VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

HUYNH TRAN TRUNG HIEU – 518H0090
TO THI BICH TUYEN – 518H0585

TEACHEQUIP MANAGEMENT SYSTEM

INFORMATION TECHNOLOGY PROJECT 2

SOFTWARE ENGINEERING

HO CHI MINH CITY, 2024

VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

HUYNH TRAN TRUNG HIEU – 518H0090
TO THI BICH TUYEN – 518H0585

TEACHEQUIP MANAGEMENT SYSTEM

INFORMATION TECHNOLOGY PROJECT 2

SOFTWARE ENGINEERING

Advised and supervised by

Mr. Nguyen Ngoc Phien

HO CHI MINH CITY, 2024

ACKNOWLEDGMENT

We would like to sincerely thank and deeply appreciate Mr. Nguyen Ngoc Phien for his unwavering assistance and guidance throughout the research and report completion.

Additionally, we would like to express our gratitude to Ton Duc Thang University's Department of Information Technology for setting up the necessary framework for my study and research on this topic. The faculty at Ton Duc Thang University has always been willing to give helpful information and their experiences with reference papers. This helps with research project execution and completion as well as general practice learning and training.

We finished the research assignment after a time of in-class study thanks to the instructors' advice, assistance, and knowledge. The group still has a lot of flaws and limits as a result of information and reasoning constraints. To help us complete our research, we would appreciate your advice and assistance. We will also do better on future research papers because of the advice from friends and lecturers. We wish for the health and happiness of all of our teachers and friends, who have always supported and cared for me.

Ho Chi Minh City, July 21, 2024

Author

(Sign and write full name)

To Thi Bich Tuyen

Huynh Tran Trung Hieu

DECLARATION OF AUTHORSHIP THE THESIS HAS BEEN ACCOMPLISHED AT TON DUC THANG UNIVERSITY

We hereby declare that this is my own project and is guided by Mr Nguyen Ngoc Phien. The research contents and results on this topic are honest and have not been published in any form before. The data in the tables for analysis, comments, and evaluation are collected by the author himself from different sources, clearly stated in the reference section.

In addition, the project also uses a number of comments, assessments as well as data from other authors, other agencies, and organizations, with citations and source annotations.

If I find any fraud, I will take full responsibility for the content of my project. Ton Duc Thang University is not related to copyright and copyright violations caused by me during the implementation process (if any).

Ho Chi Minh City, July 21, 2024

Author

(Sign and write full name)

To Thi Bich Tuyen

Huynh Tran Trung Hieu

INFORMATION TECHNOLOGY PROJECT 2

ABSTRACT

With increasingly strong development, the Information Technology industry plays an increasingly important role. Information technology has played a significant role in transforming many aspects of life.

Recognizing the importance of information technology in daily life, we have utilized the skills we have learned at Ton Duc Thang University to create a website that seeks to facilitate the examination and oversight of the use of instructional resources throughout the entire institution. In this essay, we would like to introduce and present all methods, techniques, and technologies to develop the TeachEquip Management System website.

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION AND TOPIC OVERVIEW 10
1.1 Reasons for choosing the topic 10
1.2 Objectives of the project 10
1.3 Technical of the project 11
CHAPTER 2. SYSTEM ANALYSIS AND DESIGN 12
2.1 System specification 12
2.1.1 Functional Requirement 12
2.1.2 Non-Functional Requirements 13
2.2 Use case 15
2.3 Class Diagram 35
2.3.1 Class Diagram Description 35
2.3.1.1 Role 35
2.3.1.2 Account 36
2.3.1.3 AccountDetail 37
2.3.1.3 InventoryHistory 38
2.3.1.4 Inventory 39
2.4 ERD 46
2.5 Activity Diagram 54
2.5.1 Login 54
2.5.2 Create Tool 55
2.5.3 Create Request 56
2.5.4 Approve Request 57
2.5.5 Edit Tool 58
2.6 Sequence Diagram 59
2.6.1 Login 59
2.6.2 Create Tool 60
2.6.3 Create Request 61
2.6.4 Approve Request 62
CHAPTER 3. SYSTEM ARCHITECTURE, STRUCTURE AND MAIN FUNCTION 63
3.1 Architecture 63
3.2 Structure 63
3.2.1 Frontend 63
3.2.2 Backend 64

3.3 Main function 66
3.3.1 Create Request 66
3.3.2 Update Request 66
3.3.3 Manage 67
CHAPTER 4. USER INTERFACE 68
4.1 Login 68
4.2 Dashboard/Home 68
4.3 Account 69
4.3.1 View 69
4.3.2 Create 70
4.3.3 Edit 71
4.4 Inventory 72
4.4.1 View 72
4.4.2 Edit 72
4.4.3 Create Request 73
4.4.4 Create Invoice 73
4.4.5 View Invoices 74
4.5 Request 75
4.5.1 View 75
4.5.2 History 75
4.5.3 Borrow 76
4.5.4 Create 76
4.5.5 Approve 77
4.6 Tool 77
4.6.1 View 77
4.6.2 Create 78
4.6.3 Edit 78
4.7 Supplier 79
4.7.1 View 79
4.7.2 Create 80
4.7.3 Edit 80
4.8 Category 81
4.8.1 View 81
4.8.2 Create 82

4.8.3 Edit 82
4.9 Setting 83
REFERENCES 84

LIST OF FIGURES

Figure 1 Use case 15
Figure 2 Class Diagram 35
Figure 3 ERD 46
Figure 4 Login 54
Figure 5 Create Tool 55
Figure 6 Create Request 56
Figure 7 Approve Request 57
Figure 8 Edit tool 58
Figure 9 Login 59
Figure 10 Create tool 60
Figure 11 Create Request 61
Figure 12 Approve Request 62
Figure 13 Architecture 63
Figure 14 Login 68
Figure 15. Dashboard/Home 68
Figure 16 View 69
Figure 17 Create 70
Figure 18 Edit 71
Figure 19 View 72
Figure 20 Edit 72
Figure 21 Create Request 73
Figure 22 Create Invoice 73
Figure 23 View Invoices 74
Figure 24 View 75
Figure 25 History 75
Figure 26. Borrow 76
Figure 27 Create 76
Figure 28 Approve 77
Figure 29 View 77
Figure 30 Create 78
Figure 31 Edit 78
Figure 32 View 79
Figure 33 Create 80
Figure 34 Edit 81
Figure 35. View 81
Figure 36. Create 82

Figure 37 Edit 82
Figure 38 Setting 83

LIST OF TABLES

Table 1 Functional Requirement 13
Table 2 Non-Functional Requirements 14
Table 3 Use case description of Login 17
Table 4 Use case description of Logout 18
Table 5 Use case description of View Tool Details. 19
Table 6 Use case description of Create Tool 20
Table 7 Use case description of Update Tool Information 21
Table 8 Use case description of Update Account Information 23
Table 9 Use case description of Create Category 24
Table 10 Use case description of Update Category Information 25
Table 11 Use case description of View Inventory 26
Table 12 Use case description of Create request 28
Table 13 Use case description of View Request 29
Table 14 Use case description of Dashboard 30
Table 15 Use case description of View Inventory History 31
Table 16 Use case description of Create Return Request 33
Table 17 Use case description of View Inventory Borrow 34
Table 18 Create Request Function 66
Table 19 Update Request Function 66
Table 20 Manage Function 67

CHAPTER 1. INTRODUCTION AND TOPIC OVERVIEW

1.1 Reasons for choosing the topic

With the increasing number of students, the need to use teaching aids to serve the teaching of Ton Duc Thang University also increases. That leads to the inability to thoroughly and transparently manage teaching aids, leading to loss and damage of tools. Based on that, we decided to come up with ideas and apply the knowledge we have learned to find a solution to the above problem.

In order for the equipment management team to clearly understand the schedule of using tools and easily handle and catalog all types of instructional equipment, we decided to build a website to help streamline, clarify and increase transparency in management work with our knowledge and experience.

1.2 Objectives of the project

The website is built based on the need to manage teaching tools in universities, making management easier and more transparent.

The website is divided into three main roles such as admin, manager and user. Each role will have different functions.

The website includes main functions such as managing tools, managing users, managing usage history and borrowing and returning tools.

In addition, the website can also manage inventory, declare detailed price fluctuation history of tools.

1.3 Technical of the project

- **SDLC Model:** Waterfall
- **Program language:**
  - Front end: Vuejs
  - Back-end: ASP.NET Core Api.
- **Database:** Sql server
- **IDE and code editor:** Visual Studio, Visual studio code, SQL Server Management Studio.
- **Framework:** Vuejs, TailwindCss, ASP.NET Core API, Graph api Sharepoint.
- **Document management system:** Github, Sharepoint.

CHAPTER 2. SYSTEM ANALYSIS AND DESIGN

2.1 System specification

2.1.1 Functional Requirement

Actor	Functional requirements
Admin function	- Admin can login, logout, and view the dashboard. - Admin can manage all categories, suppliers and tools information. - Admin can manage users, managers, inventories and invoices (add, delete, edit). - Admin can view requests, inventory histories and borrow histories. - Admin can make new requests and approve requests from the user.
Manager function	- Manager can login, logout, and view the dashboard. - Manager can manage all categories, suppliers and tools information. - Managers can manage users, inventories and invoices (add, delete, edit). - Manager can view the user's request, inventory histories and borrow histories. - Manager can make new requests and approve requests from the user.

User function	- Users can login, logout, view dashboard. - Users can view Inventory, tool information such as tool name, total quantity, amount borrowed. - User can make a request to borrow the tool if logged in, when the user borrows it will show the list tool and amount of borrow then the user can make a request to return the tool. - Users can edit personal information.

Table 1 Functional Requirement

2.1.2 Non-Functional Requirements

Non-Functional	Requirements
Operational	- Website display is compatible with many browsers.
Usability	- The website's layout and functions should be neat and easy to interact with.
Performance	- Improve system performance, optimize website response time. - The response time of the website when performing functions cannot exceed 10 seconds.

User interface	- Choose white, black, green and blue as the main colors for the interface. - Interface layout is suitable for managing teaching equipment at school.
Maintainability	- Code must be clear and follow variable naming rules for easy maintenance and updates.
Security	- Password must contain at least 8 characters and be encrypted. - Admin and manager can approve requests from the end of the user. - Admin and manager manage all information in the system such as category, supplier, invoice, ... - Users can view inventory to see tool information, number available and make requests to borrow tools, when they borrow they can return tools to the inventory.

Table 2 Non-Functional Requirements

2.2 Use case

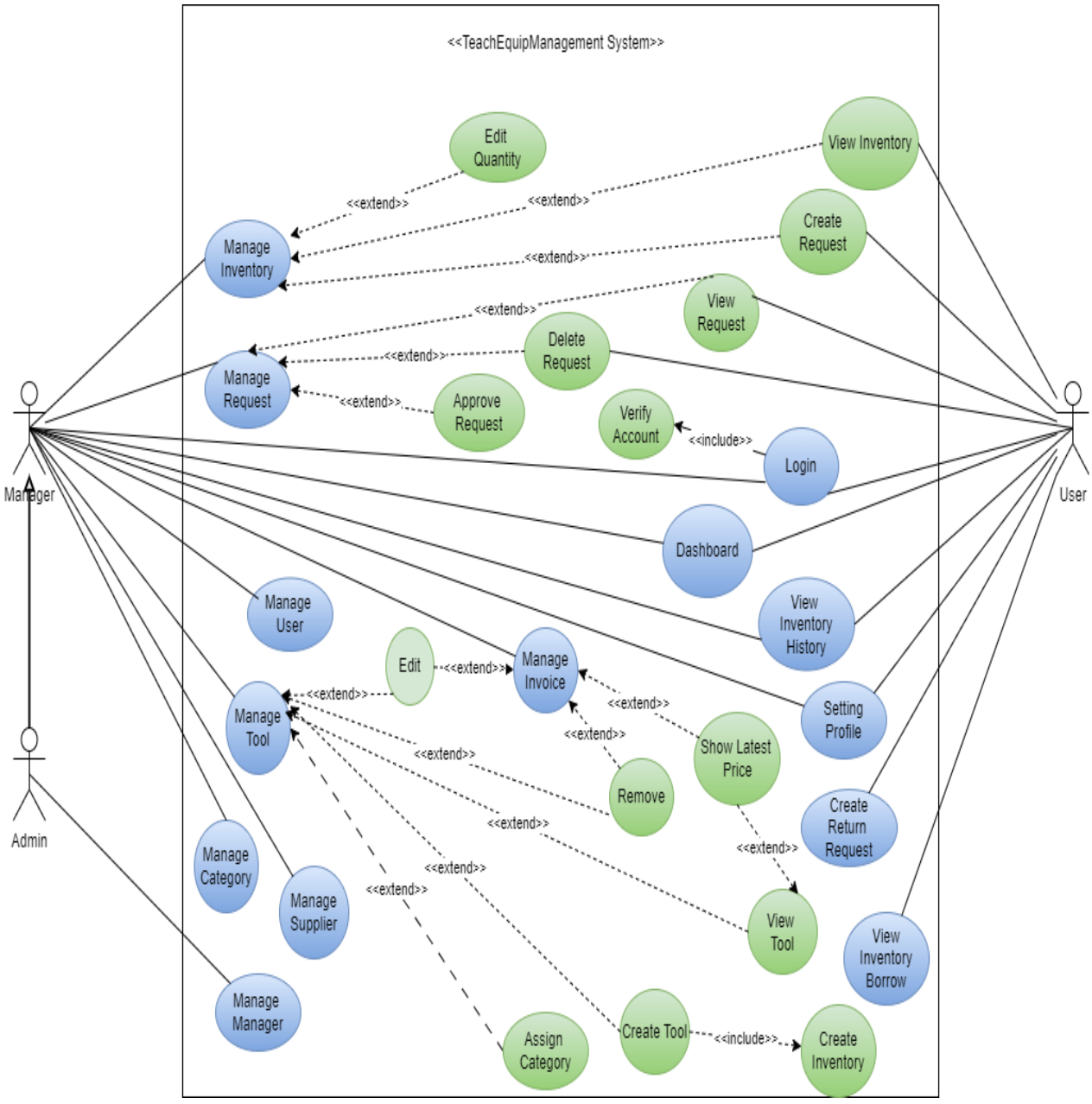

Figure 1 Use case

Use case name:	Log in	
Triggering events:	User, Manager, Admin want to login to Teachequip Management System website.	
Brief description	User, Manager, Admin login to the website to perform transaction and management functions.	
Actors:	User, Manager, Admin.	
Preconditions:	The account has been created and has been authorized.	
Postconditions:	The account is successfully logged in and the username is displayed on the website at the same time.	
Flow of activities:	Actor	System
	1. User, Manager, Admin access Teachequip Management System website. 2. User, Manager, Admin enter necessary information. 3. Click the 'Login' button to login to the website.	3.1 System authenticates login information, checks permissions. 3.2 Login successfully and display login information. 3.3 End the use case.

Alternative Flow:	3.2 Display the login information in the toolbar.
Exception conditions:	3.1 The system displays a message if the login fails.

Table 3 Use case description of Login

Use case name:	Log out	
Triggering events:	User, Admin, Manager want to log out the account.	
Brief description	User, Admin, Manager log out of the account.	
Actors:	User, Admin, Manager.	
Preconditions:	Account is logged in.	
Postconditions:	Sign out successfully.	
Flow of activities:	Actor	System
	1. The user wants to log out of the account. 2. Select the 'Sign Out' button.	2.1 The system logs out of the account. 2.2 End the use case.
Alternative Flow:	The system returns to the home page upon successful logout.	

Exception conditions:	Do not have.

Table 4 Use case description of Logout

Use case name:	View tool details.	
Triggering events:	Admin, User, Manager wants to see tool information.	
Brief description	Admin, User, Manager view tool information.	
Actors:	Admin, User, Manager.	
Preconditions:	Do not have.	
Postconditions:	Display tool details.	
Flow of activities:	Actor	System
	1. Admin, User, Manager selects the tool he wants to see detailed information about. 2. Admin, User, Manager click the 'View' button at the record you want to see detailed information about.	2.1 The system displays tool details. 2.2 End the use case.

Alternative Flow:	1.1 Displays the details of the selected tool.
Exception conditions:	Do not have.

Table 5 Use case description of View Tool Details.

