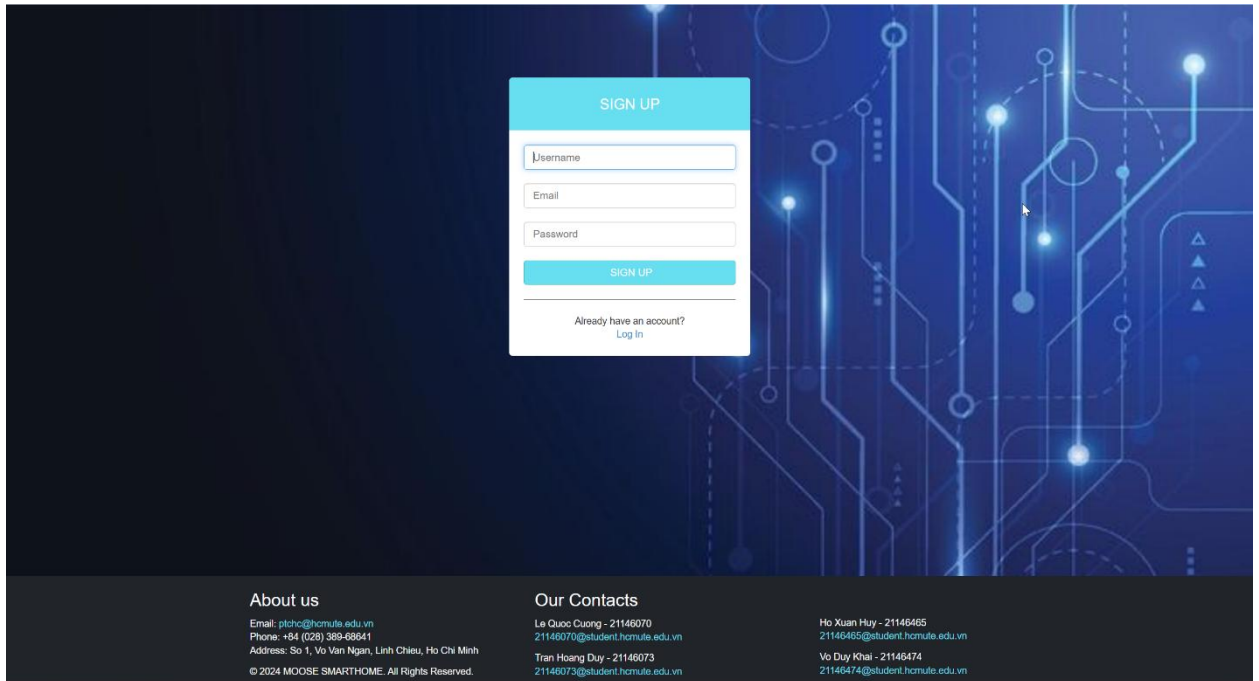
Bước 2: Điền thông tin đăng nhập của personal user đã được “Sign up” bao gồm: Username và Password. Sau đó chọn “Log In” để đăng nhập vào trang web với tư cách người dùng cá nhân. Nếu thành công, bạn sẽ được điều hướng đến trang theo dõi và giám sát các ngôi nhà mà bạn đã đăng ký như hình ảnh bên dưới.



Hình 13: Hình ảnh trang quản lý người dùng cá nhân của Personal user.

2.2.2. Đăng ký người dùng cá nhân (Sign Up)

Bước 1: Tại vị trí trang chủ HomePage, hướng đến góc trên cùng bên trái người dùng sẽ thấy hai icon bao gồm “Login” và “Sign up”. Ở đây personal user cần chọn vào “Sign up”.



The image shows a 'SIGN UP' form on a website. The form is centered and has a light blue header. It contains three input fields: 'Username', 'Email', and 'Password'. Below these fields is a blue 'SIGN UP' button. At the bottom of the form, there is a link 'Already have an account? Log in'. The background of the page is dark blue with a circuit-like pattern. The footer contains contact information for MOOSE SMARTHOME.

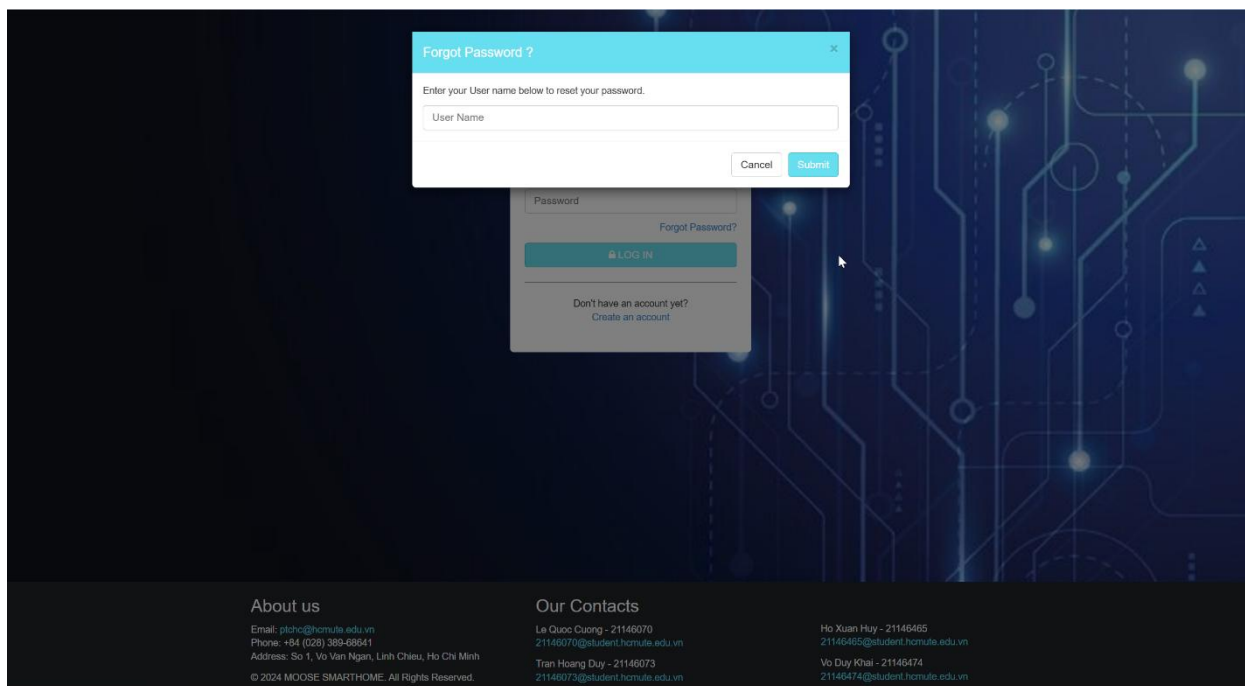
Hình 14: Hình ảnh trang Sign Up của người dùng cá nhân.

Bước 2: Điền thông tin tài khoản của personal user bao gồm: Username, Email và Password. Sau đó chọn “Sign Up” để đăng ký tài khoản mới cho người dùng cá nhân. Nếu thành công, bạn sẽ được điều hướng đến trang “**đăng nhập**” Hình 13. Sau đó bạn sẽ tiếp tục Login theo hướng dẫn ở phần trước.

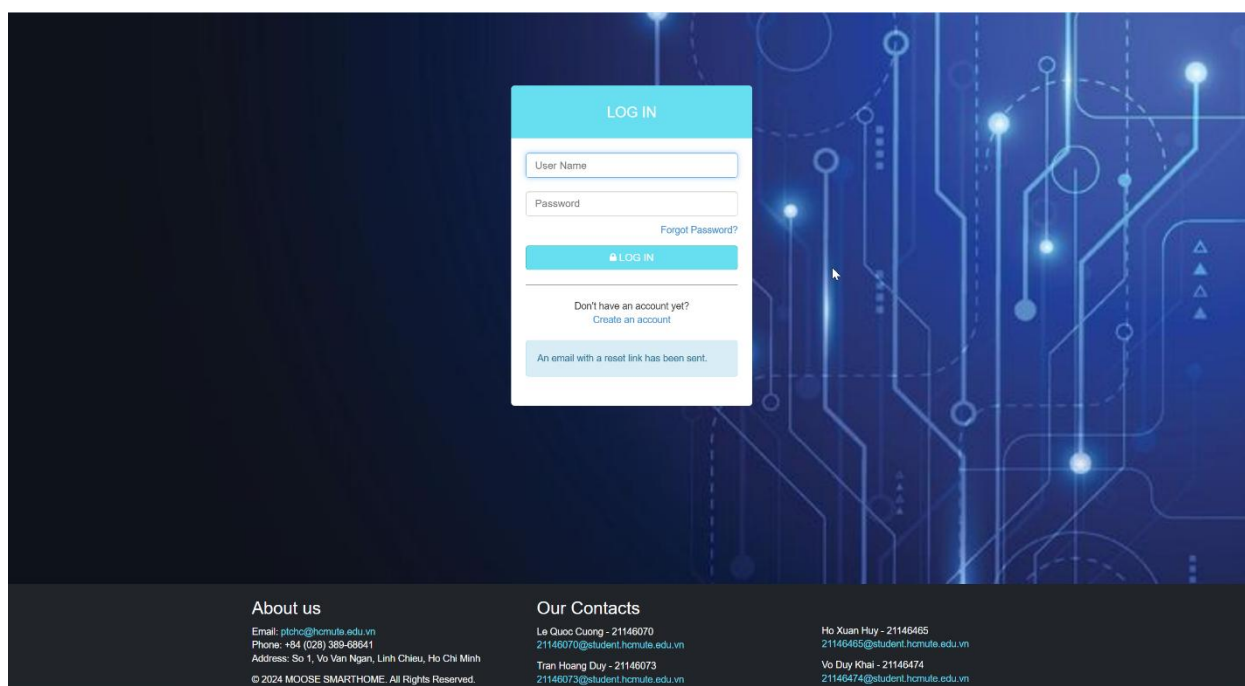
**Lưu ý: Nếu bạn đã vào trang Login, bạn vẫn có thể tạo một tài khoản mới bằng cách “Create an account” để tiến hành đăng ký một tài khoản mới như hướng dẫn ở phần này.*

2.2.3. Quên mật khẩu (Forgot Password)

Ở **Hình 12** bạn sẽ thấy dòng “Forgot Password” bên trên Login Button. Tại đây, bạn cần click chọn vào đó để tiến hành tạo lại mật khẩu mới cho tài khoản của mình bằng cách điền “User Name” mà người dùng đã đăng ký và nhấn Submit.

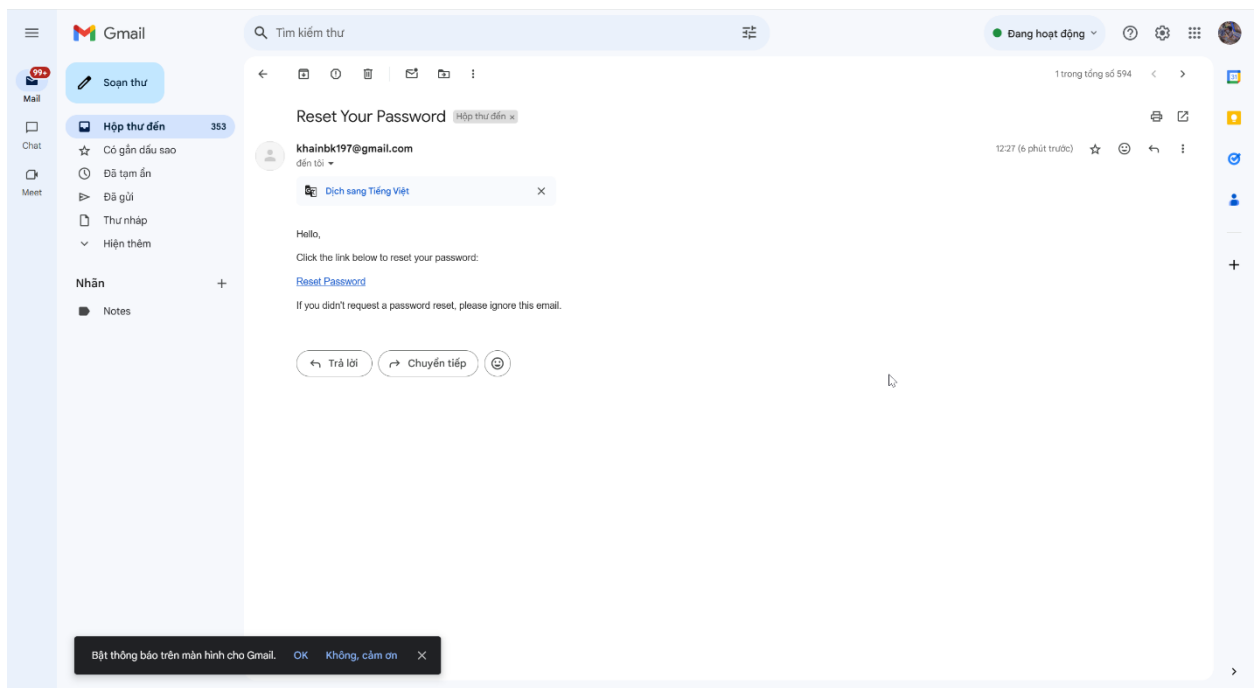


Hình 15: Hình ảnh xin cấp phép lấy lại mật khẩu.



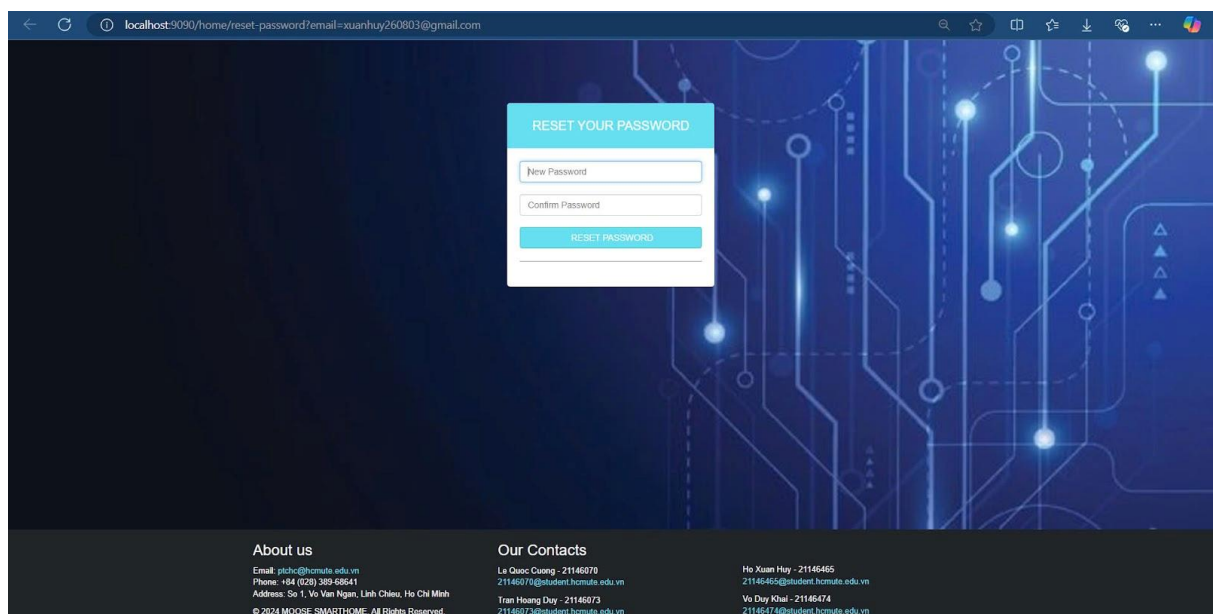
Hình 16: Hình ảnh Submit Forgot Password thành công.

Bước 1: Bạn cần nhấn vào link Reset Password trong email nhận được để được cấp phép thiết lập lại mật khẩu.



Hình 17: Hình ảnh email cấp lại mật khẩu.

Bước 2: Sau đó bạn sẽ được điều hướng đến trang “Reset Your Password”, tiến hành điền mật khẩu mới và nhấn chọn “Reset Password” để thiết lập lại mật khẩu mới. Nếu thành công sẽ có thông báo cho bạn.

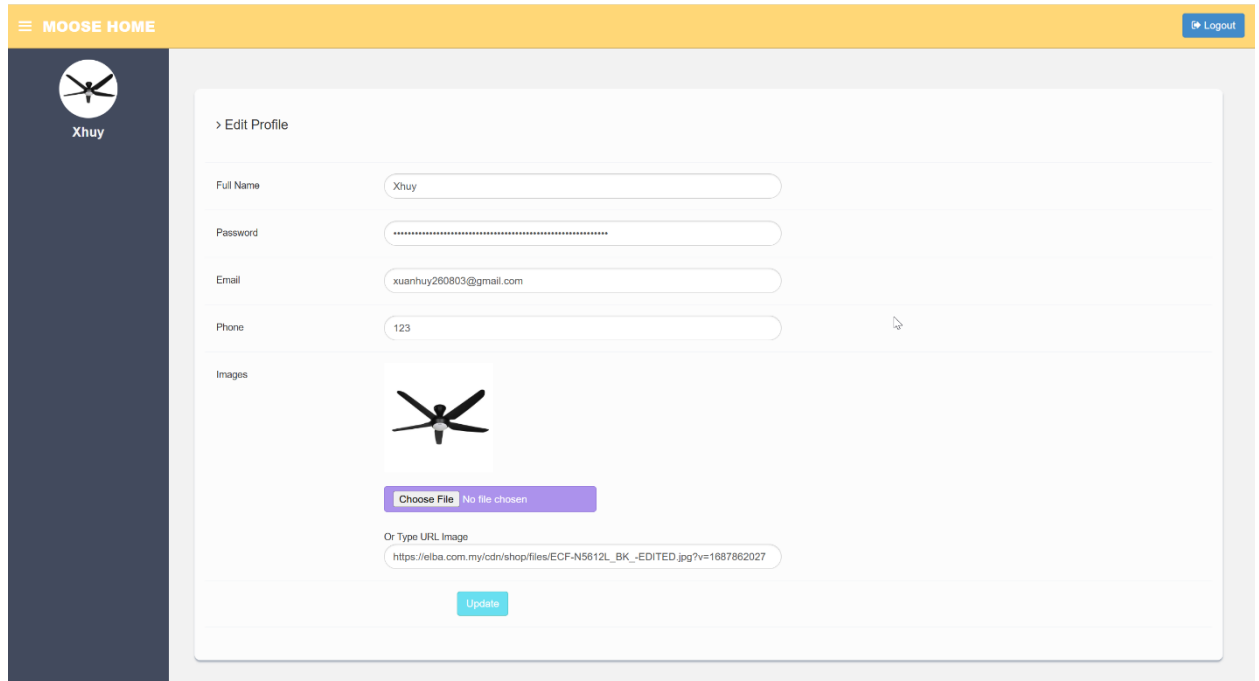


Hình 18: Hình ảnh trang Reset Your Password.

Bước 3: Sau đó bạn sẽ được điều hướng đến trang “Login” và tiến hành đăng nhập vào tài khoản của mình như thường lệ.

2.2.4. Xem và Sửa lại thông tin cá nhân (Edit & Update)

Bước 1: Bắt đầu sau khi bạn đã đăng nhập thành công, di chuyển đến góc trên cùng bên phải, bạn sẽ thấy biểu tượng người dùng. Hãy click chọn vào biểu tượng người dùng bạn sẽ được điều hướng đến trang “*Edit Profile*”.



Hình 19: Hình ảnh trang Edit Profile.

Bước 2: Tại đây bạn hãy cập nhật thông tin của bản thân theo từng mục. Sau đó chọn vào Update Button được để được cập nhật thông tin người dùng mới nhất.

2.2.5. Thao tác của với các ngôi nhà

Trong các ngôi nhà mà bạn đang ký sẽ có hai trạng thái bao gồm:

- **Active:** Bạn được phép giám sát và thao tác với các thiết bị IoT có trong ngôi nhà.
- **Lock:** Ngược lại với trạng thái Active, bởi vì Administrator đã khóa quyền giám sát và thao tác với các thiết bị IoT của bạn đối với ngôi nhà đó.

Trường hợp Active:

- + Monitor: Giám sát và thay đổi trạng thái của các thiết bị IoT của trong ngôi nhà.
- + List Equipment: Xem thông tin và tổng hợp các thiết bị có trong ngôi nhà của bạn.

Trường hợp Lock: Khác với trường hợp Active, bạn chỉ có thể chọn “*List Equipment*” bởi vì tính năng “*Monitor*” đã bị Administrator khóa.

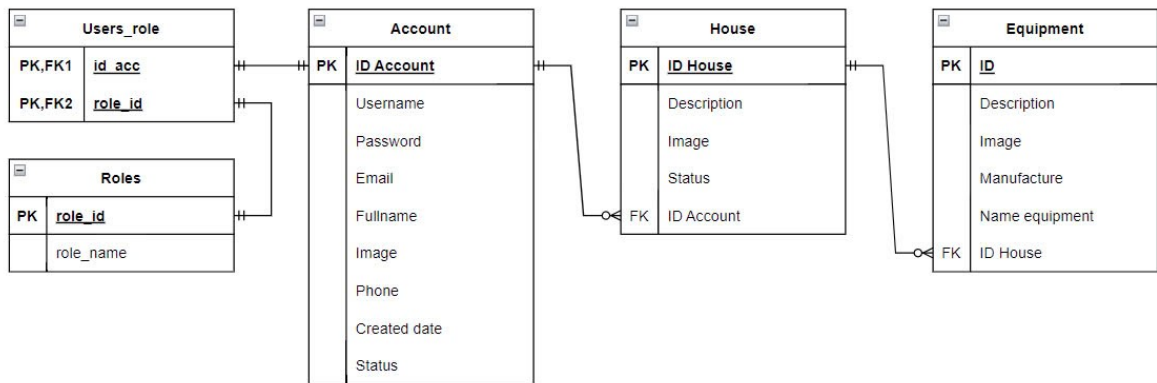
2.2.6. Đăng xuất (Logout)

Ở thao tác này, người dùng chỉ cần đơn giản là chọn vào Logout Button ở góc trên cùng bên tay phải. sau đó bạn sẽ được điều hướng quay trở lại trang chủ của MooseHouse.