Use case name:	Create Tool	
Triggering events:	Manager, Admin wants to create tools.	
Brief description	Manager, Admin can create tools.	
Actors:	Manager, Admin.	
Preconditions:	Account is logged in.	
Postconditions:	Manager, Admin can create new tools successfully.	
Flow of activities:	Actor	System

	1. Manager, Admin wants to create tools. 2. Manager, Admin click 'Tool' button then click 'Add' button. 3. Manager, Admin fills in the tool information 4. Manager, Admin attaches a photo of the tool and click the "Add" button.	4.1 System checking information was filled. 4.2 System processing information and creating tool. 4.3 End the use case.
Alternative Flow:	3.2 Display a message if the create was successful.	
Exception conditions:	3.2 Display an error message if manager fill in missing information	

Table 6 Use case description of Create Tool

Use case name:	Update Tool information
Triggering events:	Manager, Admin wants to update tool information.
Brief description	Manager, Admin can update tool information.
Actors:	Manager, Admin.

Preconditions:	Account is logged in.	
Postconditions:	Manager, Admin can update tool information successfully.	
Flow of activities:	Actor	System
	1. Manager, Admin wants to update tool information. 2. Manager, Admin update tool information. 3. Manager, Admin click the "Edit" button.	3.1 System checking information was filled. 3.2 System processing information and edit tool information. 3.3 End the use case.
Alternative Flow:	3.2 Display a message if the update was successful.	
Exception conditions:	3.2 Display an error message if manager fill in missing information	

Table 7 Use case description of Update Tool Information

Use case name:	Update Account Information
Triggering events:	Admin, Manager, User can update account information.

Brief description	Admin, Manager,User can update username, password, mail.... of the account.	
Actors:	Admin, Manager, User .	
Preconditions:	Account is logged in.	
Postconditions:	Account update successful.	
Flow of activities:	Actor	System
	1. Admin, Manager, User wants to change account information. 2. Select ‘Accounts’ and click the button ‘Edit’ at the record you want to update. 3. Admin, Manager, User in can edit, update the information in the account. 4. Press the 'Update' button.	4.1 The system checks the entered information. 4.2 Information processing. 4.3 End the use case.
Alternative Flow:	4.1.2 Display a message if the edit is successful.	

Exception conditions:	4.1.1 Display an error message if the user enters missing, incorrect information or incorrect syntax.

Table 8 Use case description of Update Account Information

Use case name:	Create category	
Triggering events:	Admin, Manager wants to create a category.	
Brief description	Admin, Manager can create a category.	
Actors:	Admin, Manager.	
Preconditions:	Account is logged in.	
Postconditions:	Admin, Manager can create a category successfully.	
Flow of activities:	Actor	System

	1. Admin, Manager wants to create a category. 2. Admin, Manager fills in the category information 3. Admin, Manager Click the "Add" button.	3.1 System checking information was filled. 3.2 System processing information and creating a category. 3.3 End the use case.
Alternative Flow:	3.2 Display a message if the create successfully.	
Exception conditions:	3.2 Display an error message if manager fill in missing information	

Table 9 Use case description of Create Category

Use case name:	Update category information
Triggering events:	Manager, Admin wants to update category information.
Brief description	Manager, Admin can update category information.
Actors:	Manager, Admin.
Preconditions:	Account is logged in.

Postconditions:	Manager, Admin can update category information successfully.	
Flow of activities:	Actor	System
	1. Manager, Admin wants to update category information. 2. Manager, Admin update category information. 3. Manager, Admin click the "Edit" button.	3.1 System checking information was filled. 3.2 System processing information and update category information. 3.3 End the use case.
Alternative Flow:	3.2 Display a message if the update was successful.	
Exception conditions:	3.2 Display an error message if manager fill in missing information	

Table 10 Use case description of Update Category Information

Use case name:	View Inventory.
Triggering events:	Admin, User, Manager wants to see inventory information.
Brief description	Admin, User, Manager view inventory information.

Actors:	Admin, User, Manager.	
Preconditions:	Account is logged in.	
Postconditions:	Display inventory details.	
Flow of activities:	Actor	System
	1. Admin, User, Manager click "Inventory"	1.1 Display inventory information of all tools in list form. 1.2 End the use case.
Alternative Flow:	1.1 Displays the information of inventory.	
Exception conditions:	Do not have.	

Table 11 Use case description of View Inventory

Use case name:	Create a request.
Triggering events:	Admin, User, Manager wants to create requests to borrow tools.

Brief description	Admin, User, Manager create request	
Actors:	Admin, User, Manager.	
Preconditions:	Account is logged in.	
Postconditions:	Create a request succesful.	
Flow of activities:	Actor	System
	1. Admin, User, Manager click “Inventory” 2. Admin, User, Manager click “request” at the tool you want to borrow. 3. Admin, User, Manager fill information need to request 4. Admin, User, Manager click “Create request”	4.1 The system validate input information 4.2 End the use case.
Alternative Flow:	4.1 Displays messages created successfully and updates the number of tools in the database.	

Exception conditions:	3.1 Display an error message if it fills in missing information.

Table 12 Use case description of Create request

Use case name:	View request.	
Triggering events:	Admin, User, Manager wants to view requests details that the user created	
Brief description	Admin, User, Manager view detail requests.	
Actors:	Admin, User, Manager.	
Preconditions:	Account is logged in.	
Postconditions:	Display request details.	
Flow of activities:	Actor	System

	1. Admin, User, Manager "Request" 2. Admin, User, Manager click "View" the record that the user created	2.1 The system displays request details. 2.2 End the use case.
Alternative Flow:	1.1 Displays the details of the selected request.	
Exception conditions:	Do not have.	

Table 13 Use case description of View Request

Use case name:	Dashboard
Triggering events:	Admin, User, Manager wants to see the dashboard.
Brief description	Admin, User, Manager view dashboard.
Actors:	Admin, User, Manager.

Preconditions:	Account is logged in.	
Postconditions:	Display dashboard	
Flow of activities:	Actor	System
	1. Manager, Admin access to system.	1.1 System display dashboard. 1.2 End the use case.
Alternative Flow:	1.1 Display the dashboard	
Exception conditions:	Do not have.	

Table 14 Use case description of Dashboard

Use case name:	View inventory history
Triggering events:	Manager, Admin and User want to see inventory history that the user created.

Brief description	Manager, Admin and User can see inventory history.	
Actors:	Manager, Admin and User	
Preconditions:	Account is logged in.	
Postconditions:	Manager, Admin and User can view inventory history that the user created.	
Flow of activities:	Actor	System
	1. Manager, Admin and User click “Inventory” 2. Manager, Admin and User click “History”	2.1 System display inventory history. 2.2 End the use case.
Alternative Flow:	2.1 Display the inventory history.	
Exception conditions:	Do not have	

Table 15 Use case description of View Inventory History

Use case name:	Create return request	
Triggering events:	Manager, Admin and User want to create a return request.	
Brief description	Manager, Admin and User can create a return request.	
Actors:	Manager, Admin and User	
Preconditions:	Account is logged in.	
Postconditions:	Manager, Admin and User can create a return request at the record that the user want to return	
Flow of activities:	Actor	System
	1. Manager, Admin and User click “Request” 2. Manager, Admin and User fill information that the user wants to return.	3.1 The system validate input information 3.2 End the use case.

	3. Manager, Admin and User click “Return” at the record that the user want to return	
Alternative Flow:	3.1 Displays messages created successfully and updates the number of tools in the database.	
Exception conditions:	3.1 Display an error message if it fills in missing information.	

Table 16 Use case description of Create Return Request

Use case name:	View inventory borrow.
Triggering events:	Manager, Admin and User want to see inventory borrowed that the user created.
Brief description	Manager, Admin and User can see borrowed history.
Actors:	Manager, Admin and User
Preconditions:	Account is logged in.

Postconditions:	Manager, Admin and User can view borrowed history that the user created.	
Flow of activities:	Actor	System
	1. Manager, Admin and User click “Inventory” 2. Manager, Admin and User click “Borrow”	2.1 System display borrow history. 2.2 End the use case.
Alternative Flow:	1.1 Display the borrowed history.	
Exception conditions:	Do not have	

Table 17 Use case description of View Inventory Borrow

2.3 Class Diagram

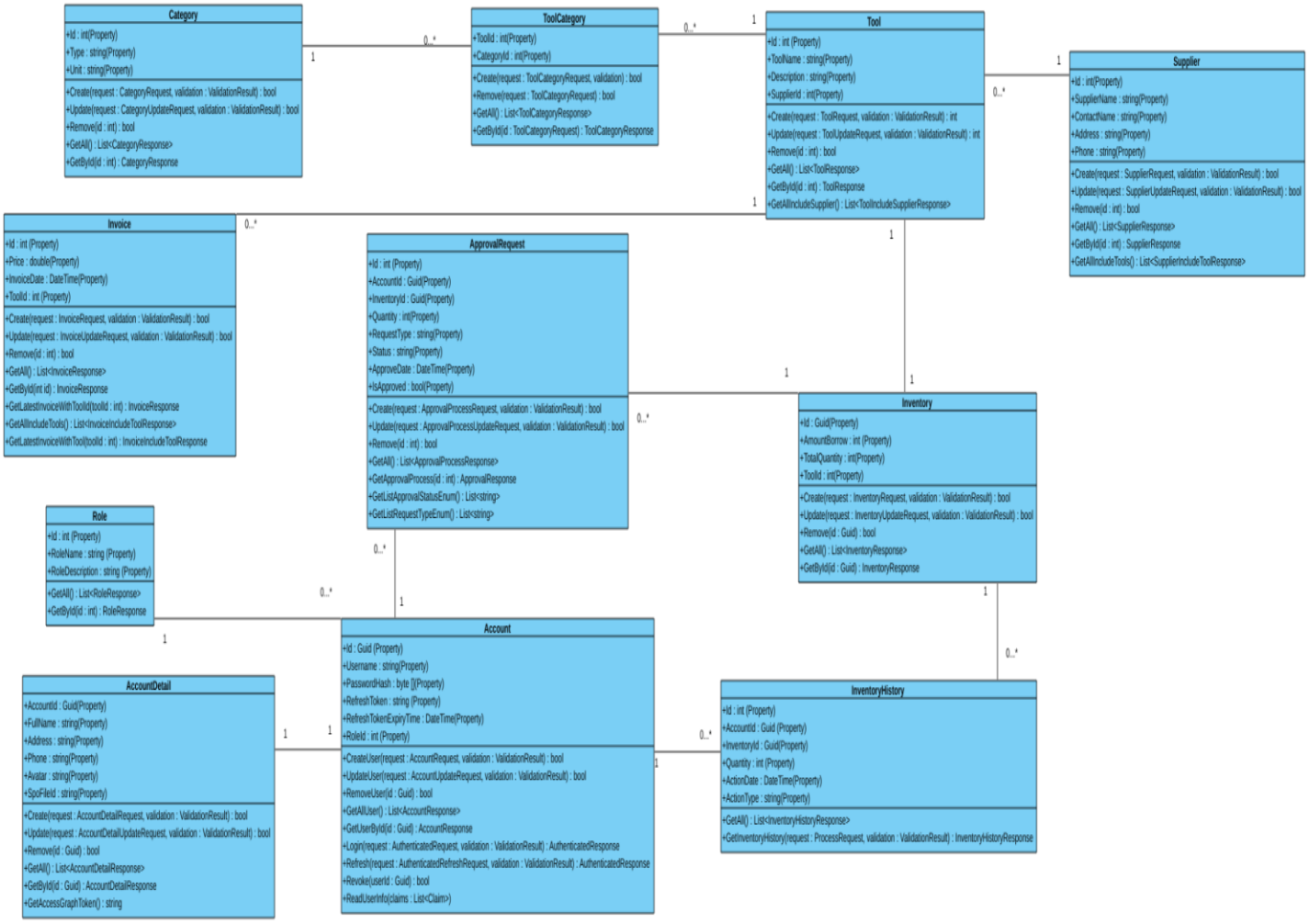

Figure 2 Class Diagram

2.3.1 Class Diagram Description

2.3.1.1 Role

Design

- ➢ <<**Property**>> + Id: int
- ➢ <<**Property**>> + RoleName: string
- ➢ <<**Property**>> + RoleDescription: string

- <<Method>> + GetAll(): List<RoleResponse>
- <<Method>> + GetById(id: int) : RoleResponse

Extend

- **RoleResponse**: Data transfer class for Role - It contains Id, RoleName, RoleDescription and remove relation object.

2.3.1.2 Account

Design

- **<<Property>>** + Id: Guid
- **<<Property>>** + Username: string
- **<<Property>>** + PasswordHash: byte[]
- **<<Property>>** + PasswordSalt: byte[]
- **<<Property>>** + RefreshToken: string
- **<<Property>>** + RefreshTokenExpiryTime: DateTime
- **<<Property>>** + RoleId: int
- <<Method>> + CreateUser(request: AccountRequest, validation: ValidationResult) : bool
- <<Method>> + UpdateUser(request: AccountUpdateRequest, validation: ValidationResult): bool
- <<Method>> + RemoveUser(id: Guid): bool
- <<Method>> + GetAllUser(): List<AccountResponse>
- <<Method>> + GetUserById(id: Guid): AccountResponse
- <<Method>> + Login(request: AuthenticatedRequest, validation: ValidationResult): AuthenticatedResponse
- <<Method>> + Refresh(request: AuthenticatedRefreshRequest, validation: ValidationResult): AuthenticatedResponse

- <<Method>> + Revoke(userId: Guid): bool
- <<Method>> + ReadUserInfo(claims: List<Claim>): bool

Extend

- **Password Hash**: a password encrypted and PasswordSalt is a key for encrypted.
- **RefreshToken**: used to generate a new access token when it expired.
- **RefreshTokenExpiryTime**: Storing expired time for access token.
- **ValidationResult**: Using for checking Request input.
- **AccountRequest**: Using for inserting new account contains Username, Password, Email and RoleId.
- **AccountUpdateRequest**: Using for updating account information such as Password, Email, RoleId and we must convey accountId too.
- **AccountResponse**: Data transfer class for Account - It contains Id, Username, PasswordHash, PasswordSalt, Email, RefreshToken, RefreshTokenExpiryTime, RoleId.
- **AuthenticatedRequest**: Request for login it must receive Username and Password.
- **AuthenticatedResponse**: Response for login - AccessToken is the token to access the system and RefreshToken is the token used for refresh new access token.

2.3.1.3AccountDetail

Design

- <<**Property**>> + AccountId: Guid
- <<**Property**>> + FullName: string
- <<**Property**>> + Address: string
- <<**Property**>> + Phone: string
- <<**Property**>> + Avatar: string
- <<**Property**>> + SpoFileId: string

- <<Method>> + Create(request: AccountDetailRequest, validation: ValidationResult): bool
- <<Method>> + Update(request: AccountDetailUpdateRequest, validation: ValidationResult): bool
- <<Method>> + Remove(id: Guid): bool
- <<Method>> + GetAll(): List<AccountDetailResponse>
- <<Method>> + GetById(id: Guid): AccountDetailResponse
- <<Method>> + GetAccessGraphToken(): string

Extend

- **SpoFileId**: DriveItem Id in Sharepoint Online.
- **AccountDetailRequest**: Using for creating new Account Detail it contains AccountId, FullName, Address, Phone.
- **AccountDetailUpdateRequest:** Using for updating Account Detail it contains AccountId, FullName, Address, Phone and FileUpload.
- **AccountDetailResponse**: Responding Account Detail information such as AccountId, FullName, Address, Phone, If it contains image then it will respond to Avatar and SpoFileId.
- **GetAccessGraphToken**: This function will generate an access token to sharepoint online.

2.3.1.3InventoryHistory

Design

- **<<Property>>** + Id: int
- **<<Property>>** + AccountId: Guid
- **<<Property>>** + InventoryId: Guid
- **<<Property>>** + Quantity: int
- **<<Property>>** + ActionDate: DateTime

- **<<Property>>** + ActionType: string
- <<Method>> + GetAll(): List<InventoryHistoryResponse>
- <<Method>> + GetInventoryHistory(request: ProcessRequest, validation: ValidationResult): InventoryHistoryResponse

Extend

- **InventoryHistoryResponse**: Response history information such as Id, AccountId, InventoryId, Quantity, ActionDate and ActionType.
- **ProcessRequest**: Request to find history combining AccountId and InventoryId.
- **ValidationResult**: Using for checking Request input.

2.3.1.4Inventory

Design

- **<<Property>>** + Id: Guid
- **<<Property>>** + AmountBorrow: int
- **<<Property>>** + TotalQuantity: int
- **<<Property>>** + ToolId: int
- <<Method>> + Create(request: InventoryRequest, validation: ValidationResult): bool
- <<Method>> + Update(request: InventoryUpdateRequest, validation: ValidationResult): bool
- <<Method>> + Remove(id: Guid): bool
- <<Method>> + GetAll(): List<InventoryResponse>
- <<Method>> + GetById(id: Guid): InventoryResponse

Extend

- **InventoryRequest**: Request to create new inventory requires TotalQuantity, AmountBorrow and ToolId.

- **ValidationResult**: Using for checking Request input.
- **InventoryUpdateRequest**: Request to update inventory requires TotalQuantity, AmountBorrow and InventoryId.
- **InventoryResponse**: Response Inventory information such as Id, TotalQuantity, AmountBorrow and ToolId.

2.3.1.5Tool

Design

- **<<Property>>** + Id: int
- **<<Property>>** + ToolName: string
- **<<Property>>** + Description: string
- **<<Property>>** + SupplierId: int
- <<Method>> + Create(request: ToolRequest, validation: ValidationResult): int
- <<Method>> + Update(request: ToolUpdateRequest, validation: ValidationResult): int
- <<Method>> + Remove(id: Guid): bool
- <<Method>> + GetAll(): List<ToolResponse>
- <<Method>> + GetById(id: int): ToolResponse
- <<Method>> + GetAllIncludeSupplier(): List<ToolIncludeSupplierResponse>

Extend

- **ToolRequest**: Requesting to create a new tool, it contains ToolName, Description and SupplierId.
- **ValidationResult**: Using for checking Request input
- **ToolUpdateRequest**: Requesting to update tool information, it contains Id, ToolName, Description and SupplierId.

- **ToolResponse**: Response tool information such as Id, ToolName, Description and SupplierId.
- **ToolIncludeSupplierResponse**: Response tool information include supplier information such as Id, ToolName, Description and Supplier information.

2.3.1.6Supplier

Design

- **<<Property>>** + Id: int
- **<<Property>>** + SupplierName: string
- **<<Property>>** + ContactName: string
- **<<Property>>** + Address: string
- **<<Property>>** + ContactName: string
- **<<Property>>** + Phone: string
- <<Method>> + Create(request: SupplierRequest, validation: ValidationResult) : bool
- <<Method>> + Update(request: SupplierUpdateRequest, validation: ValidationResult) : bool
- <<Method>> + Remove(id: Guid): bool
- <<Method>> + GetAll() : List<SupplierResponse>
- <<Method>> + GetById(id: int) : SupplierResponse
- <<Method>> + GetAllIncludeTools(): List<SupplierIncludeToolResponse>

Extend

- **SupplierRequest**: Requesting to create a new tool, it contains ToolName, Description and SupplierId.
- **ValidationResult**: Using for checking Request input.

- **SupplierUpdateRequest**: Requesting to update supplier, it contains Id, SupplierName, ContactName, Address and Phone.
- **SupplierResponse**: Response Supplier information such as Id, SupplierName, ContactName, Address and Phone.
- **SupplierIncludeToolResponse**: Response Supplier information include list tools information.

2.3.1.7 ApprovalRequest

Design

- **<<Property>>** + Id: int
- **<<Property>>** + AccountId: Guid
- **<<Property>>** + InventoryId: Guid
- **<<Property>>** + Quantity: int
- **<<Property>>** + RequestType: string
- **<<Property>>** + Status: string
- **<<Property>>** + ApproveDate: DateTime
- **<<Property>>** + IsApproved: bool.
- <<Method>> + Create(request: ApprovalProcessRequest, validation: ValidationResult): bool
- <<Method>> + Update(request: ApprovalProcessUpdateRequest, validation: ValidationResult): bool
- <<Method>> + Remove(id: int): bool
- <<Method>> + GetAll(): List<ApprovalProcessResponse>
- <<Method>> + GetApprovalProcess(id: int): ApprovalResponse
- <<Method>> + GetListApprovalStatusEnum(): List<string>
- <<Method>> + GetListRequestTypeEnum(): List<string>

Extend

- **GetListApprovalStatusEnum**: Return status approval requests such as Pending and Accept.
- **GetListRequestTypeEnum**: Return list request type such as Borrow, Return, Buy and Sell.

2.3.1.8ToolCategory

Design

- **<<Property>>** + ToolId: Guid
- **<<Property>>** + CategoryId: Guid
- <<Method>> + Create(request: ToolCategoryRequest, validation:): bool
- <<Method>> + Remove(request: ToolCategoryRequest): bool
- <<Method>> + GetAll(): List<ToolCategoryResponse>
- <<Method>> + GetById(id: ToolCategoryRequest): ToolCategoryResponse

Extend

- **ToolCategoryRequest**: Requesting for update ToolCategory.
- **ToolCategoryResponse**: Response Tool and Category information.

2.3.1.9Category

Design

- **<<Property>>** + Id: int
- **<<Property>>** + Type: string
- **<<Property>>** + Unit: string
- <<Method>> + Create(request: CategoryRequest, validation: ValidationResult): bool
- <<Method>> + Update(request: CategoryUpdateRequest, validation: ValidationResult): bool

- <<Method>> + Remove(id: int): bool
- <<Method>> + GetAll(): List<CategoryResponse>
- <<Method>> + GetById(id: int): CategoryResponse

Extend

- **CategoryRequest**: Requesting for create category, it contains Type and Unit.
- **CategoryUpdateRequest**: Requesting for update category, it requires Id, Type and Unit.
- **CategoryResponse**: Response Category information such as Id, Type and Unit.

2.3.1.10 Invoice

Design

- **<<Property>>** + Id: int
- **<<Property>>** + Price: double
- **<<Property>>** + InvoiceDate: Datetime
- **<<Property>>** +ToolId: int
- <<Method>> + Create(request: InvoiceRequest, validation: ValidationResult) : bool
- <<Method>> + Update(request: InvoiceUpdateRequest, validation: ValidationResult) : bool
- <<Method>> + Remove(id: int) : bool
- <<Method>> + GetAll(): List<InvoiceResponse>
- <<Method>> + GetById(int id): InvoiceResponse
- <<Method>> + GetLatestInvoiceWithToolId(toolId: int): InvoiceResponse
- <<Method>> + GetAllIncludeTools(): List<InvoiceIncludeToolResponse>
- <<Method>> + GetLatestInvoiceWithTool(toolId: int) : InvoiceIncludeToolResponse

Extend

- **InvoiceRequest**: Requesting create new invoice, it contains Price and ToolId.
- **InvoiceUpdateRequest**: Requesting update invoice information requires Id and Price.
- **InvoiceResponse**: Response Invoice Information such as Id, Price, InvoiceDate and ToolId.
- **GetLatestInvoiceWithToolId**: Function return the latest invoice for each tool.
- **GetAllIncludeTools**: Get All Invoice belongs to tool.
- **GetLatestInvoiceWithTool**: Get All Latest invoice for each tool.

2.4 ERD

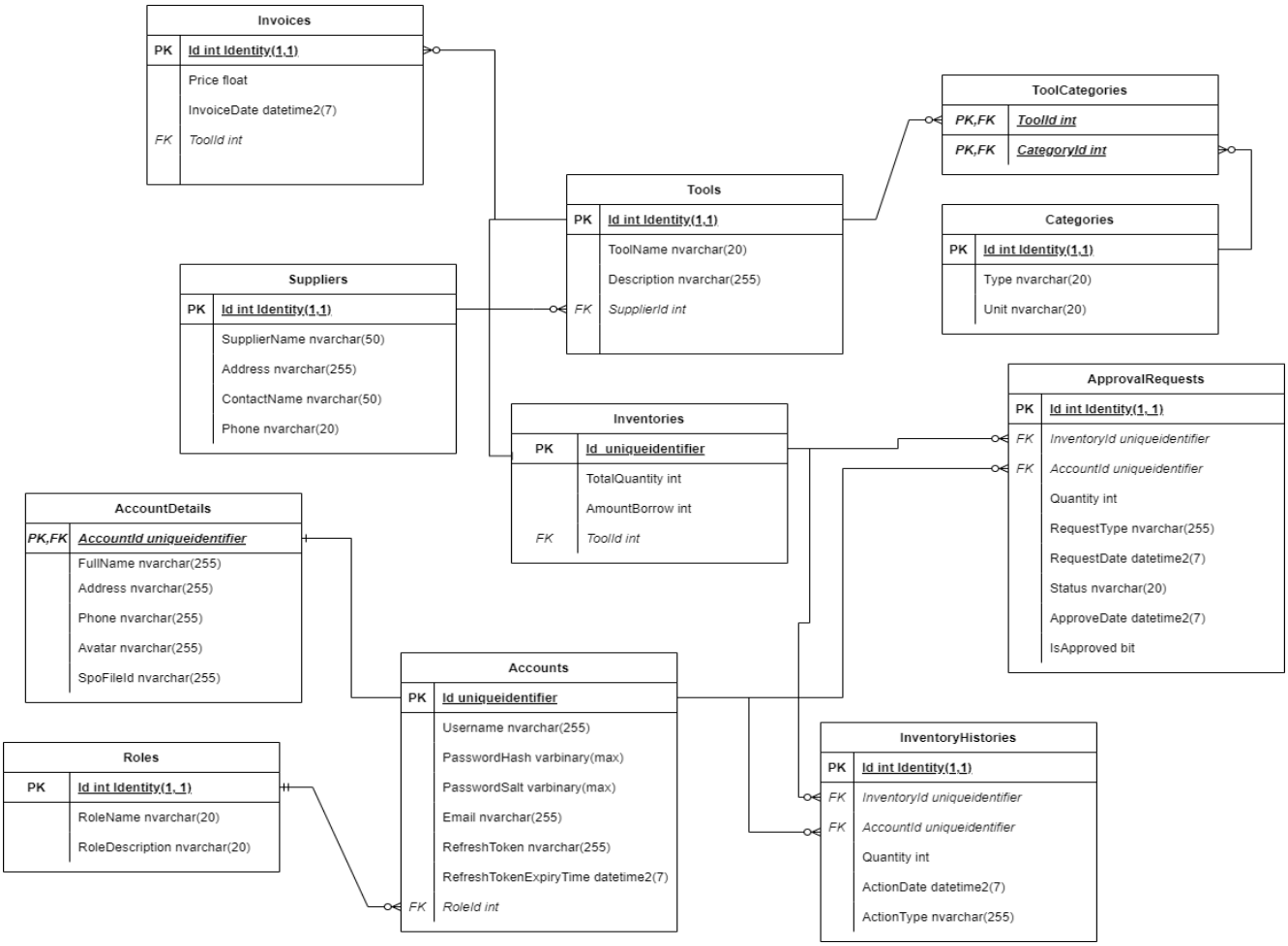

Figure 3 ERD

2.4.1 Accounts table

Field

- **Id(Primary key) - UniqueIdentifier:** User identification
- **Username - Nvarchar(255):** Name of user
- **PasswordHash - Varbinary(max):** Password Hash encrypted for security purpose

- **PasswordSalt - Varbinary(max):** Password Salt is a key for using to decrypt for passwordHash
- **Email - Nvarchar(255):** Email address of user
- **RoleId - Int:** Used to distinguish what role a user has.
- **RefreshToken(Foreign key) - Nvarchar(255):** Store Refresh Token to refresh access token when it expires.
- **RefreshTokenExpiryTime - Datetime2(7):** Store Time Expiry of access token

Relationship

- Each Accounts will have a corresponding Roles
- Each Account has only one AccountDetails which contains Account information.
- Each Account will have multiple ApprovalRequest and InventoryHistories.

2.4.2 Roles table

Field

- **Id(Primary key) - Int Identity(1, 1):** Used to identify role types.
- **RoleName - Nvarchar(20):** The name of that role
- **RoleDescription - Nvarchar(20):** Describe your role in the system

Relationship

- Each role will have many users.

2.4.3 AccountDetails table

Field

- **AccountId(Primary key, Foreign key) - Uniqueidentifier:** User identification which references to Accounts table id.
- **FullName - Nvarchar(255):** Full Name of user
- **Address - Nvarchar(255):** Address of user
- **Phone - Nvarchar(255):** Phone number of user which must be between 8 and 14 number.
- **Avatar - Nvarchar(255):** Avatar's link at Sharepoint online.
- **SpoFileId - Nvarchar(255):** DriveItem at Sharepoint online for Image File.

Relationship

- Each account detail will have a corresponding account.
- An account will have only one account detail and an account detail just has one account.

2.4.4 Tools table

Field

- **Id(Primary key) - Int Identity(1, 1):** Used to identify tools, when inserting a new tool it will automatically increase by 1.
- **ToolName - Nvarchar(20):** Name of tool, it will be limited to 20 characters.
- **Description - Nvarchar(255):** Description of Tool, it will be limited to 255 characters, and just an optional so we can get in equal "".
- **SupplierId(Foreign key) - Int:** Identify of supplier, this field is a foreign key so it will link to Supplier table id.

Relationship

- Each Tool will have a corresponding Supplier.

- Each Tool can have many Invoices.
- Each Tool can have multiple categories through the tool categories table.
- Each Tool will have a corresponding Inventory.

2.4.5 Categories table

Field

- **Id(Primary key) - Int Identity(1, 1):** Used to identify category, when inserting new category, it will be increased by 1.
- **Type - Nvarchar(20):** Name of type category, it will be limited to 20**.**
- **Unit - Nvarchar(20):** Units of teaching tools (for example Pieces, liters), it will be limited to 20.

Relationship

- Each Category can belong to multiple Tools throughout the ToolCategories table.

2.4.6 ToolCategories

Field

- **CategoryId(Primary key, Foreign key):** Used to identify categories which will link to the category table.
- **ToolId(Primary key, Foreign key):** Used to identify tools which will link to the Tool table.

Relationship

- ToolCategories will depend on Categories and Tools, so when a category or tool is removed from the system then all records will be deleted cascade.

2.4.7 Invoices table

Field

- **Id(Primary key) - Int Identity(1, 1):** Identify of Invoice, it will automatically increase by 1.
- **Price - Float**: Price of tool
- **InvoiceDate - Datetime2(7)**: Instrument valuation date
- **ToolId(Foreign key) - Int:** Identify of Tool which will link to Tool table.

Relationship

- Each invoice is belongs to tool

2.4.8 Suppliers table

Field

- **Id(Primary key) - Int Identity(1, 1):** Identification of Supplier, it will automatically increase by 1 when inserting.
- **SupplierName - Nvarchar(50):** Name of supplier, it limits to 50.
- **ContactName - Nvarchar(255):** Name of contact's supplier, it limits to 255.
- **Address - Nvarchar(50):** Address of supplier's company.
- **Phone - Nvarchar(20):** Phone number of supplier, it must be digit between 8 and 14.

Relationship

- Each Supplier will provide many tools.

2.4.9 Inventories table

Field

- **Id(Primary key) - UniqueIdentifier:** Identification of Inventories.
- **TotalQuantity - Int:** It shows the user knows the total available inventory in the system.
- **AmountBorrow - Int:** It shows users know total borrow of inventory in the system.
- **ToolId(Foreign key) - Int:** Identifies the Tool and links to Tool table.

Relationship

- Each Inventory will have multiple ApprovalRequest and InventoryHistories.
- Each Inventory belongs to the corresponding tool.

2.4.10 ApprovalRequests table

Field

- **Id(Primary key) - Int Identity(1, 1):** Identification of Request, it will automatically increase by 1 when inserting.
- **InventoryId(Foreign key) - UniqueIdentifier:** Identify of Inventory, it links to Inventories table.
- **AccountId(Foreign key) - UniqueIdentifier:** Identify of AccountId, it links to the Accounts table.
- **Quantity - Int:** Quantity when making a request for a tool, it must be greater than 0.
- **RequestType - Nvarchar(255):** Type of Request, it has Borrow and Return for users to make requests. Admin and manager can make a request Buy and Sell for tools in Inventory.

- **RequestDate - Datetime2(7):** When making a request, it will automatically get a new time.
- **Status - Nvarchar(20):** Status of Request, when we make a request it will store "Pending" when the admin or manager approves the request then it will change to "Accept".
- **ApproveDate - Datetime2(7):** It automatically gets a new time when approving a request.
- **IsApproved - Bit:** Let admin and manager know if a request is approved or not?.

Relationship

- Each ApprovalRequest will reference Account and Inventory to make sure they exist, if they are removed from the system then record delete cascade.

2.4.11 Inventory Histories table

Field

- **Id(Primary key) - Int Identity(1, 1):** Identification of History, it will automatically increase by 1 when inserting.
- **InventoryId(Foreign key) - UniqueIdentifier:** Identify of Inventory, it links to Inventories table.
- **AccountId(Foreign key) - UniqueIdentifier:** Identify of AccountId, it links to the Accounts table.
- **Quantity - Int:** Quantity when making a request for a tool, it must be greater than 0.
- **ActionType - Nvarchar(255):** Type of History, it depends on the request approval status.
- **ActionDate - Datetime2(7):** When inserting a new History, it will automatically get a new time.