3. CHỨC NĂNG

Moose Home cung cấp một loạt các chức năng được thiết kế để hỗ trợ người dùng quản lý và tối ưu hóa hệ thống nhà thông minh. Các chức năng chính bao gồm:

3.1. Biểu đồ Quan hệ thực thể (Entity Relationship Diagram)



Hình 20: Entity Relationship Diagram (ERD).

– Bảng Account

+ **Mô tả:** Quản lý thông tin tài khoản người dùng trong hệ thống.

+ **Thuộc tính:**

- ID Account (Khóa chính): Định danh duy nhất cho mỗi tài khoản.
- Username, Password: Thông tin đăng nhập của người dùng.
- Email, Phone, Fullname, Image: Thông tin cá nhân của người dùng.
- Created date: Ngày tạo tài khoản.
- Status: Trạng thái tài khoản (kích hoạt hoặc không).
- is_admin: Xác định tài khoản có quyền admin hay không.

+ **Quan hệ:** Một tài khoản có thể quản lý nhiều ngôi nhà (mỗi quan hệ 1 - N với bảng House).

– Bảng House

+ **Mô tả:** Đại diện cho thông tin ngôi nhà trong hệ thống MooseHome.

+ **Thuộc tính:**

- ID House (Khóa chính): Định danh duy nhất cho mỗi ngôi nhà.
- Description: Mô tả chi tiết về ngôi nhà.
- Image: Hình ảnh của ngôi nhà.
- Status: Trạng thái hoạt động của ngôi nhà.
- ID Account (Khóa ngoại): Liên kết với tài khoản quản lý ngôi nhà.

+ **Quan hệ:**

- Một ngôi nhà thuộc về một tài khoản.
- Một ngôi nhà có thể chứa nhiều thiết bị (mối quan hệ 1-N với bảng Equipment).

– **Bảng Equipment**

+ **Mô tả:** Quản lý thông tin về các thiết bị IoT trong hệ thống.

+ **Thuộc tính:**

- ID (Khóa chính): Định danh duy nhất cho mỗi thiết bị.
- Description: Mô tả chi tiết về thiết bị.
- Image: Hình ảnh minh họa thiết bị.
- Manufacture: Nhà sản xuất thiết bị.
- Name equipment: Tên của thiết bị.
- ID House (Khóa ngoại): Liên kết với ngôi nhà chứa thiết bị.

+ **Quan hệ:** Một thiết bị chỉ thuộc về một ngôi nhà.

Tóm tắt mối quan hệ:

- **Account ↔ House**: Một tài khoản có thể quản lý nhiều ngôi nhà, nhưng mỗi ngôi nhà chỉ thuộc về một tài khoản.
- **House ↔ Equipment**: Một ngôi nhà có thể chứa nhiều thiết bị IoT, nhưng mỗi thiết bị chỉ thuộc về một ngôi nhà.

3.2. Chức năng Security

```
@Bean
public UserDetailsService userDetailsService() {
    return new UserService();
}
```

Khai báo một bean kiểu `UserDetailsService`, là giao diện do Spring Security cung cấp, có trách nhiệm lấy thông tin người dùng (username, password, vai trò, v.v.) để phục vụ cho việc xác thực.

```
@Bean
public BCryptPasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

Sử dụng thuật toán **BCrypt** để mã hóa mật khẩu. Đây là thuật toán bảo mật cao, được khuyến nghị trong việc lưu trữ mật khẩu.

```
@Bean
public DaoAuthenticationProvider authenticationProvider() {
    DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
    authProvider.setUserDetailsService(customUserDetailsService);
    authProvider.setPasswordEncoder(passwordEncoder());
    return authProvider;
}
```

Phương thức **authenticationProvider()** trong lớp cấu hình bảo mật là một Bean của Spring Framework. Phương thức này được sử dụng để thiết lập và trả về một đối tượng **DaoAuthenticationProvider**, chịu trách nhiệm xử lý xác thực người dùng dựa trên cơ sở dữ liệu.

```
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(customUserDetailsService)
        .passwordEncoder(passwordEncoder());
}
```

Phương thức `configure(AuthenticationManagerBuilder auth)`:

- + Cấu hình cách Spring Security xác thực người dùng.
- + Sử dụng `UserDetailsService` để tải thông tin người dùng.
- + Sử dụng `PasswordEncoder (BCrypt)` để kiểm tra mật khẩu một cách an toàn.
- + Đảm bảo rằng quá trình xác thực hoạt động hiệu quả và bảo mật cao.

```
@Bean
public AuthenticationManager authenticationManager(AuthenticationConfiguration
authConfig) throws Exception {
    final List<GlobalAuthenticationConfigurerAdapter> configurers = new
    ArrayList<>();
    configurers.add(new GlobalAuthenticationConfigurerAdapter() {
        @Override
        public void configure(AuthenticationManagerBuilder auth)
        throws Exception {
            auth.userDetailsService(customUserDetailsService);
        }
    });
    return authConfig.getAuthenticationManager();
}
```

Phương thức **`authenticationManager()`**: Tạo một **`AuthenticationManager`** sử dụng các cấu hình toàn cục. Gắn một **`UserDetailsService`** để tải thông tin người dùng. Đảm bảo quá trình xác thực linh hoạt, bảo mật, và dễ dàng mở rộng với các cơ chế xác thực bổ sung.

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    return http
        .csrf(csrf -> csrf.disable())
        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/home/**", "/assets/**").permitAll()
            .requestMatchers("/user/**", "/user/profile/**").hasAnyAuthority("ROLE_USER", "
            ROLE_ADMIN")
            .requestMatchers("/admin/**").hasAuthority("ROLE_ADMIN")
            .requestMatchers(HttpMethod.GET, "/api/**").permitAll()
            .requestMatchers("/api/**").permitAll()
        )
        .formLogin(login -> login
            .loginPage("/home/login")
            .loginProcessingUrl("/home/login")
            .defaultSuccessUrl("/home/waiting", true)
            .failureUrl("/home/login?error=Incorrect User or Password")
        )
    ;
}
```

```

        .permitAll()
    )
    .logout(logout -> logout
        .logoutUrl("/home/logout")
        .logoutSuccessUrl("/home/")
        .clearAuthentication(true)
        .deleteCookies("JSESSIONID")
        .permitAll() )
    .exceptionHandling(handling -> handling.accessDeniedPage("/403"))
    .build();
}

```

+ Quản lý quyền truy cập:

- Các URL bắt đầu bằng */home/* và */assets/* được phép truy cập mà không cần xác thực hay quyền truy cập.
- Chỉ cho phép người dùng có vai trò *ROLE_USER* hoặc *ROLE_ADMIN* truy cập vào các URL bắt đầu với */user/* hoặc */user/profile/*.
- Chỉ người dùng có vai trò *ROLE_ADMIN* mới có quyền truy cập vào các URL bắt đầu với */admin/*.
- Chỉ định rằng tất cả các yêu cầu HTTP phương thức GET đối với các URL bắt đầu với */api/* sẽ được phép truy cập công khai, không yêu cầu xác thực.
- Tất cả các yêu cầu HTTP (bao gồm cả POST, PUT, DELETE, v.v.) đối với các URL bắt đầu với */api/* cũng sẽ được phép truy cập công khai mà không cần xác thực

+ Đăng nhập (Authentication)

- Cho phép mọi người dùng (kể cả chưa đăng nhập) truy cập vào các URL liên quan đến đăng nhập và xử lý đăng nhập mà không cần xác thực.
- Khi người dùng chưa đăng nhập và cố gắng truy cập một trang yêu cầu xác thực, họ sẽ được chuyển hướng đến trang */home/login*.
- Sau khi người dùng đăng nhập thành công, họ sẽ được chuyển hướng đến trang */home/waiting*.

- Nếu người dùng nhập sai tên người dùng hoặc mật khẩu, họ sẽ được chuyển hướng lại đến trang đăng nhập */home/login* và một tham số lỗi (error = Incorrect User or Password)

+ Đăng xuất

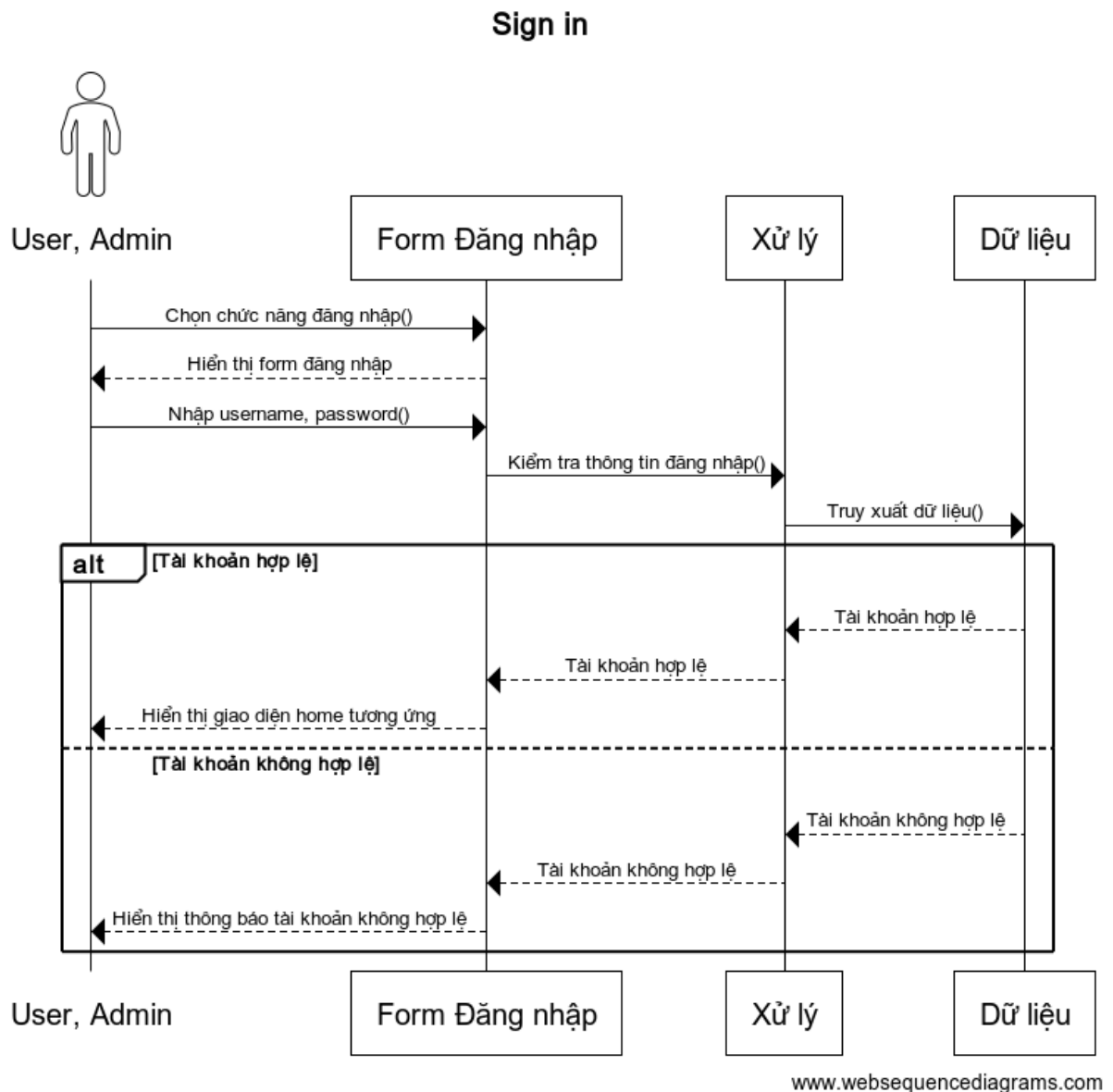
- Khi người dùng muốn đăng xuất khỏi ứng dụng, họ sẽ gửi yêu cầu HTTP đến URL này */home/logout*.
- Sau khi người dùng đăng xuất thành công, họ sẽ được chuyển hướng đến trang */home/*.
- Xóa tất cả thông tin xác thực (authentication) của người dùng khi họ đăng xuất.
- Xóa cookie phiên (session cookie) có tên là JSESSIONID khi người dùng đăng xuất
- Cho phép mọi người dùng (bao gồm người dùng chưa đăng nhập) truy cập vào URL đăng xuất và thực hiện hành động đăng xuất mà không cần phải xác thực.

3.3. Các chức năng xác thực người dùng

Mô tả: Người dùng có thể tạo tài khoản, đăng nhập, đặt lại mật khẩu và cập nhật thông tin cá nhân.

3.3.1. Chức năng đăng nhập

– Diagrams



Hình 21: Sequence Diagram trang Sign in

Đăng nhập (Login)

```

@GetMapping("/login")
public String showLoginPage(@RequestParam(value = "error", required = false) String error, Model model) {
    if (error != null) {
        model.addAttribute("error", error);
    }
    model.addAttribute("loginForm", new Account());
    return "login_temp.html";
}
  
```

+ **Hiển thị trang đăng nhập:** Khi người dùng truy cập vào URL /login, hàm này sẽ được gọi và trả về trang đăng nhập (login_temp.html).

- + Hiển thị thông báo lỗi (nếu có): Nếu tham số error tồn tại trong URL (ví dụ khi đăng nhập không thành công), hàm sẽ thêm thông báo lỗi vào model, giúp hiển thị lỗi trên trang đăng nhập
- + Tạo đối tượng đăng nhập trống: Hàm tạo một đối tượng Account trống và thêm vào model dưới tên "loginForm". Điều này giúp hiển thị một form đăng nhập rỗng, sẵn sàng để người dùng nhập tên đăng nhập và mật khẩu.
- + Trả về view đăng nhập: Cuối cùng, hàm trả về tên của trang HTML (login_temp.html), nơi chứa form đăng nhập và các thông báo lỗi (nếu có).

Phục vụ cho việc hiển thị trang đăng nhập với khả năng hiển thị thông báo lỗi khi đăng nhập không thành công và chuẩn bị form đăng nhập cho người dùng.

```
@GetMapping("/waiting")
public String checkIfAdmin(Model model) {
    Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
    if (authentication == null || !authentication.isAuthenticated() ||
authentication.getPrincipal() instanceof String) {
        model.addAttribute("error", "Account does not exist");
        return "login_temp.html"; // Quay lại trang đăng nhập
    }

    String username = authentication.getName();
    Account account = userService.findbyUser(username);

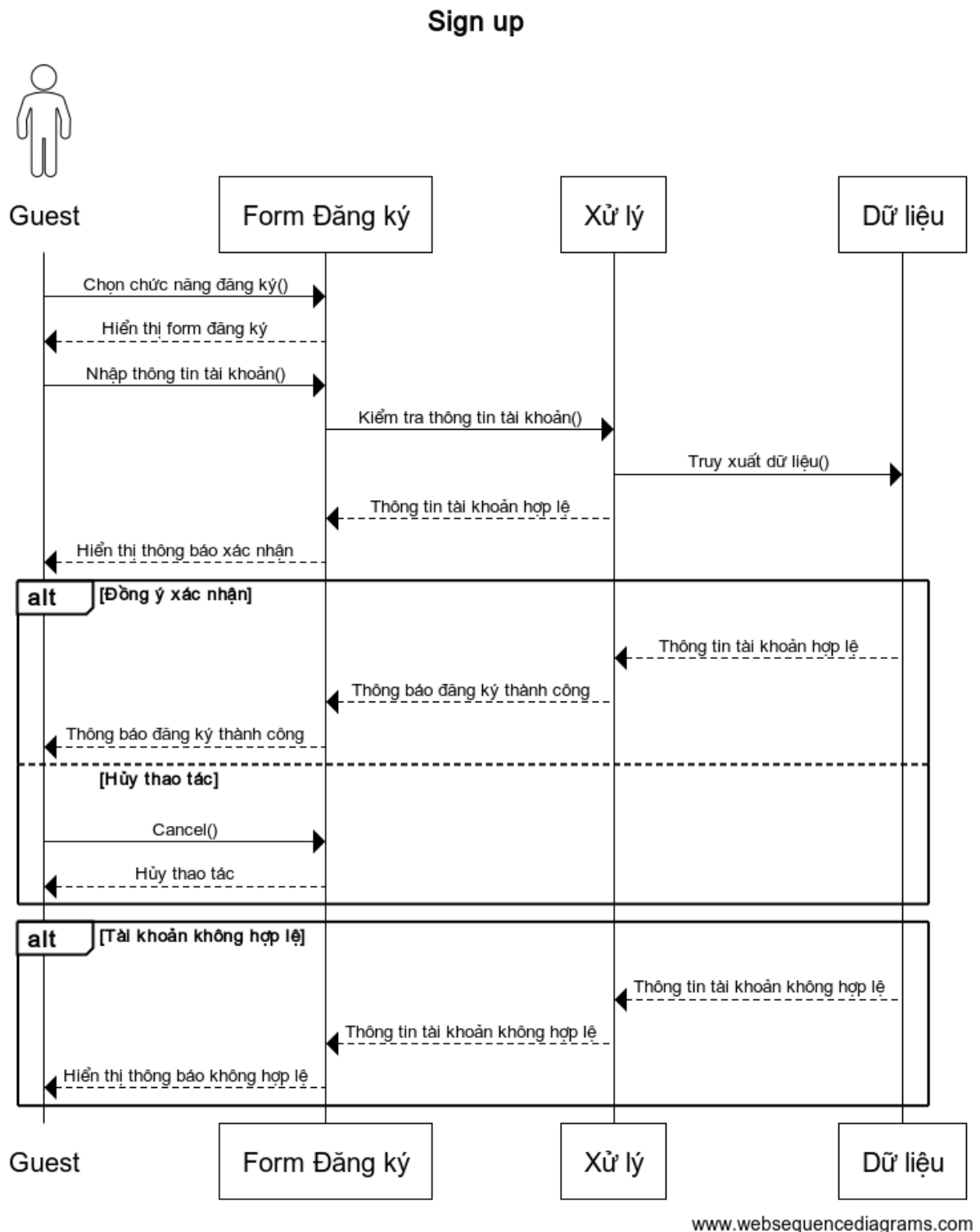
    if (account != null && !account.isStatus()) {
        SecurityContextHolder.clearContext();
        model.addAttribute("error", "Account is locked");
        return "login_temp.html";
    }
    if (authentication.getAuthorities().stream().anyMatch(role ->
role.getAuthority().equals("ROLE_ADMIN"))) {
        return "redirect:/admin/account";
    }

    return "redirect:/user/home";
}
```

Xác thực người dùng, đảm bảo rằng chỉ những người dùng hợp lệ, có tài khoản không bị khóa và có quyền truy cập phù hợp mới có thể tiếp tục truy cập trang tiếp theo. Nếu một trong các điều kiện không được thỏa mãn (chưa đăng nhập, tài khoản

bị khóa, không phải admin), người dùng sẽ được chuyển hướng về trang đăng nhập với thông báo lỗi thích hợp.

3.3.2. Chức năng đăng ký



Hình 22: Sequence Diagram trang Sign up.

```
@GetMapping("/register")
public String showRegisterForm(Model model) {
    model.addAttribute("registerForm", new Account());
    return "register_temp.html";
}
```

Giúp hiển thị trang đăng ký cho người dùng, đồng thời tạo ra một đối tượng Account trống để bind dữ liệu từ form đăng ký của người dùng. Người dùng sẽ nhập thông tin vào form và khi gửi, thông tin sẽ được xử lý theo các bước tiếp theo.