Relationship

- Each InventoryHistories will reference Account and Inventory to make sure they exist, if they are removed from the system then record delete cascade.

2.5 Activity Diagram

2.5.1 Login

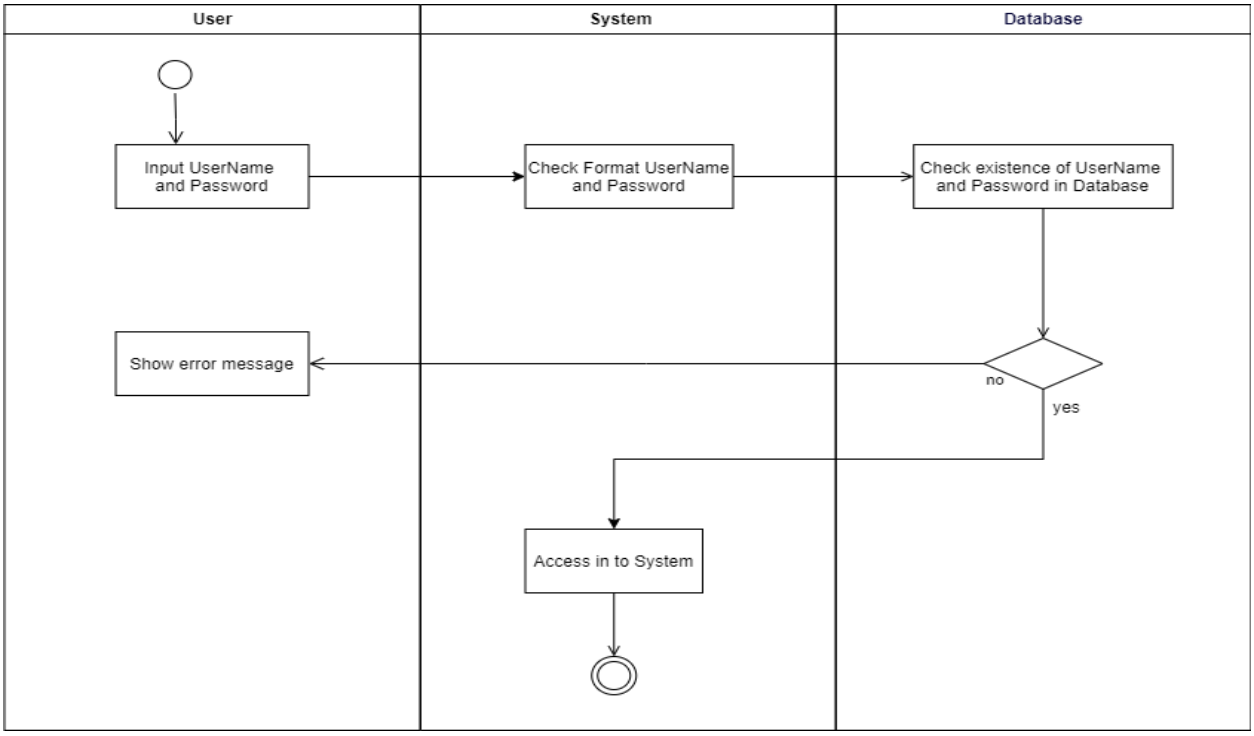

Figure 4 Login

2.5.2 Create Tool

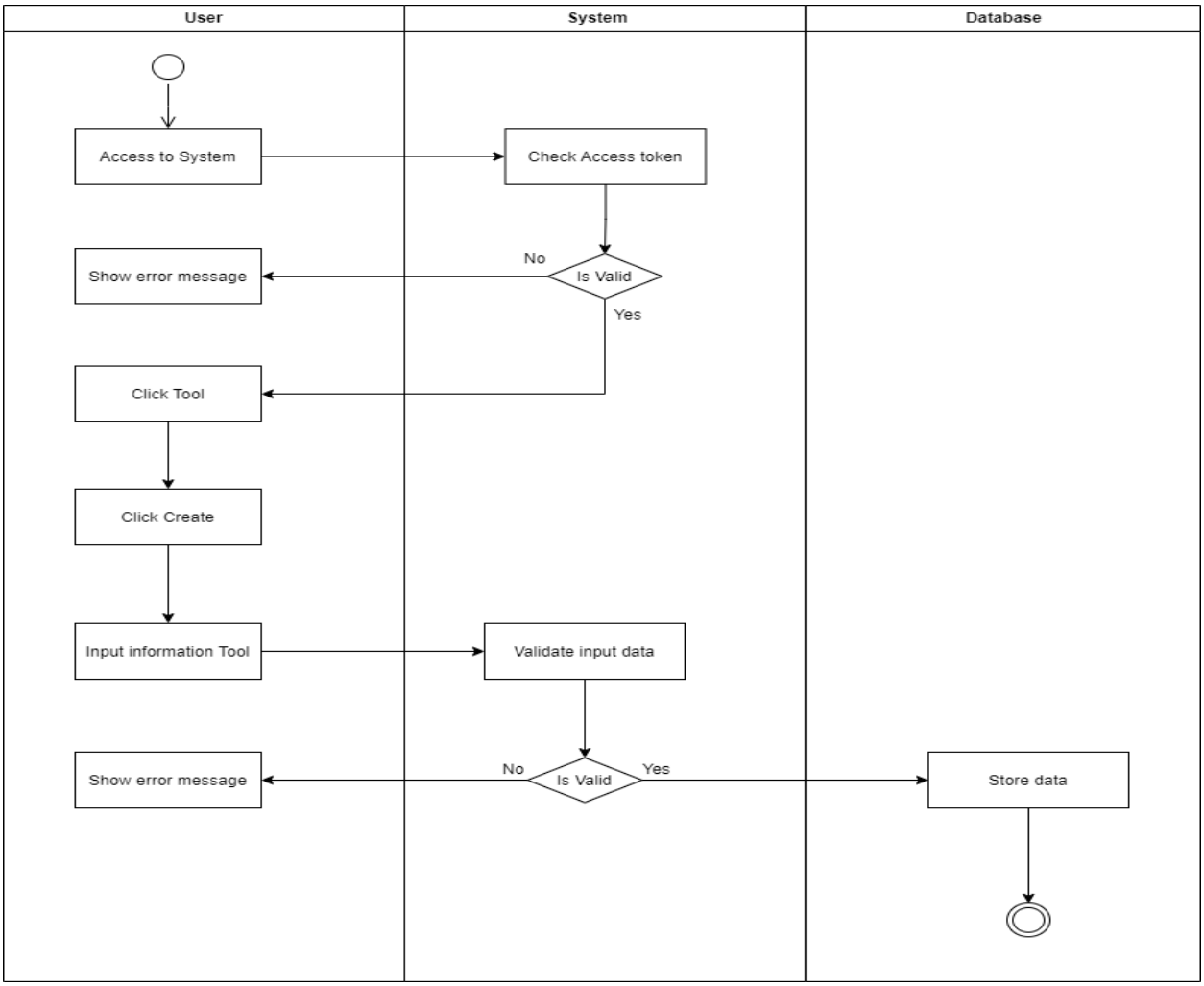

Figure 5 Create Tool

2.5.3 Create Request

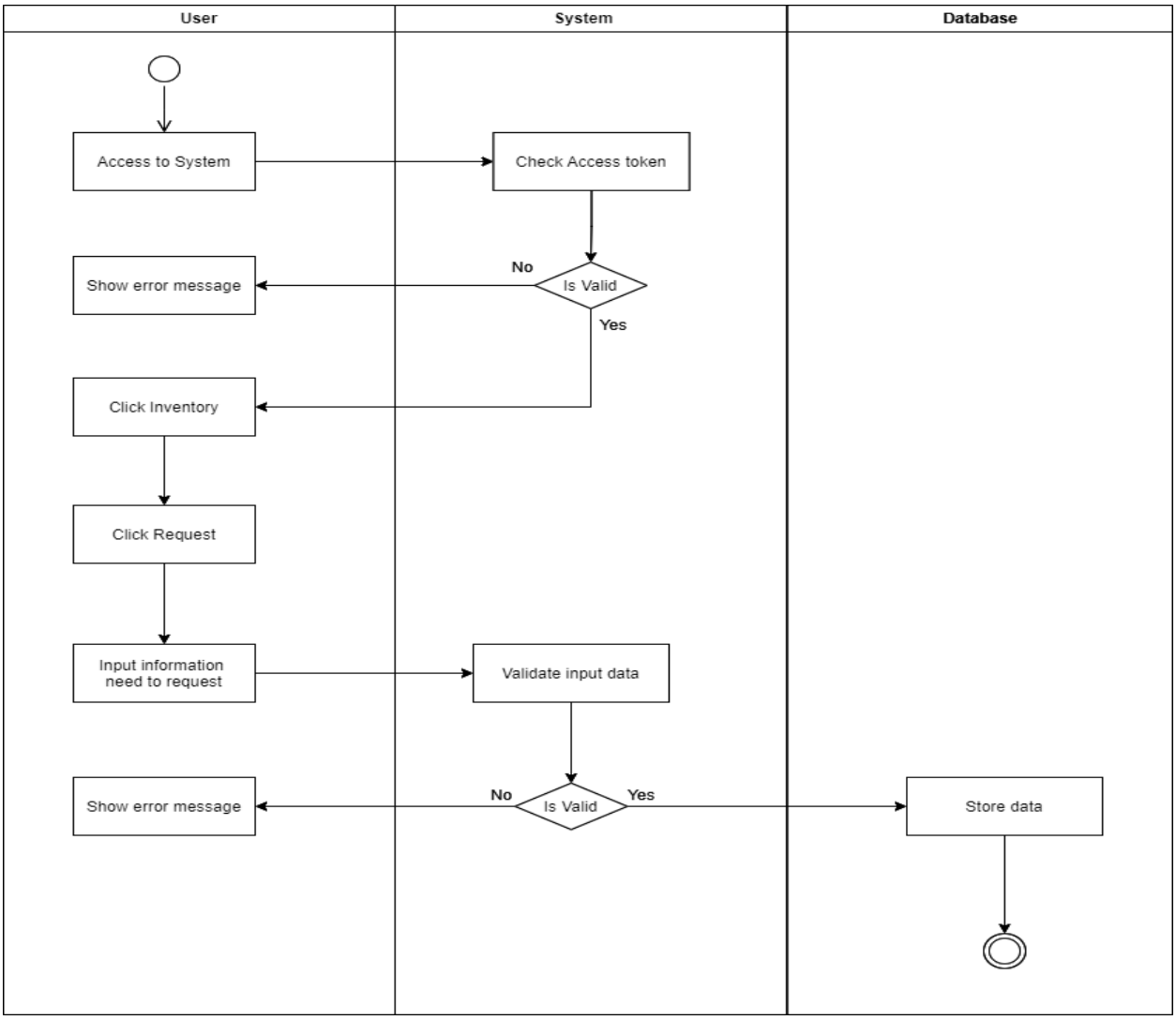

Figure 6 Create Request

2.5.4 Approve Request

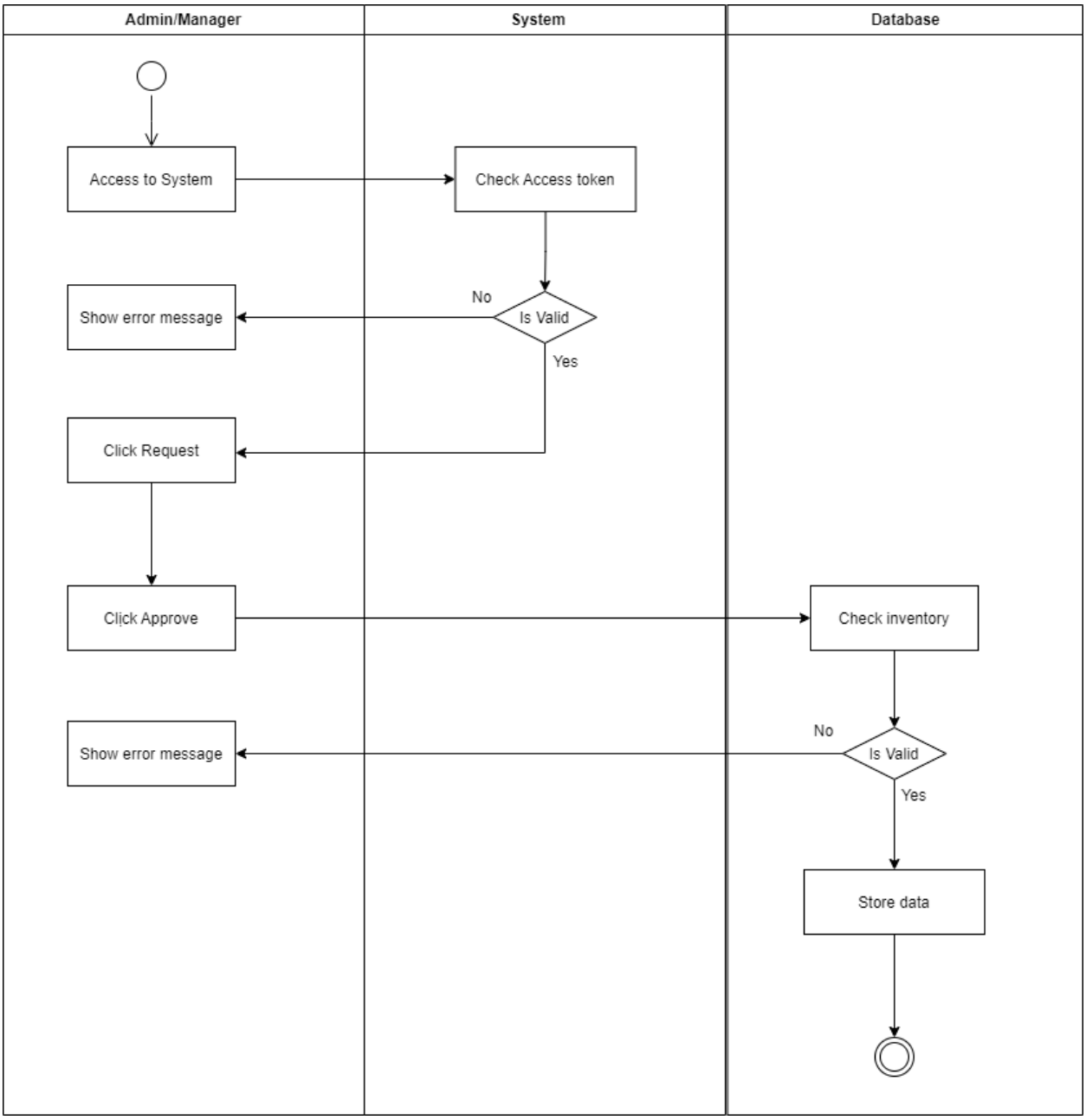

Figure 7 Approve Request

2.5.5 Edit Tool

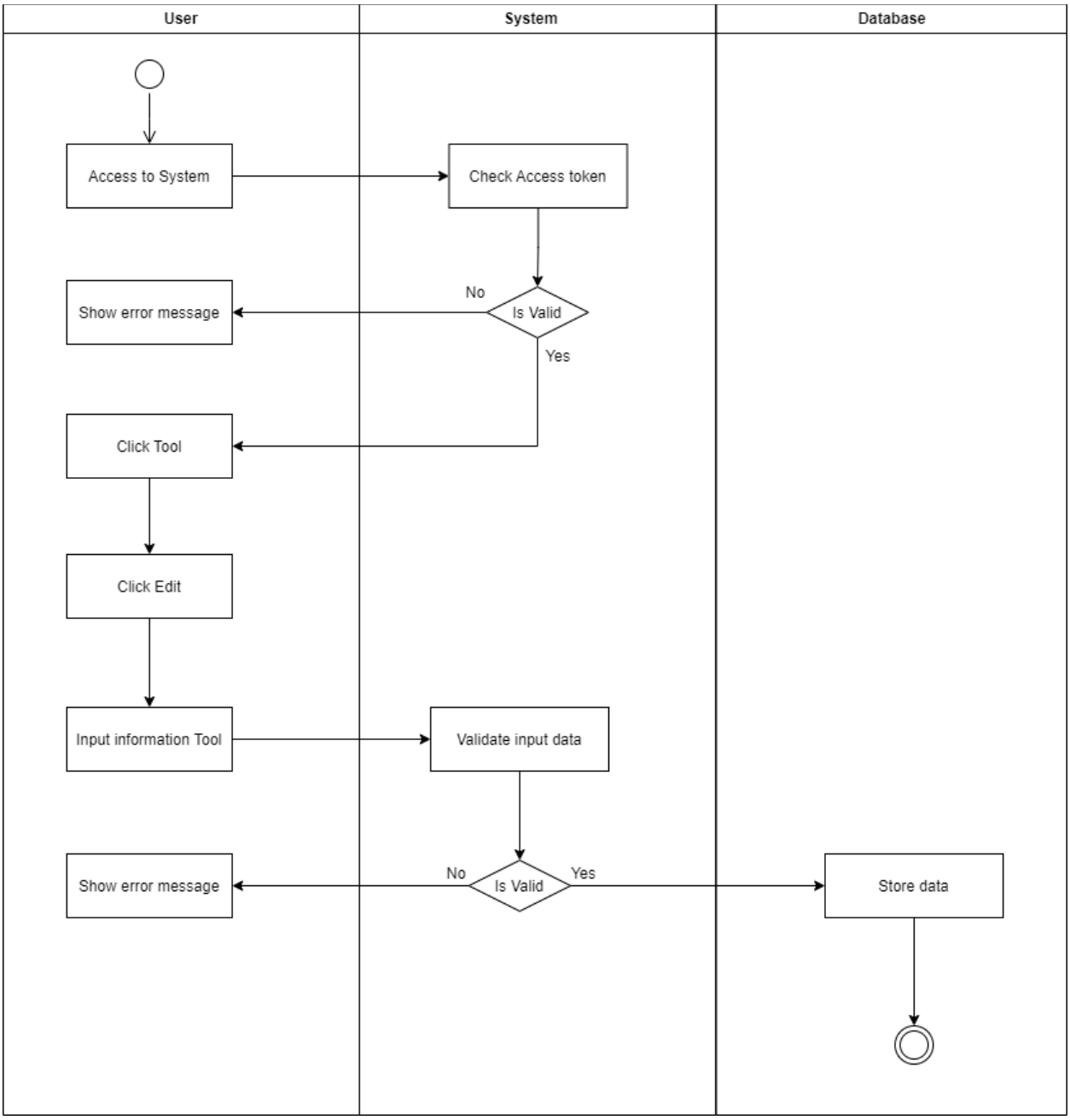

Figure 8 Edit tool

2.6 Sequence Diagram

2.6.1 Login

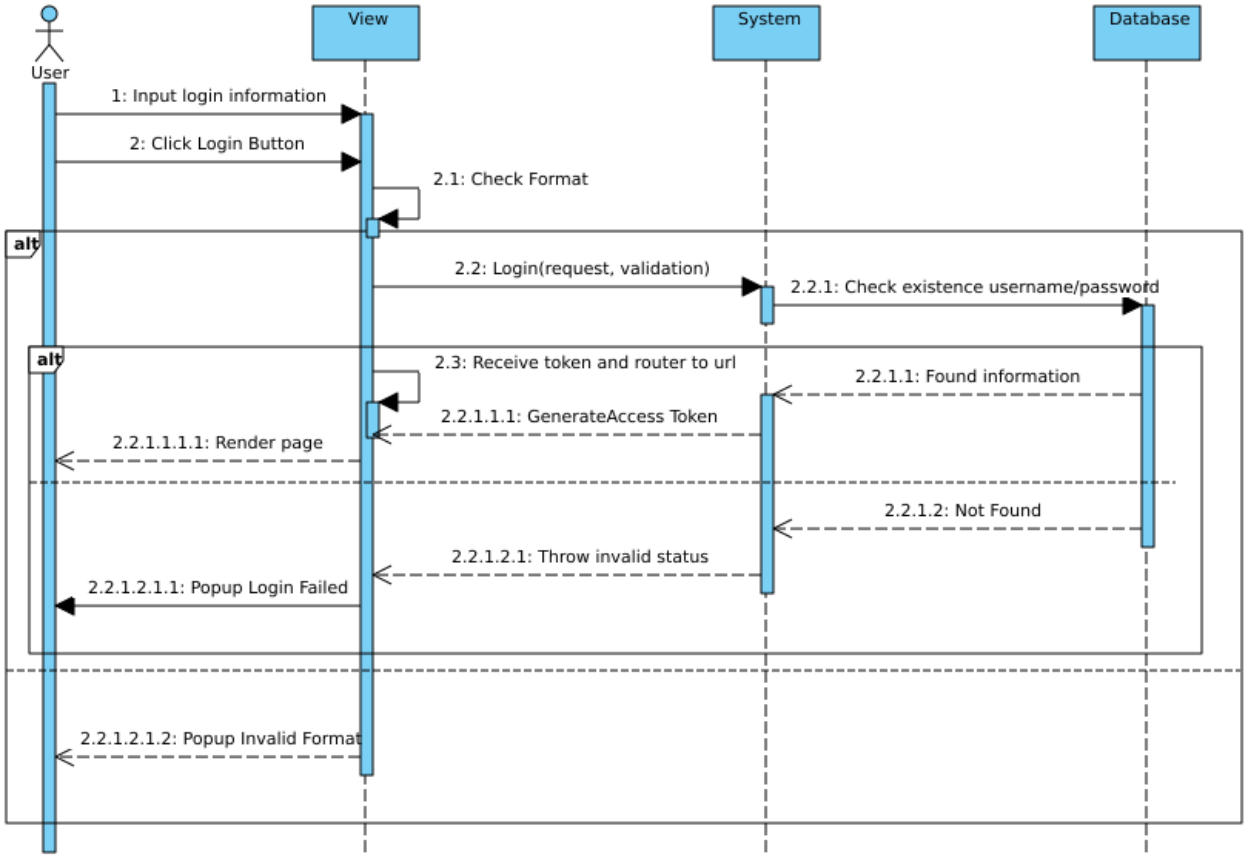

Figure 9 Login

2.6.2 Create Tool

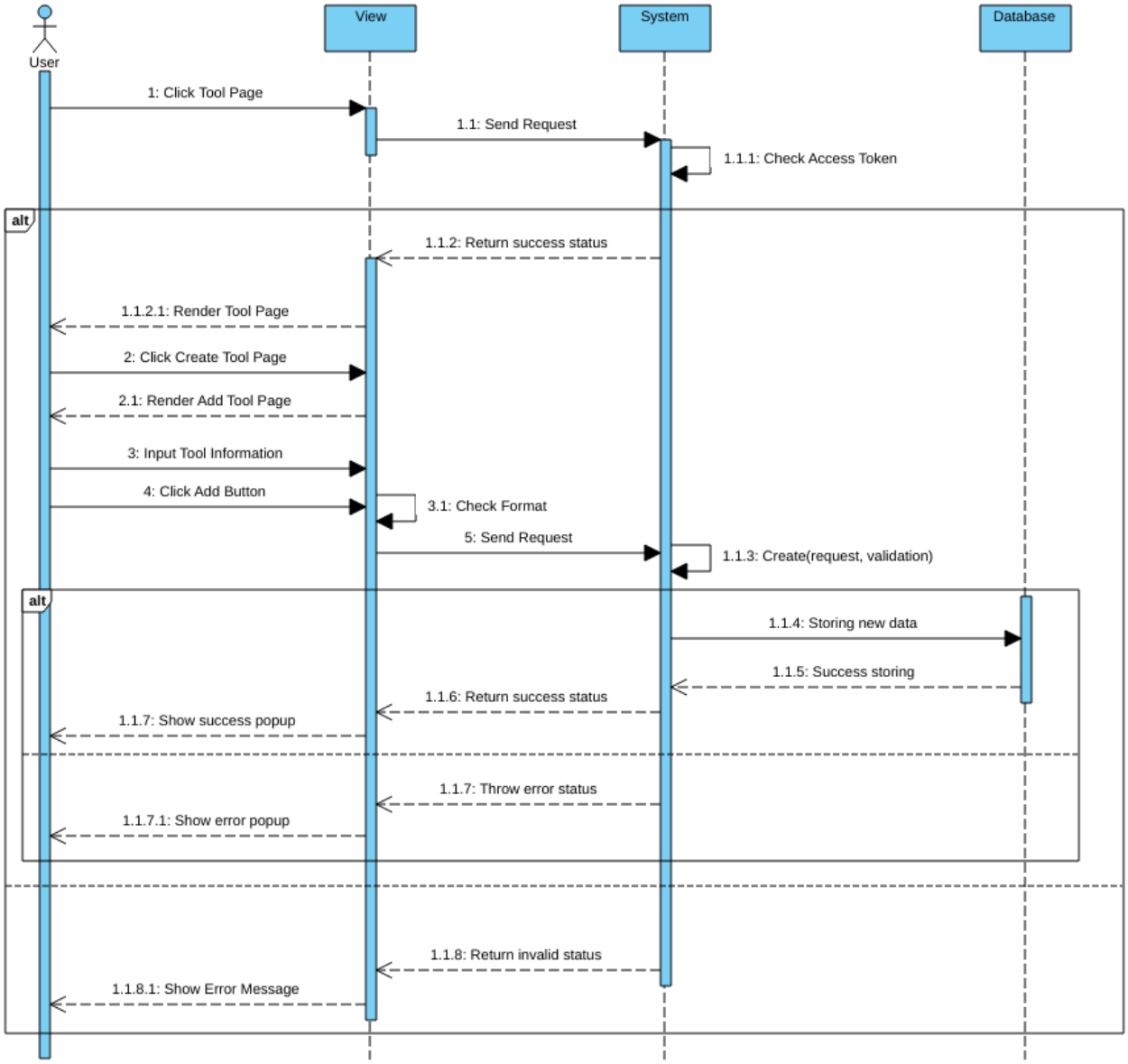

Figure 10 Create tool

2.6.3 Create Request

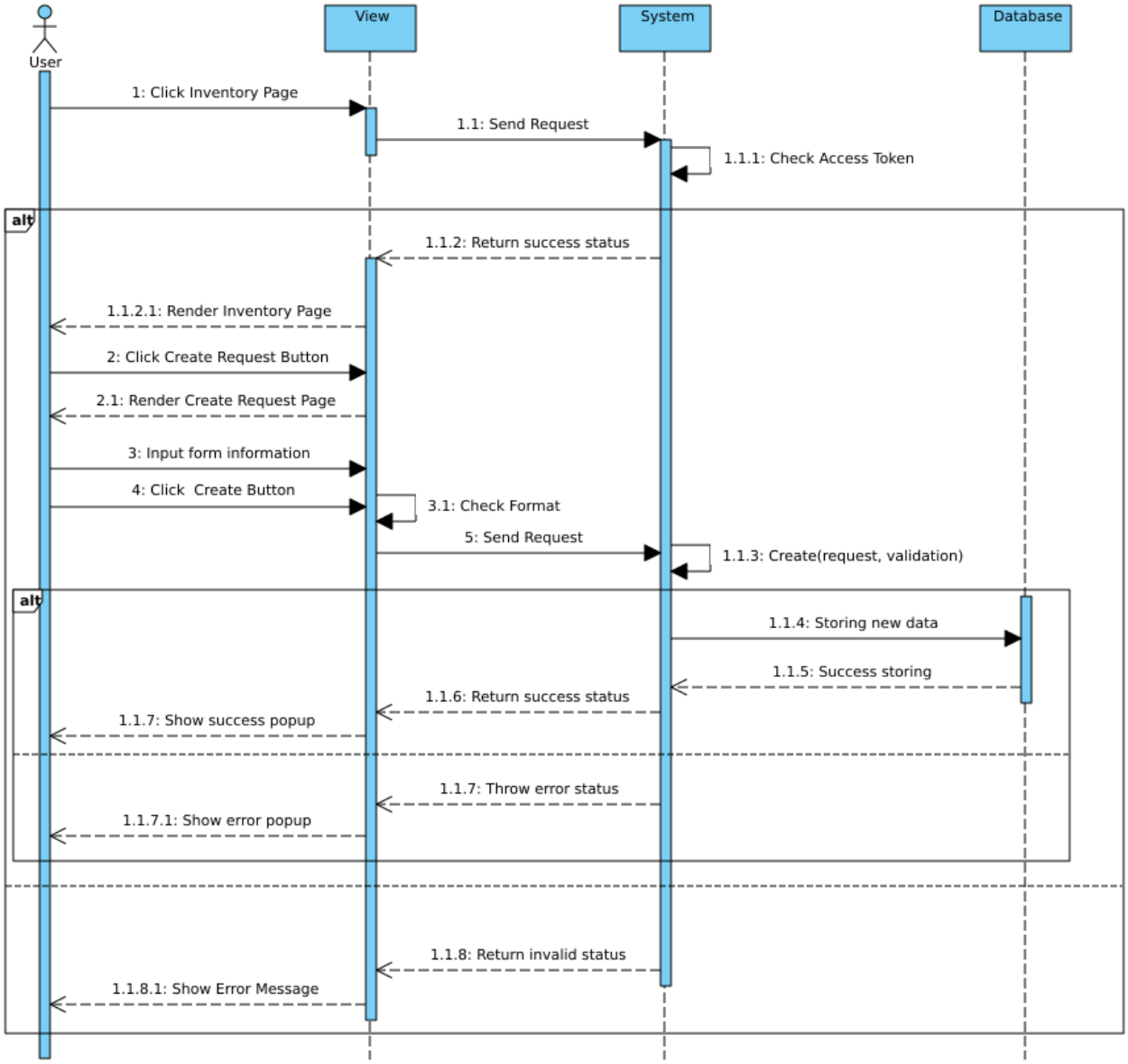

Figure 11 Create Request

2.6.4 Approve Request

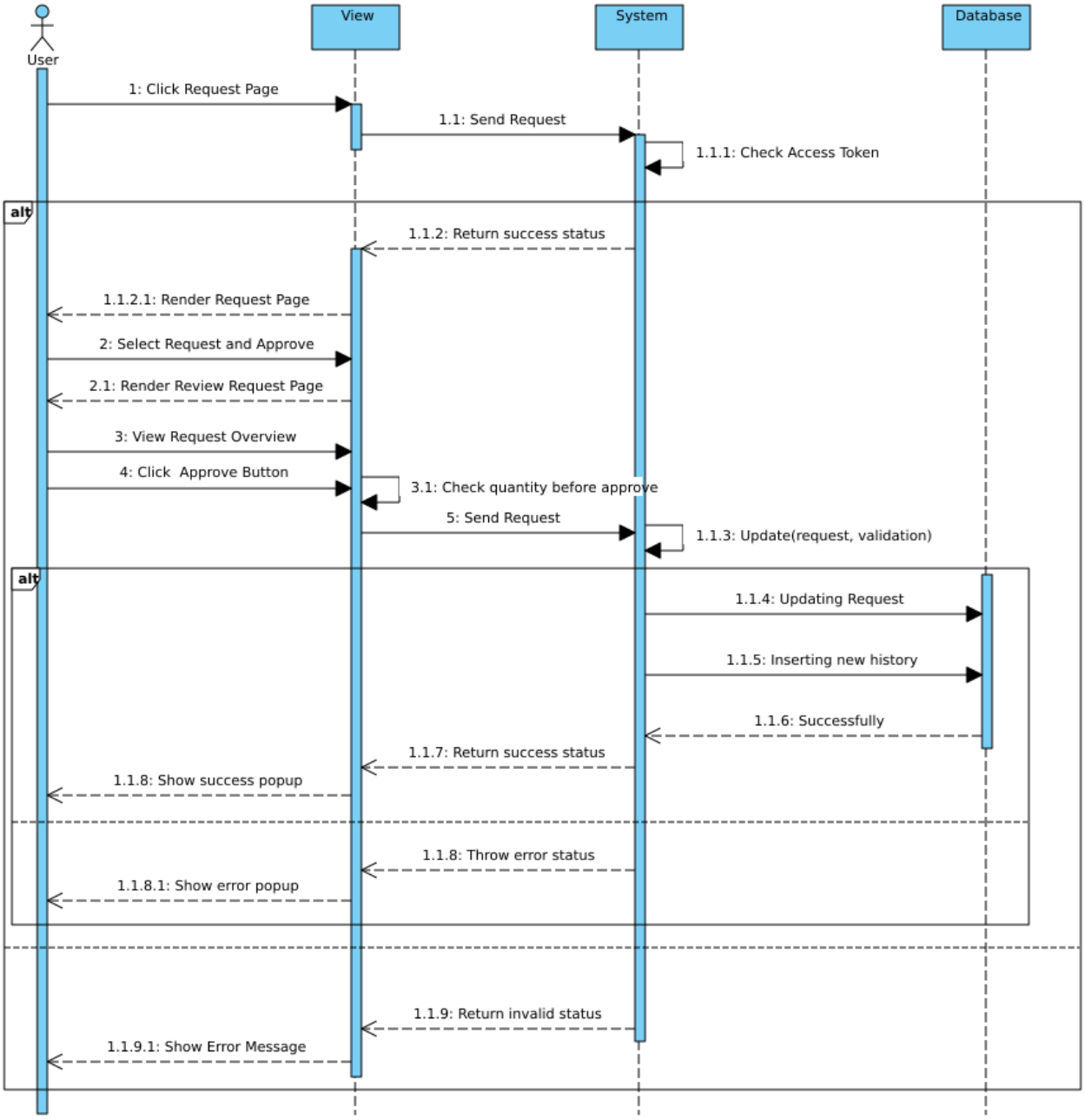

Figure 12 Approve Request

CHAPTER 3. SYSTEM ARCHITECTURE, STRUCTURE AND MAIN FUNCTION

3.1 Architecture

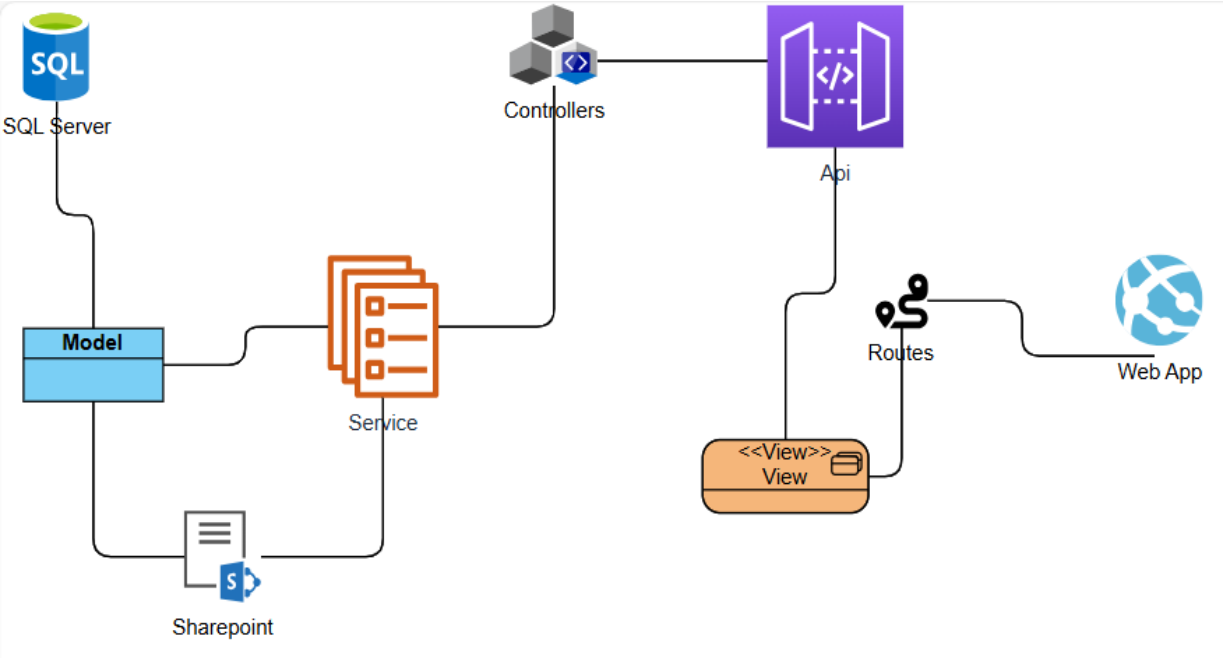

Figure 13 Architecture

3.2 Structure

3.2.1 Frontend

Assets: storing static files such as images, svg, css, scss.

Components: storing components of view such as Dashboard, Navbar, UserProfile, …

Router: mapping url in the system

Store: managing vue state

Views: Storing page in each router.

3.2.2 Backend

TeachEquipManagement.WebAPI:

Logs: folder save all log of serilog for console system.

Controllers: Presentation of backend, it will render api to make an action in an app with a database.

MigrationInitial.cs: DataContext Helper to require for program.cs to automatically seed data to database.

TeachEquipManagement.BLL:

Services: Concrete class implements all IService.

IServices: All interfaces for business layer.

ManageServices: Apply Unit of work pattern to manage all services.

FluentValidator: Configuration files validate data input for request in api.

BusinessModels: Define all Request Dtos and Response Dtos for system, it also contains Enums and Common files such as Api Response.

AutoMapperProfile: Configuration file setting of automapper using for defining mapping between Dtos file and Class Model.

TeachEquipManagement.DAL:

EFContext: Storing DataContext of Entity framework core, defining all class models and their relationship, config attributes.

IGenericRepository: Define common interface using generic for all class models.

GenericRepository: Implement Generic Repository used for all class models.

IRepositories: All interfaces query model.

Repositories: Implementation of all interfaces query.

Migrations: Storing all versions of database systems.

Models: Folder contains all class models for backend.

SeedData: Folder containing all configuration files of entity framework core using seed data.

UnitOfWorks: Configuration of managing all repository and class models in the system.

TeachEquipManagement.Utilities:

Helper: Storing all helpers functions such as HashPassword, RestSharp for working with api, extension for enum to get description attribute value.

CommonModels: Storing all models frequently used in an app.

3.3 Main function

3.3.1 Create Request

Step	Operation
1	- Login into the system.
2	- Click on Inventory page, choose tool item then click on button request.
3	- Move to the request page select type of request and input quantity
4	- Click to Create Request it will store at Request page

Table 18 Create Request Function

3.3.2 Update Request

Step	Operation
1	- Login into the system with an admin/manager account.
2	- Click on the Request page, choose the request you want to approve then click the approve button.
3	- Move to the approve page, overview request before confirm
4	- Click to confirm the button will process in the system.

Table 19 Update Request Function

3.3.3 Manage

Step	Operation
1	- Login into the system with an admin/ manager account.
2	- Click on any page in the system.
3	- Enable to Create/Edit/Delete/View Data in all pages. With a Manager can easily reach all pages but at account page it just allows operations with the user.
4	If you want to view data then click on any page in the sidebar, choose the create button to move to add page, at view list data we can select item and click edit move to edit page or remove button to remove page.