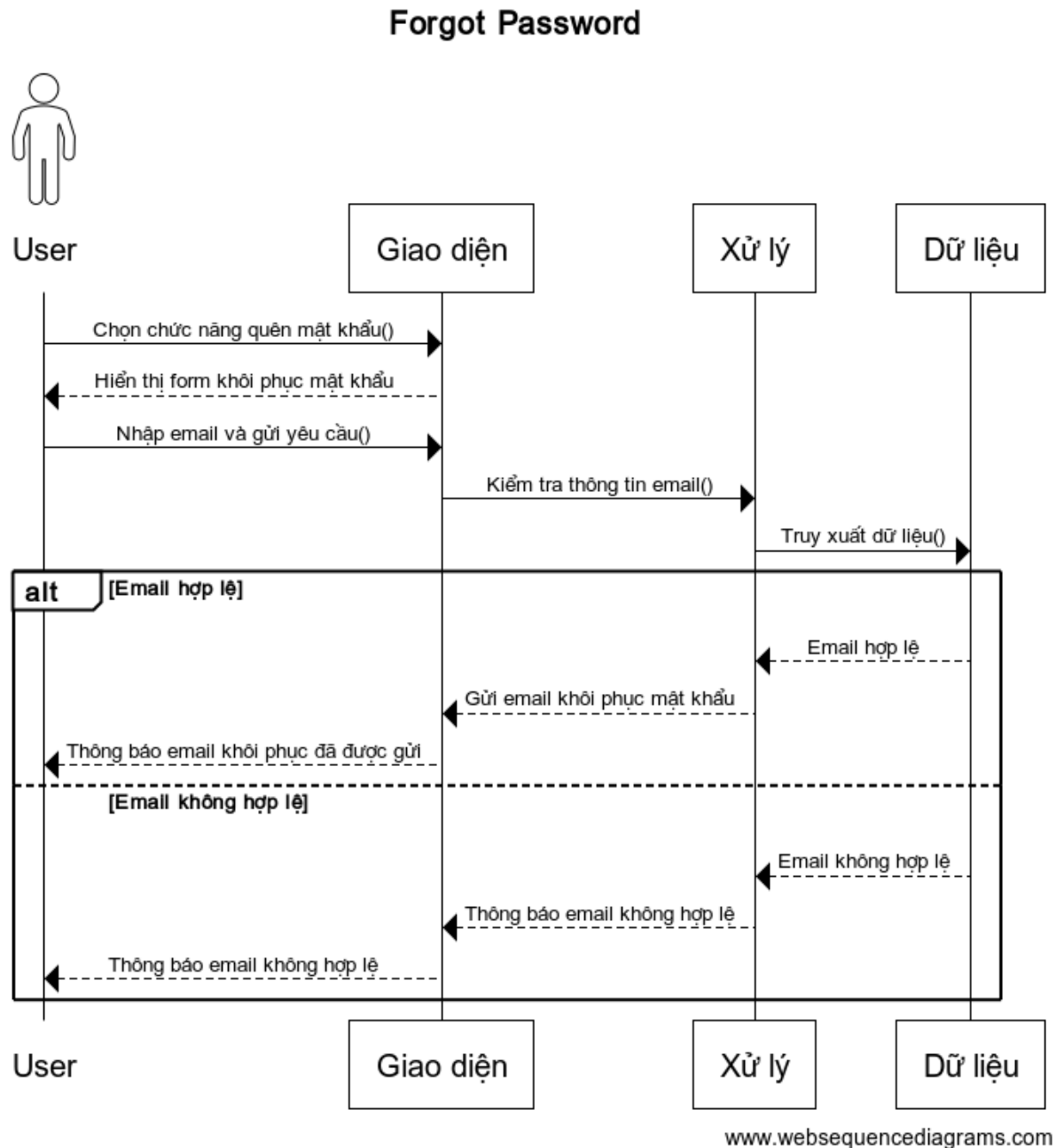
```
@PostMapping("/register")
public String processRegister(@ModelAttribute Account registerForm, Model model) {
    if (userService.CheckUserExist(registerForm.getUsername())) {
        model.addAttribute("error", "Username is existed");
        model.addAttribute("registerForm", new Account());
        return "register_temp.html";
    }

    if (userService.CheckEmailExist(registerForm.getEmail())) {
        model.addAttribute("error", "Email is existed");
        model.addAttribute("registerForm", new Account());
        return "register_temp.html";
    }

    Roles role = roleRepository.findByName("USER").get();
    registerForm.setPassword(passwordEncoder.encode(registerForm.getPassword(
)));
    registerForm.setRoles(Collections.singleton(role));
    registerForm.setCrDate(LocalDate.now());
    registerForm.setStatus(true);
    userService.insert(registerForm);
    return "redirect:/home/login";
}
```

- + Kiểm tra xem tên đăng nhập và email đã tồn tại trong cơ sở dữ liệu chưa.
- + Mã hóa mật khẩu trước khi lưu.
- + Gán vai trò, ngày tạo tài khoản và trạng thái tài khoản cho người dùng.
- + Lưu tài khoản người dùng mới vào cơ sở dữ liệu.
- + Nếu mọi thứ hợp lệ, chuyển hướng người dùng đến trang đăng nhập.

3.3.3. Chức năng quên mật khẩu



Hình 23: Sequence Diagram trang Forgot Password.

Gửi email:

```
spring.mail.host=smtp.gmail.com
spring.mail.port=465
spring.mail.username=khainbk197@gmail.com
spring.mail.password=blnd vdttd adna ntda
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=false
spring.mail.properties.mail.debug=true
```

```
spring.mail.properties.mail.smtp.ssl.enable=true
```

Có chức năng cấu hình kết nối và gửi email từ ứng dụng Spring Boot qua máy chủ SMTP của Gmail.

```
public void sendEmail(String to, String subject, String body) {  
    SimpleMailMessage message = new SimpleMailMessage();  
    message.setFrom("khainbk197@gmail.com");  
    message.setTo(to);  
    message.setSubject(subject);  
    message.setText(body);  
    mailSender.send(message);  
}
```

Cho phép ứng dụng gửi một email đơn giản từ địa chỉ *khainbk197@gmail.com* tới địa chỉ email nhận, với tiêu đề và nội dung do người gọi phương thức cung cấp.

```
public void sendResetPasswordEmail(String email) {  
    String resetUrl = "http://localhost:9090/home/reset-password?email=" + email;  
    String subject = "Reset Your Password";  
    String message = "<p>Hello,</p>" +  
        "<p>Click the link below to reset your password:</p>" +  
        "<a href=\"" + resetUrl + "\">Reset Password</a>" +  
        "<p>If you didn't request a password reset, please ignore this  
email.</p>";  
    MimeMessage mimeTypeMessage = mailSender.createMimeMessage();  
    try {  
        MimeMessageHelper helper = new MimeMessageHelper(mimeTypeMessage,  
true);  
        helper.setTo(email);  
        helper.setSubject(subject);  
        helper.setText(message, true);  
        mailSender.send(mimeTypeMessage);  
    } catch (MessagingException e) {  
        e.printStackTrace();  
        throw new RuntimeException("Failed to send email");  
    }  
}
```

Các bước thực hiện:

- + Tạo URL reset mật khẩu: Dùng địa chỉ email của người dùng để tạo ra URL reset mật khẩu.

- + Cấu hình email: Cài đặt các thông tin cơ bản như người nhận (setTo), tiêu đề (setSubject), và nội dung (setText với tham số true để hỗ trợ HTML).
- + Gửi email bằng mailSender
- + Xử lý lỗi: Nếu có lỗi trong quá trình gửi email (ví dụ: lỗi cấu hình hoặc kết nối), một MessagingException sẽ được ném ra và chương trình sẽ in lỗi và ném một ngoại lệ mới (RuntimeException).

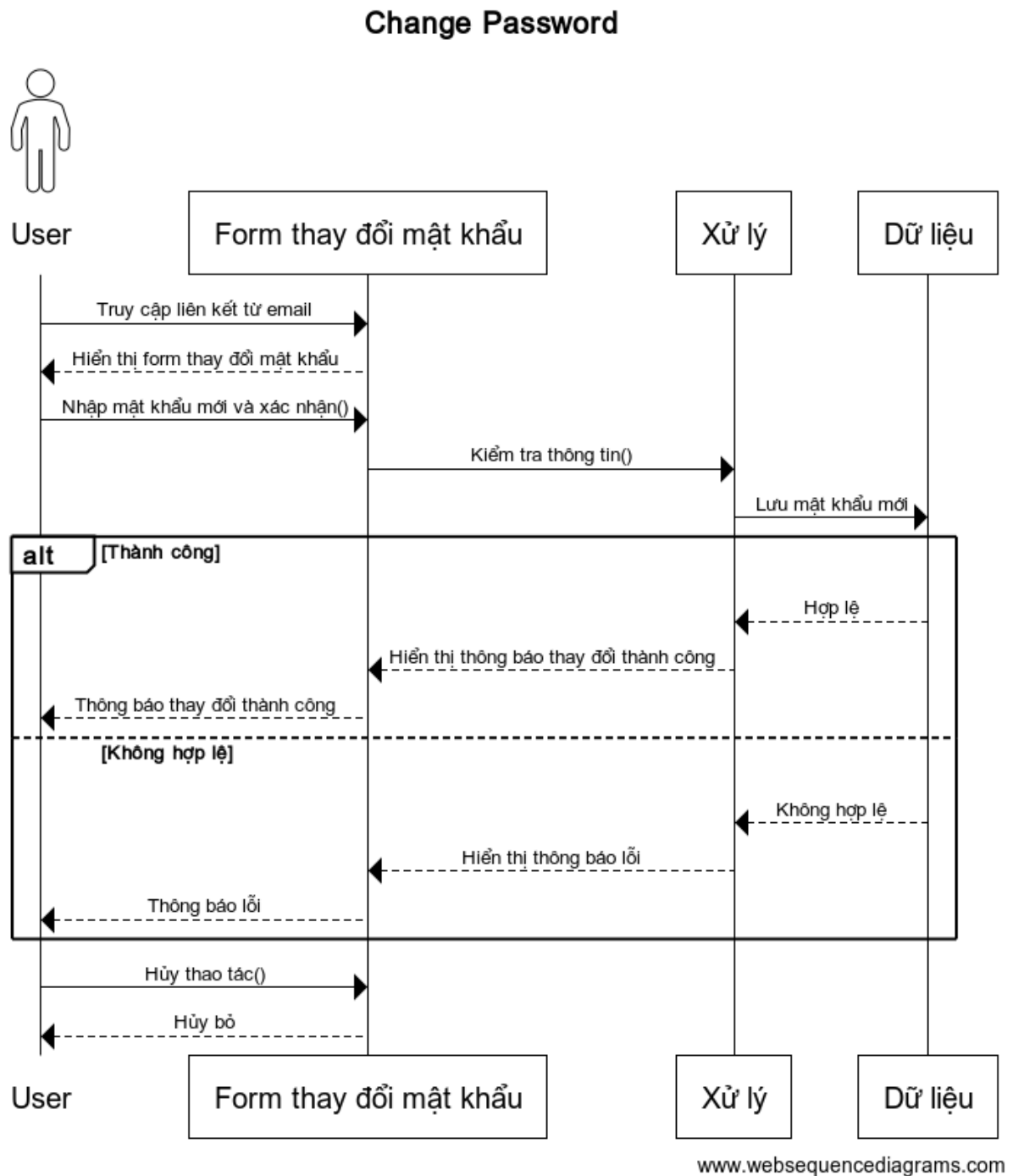
```

public String processForgotPassword(@RequestParam("username") String username,
Model model) {
    try {
        Account account = userService.findbyUser(username);
        if (account != null) {
            emailService.sendResetPasswordEmail(account.getEmail());
            model.addAttribute("message", "An email with a reset link has been sent.");
        }
        else {
            model.addAttribute("message", "UserName isn't exist");
        }
    } catch (Exception e) {
        model.addAttribute("error", "There was an error sending the email. Please
try again.");
    }
    return "login_temp.html";
}

```

- + Lấy tên đăng nhập người dùng từ form quên mật khẩu.
- + Kiểm tra xem tài khoản người dùng có tồn tại không.
- + Nếu có, gửi một email chứa liên kết khôi phục mật khẩu.
- + Nếu không có, thông báo rằng tài khoản không tồn tại.
- + Nếu có lỗi xảy ra trong quá trình gửi email, hiển thị thông báo lỗi cho người dùng.
- + Sau khi hoàn tất, người dùng sẽ được chuyển hướng về trang đăng nhập.

3.3.4. Chức năng thay đổi mật khẩu



Hình 24: Sequence Diagram trang Change Password.

Thay đổi mật khẩu qua email:

```

@GetMapping("/reset-password")
public String resetPasswordPage(@RequestParam("email") String email, Model model) {
    model.addAttribute("email", email);
    return "reset_psw_temp.html";
}
  
```

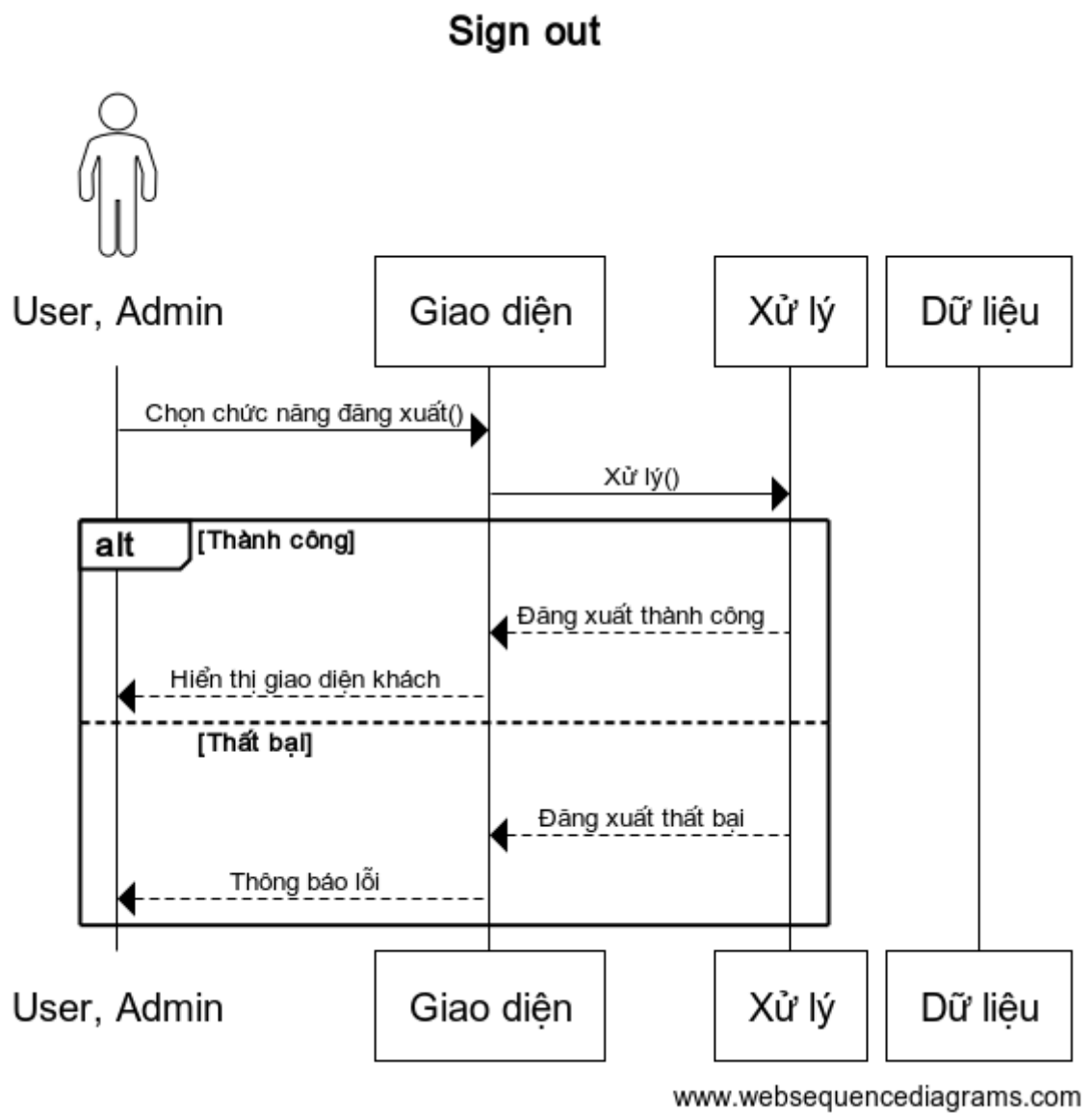
Hiện thị trang đặt lại mật khẩu cho người dùng sau khi họ đã nhận được liên kết đặt lại mật khẩu qua email.

```
@PostMapping("/update-password")
public String updatePassword(@RequestParam("email") String email,
    @RequestParam("newPassword") String newPassword,
    @RequestParam("confirmPassword") String confirmPassword, Model model) {
    if (!newPassword.equals(confirmPassword)) {
        model.addAttribute("error", "Passwords do not match");
        model.addAttribute("email", email);
        return "reset_psw_temp.html";
    }
    userService.UpdatePswbyEmail(email, confirmPassword);
    model.addAttribute("message", "Password updated successfully");
    model.addAttribute("loginForm", new Account());
    return "login_temp.html";
}
```

Xử lý yêu cầu thay đổi mật khẩu của người dùng. Các bước thực hiện gồm:

- + Kiểm tra xem mật khẩu mới và mật khẩu xác nhận có khớp không.
- + Nếu không khớp, hiển thị thông báo lỗi và quay lại trang thay đổi mật khẩu.
- + Nếu mật khẩu khớp, cập nhật mật khẩu mới trong cơ sở dữ liệu.
- + Sau khi cập nhật mật khẩu thành công, chuyển hướng người dùng đến trang đăng nhập để họ có thể đăng nhập lại với mật khẩu mới.

3.3.5. Chức năng đăng xuất

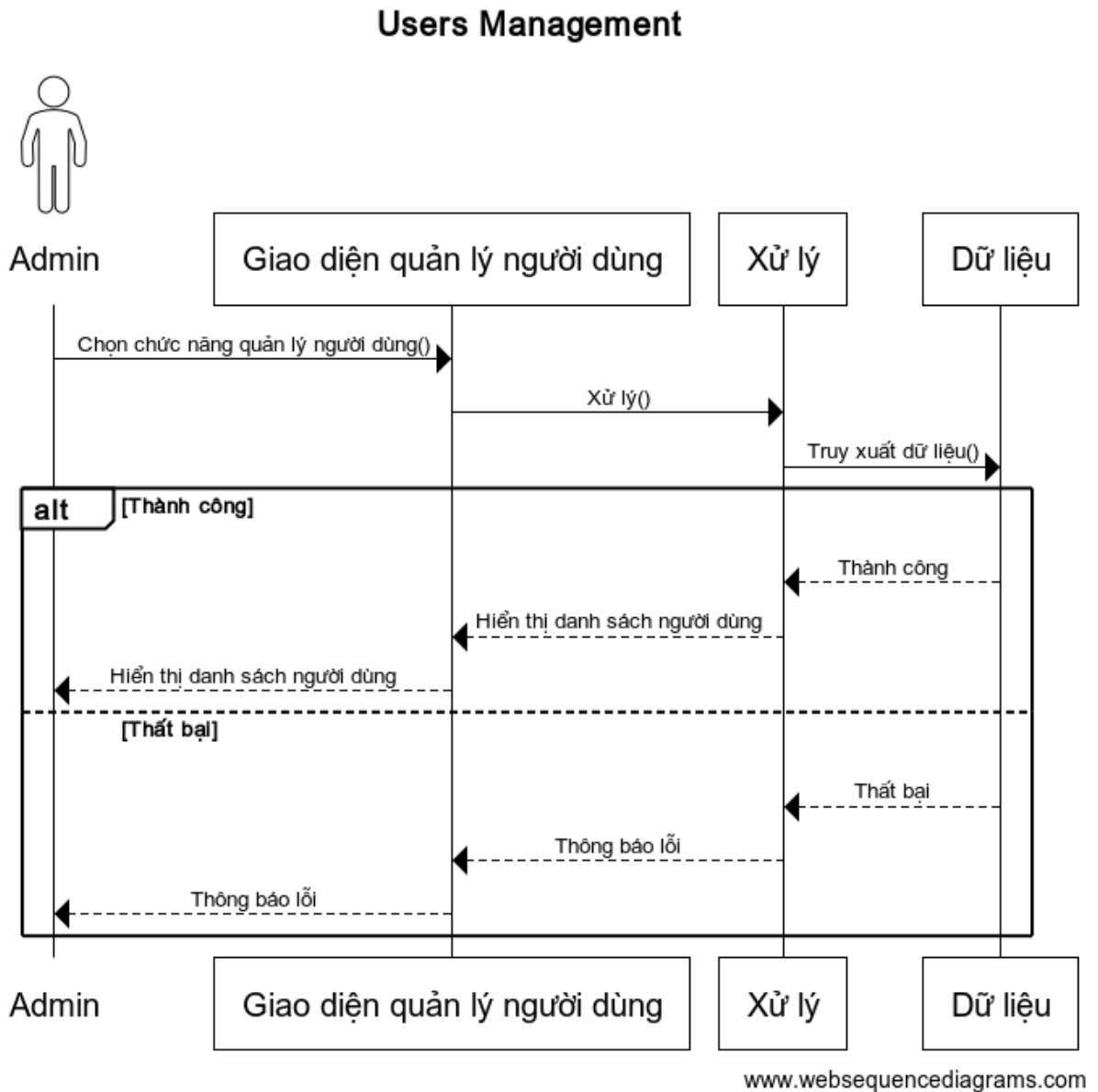


Hình 25: Sequence Diagram trang Sign Out.

3.4. Administrator

3.4.1. Account

a. *Hiển thị danh sách người dùng*



Hình 26: Sequence Diagram Quản lý danh sách người dùng.