Table 20 Manage Function

CHAPTER 4. USER INTERFACE

4.1 Login

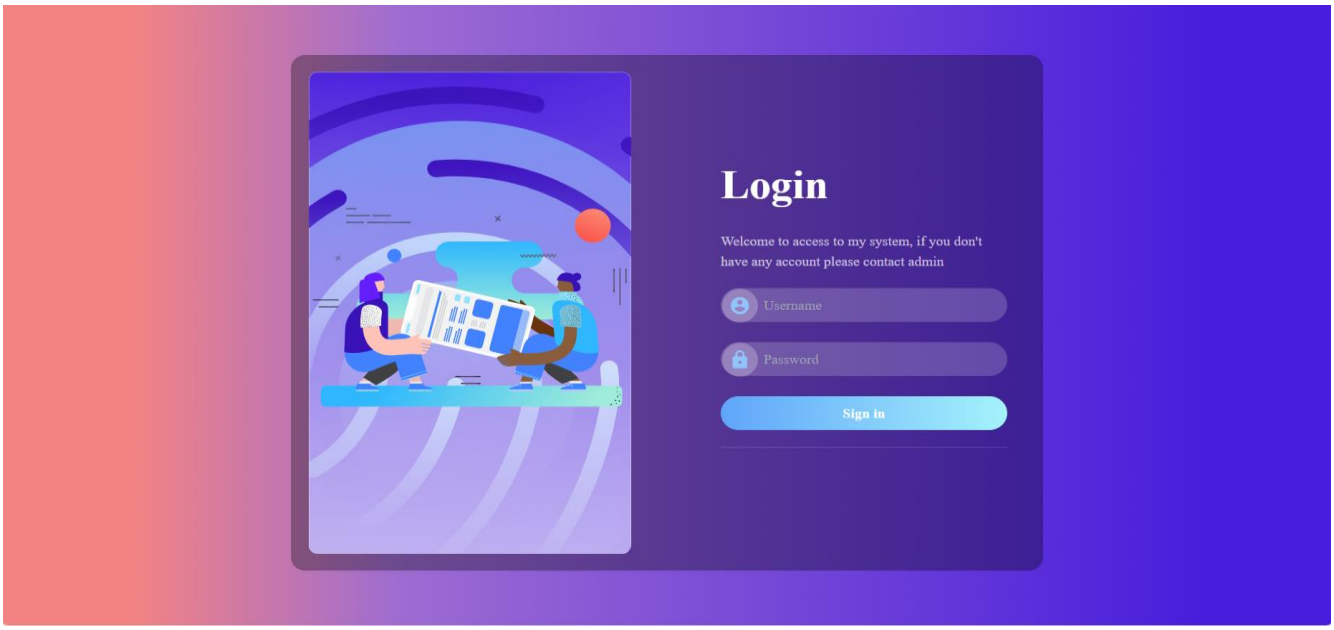

Figure 14 Login

4.2 Dashboard/Home

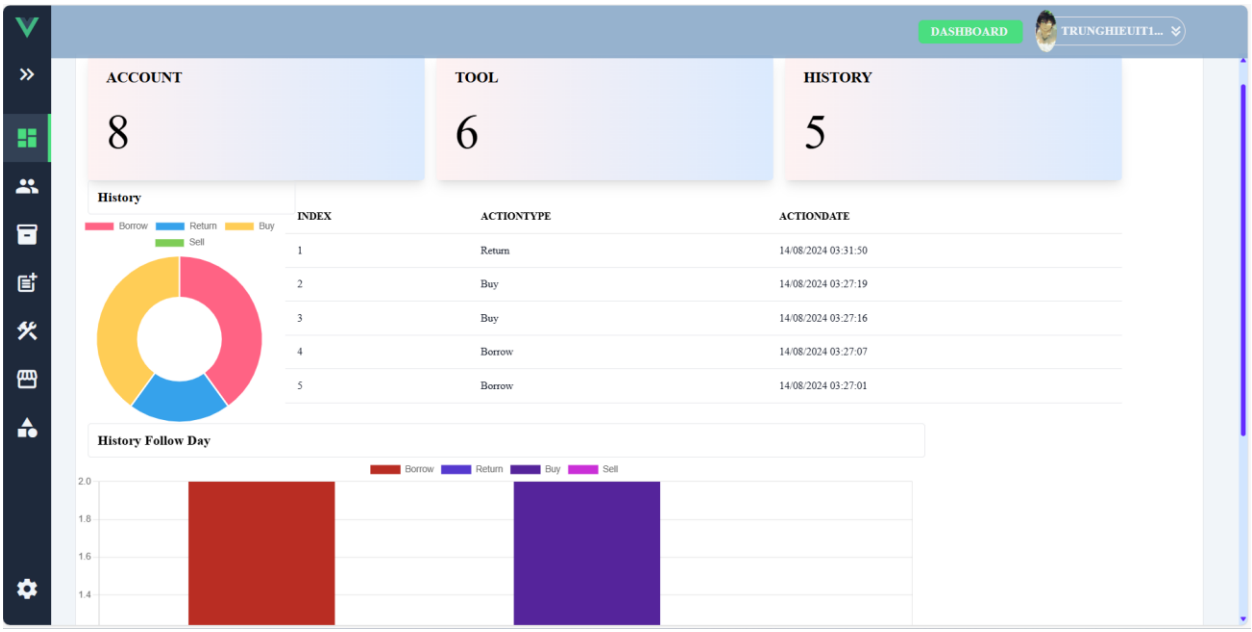

Figure 15. Dashboard/Home

4.3 Account

4.3.1 View

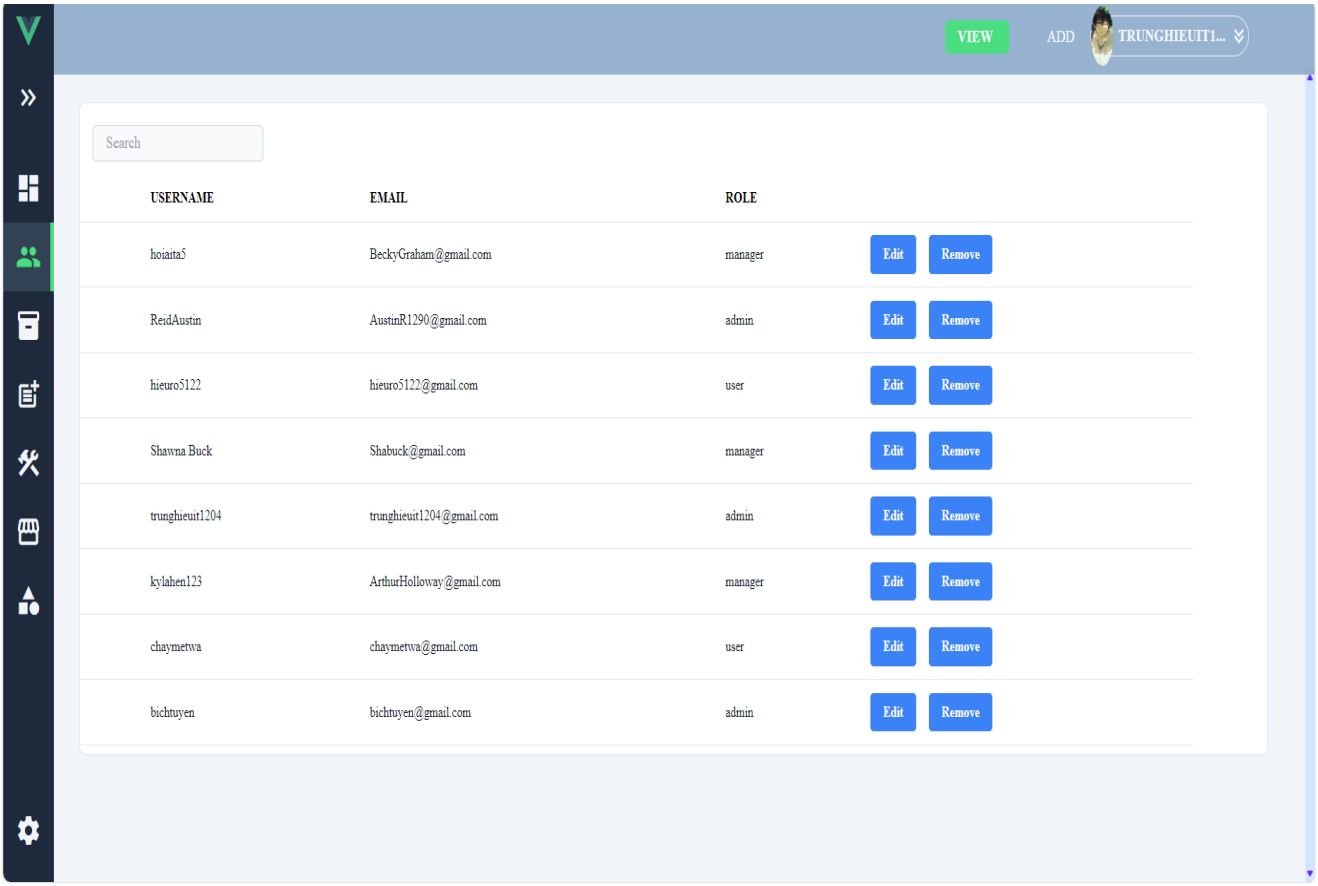

Figure 16 View

4.3.2 Create

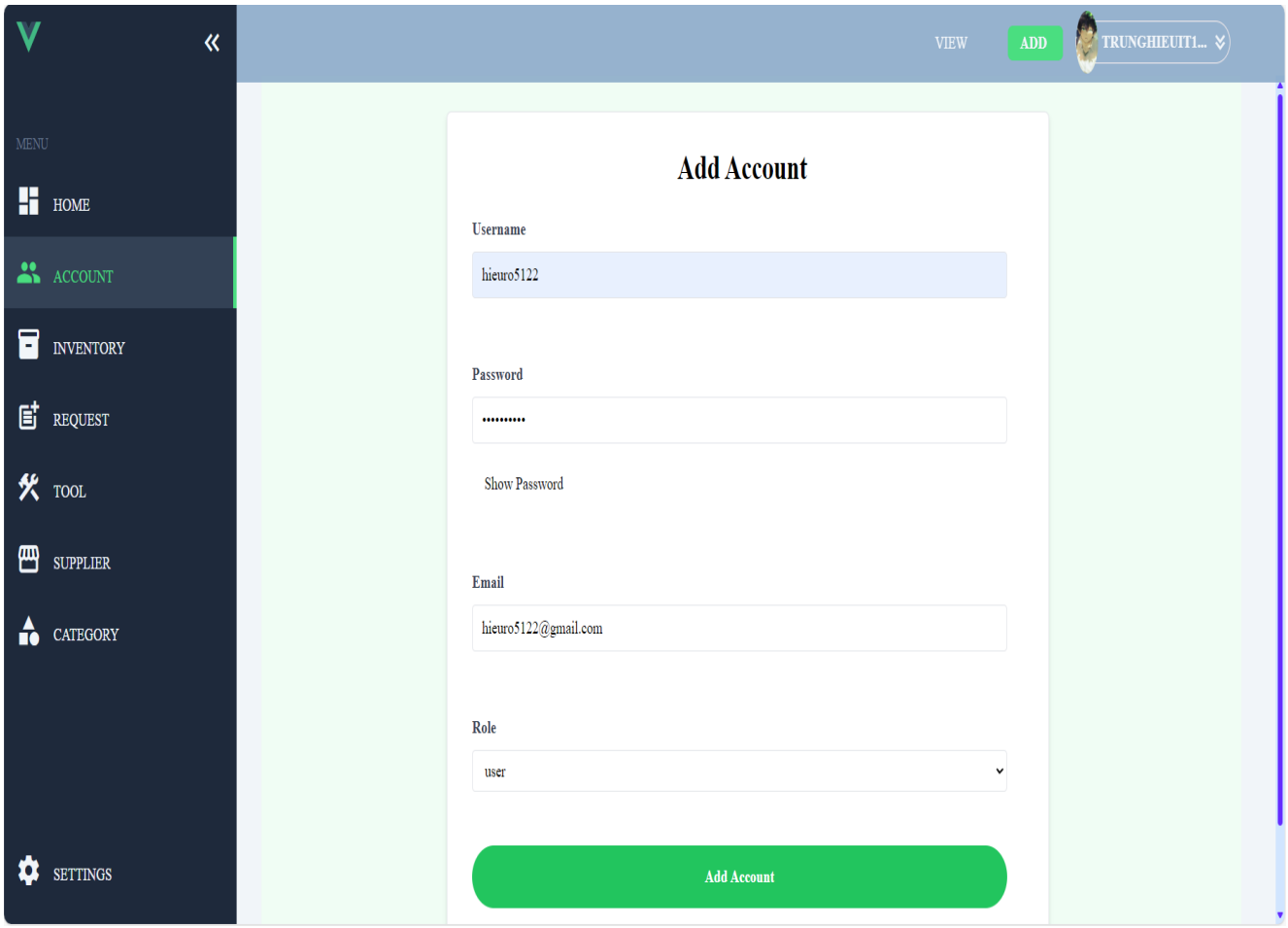

Figure 17 Create

4.3.3 Edit

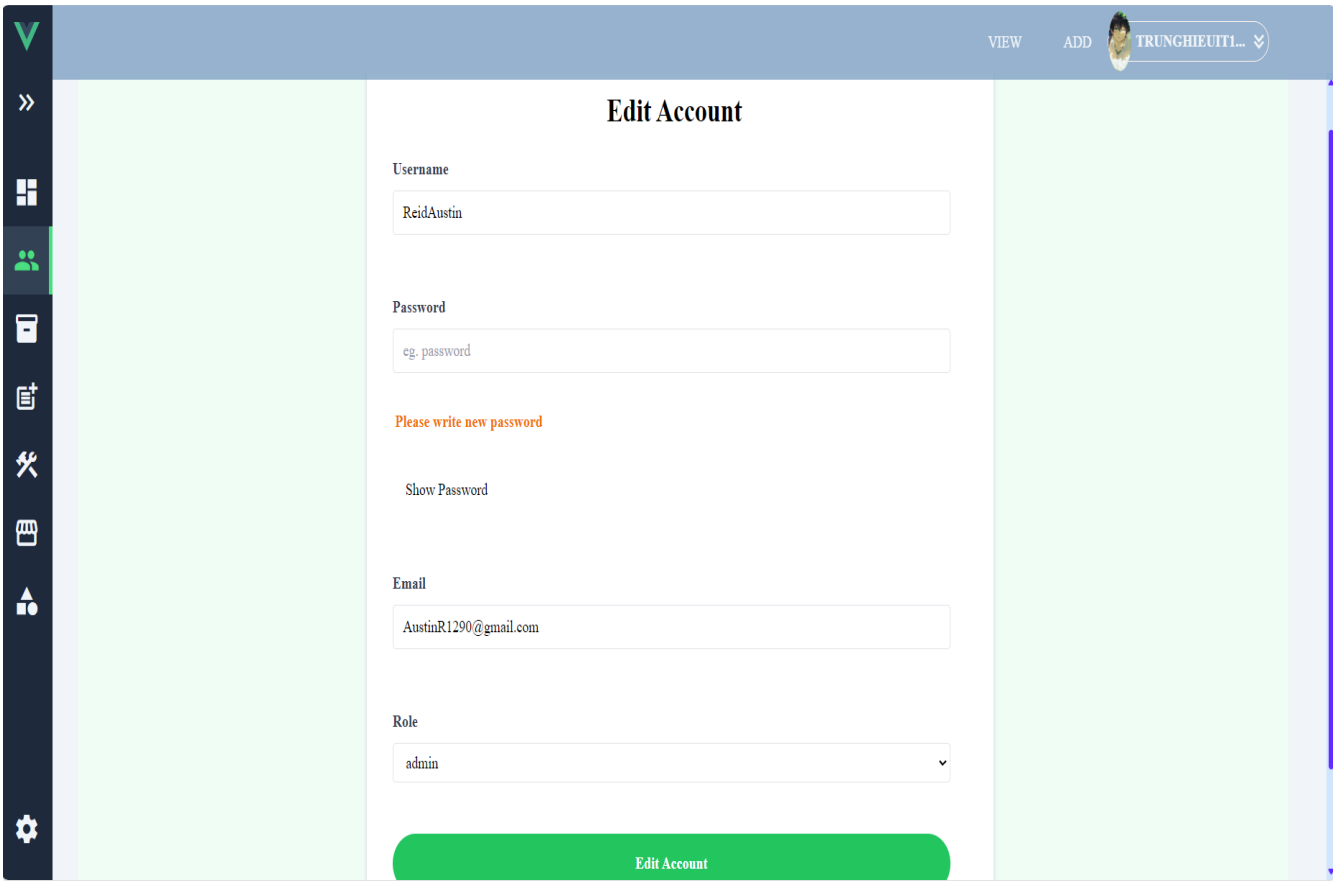

Figure 18 Edit

4.4 Inventory

4.4.1 View

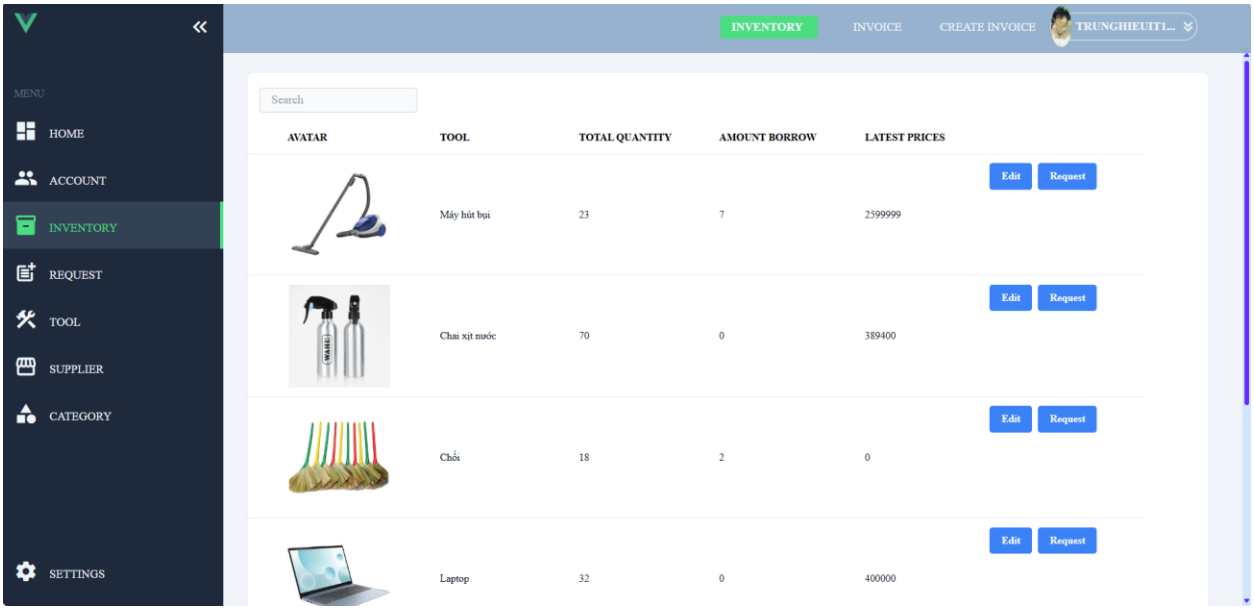

Figure 19 View

4.4.2 Edit

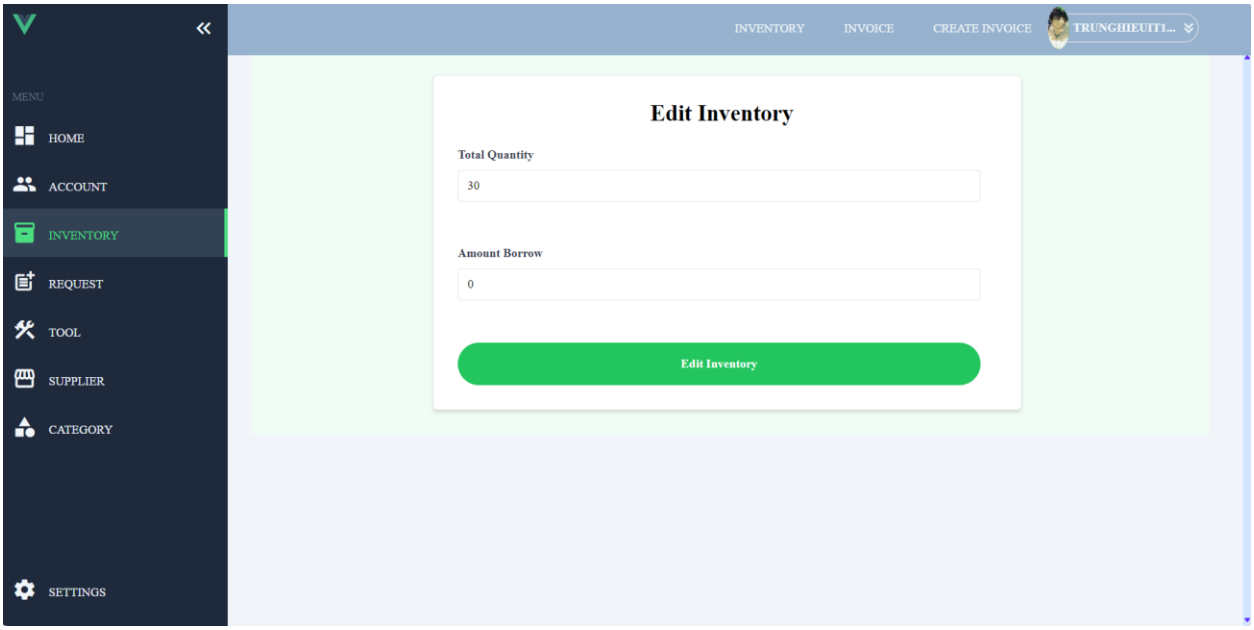

Figure 20 Edit

4.4.3 Create Request

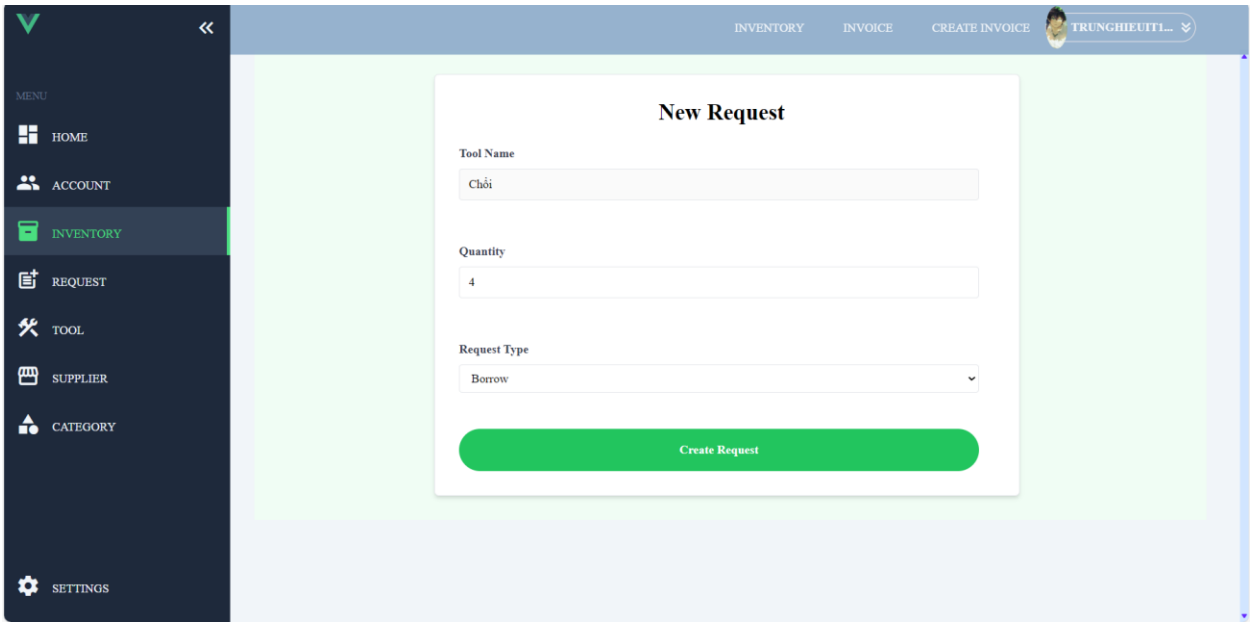

Figure 21 Create Request

4.4.4 Create Invoice

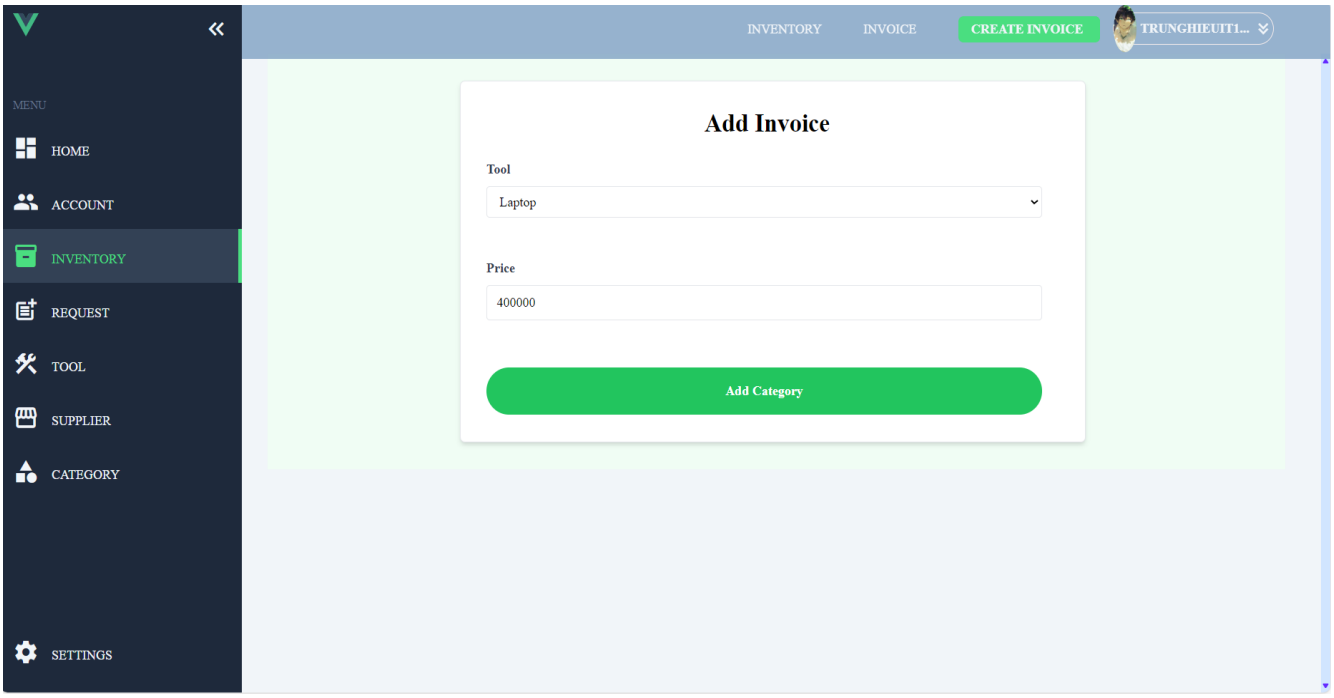

Figure 22 Create Invoice

4.4.5 View Invoices

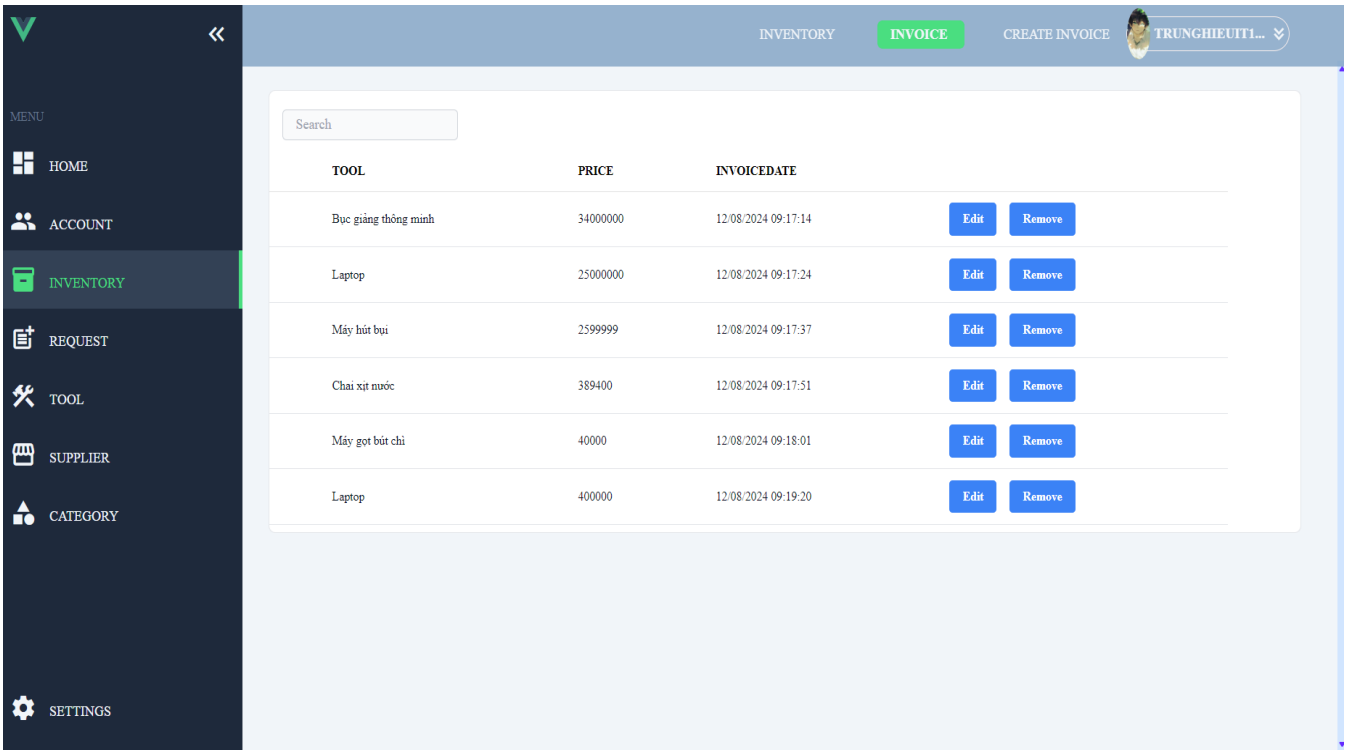