Hiển thị danh sách người dùng:

```

@RequestMapping("")
public String all(Model model) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    if (authentication != null && authentication.isAuthenticated()) {
        String username = authentication.getName();
        Account user = userService.findbyUser(username);
    }
}
  
```

```

        if (user != null) {
            model.addAttribute("fullname", user.getFullName());
            model.addAttribute("user", user);
        } else {
            model.addAttribute("fullname", "User not found");
        }
    } else {
        model.addAttribute("fullname", "User not authenticated");
    }
    List<Account> list = accService.findAll();
    model.addAttribute("acc", list);
    return "account/list_acc.html";
}

```

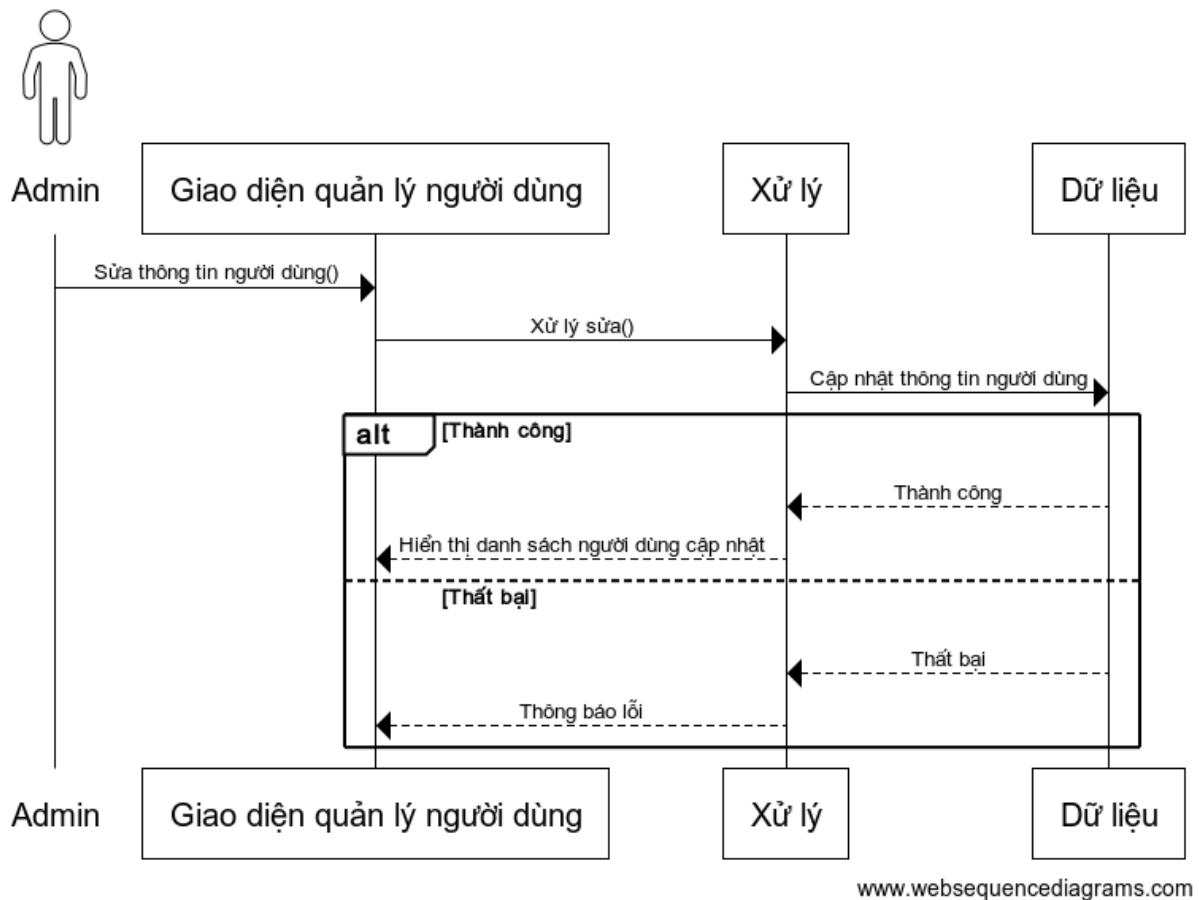
Các bước thực hiện:

- + Lấy đối tượng Authentication từ SecurityContextHolder, chứa thông tin xác thực của người dùng hiện tại (nếu có).
- + Nếu authentication không phải null và người dùng đã được xác thực (isAuthenticated() trả về true), tiến hành các bước tiếp theo.
- + Lấy tên người dùng từ authentication.getName() và tìm thông tin tài khoản người dùng đó trong cơ sở dữ liệu thông qua userService.findbyUser(username).
- + Nếu tìm thấy người dùng, thêm tên đầy đủ và thông tin tài khoản vào model để hiển thị trên giao diện. Nếu không tìm thấy, thêm thông báo "User not found" vào model.
- + Nếu người dùng chưa đăng nhập, thêm thông báo "User not authenticated" vào model.
- + Lấy tất cả các tài khoản người dùng từ cơ sở dữ liệu thông qua accService.findAll() và thêm danh sách này vào model.
- + Trả về trang HTML (account/list_acc.html) để hiển thị thông tin người dùng và danh sách tài khoản.

b. Chức năng thay đổi trạng thái người dùng người dùng cá nhân

– Diagrams

Edit User



Hình 27: Sequence Diagram trang Edit User.

Sửa người dùng:

```

@GetMapping("/edit/{id}")
public ModelAndView edit(ModelMap model, @PathVariable("id") Long userId) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String username = authentication.getName();
    Account user = userService.findbyUser(username);
    if (user != null && user instanceof Account) {
        model.addAttribute("fullname", user.getFullName());
        model.addAttribute("user", user);
    } else {
        model.addAttribute("fullname", "err");
    }
    if (user != null && user instanceof Account) {
        model.addAttribute("user", user);
    }
    Optional<Account> optAccount = accService.findById(userId);
    Account acc = new Account();
  
```

```

    if (optAccount.isPresent()) {
        Account entity = optAccount.get();
        BeanUtils.copyProperties(entity, acc);

        model.addAttribute("acc", acc);
        return new ModelAndView("account/edit_account_admin.html", model);
    }
    model.addAttribute("message", "Account is not existed");
    return new ModelAndView("redirect:/admin/account", model);
}

```

Các bước thực hiện:

- + Lấy thông tin xác thực của người dùng hiện tại (đang đăng nhập) từ SecurityContextHolder và tìm thông tin tài khoản người dùng đó trong cơ sở dữ liệu.
- + Nếu người dùng đã đăng nhập và tồn tại trong cơ sở dữ liệu, thông tin đầy đủ (tên người dùng và tài khoản) sẽ được thêm vào model để hiển thị trên giao diện. Nếu không tìm thấy, sẽ thêm thông báo lỗi vào model.
- + Dùng accService.findById(userId) để tìm tài khoản cần chỉnh sửa từ cơ sở dữ liệu bằng userId được truyền qua đường dẫn.
- + Nếu tài khoản được tìm thấy (optAccount.isPresent()), sao chép các thuộc tính của tài khoản đó vào đối tượng acc để có thể chỉnh sửa và hiển thị lên form. Đối tượng acc được thêm vào model.
- + Nếu tài khoản tồn tại, trả về trang edit_account_admin.html để hiển thị form chỉnh sửa tài khoản, kèm theo thông tin tài khoản trong model.
- + Nếu tài khoản không tồn tại, thêm thông báo lỗi vào model và chuyển hướng người dùng về trang danh sách tài khoản quản trị viên (/admin/account).

Lưu thông tin sau khi chỉnh sửa:

```

@PostMapping("/save")
public ModelAndView saveOrUpdate(@RequestParam Long id, @RequestParam
boolean status, ModelMap model) {
    Optional<Account> optAcc = accService.findById(id);
    Account acc = new Account();
    if (optAcc.isPresent()) {
        Account entity = optAcc.get();
        BeanUtils.copyProperties(entity, acc);

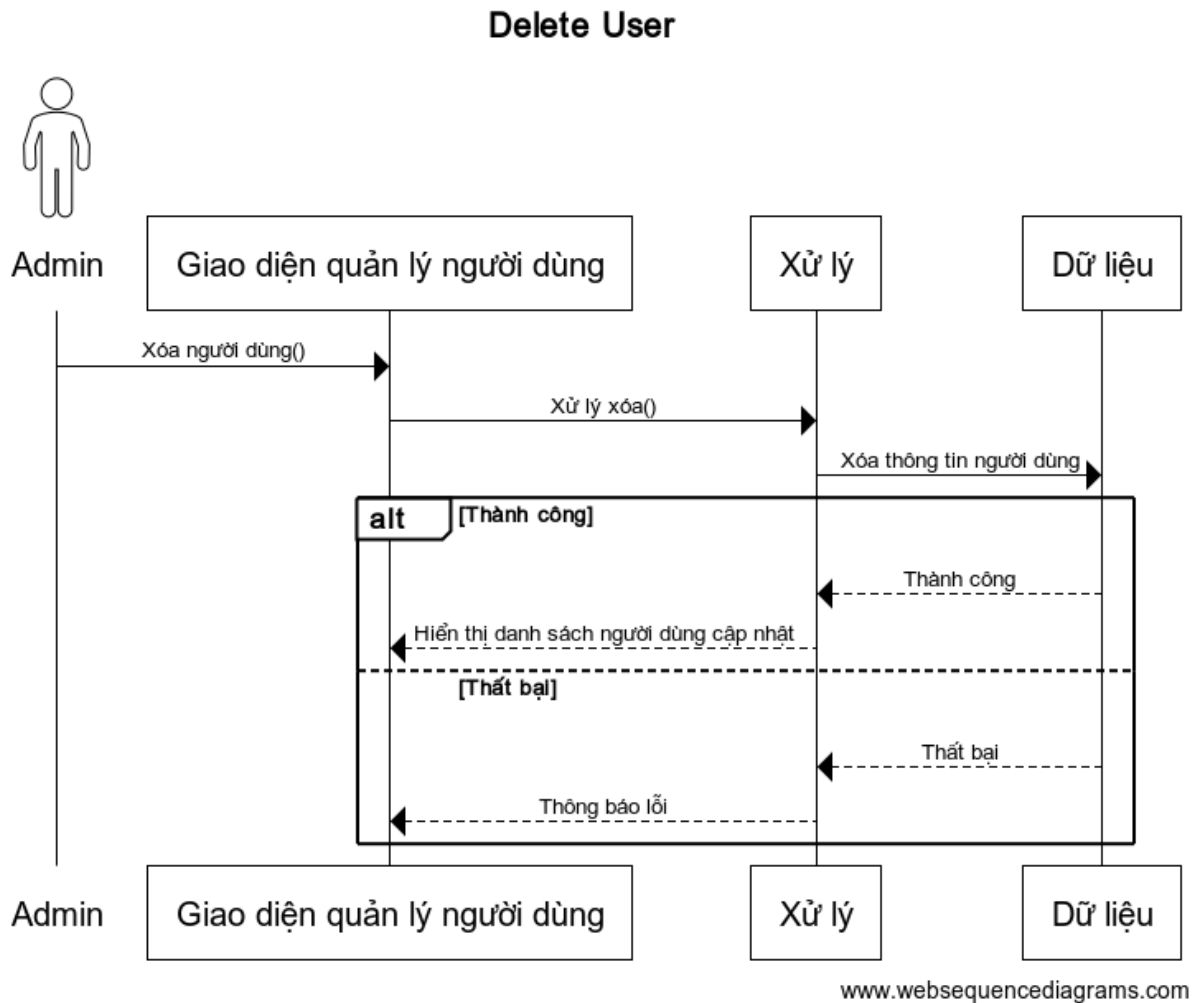
```

```
}  
acc.setStatus(status);  
accService.save(acc);  
return new ModelAndView("redirect:/admin/account", model);  
}
```

Các bước thực hiện:

- + Dùng `accService.findById(id)` để tìm tài khoản theo id được truyền từ yêu cầu. Kết quả trả về là một đối tượng `Optional<Account>`.
- + Tạo một đối tượng `Account` mới
- + Nếu tài khoản tồn tại (kiểm tra bằng `optAcc.isPresent()`), sao chép các thuộc tính của tài khoản hiện tại (`entity`) vào đối tượng `acc` mới. Điều này đảm bảo rằng tất cả các thông tin từ tài khoản gốc được chuyển sang đối tượng mới.
- + Sau khi sao chép các thuộc tính từ tài khoản gốc, cập nhật trạng thái của tài khoản (`status`) theo giá trị được truyền từ yêu cầu.
- + Sau khi cập nhật trạng thái, lưu đối tượng tài khoản vào cơ sở dữ liệu.
- + Sau khi lưu thành công, chuyển hướng người dùng về trang quản lý tài khoản (`/admin/account`).

c. Xóa người dùng



Hình 28: Sequence Diagram chức năng xóa người dùng.

Xóa người dùng:

```

@GetMapping("/delete/{id}")
public ModelAndView delete(ModelMap model, @PathVariable("id") Long Id) {
    Optional<Account> optAccount = accService.findById(Id);
    optAccount.get().getRoles().clear();
    accService.save(optAccount.get());
    accService.deleteById(Id);
    model.addAttribute("message", "Account is deleted");
    return new ModelAndView("redirect:/admin/account", model);
}
  
```

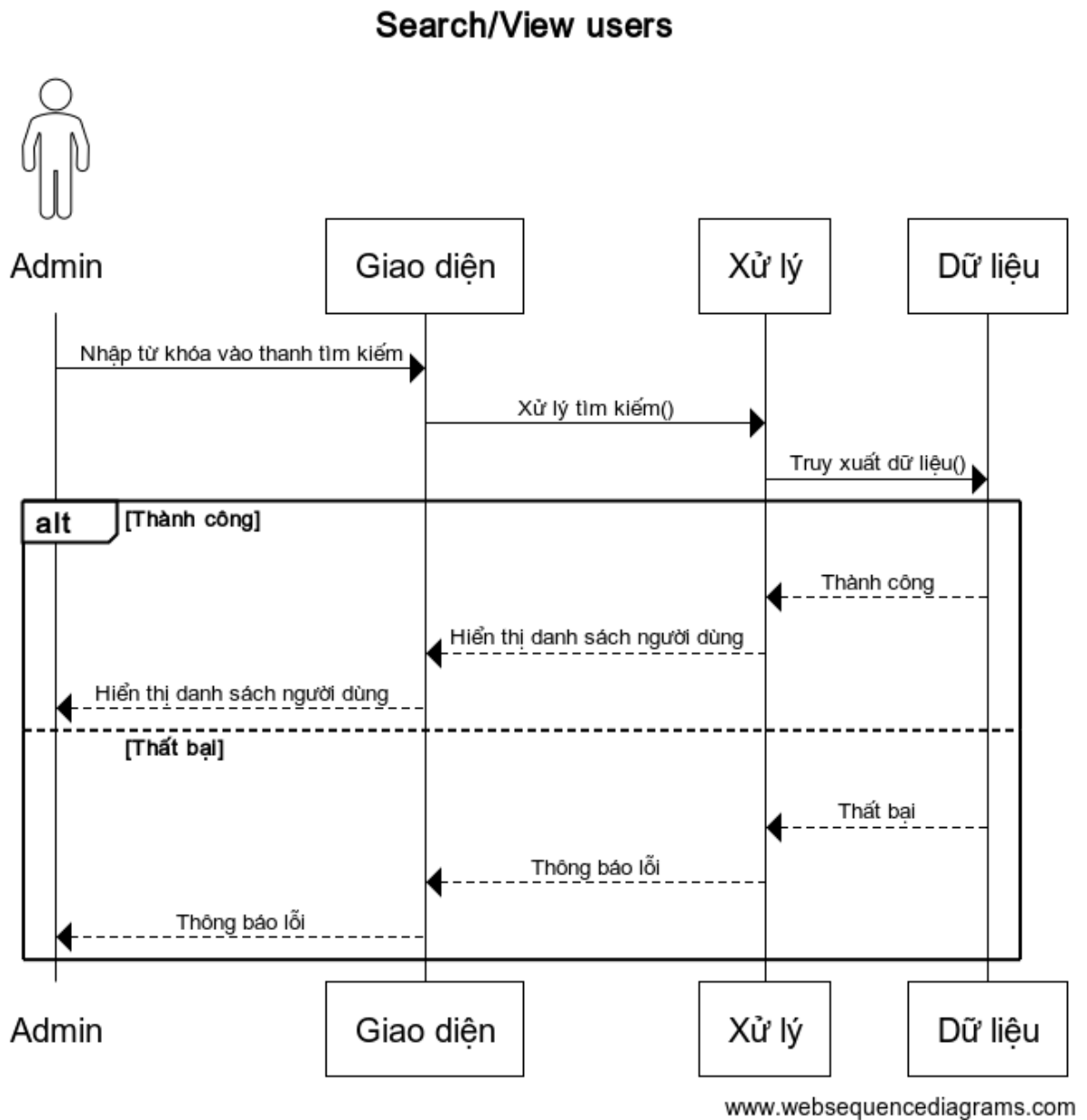
Các bước thực hiện:

- + Dùng `accService.findById(Id)` để tìm tài khoản cần xóa theo Id được truyền từ đường dẫn URL. Kết quả trả về là một đối tượng `Optional<Account>`.

- + Nếu tài khoản được tìm thấy, lấy các vai trò (roles) của tài khoản và xóa tất cả các vai trò (clear).
- + Lưu lại tài khoản sau khi đã xóa các vai trò của nó, đảm bảo rằng thông tin tài khoản được cập nhật.
- + Tiến hành xóa tài khoản khỏi cơ sở dữ liệu bằng cách sử dụng `accService.deleteById(Id)`.
- + Sau khi xóa tài khoản, thông báo "Account is deleted" được thêm vào model để thông báo cho người dùng. Sau đó, chuyển hướng đến trang danh sách tài khoản quản trị viên (`/admin/account`).

Hàm này thực hiện xóa một tài khoản người dùng bằng cách tìm tài khoản theo id, xóa các vai trò của tài khoản, cập nhật lại tài khoản và xóa tài khoản khỏi cơ sở dữ liệu. Sau khi xóa, người dùng sẽ được chuyển hướng đến trang quản lý tài khoản với thông báo thành công.

d. Tìm kiếm người dùng



Hình 29: Sequence Diagram trang Tìm kiếm người dùng.

Tìm kiếm người dùng:

```

@GetMapping("/find")
public String findUsername(Model model, @RequestParam String userfind) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String username = authentication.getName();
    Account user = userService.findbyUser(username);
    if (user != null) {
        model.addAttribute("user", user);
    }
}

```



```

Optional<Account> acc = accService.findByUsername(userfind);
if (acc.isPresent()) {
    model.addAttribute("acc", acc.get());
} else {
    model.addAttribute("acc", null);
}
return "account/list_acc";
}

```

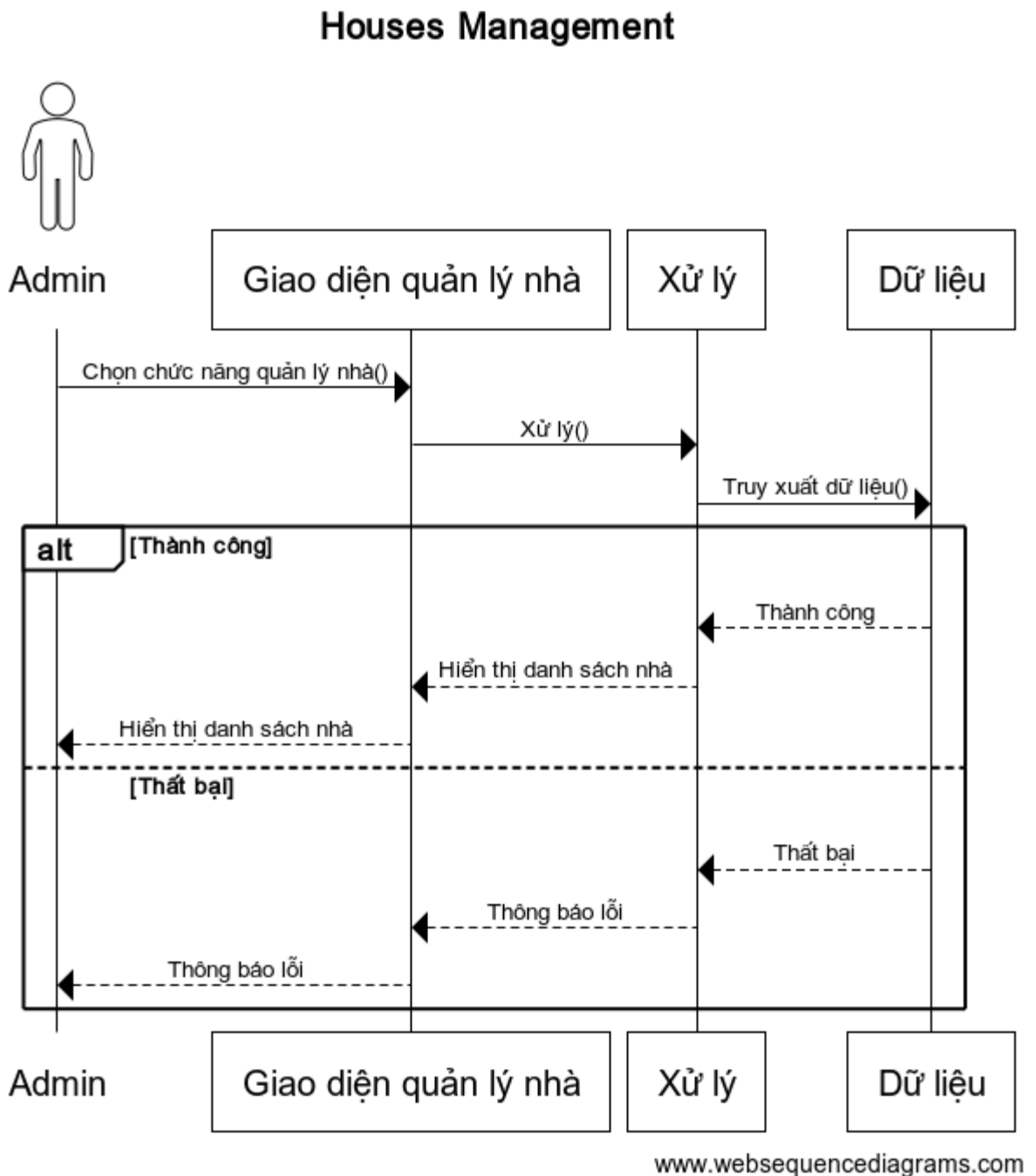
Các bước thực hiện:

- + Lấy thông tin người dùng hiện tại từ phiên làm việc thông qua sử dụng `SecurityContextHolder.getContext().getAuthentication()` để lấy thông tin xác thực (authentication) của người dùng hiện tại. Sau đó lấy tên người dùng thông qua phương thức `authentication.getName()`.
- + Tìm tài khoản của người dùng hiện tại thông qua `userService.findbyUser(username)`
- + Tìm kiếm tài khoản theo tên người dùng (userfind) thông qua `accService.findByUsername(userfind)`
- + Trả về trang hiển thị danh sách tài khoản “*account/list_acc*”

3.4.2. House Management

Mô tả: Chức năng này cho phép Admin thực hiện các thao tác quản lý thông tin ngôi nhà, bao gồm xem danh sách nhà, thêm mới, chỉnh sửa và xóa nhà

a. Hiện thị danh sách các ngôi nhà



Hình 30: Sequence Diagram trang Hiện thị danh sách các ngôi nhà.

```

@GetMapping("/list-house/{id}")
public String all(@PathVariable Long id, Model model) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String username=authentication.getName();
    Account user = userService.findbyUser(username);
}
  
```

```

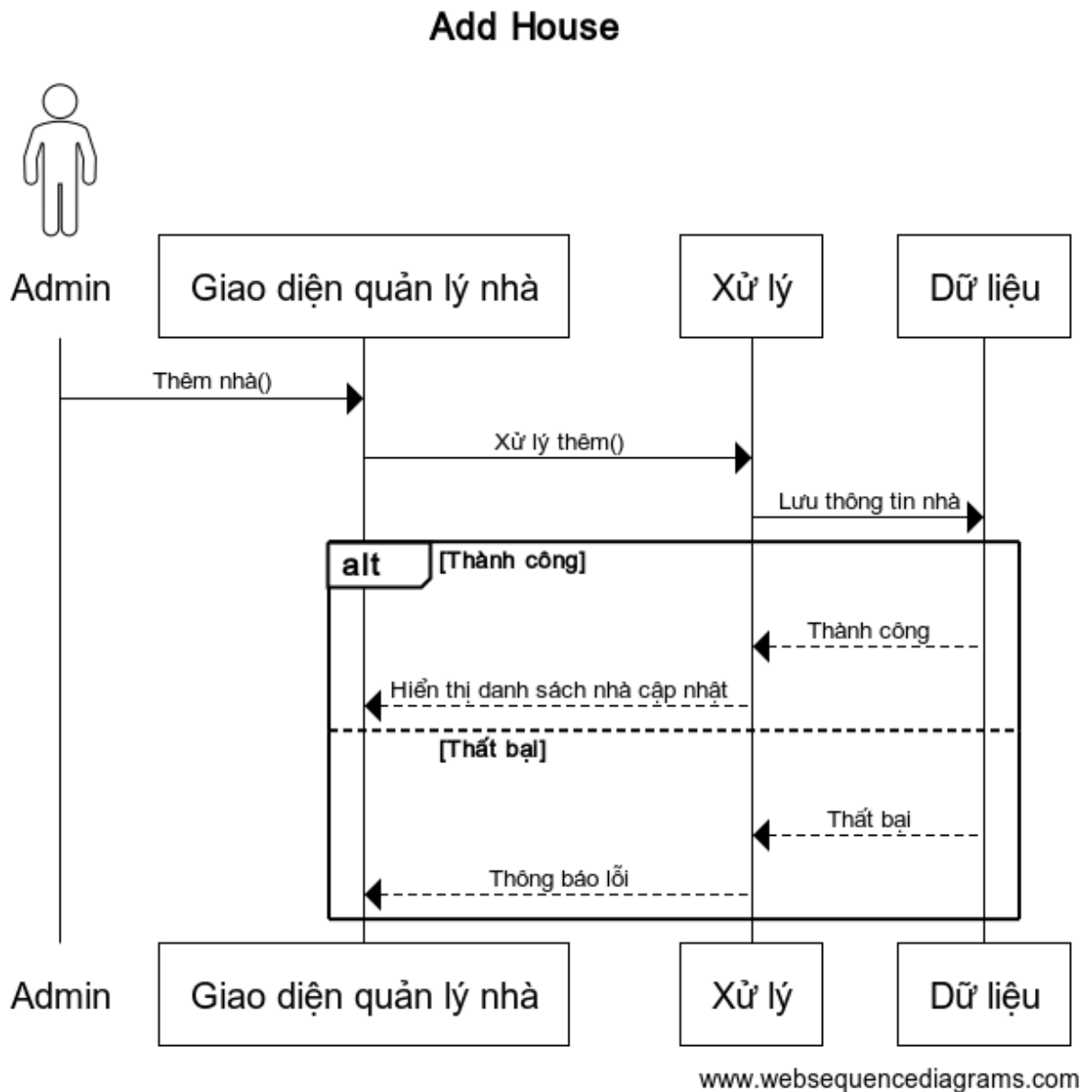
    if (user != null && user instanceof Account) {
        model.addAttribute("user", user);
    }
    Optional<Account> acc = accService.findById(id);
    if (acc.isPresent()) {
        List<Houses> houses = houseService.findByAccount(acc.get());
        model.addAttribute("houses", houses);
        model.addAttribute("id_acc", id);
    } else {
        model.addAttribute("message", "Account not found");
    }
    return "list-house/list_house_of_acc.html";
}

```

Các bước thực hiện:

- + Lấy thông tin người dùng hiện tại từ SecurityContextHolder để truy xuất tên người dùng đăng nhập.
- + Tìm tài khoản người dùng trong cơ sở dữ liệu thông qua tên người dùng (username).
- + Nếu tài khoản người dùng tồn tại, thêm thông tin người dùng vào model để hiển thị trong view.
- + Kiểm tra sự tồn tại của tài khoản với ID được cung cấp từ URL. Nếu tài khoản tồn tại, tiếp tục lấy danh sách các căn nhà (houses) của tài khoản đó thông qua phương thức findByAccount.
- + Thêm danh sách căn nhà và ID tài khoản vào model để hiển thị trong view.
- + Nếu tài khoản không tồn tại, thêm thông báo lỗi vào model (Account not found).
- + Trả về trang view list-house/list_house_of_acc.html để hiển thị danh sách các căn nhà của tài khoản.

b. Thêm nhà



Hình 31: Sequence Diagram trang Add House.

```

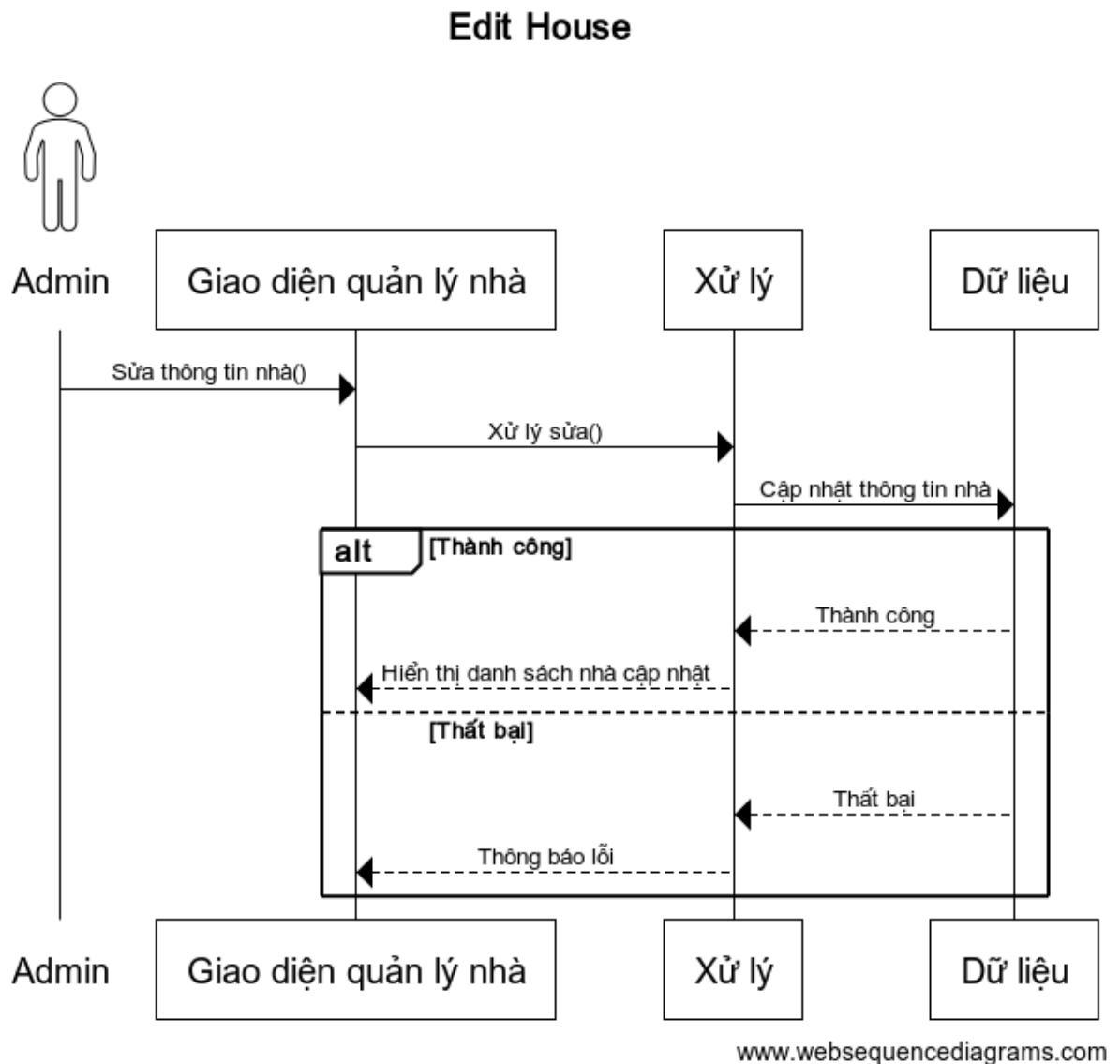
@GetMapping("/list-house/add/{id}")
public String add(@PathVariable Long id, Model model) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String username = authentication.getName();
    Account user = userService.findbyUser(username);
    if (user != null && user instanceof Account) {
        model.addAttribute("user", user);
    }
    Houses listHouse = new Houses();
}
  
```

```
model.addAttribute("id_acc", id);  
model.addAttribute("house",listHouse);  
listHouse.setIsEdit(false);  
return "list-house/add_edit_house.html";  
}
```

Các bước thực hiện:

- + Lấy thông tin người dùng hiện tại từ SecurityContextHolder và tìm tài khoản người dùng trong cơ sở dữ liệu.
- + Kiểm tra tài khoản người dùng: Nếu tài khoản người dùng tồn tại, thêm thông tin vào model để hiển thị.
- + Khởi tạo đối tượng Houses mới, thiết lập thuộc tính isEdit thành false (vì đây là thêm mới, không chỉnh sửa).
- + Thêm dữ liệu vào model: Thêm ID tài khoản người dùng và đối tượng Houses vào model để truyền tới view.
- + Trả về view: Chuyển hướng người dùng tới trang add_edit_house.html để nhập thông tin căn nhà mới.

c. Sửa thông tin của ngôi nhà



Hình 32: Sequence Diagram trang Edit House.

```

@GetMapping("/list-house/edit/{id}")
public ModelAndView edit (ModelMap model,@PathVariable("id") String houseId) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String username=authentication.getName();
    Account user = userService.findbyUser(username);
    if (user != null && user instanceof Account) {
        model.addAttribute("user", user);
    }
    Optional<Houses> optHouse =houseService.findById(houseId);
    Houses houseModel = new Houses();
  
```

```

    if (optHouse.isPresent()) {
        Houses entity = optHouse.get();
        BeanUtils.copyProperties(entity, houseModel);
        houseModel.setIsEdit(true);
        Long id = houseModel.getAcc().getId();
        model.addAttribute("house", houseModel);
        model.addAttribute("id_acc", id);
        return new ModelAndView("list-house/add_edit_house.html", model);
    }
    model.addAttribute("message", "House is not existed");
    return new ModelAndView("redirect:/admin/list-house", model);
}

```

Các bước thực hiện:

- + Lấy thông tin người dùng hiện tại từ SecurityContextHolder để truy xuất tên người dùng đăng nhập.
- + Tìm tài khoản người dùng trong cơ sở dữ liệu dựa trên tên người dùng (username) từ SecurityContext.
- + Nếu tài khoản người dùng tồn tại, thêm thông tin người dùng vào model để hiển thị trong view.
- + Tìm căn nhà với ID được cung cấp trong URL (houseId) từ dịch vụ houseService qua phương thức findById.
- + Nếu căn nhà tồn tại: Sao chép các thuộc tính của căn nhà tìm thấy vào đối tượng houseModel mới. Đánh dấu căn nhà là đang chỉnh sửa bằng cách thiết lập setIsEdit(true). Lấy ID của tài khoản (account) liên kết với căn nhà và thêm vào model. Trả về trang chỉnh sửa căn nhà (add_edit_house.html) với dữ liệu đã chuẩn bị trong model.
- + Nếu không tìm thấy căn nhà: Thêm thông báo lỗi vào model ("House is not existed"). Chuyển hướng về trang danh sách căn nhà /admin/list-house.

d. Lưu thông tin sau khi thêm hoặc sửa

```

@PostMapping("/list-house/save/{id_acc}")
public ModelAndView saveOrUpdate(@PathVariable Long id_acc, ModelMap model,
    @Valid @ModelAttribute("house") Houses houseModel, BindingResult
    result,
    @RequestParam("imageFile") MultipartFile imageFile) {
    if (result.hasErrors()) {

```

```

        return new ModelAndView("admin/list-house", model);
    }
    Optional<Account> acc = accService.findById(id_acc);
    if (!acc.isPresent()) {
        model.addAttribute("message", "Account not found");
        return new ModelAndView("/admin/list-house/{id_acc}", model);
    }
    Houses house = new Houses();
    String uploadPath = "D:\\upload";
    File uploadDir = new File(uploadPath);
    if (!uploadDir.exists()) {
        uploadDir.mkdir();
    }
    try {
        if (!imageFile.isEmpty()) {
            String originalFilename = imageFile.getOriginalFilename();
            int index = originalFilename.lastIndexOf(".");
            String ext = originalFilename.substring(index + 1);
            String fname = System.currentTimeMillis() + "." + ext;
            imageFile.transferTo(new File(uploadPath + "/" + fname));
            houseModel.setImage(fname);
        }
        BeanUtils.copyProperties(houseModel, house);
        house.setAcc(acc.get());
        houseService.save(house);
        model.addAttribute("message", "House saved successfully");
    } catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("message", "Error saving House");
    }
    String message = houseModel.getIsEdit() ? "House is EDIT" : "House is SAVE";
    model.addAttribute("message", message);
    return new ModelAndView("redirect:/admin/list-house/{id_acc}", model);
}

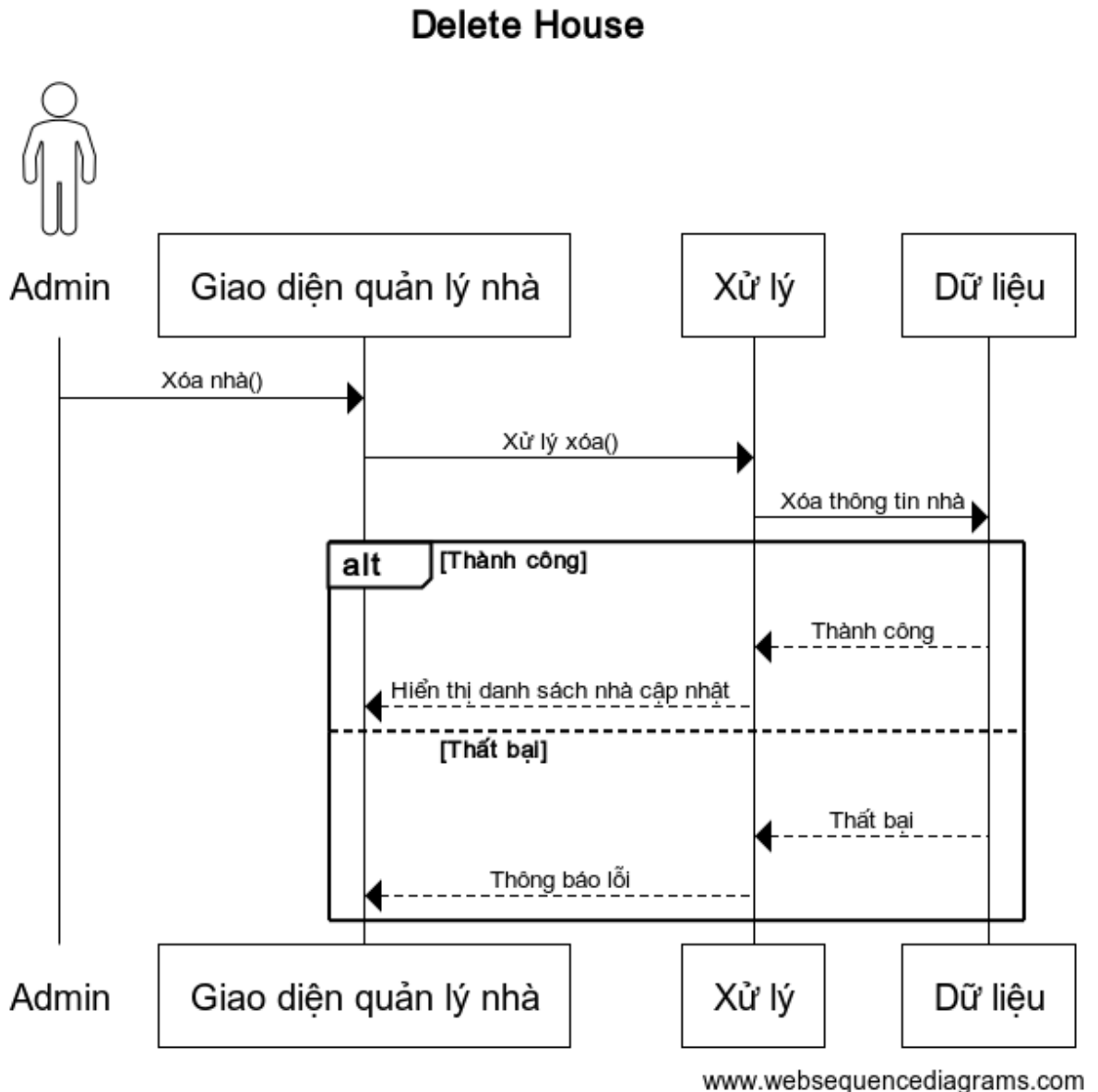
```

Các bước thực hiện:

- + Kiểm tra lỗi hợp lệ: Nếu có lỗi từ form, trả về trang danh sách căn nhà với các lỗi hiển thị.
- + Tìm tài khoản người dùng: Kiểm tra nếu tài khoản người dùng với id_acc tồn tại. Nếu không, hiển thị thông báo lỗi.
- + Chuẩn bị thư mục và đường dẫn ảnh: Kiểm tra và tạo thư mục lưu trữ ảnh nếu chưa tồn tại.
- + Xử lý ảnh tải lên: Nếu có ảnh, lưu ảnh vào thư mục với tên duy nhất.

- + Cập nhật thông tin căn nhà: Sao chép dữ liệu từ form vào đối tượng căn nhà, liên kết tài khoản người dùng với căn nhà, và lưu căn nhà vào cơ sở dữ liệu.
- + Thông báo và chuyển hướng: Thêm thông báo thành công và chuyển hướng người dùng trở lại danh sách căn nhà.

e. Xóa nhà



Hình 33: Sequence Diagram trang Delete House.

```

@GetMapping("/list-house/delete/{id}")
public ModelAndView delete(ModelMap model, @PathVariable("id") String houseId) {
    Optional<Houses> optHouse = houseService.findById(houseId);
  
```

```

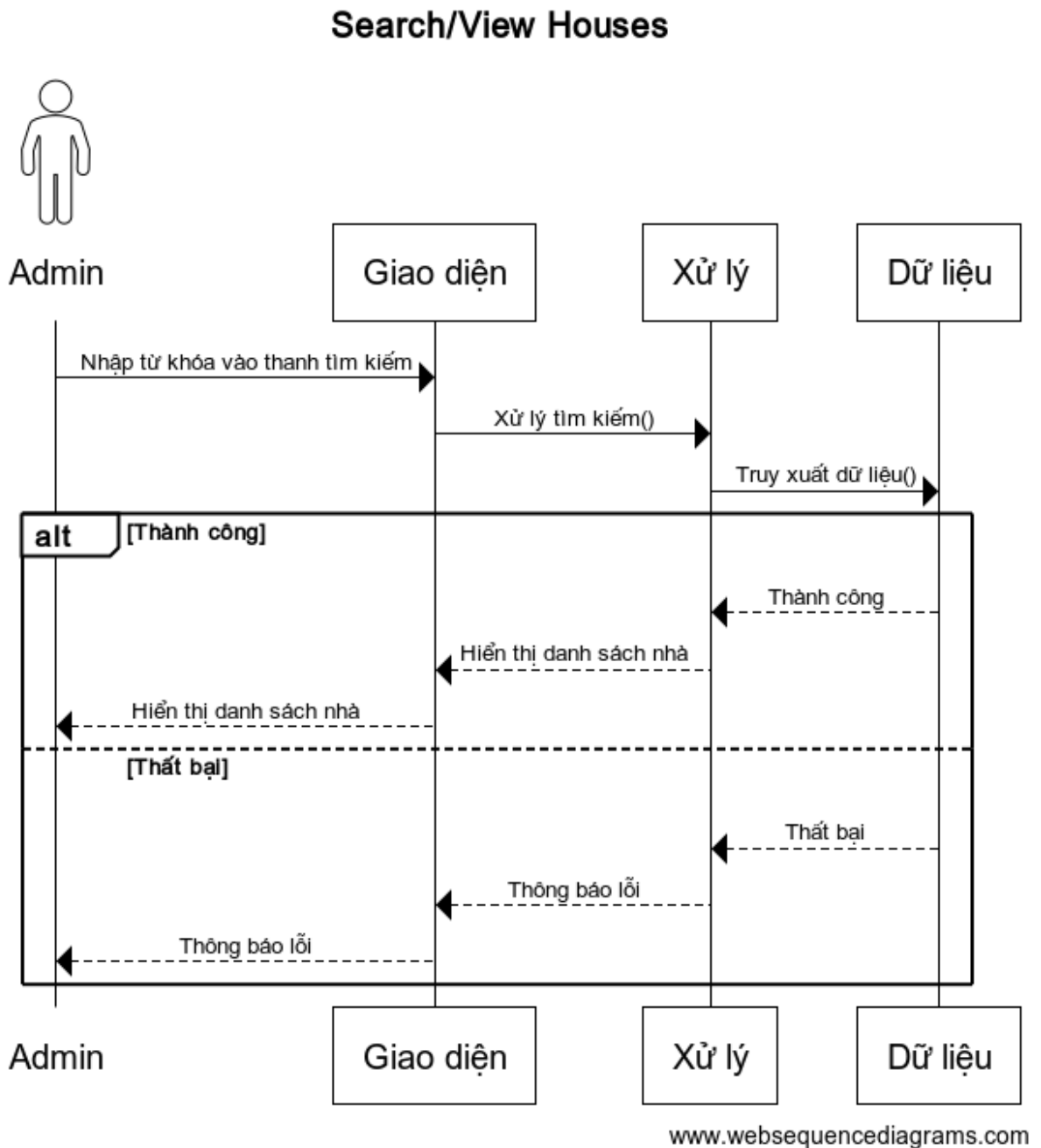
    if (optHouse.isPresent()) {
        houseService.deleteById(houseId);
        Houses entity = optHouse.get();
        Long id = entity.getAcc().getId();
        model.addAttribute("message", "House is deleted");
        model.addAttribute("id_acc", id);
    } else {
        model.addAttribute("message", "House not found");
    }
    return new ModelAndView("redirect:/admin/list-house/{id_acc}", model);
}

```

Các bước thực hiện:

- + Lấy thông tin căn nhà: Sử dụng `houseService.findById(houseId)` để tìm căn nhà với `houseId` được truyền vào.
- + Kiểm tra sự tồn tại của căn nhà: Nếu căn nhà tồn tại (`optHouse.isPresent()`), tiếp tục với các bước tiếp theo.
- + Xóa căn nhà: Gọi `houseService.deleteById(houseId)` để xóa căn nhà khỏi cơ sở dữ liệu. Lấy thông tin tài khoản người dùng: Lấy id của tài khoản người dùng liên kết với căn nhà đã xóa và thêm vào model.
- + Hiển thị thông báo: Thêm thông báo "House is deleted" vào model và chuyển hướng người dùng trở lại trang danh sách căn nhà của người dùng.
- + Xử lý trường hợp không tìm thấy căn nhà: Nếu không tìm thấy căn nhà, thêm thông báo lỗi "House not found" vào model.
- + Cuối cùng, phương thức trả về một đối tượng `ModelAndView`, chuyển hướng người dùng trở lại trang danh sách căn nhà của người dùng.

f. Tìm nhà



Hình 34: Sequence Diagram trang Search Houses.

```

@GetMapping("/house/find")
public String findidhouse(Model model, @RequestParam String idhousefind) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String username=authentication.getName();
    Account user = userService.findbyUser(username);
    if (user != null && user instanceof Account) {

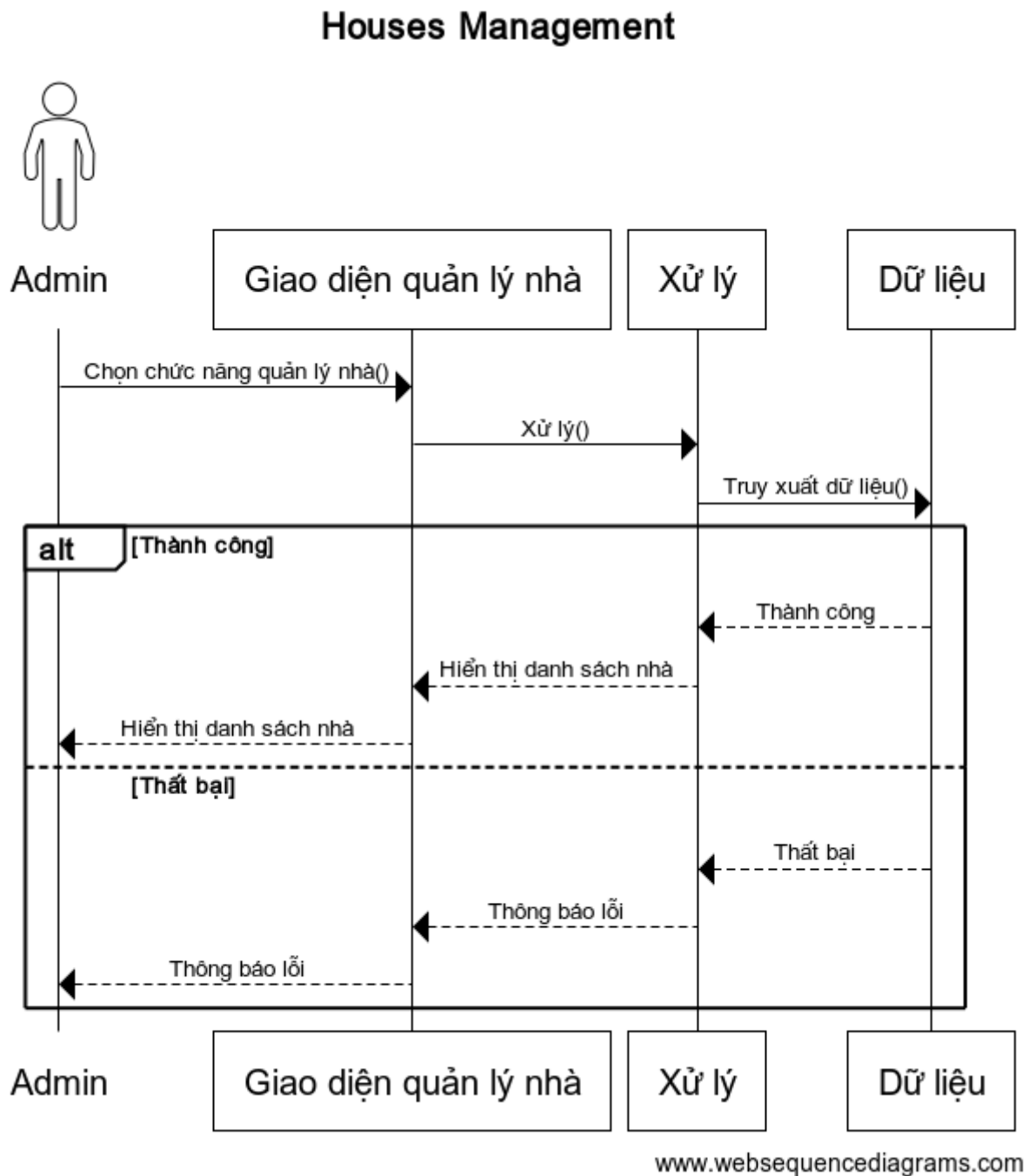
```

```
        model.addAttribute("user", user);
    }
    Optional<Houses> houses = houseService.findById(idhousefind);
    if (houses.isPresent()) {
        model.addAttribute("houses", houses.get());
    } else {
        model.addAttribute("houses", null);
    }
    return "list-house/home_admin_temp.html";
}
```

Các bước thực hiện:

- + Lấy thông tin người dùng: Lấy thông tin người dùng hiện tại từ SecurityContext.
- + Kiểm tra sự tồn tại của tài khoản: Nếu tài khoản người dùng tồn tại, thêm vào model.
- + Tìm căn nhà: Tìm căn nhà theo idhousefind từ service houseService.
- + Hiển thị kết quả: Trả về view list-house/home_admin_temp.html để hiển thị thông tin căn nhà tìm thấy.

g. Hiện thị danh sách tất cả ngôi nhà



Hình 35: Sequence Diagram trang Hiện thị danh sách các ngôi nhà.

```

@GetMapping("/house")
public String home(HttpSession session, Model model) {
    Authentication authentication =
    SecurityContextHolder.getContext().getAuthentication();
    String username=authentication.getName();
    Account user = userService.findbyUser(username);
}
  
```

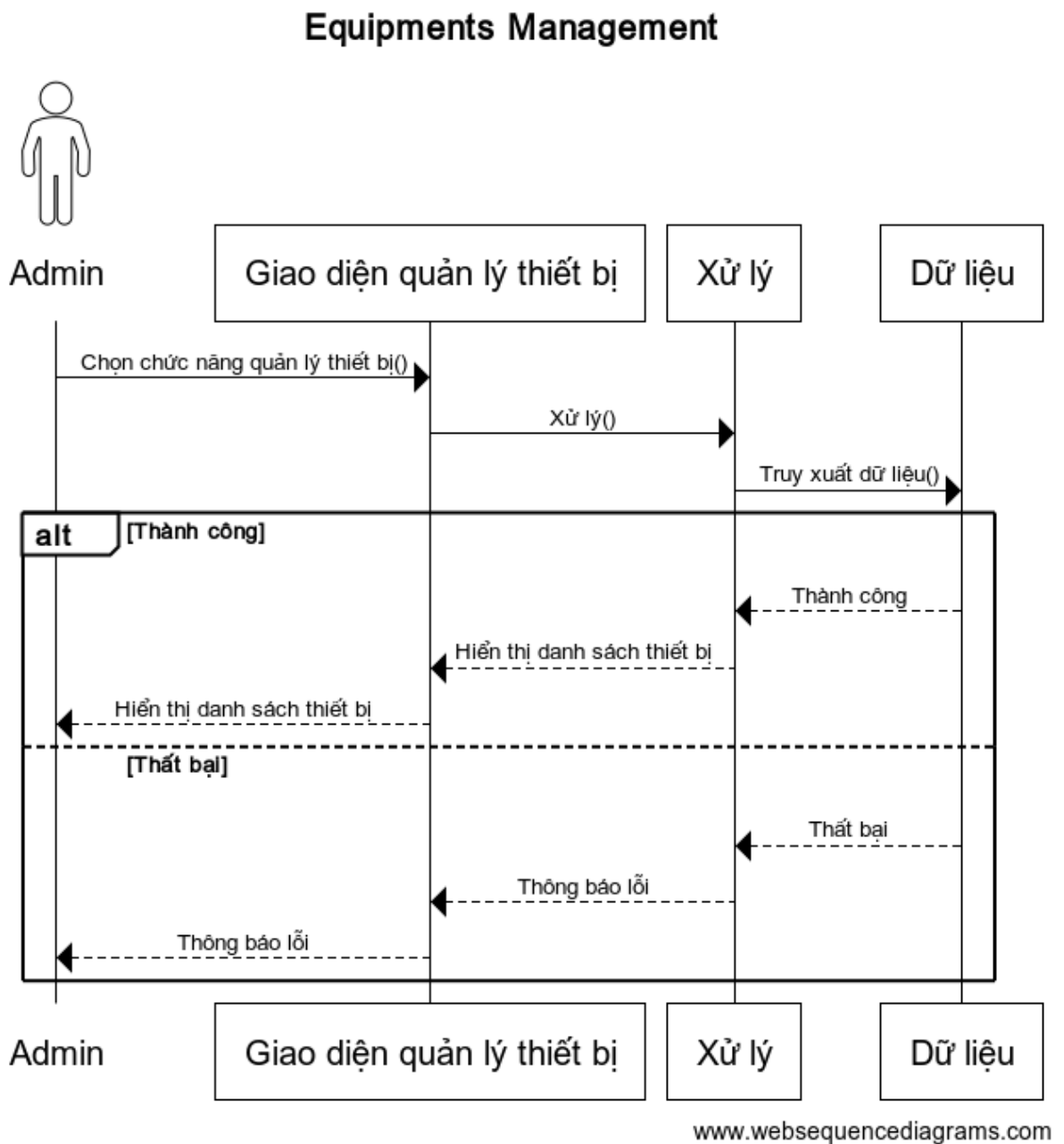
```
if (user != null && user instanceof Account) {  
    model.addAttribute("user", user);  
}  
List<Houses> houses = houseService.findAll();  
model.addAttribute("houses", houses);  
return "list-house/home_admin_temp.html";  
}
```

Các bước thực hiện:

- + Lấy thông tin người dùng hiện tại: Sử dụng SecurityContextHolder để lấy tên người dùng đã đăng nhập.
- + Kiểm tra sự tồn tại của tài khoản người dùng: Nếu tài khoản người dùng tồn tại, thêm thông tin vào model.
- + Lấy danh sách tất cả căn nhà: Gọi phương thức findAll() từ houseService để lấy danh sách căn nhà.
- + Trả về view: Trả về trang list-house/home_admin_temp.html và truyền danh sách căn nhà vào model để hiển thị.

3.4.3. Equipment

a. *Hiển thị danh sách thiết bị*



Hình 36: Sequence Diagram trang hiển thị danh sách các thiết bị.

```

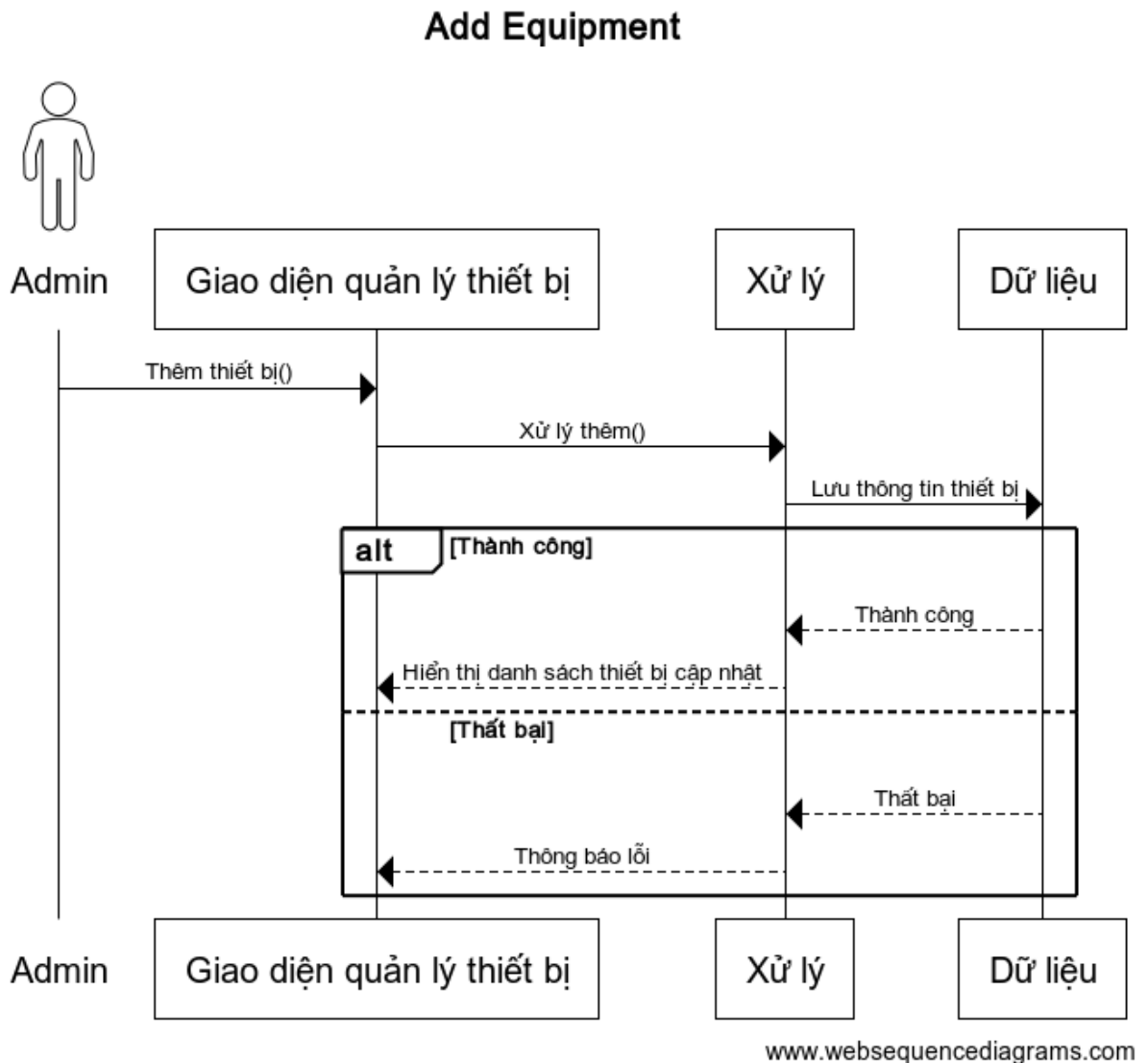
@GetMapping("/{id}")
public String find_id(@PathVariable String id, Model model) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String username=authentication.getName();
    Account user = userService.findbyUser(username);
}
    
```

```
    if (user != null && user instanceof Account) {  
        model.addAttribute("user", user);  
    }  
    List<Equipments> equip = equipService.findByHouseId(id);  
    model.addAttribute("equip", equip);  
    model.addAttribute("idHouse", id);  
    return "equip/list_equip_of_house.html";  
}
```

Các bước thực hiện:

- + Lấy thông tin người dùng đã đăng nhập từ SecurityContextHolder.
- + Lấy danh sách thiết bị (Equipments) theo ID nhà từ equipService.
- + Thêm thông tin người dùng và danh sách thiết bị vào model.
- + Trả về trang list_equip_of_house.html để hiển thị danh sách thiết bị của nhà.

b. Thêm thiết bị



Hình 37: Sequence Diagram chức năng thêm thiết bị.

```

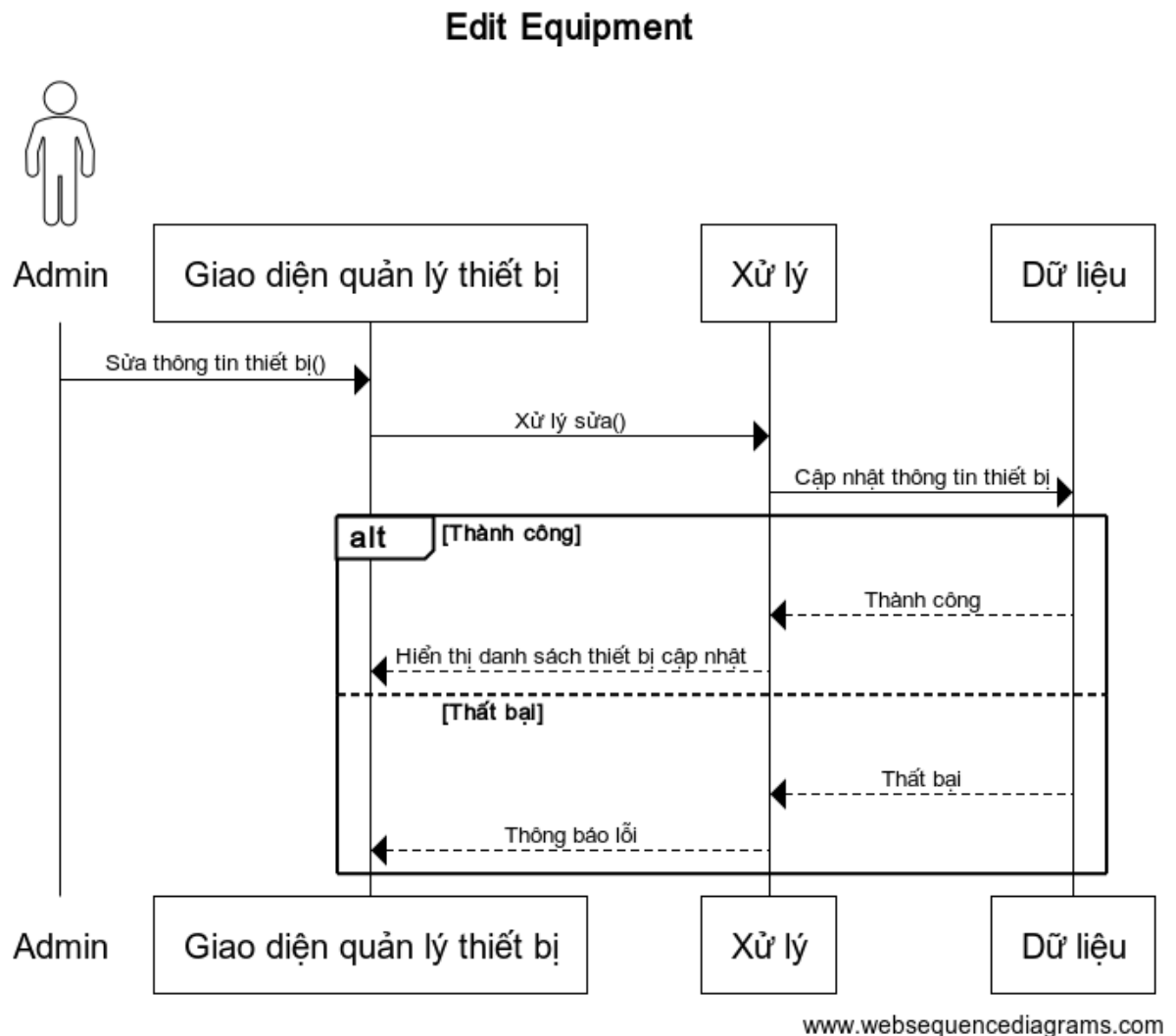
@GetMapping("/add/{id}")
public String add(@PathVariable String id, Model model) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String username = authentication.getName();
    Account user = userService.findbyUser(username);
    if (user != null && user instanceof Account) {
        model.addAttribute("user", user);
    }
    Equipments listequip = new Equipments();
    model.addAttribute("idHouse", id);
    model.addAttribute("house", listequip);
}
  
```

```
listequip.setIsEdit(false);  
return "equip/add_edit_equip.html";  
}
```

Các bước thực hiện:

- + Lấy thông tin người dùng đã đăng nhập: Từ SecurityContextHolder, lấy tên người dùng và tìm tài khoản của người dùng trong cơ sở dữ liệu thông qua userService.findbyUser(username).
- + Chuyển thông tin người dùng vào model: Nếu người dùng tồn tại, thêm thông tin tài khoản vào model.
- + Khởi tạo đối tượng Equipments mới: Tạo một đối tượng Equipments trống để bind dữ liệu khi thêm thiết bị mới.
- + Cung cấp ID nhà vào model: Thêm ID của nhà (idHouse) vào model để sử dụng khi thêm thiết bị.
- + Trả về trang "add_edit_equip.html": Chuyển hướng đến trang để người dùng có thể thêm thiết bị mới cho ngôi nhà.

c. Sửa thiết bị



Hình 38: Sequence Diagram trang Edit Equipment.

```

@GetMapping("/edit/{id}")
public ModelAndView edit (ModelMap model,@PathVariable("id") Long Id) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String username=authentication.getName();
    Account user = userService.findbyUser(username);
    if (user != null && user instanceof Account) {
        model.addAttribute("user", user);
    }
    Optional<Equipments> optEquip =equipService.findById(Id);
    Equipments equipModel = new Equipments();
    if (optEquip.isPresent()) {
        Equipments entity =optEquip.get();
    }
}
  
```

```

        BeanUtils.copyProperties(entity, equipModel);
        equipModel.setIsEdit(true);
        String id = equipModel.getHouse().getIdHouse();
        model.addAttribute("equip", equipModel);
        model.addAttribute("idHouse", id);
        return new ModelAndView("equip/add_edit_equip.html", model);
    }
    model.addAttribute("message", "Equipment is not existed");
    return new ModelAndView("redirect:/admin/equipment", model);
}

```

Các bước thực hiện:

- + Lấy thông tin người dùng: Dùng SecurityContextHolder để lấy tên người dùng và tìm tài khoản qua userService.findbyUser(username).
- + Thêm thông tin người dùng vào model: Nếu người dùng tồn tại, thêm thông tin vào model.
- + Lấy thiết bị theo ID: Dùng equipService.findById(Id) để lấy thiết bị cần chỉnh sửa.
- + Sao chép và chỉnh sửa thông tin thiết bị: Nếu thiết bị tồn tại, sao chép thông tin vào đối tượng equipModel và đánh dấu thiết bị đang chỉnh sửa.
- + Thêm thông tin thiết bị vào model: Thêm đối tượng thiết bị và ID nhà vào model.
- + Chuyển hướng tới trang chỉnh sửa thiết bị: Trả về trang add_edit_equip.html để người dùng chỉnh sửa thiết bị.
- + Xử lý lỗi: Nếu thiết bị không tồn tại, thông báo lỗi và chuyển hướng về danh sách thiết bị.

d. Lưu thông tin thiết bị khi thêm hoặc sửa

```

@PostMapping("/save/{idHouse}")
public ModelAndView saveOrUpdate(@PathVariable String idHouse, ModelMap model,
    @Valid @ModelAttribute("house") Equipments equipModel, BindingResult
    result,
    @RequestParam("imageFile") MultipartFile imageFile) {
    if (result.hasErrors()) {
        return new ModelAndView("equip/add", model);
    }
    Optional<Houses> house = houseService.findById(idHouse);
    if (!house.isPresent()) {

```

```

        model.addAttribute("message", "Account not found");
        return new ModelAndView("/admin/equipments/{idHouse}", model);
    }
    Equipments equip = new Equipments();
    String uploadPath = "D:\\upload";
    File uploadDir = new File(uploadPath);
    if (!uploadDir.exists()) {
        uploadDir.mkdir();
    }
    try {
        if (!imageFile.isEmpty()) {
            String originalFilename = imageFile.getOriginalFilename();
            int index = originalFilename.lastIndexOf(".");
            String ext = originalFilename.substring(index + 1);
            String fname = System.currentTimeMillis() + "." + ext;
            imageFile.transferTo(new File(uploadPath + "/" + fname));
            equipModel.setImage(fname);
        }
        BeanUtils.copyProperties(equipModel, equip);
        equip.setHouse(house.get());
        equipService.save(equip);
        model.addAttribute("message", "Equipment saved successfully")
    } catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("message", "Error saving Equipment");
    }
    String message = equipModel.getIsEdit() ? "Equipment is EDIT" : "Equipment is
SAVE";
    model.addAttribute("message", message);
    return new ModelAndView("redirect:/admin/equipments/{idHouse}", model);
}

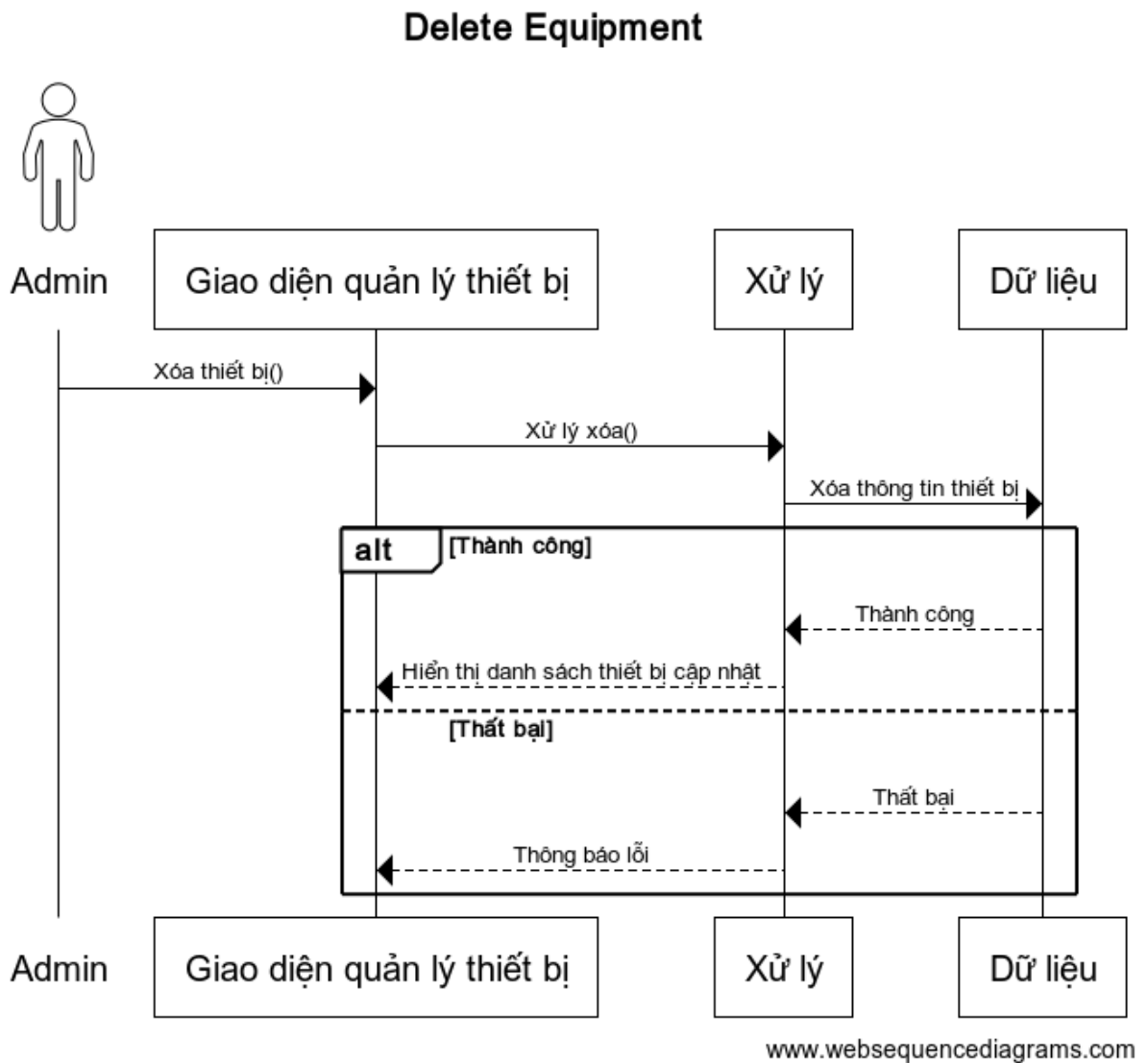
```

Các bước thực hiện:

- + Kiểm tra lỗi form: Nếu có lỗi, quay lại trang thêm thiết bị (equip/add).
- + Kiểm tra sự tồn tại của nhà: Nếu nhà không tồn tại, thông báo lỗi và quay lại trang quản lý thiết bị.
- + Tạo đối tượng thiết bị mới: Tạo một đối tượng Equipments mới.
- + Tải ảnh lên: Lưu ảnh vào thư mục và gán tên ảnh vào đối tượng thiết bị.
- + Sao chép thuộc tính: Sao chép dữ liệu từ form vào đối tượng Equipments.
- + Liên kết thiết bị với nhà: Gán nhà cho thiết bị.
- + Lưu thiết bị: Lưu đối tượng thiết bị vào cơ sở dữ liệu.
- + Thông báo kết quả: Hiện thị thông báo thành công hoặc thất bại.

- + Hiển thị thông báo thao tác: Hiển thị thông báo "Equipment is EDIT" hoặc "Equipment is SAVE".
- + Chuyển hướng: Sau khi lưu, chuyển hướng về trang danh sách thiết bị của nhà.

e. Xóa thiết bị



Hình 39: Sequence Diagram chức năng Delete Equipment.

```

@GetMapping("/delete/{id}")
public ModelAndView delete(ModelMap model, @PathVariable("id") Long Id) {
    Optional<Equipments> optequip= equipService.findById(Id);
    if (optequip.isPresent()) {
        equipService.deleteById(Id);
    }
}
  
```

```

        Equipments entity = optequip.get();
        String id = entity.getHouse().getIdHouse();
        model.addAttribute("message", "Equipment is deleted");
        model.addAttribute("idHouse", id);
    } else {
        model.addAttribute("message", "House not found");
    }
    return new ModelAndView("redirect:/admin/equipments/{idHouse}", model);
}

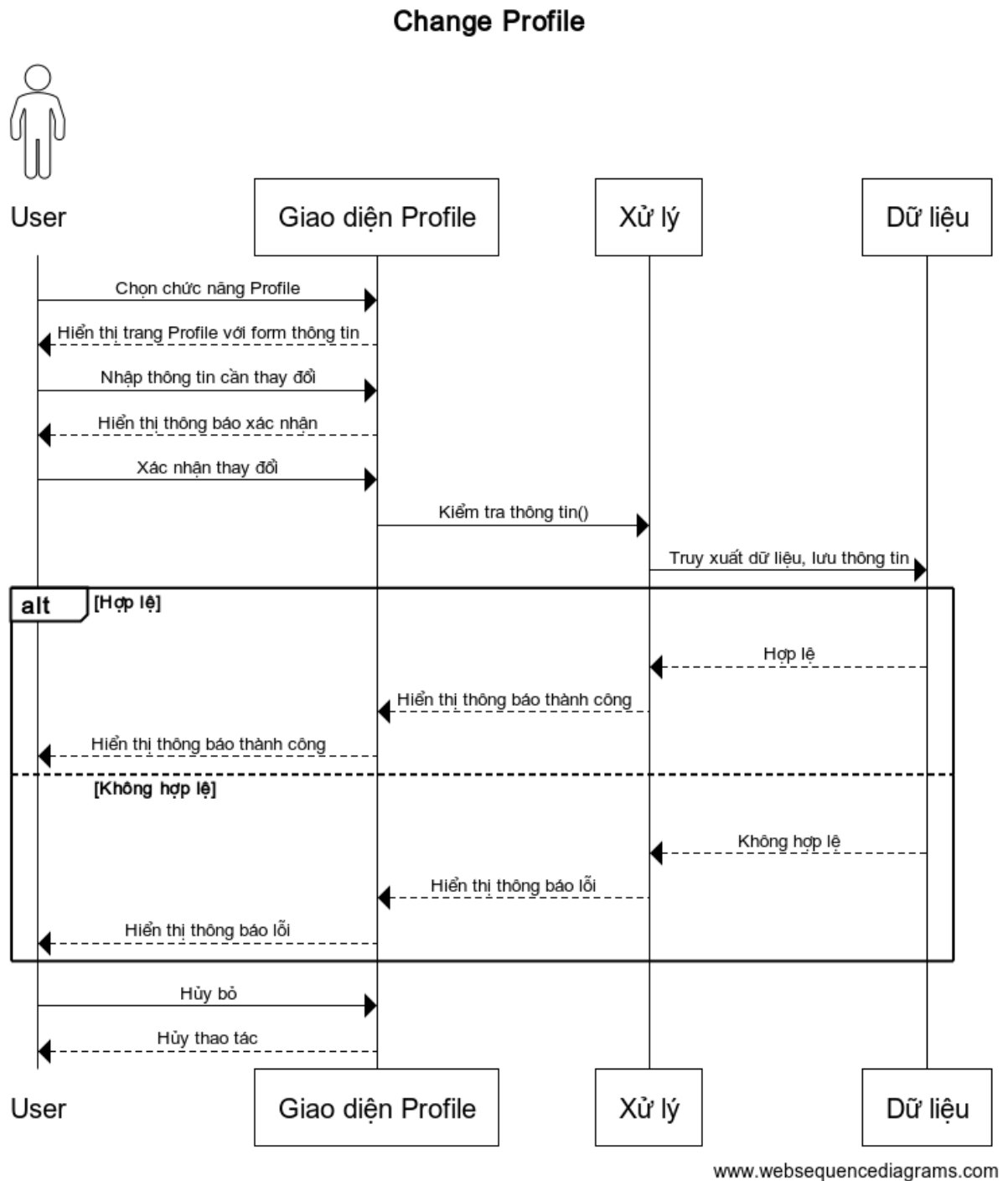
```

Các bước thực hiện:

- + Tìm thiết bị theo ID: Sử dụng `equipService.findById(Id)` để tìm thiết bị dựa trên ID được truyền vào.
- + Kiểm tra sự tồn tại của thiết bị: Nếu thiết bị tồn tại (`optequip.isPresent()`), tiếp tục xóa thiết bị. Nếu không, thông báo lỗi "House not found".
- + Xóa thiết bị: Nếu thiết bị tồn tại, thực hiện xóa bằng cách sử dụng `equipService.deleteById(Id)`.
- + Lấy ID của nhà liên kết với thiết bị: Sau khi xóa thiết bị, lấy `idHouse` từ nhà liên kết với thiết bị để chuyển hướng.
- + Thông báo kết quả: Thêm thông báo "Equipment is deleted" vào mô hình để hiển thị cho người dùng.
- + Chuyển hướng về danh sách thiết bị của nhà: Sau khi xóa thành công, chuyển hướng về trang danh sách thiết bị của nhà với ID nhà (`idHouse`).

3.5. User

3.5.1. Edit Profile



Hình 40: Sequence Diagram chức năng Edit Profile.

@GetMapping("/edit/{id}")


```

public ModelAndView edit(ModelMap model, @PathVariable("id") Long
userId) {
    Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
    String username = authentication.getName();
    Account user = userService.findbyUser(username);
    if (user != null && user instanceof Account) {
        model.addAttribute("fullname", user.getFullName());
        model.addAttribute("user", user);
    }
    Account acc = new Account();
    BeanUtils.copyProperties(user, acc);
    model.addAttribute("acc", acc);
    model.addAttribute("id_acc", acc.getId());
    return new ModelAndView("account/edit_profile.html", model);
}

```

Các bước thực hiện:

- + Nhận yêu cầu GET từ đường dẫn /edit/{id}, trong đó {id} là ID của người dùng.
- + Lấy thông tin người dùng từ SecurityContextHolder, sử dụng thông tin xác thực hiện tại (username).
- + Tìm người dùng trong cơ sở dữ liệu thông qua dịch vụ userService.findbyUser(username).
- + Kiểm tra người dùng hợp lệ (không null và là đối tượng Account).
- + Chuyển thông tin người dùng vào đối tượng Account mới (acc).
- + Cập nhật thông tin người dùng (ví dụ: tên đầy đủ và đối tượng người dùng) vào mô hình (model).
- + Trả về trang view account/edit_profile.html, kèm theo các thông tin người dùng đã được thêm vào mô hình.

Lưu thông tin sau khi sửa:

```

@PostMapping("/save")
public ModelAndView saveOrUpdate(ModelMap model, @Valid
    @ModelAttribute("acc") Account accModel, BindingResult result,
    @RequestParam("imageFile") MultipartFile imageFile) {

```

```

        if (result.hasErrors()) {
            return new ModelAndView("account/edit_profile", model);
        }
        Account acc = new Account();
        String uploadPath = "D:\\upload";
        File uploadDir = new File(uploadPath);
        if (!uploadDir.exists()) {
            uploadDir.mkdir();
        }
        try {
            if (!imageFile.isEmpty()) {
                String originalFilename = imageFile.getOriginalFilename();
                int index = originalFilename.lastIndexOf(".");
                String ext = originalFilename.substring(index + 1);
                String fname = System.currentTimeMillis() + "." + ext;
                imageFile.transferTo(new File(uploadPath + "/" + fname));
                accModel.setImage(fname);
            }
            if
(!accModel.getPassword().equals(userService.findbyUser(accModel.getUsername()).getPassword())) {
                String hashedPassword =
passwordEncoder.encode(accModel.getPassword());
                accModel.setPassword(hashedPassword);
            }
            BeanUtils.copyProperties(accModel, acc);
            acc.setStatus(true);
            Roles role = roleRepository.findByName("USER").get();
            acc.setRoles(Collections.singleton(role));
            accService.save(acc);
            Authentication authentication =
new UsernamePasswordAuthenticationToken(
accModel.getUsername(), accModel.getPassword(), accModel.getAuthorities());
SecurityContextHolder.getContext().setAuthentication(authentication);
        } catch (Exception e) {
            e.printStackTrace();
            model.addAttribute("message", "Error saving profile");
        }
        return new ModelAndView("redirect:/user/home", model);
    }
}

```

Các bước thực hiện:

- + Nhận yêu cầu POST từ đường dẫn /save khi người dùng gửi thông tin chỉnh sửa tài khoản.

- + Kiểm tra lỗi xác thực: Nếu có lỗi (dựa trên BindingResult), trả về trang chỉnh sửa hồ sơ (account/edit_profile).
- + Tạo đối tượng Account mới để lưu thông tin từ mẫu (accModel).
- + Tạo thư mục lưu ảnh nếu chưa có thư mục D:\\upload.
- + Xử lý tệp hình ảnh: Kiểm tra xem tệp hình ảnh có tồn tại không. Nếu có, lưu tệp hình ảnh vào thư mục và đặt tên mới cho nó.
- + Mã hóa mật khẩu: Nếu mật khẩu đã thay đổi, mã hóa mật khẩu và gán lại cho accModel.
- + Sao chép thông tin từ accModel sang acc (đối tượng Account mới).
- + Cập nhật trạng thái tài khoản (status = true) và thiết lập vai trò người dùng là "USER".
- + Lưu đối tượng Account vào cơ sở dữ liệu thông qua accService.
- + Cập nhật thông tin xác thực trong SecurityContextHolder để phản ánh trạng thái người dùng mới.
- + Xử lý lỗi: Nếu có lỗi xảy ra trong quá trình lưu, thông báo lỗi sẽ được hiển thị.
- + Chuyển hướng người dùng tới trang chính (/user/home) sau khi lưu thành công.

3.5.2. Xem danh sách các ngôi nhà của user

Xem danh sách nhà của user.

```
@GetMapping("/home")
public String home(Model model) {
    Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
    if (authentication != null && authentication.isAuthenticated()) {
        String username = authentication.getName();
        Account user=userService.findbyUser(username);
        model.addAttribute("user", user);
        long id = user.getId();
        Optional<Account> acc = accService.findById(id);
        if (acc.isPresent()) {
            List<Houses> houses = houseService.findByAccount(acc.get());
            model.addAttribute("houses", houses);
        }
    }
}
```

```

        model.addAttribute("id_acc", id);
    } else {
        model.addAttribute("message", "Account not found");
    }
} else {
    model.addAttribute("message", "User is not authenticated");
}
return "list-house/list_house_of_user.html";
}

```

Các bước thực hiện:

- + Kiểm tra xác thực người dùng: Dùng SecurityContextHolder để xác định người dùng đã đăng nhập chưa.
- + Lấy thông tin người dùng: Truy vấn thông tin người dùng từ cơ sở dữ liệu bằng tên người dùng.
- + Thêm thông tin vào model: Thêm đối tượng người dùng vào model.
- + Lấy danh sách nhà của người dùng: Tìm tài khoản và danh sách nhà của người dùng từ cơ sở dữ liệu.
- + Thông báo nếu không tìm thấy tài khoản: Nếu không tìm thấy tài khoản hoặc nhà, hiển thị thông báo lỗi.
- + Trả về view: Hiển thị danh sách nhà của người dùng trên trang list-house/list_house_of_user.html.

3.5.3. Xem danh sách thiết bị của nhà:

```

@GetMapping("/equipments/{id}")
public String find_id(@PathVariable String id, Model model) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String username=authentication.getName();
    Account user = userService.findbyUser(username);
    if (user != null && user instanceof Account) {
        model.addAttribute("user", user);
    }
    List<Equipments> equip = equipService.findByHouseId(id);
    model.addAttribute("equip", equip);
    model.addAttribute("idHouse", id);
    return "equip/list_equip_of_house_user.html";
}

```

}

Các bước thực hiện:

- + Lấy thông tin người dùng đã đăng nhập: Sử dụng SecurityContextHolder để lấy tên người dùng đã đăng nhập và tìm kiếm thông tin người dùng từ dịch vụ userService.
- + Thêm thông tin người dùng vào model: Nếu người dùng tồn tại, thêm đối tượng người dùng vào model để có thể hiển thị trên giao diện.
- + Lấy danh sách thiết bị của ngôi nhà: Truy vấn danh sách thiết bị (Equipments) liên kết với id của ngôi nhà từ dịch vụ equipService.
- + Thêm thông tin vào model: Thêm danh sách thiết bị và idHouse vào model để hiển thị trên giao diện.
- + Trả về trang hiển thị danh sách thiết bị: Chuyển hướng đến trang equip/list_equip_of_house_user.html để hiển thị danh sách thiết bị của ngôi nhà.

3.6. Internet of Things (IoT)

Đầu tiên vào trang điều khiển.

```
@GetMapping("/monitor/{idHouse}")
public String led(Model model, @PathVariable String idHouse) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String username = authentication.getName();
    Account user = userService.findbyUser(username);
    if (user != null && user instanceof Account) {
        model.addAttribute("user", user);
    }
    List<Equipments> equip = equipService.findbyHouseId(idHouse);
    List<Equipments> button = equipService.findbySensor(equip, "Switch");
    List<Equipments> dht11 = equipService.findbySensor(equip, "DHT11");

    Set<Roles> roles = user.getRoles();
```

```

        model.addAttribute("roles", roles);
        model.addAttribute("button", button);
        model.addAttribute("dht11", dht11);
        model.addAttribute("idHouse", idHouse);
        model.addAttribute("ledStatus", ledStatus ? 1 : 0);
        return "/monitor";
    }

```

- + Lấy thông tin người dùng đã đăng nhập: Dùng SecurityContextHolder để lấy tên người dùng và tìm thông tin người dùng.
- + Thêm người dùng vào model: Nếu người dùng tồn tại, thêm vào model.
- + Lấy danh sách thiết bị theo nhà: Truy vấn thiết bị từ equipService theo idHouse.
- + Lọc thiết bị theo loại cảm biến: Lọc các thiết bị "Switch" và "DHT11".
- + Lấy quyền của người dùng: Lấy và thêm các quyền (roles) vào model.
- + Thêm dữ liệu vào model: Thêm danh sách thiết bị, quyền và trạng thái đèn vào model.
- + Trả về trang giám sát: Chuyển hướng đến trang /monitor để hiển thị thông tin.

3.6.1. Giám sát thiết bị

Code ESP32 đọc và gửi giá trị nhiệt độ và độ ẩm :Sử dụng thư viện để gửi chuỗi Json lên web.

```

if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor!");
} else {
    http.begin(serverURL + "/updateSensorData");
    String jsonData = "{\"temperature\": " + String(temperature) + ", \"humidity\": " +
String(humidity) + "}";
    http.addHeader("Content-Type", "application/json");
    int postCode = http.POST(jsonData);
    if (postCode == 200) {
        Serial.println("Successfully sent temperature and humidity data.");
    } else {
        Serial.println("Failed to send temperature and humidity data.");
    }
}
http.end();

```

```

@PostMapping("/updateSensorData")
public ResponseEntity<String> updateSensorData(@RequestBody String jsonData) {
    try {
        String[] parts = jsonData.replaceAll("{}\\\"", "").split(",");
        for (String part : parts) {
            String[] keyValue = part.split(":");
            if (keyValue[0].trim().equals("temperature")) {
                temperature = Double.parseDouble(keyValue[1].trim());
            } else if (keyValue[0].trim().equals("humidity")) {
                humidity = Double.parseDouble(keyValue[1].trim());
            }
        }
        return ResponseEntity.ok("Sensor data received and processed successfully.");
    } catch (Exception e) {
        return ResponseEntity.status(400).body("Invalid data format");
    }
}

```

Xử lý chuỗi nhận được từ ESP32 và lấy mỗi thông số giá trị sau đó lưu vào các biến tương ứng.

```

function updateStatus() {
    fetch('/api/getTemperature')
        .then(response => response.json())
        .then(data => {
            document.getElementById('temperatureStatus').textContent =
            `${data.temperature} °C`;
            document.getElementById('humidityStatus').textContent =
            `${data.humidity} %`;
        }).catch(error => console.error('Error fetching temperature:', error));
}

setInterval(updateStatus, 500);
window.onload = updateStatus;

```

```

@GetMapping("/getTemperature")
public ResponseEntity<Map<String, Double>> getTemperature() {
    Map<String, Double> response = new HashMap<>();
    response.put("temperature", temperature);
    response.put("humidity", humidity);
}

```

```

    return ResponseEntity.ok(response);
}

```

Sử dụng hàm `updateStatus` này mỗi 500ms để cập nhật hiển thị giá trị nhiệt độ và độ ẩm thông qua id “`temperatureStatus`” và “`humidityStatus`”

```

<div class="content" th:if="${dht11 != null and !dht11.isEmpty()}" >
    <div class="row justify-content-center">
        <!-- Nhiệt độ -->
        <div class="col-md-4">
            <div class="card_iot">
                <div class="card_header">
                    <i class="fas fa-thermometer-half icon-large"></i> Temperature
                </div>
                <div class="card-body">
                    <h5 id="temperatureStatus">Loading temperature...</h5>
                    <h5>Temperature</h5>
                </div>
            </div>
        </div>
        <!-- Độ ẩm -->
        <div class="col-md-4">
            <div class="card_iot">
                <div class="card_header">
                    <i class="fas fa-tint icon-large"></i> Humidity
                </div>
                <div class="card-body">
                    <h5 id="humidityStatus">Loading humidity...</h5>
                    <h5>Humidity</h5>
                </div>
            </div>
        </div>
    </div>
</div>

```

3.6.2. Điều khiển thiết bị

```

<div class="row">
    <div th:each="equip : ${button}">
        <div class="col-md-3">
            <div class="card_iot">
                <div class="card_header">
                    <i class="fas fa-lightbulb icon-large"></i> <p th:text="
                    ${equip.description}" /></p>

```



```

        </div>
        <div class="card-body">
        <!-- <a class="btn-control" id="lightSwitch1">ON</a> -->
        <button th:id="'ledStatus_' + ${equip.id}"
                class="btn-control"
                th:onclick="'toggleLED(' + ${equip.id} + ')'"> ON
        </button>
        </div>
    </div>
</div>
</div>

```

Hiển thị các Switch được khai báo và thực hiện hàm toggleLED mỗi khi click vào nó

```

function toggleLED(equipId) {
    fetch(`/api/toggleLed/${equipId}`, { method: 'POST' })
        .then(response => response.json())
        .then(data => {
            if (equipId === 2) {
                const statusElement2 = document.getElementById('ledStatus_2');
                statusElement2.textContent = `${data.ledStatus2 === 1 ? 'ON' : 'OFF'} `;
            } else if (equipId === 3) {
                const statusElement3 = document.getElementById('ledStatus_3');
                statusElement3.textContent = `${data.ledStatus3 === 1 ? 'ON' : 'OFF'} `;
            } else if (equipId === 4) {
                const statusElement4 = document.getElementById('ledStatus_4');
                statusElement4.textContent = `${data.ledStatus4 === 1 ? 'ON' : 'OFF'} `;
            } else if (equipId === 5) {
                const statusElement5 = document.getElementById('ledStatus_5');
                statusElement5.textContent = `${data.ledStatus5 === 1 ? 'ON' : 'OFF'} `;
            }
        }).catch(error => console.error(` Error toggling LED ${equipId}:`, error));
}

```

Hàm toggleLED thay đổi trạng thái của các ledStatus mỗi lần nhấn. Khi nhấn nút nó sẽ ánh xạ tới URL `"/api/toggleLed/${equipId}"` để thực hiện việc thay đổi trạng thái của led và trả lại 1 chuỗi JSON.

```

String url = serverURL + "/ledStatus";
http.begin(url);
int httpCode = http.GET();

if (httpCode == 200) {

```

```

        payload = http.getString();
        Serial.println("Received response for equipId: ");
    } else {
        Serial.println("Failed to get data for equipId: ");
    }
    http.end();
    StaticJsonDocument<200> doc;
    if (deserializeJson(doc, payload) == DeserializationError::Ok) {
        int ledStatus2 = doc["ledStatus2"];
        int ledStatus3 = doc["ledStatus3"];
        int ledStatus4 = doc["ledStatus4"];
        int ledStatus5 = doc["ledStatus5"];
    }

```

```

@PostMapping("/toggleLed/{equipId}")
public ResponseEntity<Map<String, Object>> toggleLedFromWeb(@PathVariable Long equipId) {
    Optional<Equipments> equipment = equipService.findById(equipId);
    Map<String, Object> response = new HashMap<>();
    if (equipId == 2){
        ledStatus2 = !ledStatus2;
        response.put("ledStatus"+equipId, ledStatus2 ? 1 : 0); // Trạng thái LED mới
        (1: ON, 0: OFF)
    }
    else if (equipId == 3) {
        ledStatus3 = !ledStatus3;
        response.put("ledStatus"+equipId, ledStatus3 ? 1 : 0); // Trạng thái LED mới
        (1: ON, 0: OFF)
    }
    else if (equipId == 4) {
        ledStatus4 = !ledStatus4;
        response.put("ledStatus"+equipId, ledStatus4 ? 1 : 0); // Trạng thái LED mới
        (1: ON, 0: OFF)
    }
    else if (equipId == 5) {
        ledStatus5 = !ledStatus5;
        response.put("ledStatus"+equipId, ledStatus5 ? 1 : 0); // Trạng thái LED mới
        (1: ON, 0: OFF)
    }
    response.put("id", equipment.get().getId());
    response.put("name", equipment.get().getSensor());
    return ResponseEntity.ok(response);
}

```

ESP32 sẽ gửi yêu cầu truy cập phương thức GET để lấy chuỗi JSON.

```

@GetMapping("/ledStatus")
public ResponseEntity<Map<String, Object>> getLedStatus() {
    Map<String, Object> response = new HashMap<>();
    response.put("ledStatus2", ledStatus2 ? 1 : 0); // Trạng thái LED (1: ON, 0: OFF)
    response.put("ledStatus3", ledStatus3 ? 1 : 0); // Trạng thái LED (1: ON, 0: OFF)
    response.put("ledStatus4", ledStatus4 ? 1 : 0);
    response.put("ledStatus5", ledStatus5 ? 1 : 0);
    return ResponseEntity.ok(response);
}

```

Code ESP32 nhận chuỗi JSON:

```

String url = serverURL + "/ledStatus";
http.begin(url);
int httpCode = http.GET();
if (httpCode == 200) {
    payload = http.getString();
    Serial.println("Received response for equipId: ");
} else {
    Serial.println("Failed to get data for equipId: ");
}
http.end();
StaticJsonDocument<200> doc;
if (deserializeJson(doc, payload) == DeserializationError::Ok) {
    int ledStatus2 = doc["ledStatus2"];
    int ledStatus3 = doc["ledStatus3"];
    int ledStatus4 = doc["ledStatus4"];
    int ledStatus5 = doc["ledStatus5"];
}

```

Code ESP32 dùng để điều khiển các chân IO sau :

```

if (ledStatus2 == 1) {
    digitalWrite(ledPin2, HIGH);
    Serial.println("LED 2 is ON");
} else {
    digitalWrite(ledPin2, LOW);
    Serial.println("LED 2 is OFF");
}
if (ledStatus3 == 1) {
    digitalWrite(ledPin3, HIGH);
    Serial.println("LED 3 is ON");
} else {
    digitalWrite(ledPin3, LOW);
    Serial.println("LED 3 is OFF");
}
if (ledStatus4 == 1) {
    digitalWrite(ledPin4, HIGH);
    Serial.println("LED 4 is ON");
} else {

```

```
        digitalWrite(ledPin4, LOW);  
        Serial.println("LED 4 is OFF");  
    }  
    if (ledStatus5 == 1) {  
        digitalWrite(ledPin5, HIGH);  
        Serial.println("LED 5 is ON");  
    } else {  
        digitalWrite(ledPin5, LOW);  
        Serial.println("LED 5 is OFF");  
    }  
}
```

4. ĐIỀU KHOẢN PHÁP LÝ VÀ TUÂN THỦ

4.1. Điều Khoản Dịch Vụ

Trang web MooseHome cung cấp các điều khoản dịch vụ nhằm đảm bảo người dùng hiểu rõ và tuân thủ các quy định khi sử dụng hệ thống. Các nội dung chính bao gồm:

- + **Quyền và trách nhiệm của người dùng:** Người dùng phải cung cấp thông tin chính xác, bảo mật tài khoản và sử dụng dịch vụ đúng mục đích. Hành vi sử dụng sai cách hoặc trái pháp luật sẽ bị xử lý theo quy định.
- + **Quyền sở hữu trí tuệ:** Tất cả nội dung, hình ảnh và tài liệu trên trang web thuộc quyền sở hữu của MooseHome hoặc các bên cấp phép hợp pháp. Nghiêm cấm sao chép, phân phối hoặc sử dụng trái phép.
- + **Chính sách bảo mật:** Thông tin cá nhân của người dùng được thu thập và sử dụng theo quy định pháp luật Việt Nam, bao gồm Luật An ninh mạng 2018 và Nghị định 13/2023/NĐ-CP về bảo vệ dữ liệu cá nhân.

4.2. Tiêu Chuẩn Tuân Thủ

Trang web MooseHome cam kết tuân thủ các quy định pháp luật Việt Nam và các tiêu chuẩn quốc tế liên quan đến bảo mật dữ liệu và an ninh mạng:

- + **Luật An Ninh Mạng 2018:** Đảm bảo an toàn thông tin cá nhân và dữ liệu người dùng trong quá trình sử dụng dịch vụ.
- + **Nghị định 13/2023/NĐ-CP:** Bảo vệ quyền riêng tư và quyền kiểm soát dữ liệu cá nhân của người dùng.
- + **GDPR (General Data Protection Regulation):** Áp dụng đối với người dùng tại Liên minh Châu Âu.
- + **PCI-DSS (Payment Card Industry Data Security Standard):** Bảo đảm an toàn cho các giao dịch thanh toán trực tuyến.

4.3. Cam Kết Bảo Mật và Quyền Riêng Tư

Chúng tôi cam kết:

- + **Bảo vệ thông tin cá nhân:** Mọi dữ liệu cá nhân được lưu trữ và xử lý một cách an toàn, tuân thủ các quy định pháp lý.
- + **Minh bạch dữ liệu:** Người dùng có quyền yêu cầu truy cập, chỉnh sửa hoặc xóa dữ liệu cá nhân của mình bất kỳ lúc nào.
- + **Phòng chống tấn công mạng:** Hệ thống được trang bị các biện pháp bảo mật tiên tiến để ngăn chặn các hành vi truy cập trái phép hoặc tấn công mạng.

5. KẾT LUẬN

Tài liệu hướng dẫn sử dụng và phát triển MooseHome là công cụ hữu ích để giúp người dùng và nhà phát triển tận dụng tối đa những tính năng của nền tảng quản lý nhà thông minh. Chúng tôi mong rằng thông qua tài liệu này, bạn sẽ dễ dàng quản lý và kiểm soát các thiết bị IoT trong nhà, đồng thời hiểu rõ hơn về cách hệ thống hoạt động.

Sự thành công của hệ sinh thái MooseHome không chỉ phụ thuộc vào công nghệ mà còn nhờ vào sự đóng góp ý kiến và trải nghiệm của bạn. Hãy luôn chia sẻ phản hồi để chúng tôi không ngừng cải tiến và mang lại những giá trị tốt nhất cho ngôi nhà thông minh của bạn.

Cảm ơn bạn đã sử dụng MooseHome và đồng hành cùng chúng tôi trong hành trình xây dựng một cuộc sống tiện nghi và hiện đại hơn!

TÀI LIỆU THAM KHẢO

1. Spring Framework Documentation

URL: <https://spring.io/projects/spring-framework>

Miêu tả: Tài liệu chính thức của Spring Framework, cung cấp hướng dẫn chi tiết về cách sử dụng Spring Security, Spring Boot.

2. Baeldung: Spring Tutorials

URL: <https://www.baeldung.com/>

Miêu tả: Hướng dẫn từng bước về cách triển khai Spring Boot, Spring Security, xử lý email qua SMTP.

3. Gmail SMTP Documentation

URL: <https://support.google.com/mail/answer/7126229?hl=en>

Miêu tả: Hướng dẫn chi tiết cách cấu hình và sử dụng SMTP của Gmail để gửi email.

4. Oracle Java Documentation

URL: <https://docs.oracle.com/en/java/>

Miêu tả: Tài liệu chính thức của Oracle Java, hỗ trợ triển khai các đoạn mã Java trong ứng dụng.

5. MooseHome User Guide

Miêu tả: Hướng dẫn sử dụng hệ thống MooseHome, bao gồm các sơ đồ ERD, Sequence Diagram, và mã nguồn mẫu (tài liệu nội bộ từ nhóm dự án).

6. Thư viện mã hóa Bcrypt

URL: <https://mvnrepository.com/artifact/org.springframework.security/spring-security-crypto>

Miêu tả: Thư viện hỗ trợ mã hóa mật khẩu bảo mật cao, được sử dụng trong dự án.

7. Entity Relationship Diagram (ERD) Design

URL: <https://www.lucidchart.com/pages/er-diagram-tool>

Miêu tả: Hướng dẫn và công cụ thiết kế sơ đồ ERD, liên quan đến phân tích mối quan hệ giữa các bảng.

8. Trang web hỗ trợ mã nguồn mở GitHub

URL: <https://github.com/>

Miêu tả: Lưu trữ mã nguồn mở và các ví dụ triển khai liên quan đến các chức năng trong dự án.

9. Stack Overflow

URL: <https://stackoverflow.com/>

Miêu tả: Cộng đồng hỗ trợ lập trình, giải quyết các vấn đề kỹ thuật liên quan đến Spring Boot, bảo mật, và quản lý cơ sở dữ liệu.

10. Apache Maven Repository

URL: <https://mvnrepository.com/>

Miêu tả: Nguồn tải các thư viện mã nguồn mở, bao gồm Spring Security, Spring Boot, và các công cụ mã hóa.

11. Java Persistence API (JPA) Documentation

URL: <https://docs.oracle.com/javaee/7/tutorial/persistence-intro.htm>

Miêu tả: Tài liệu hướng dẫn sử dụng JPA trong việc quản lý cơ sở dữ liệu và tích hợp với Spring Data.

12. Thư viện Apache Commons

URL: <https://commons.apache.org/>

Miêu tả: Các công cụ hỗ trợ xử lý dữ liệu, thao tác file, và cấu hình hệ thống.

13. Hướng dẫn thiết kế Sequence Diagram

URL: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/>

Miêu tả: Hướng dẫn tạo và sử dụng Sequence Diagram trong phân tích và thiết kế hệ thống.

14. JavaMail API Documentation

URL: <https://javaee.github.io/javamail/>

Miêu tả: Hướng dẫn sử dụng JavaMail API để gửi và nhận email trong các ứng dụng Java.

15. Post`man API Testing

URL: <https://www.postman.com/>

Miêu tả: Công cụ kiểm tra API, hỗ trợ phát triển và kiểm thử các tính năng REST API của MooseHome.