Figure 23 View Invoices

4.5 Request

4.5.1 View

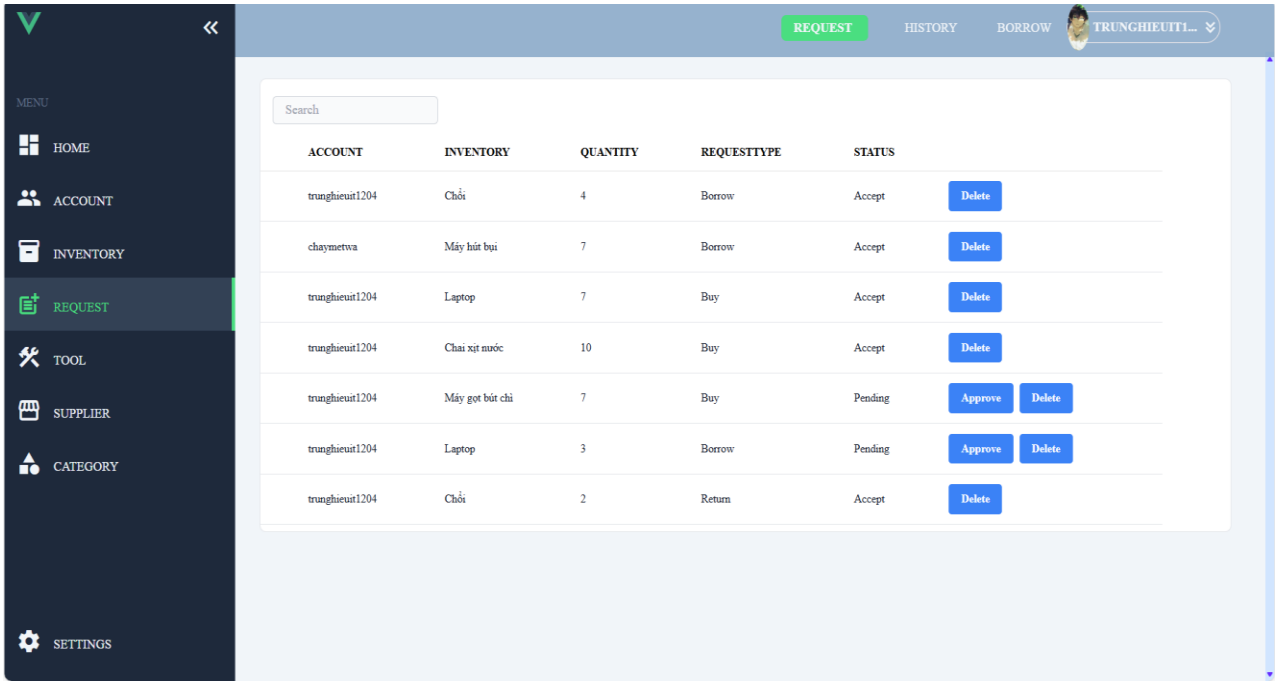

Figure 24 View

4.5.2 History

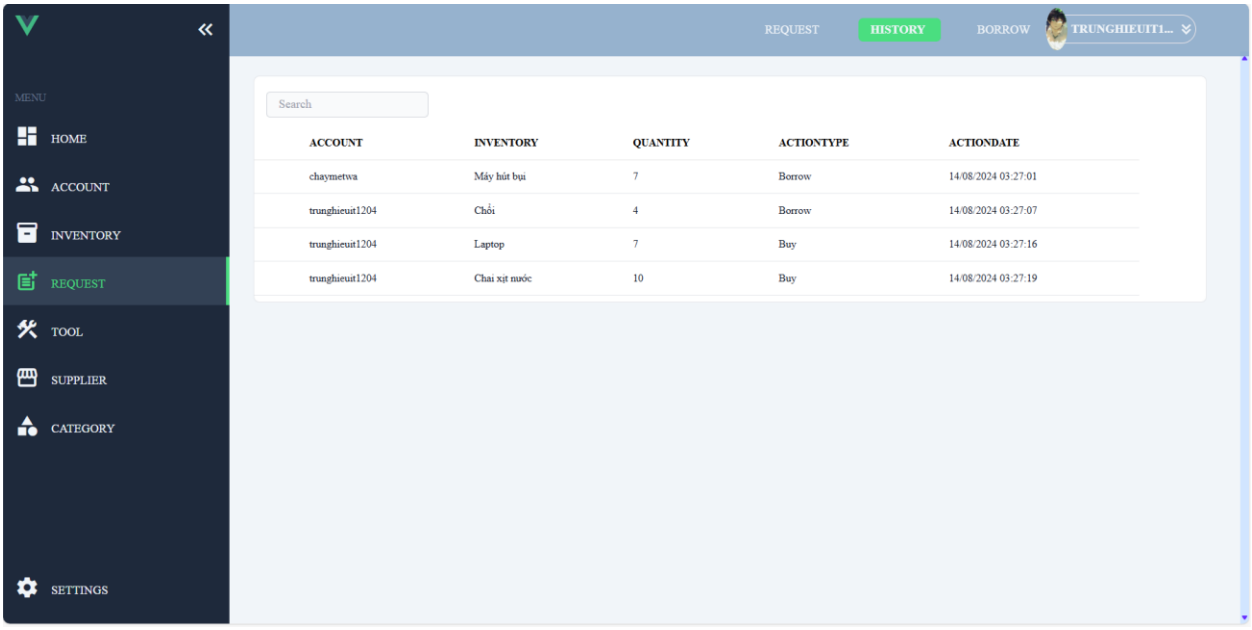

Figure 25 History

4.5.3 Borrow

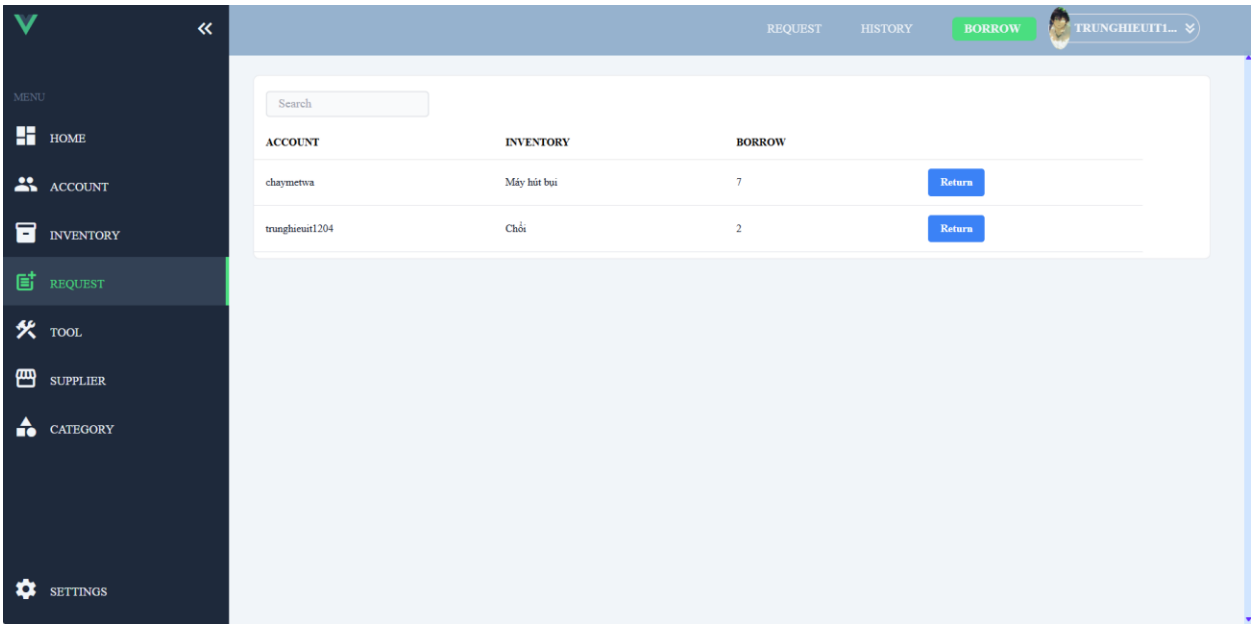

Figure 26. Borrow

4.5.4 Create

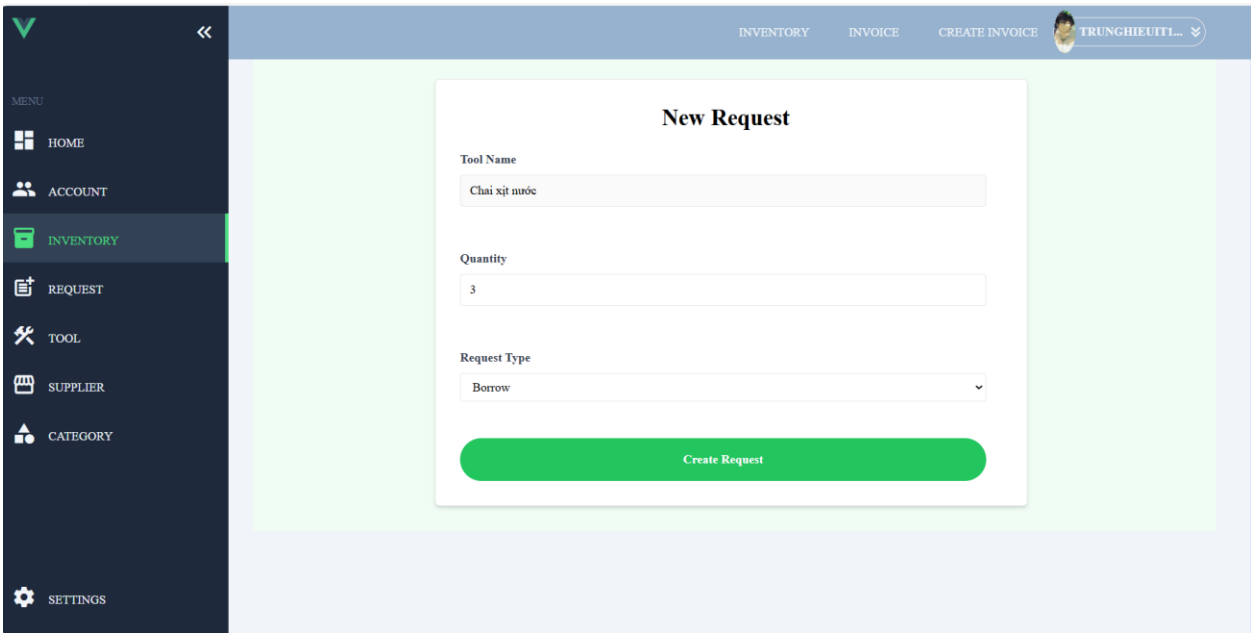

Figure 27 Create

4.5.5 Approve

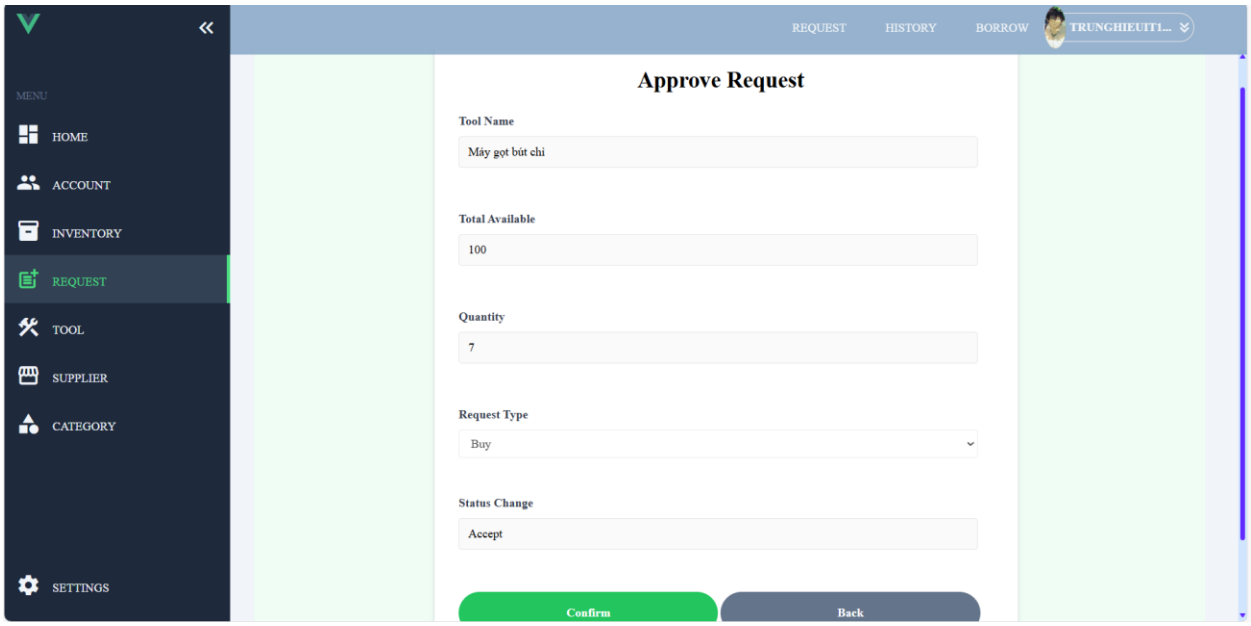

Figure 28 Approve

4.6 Tool

4.6.1 View

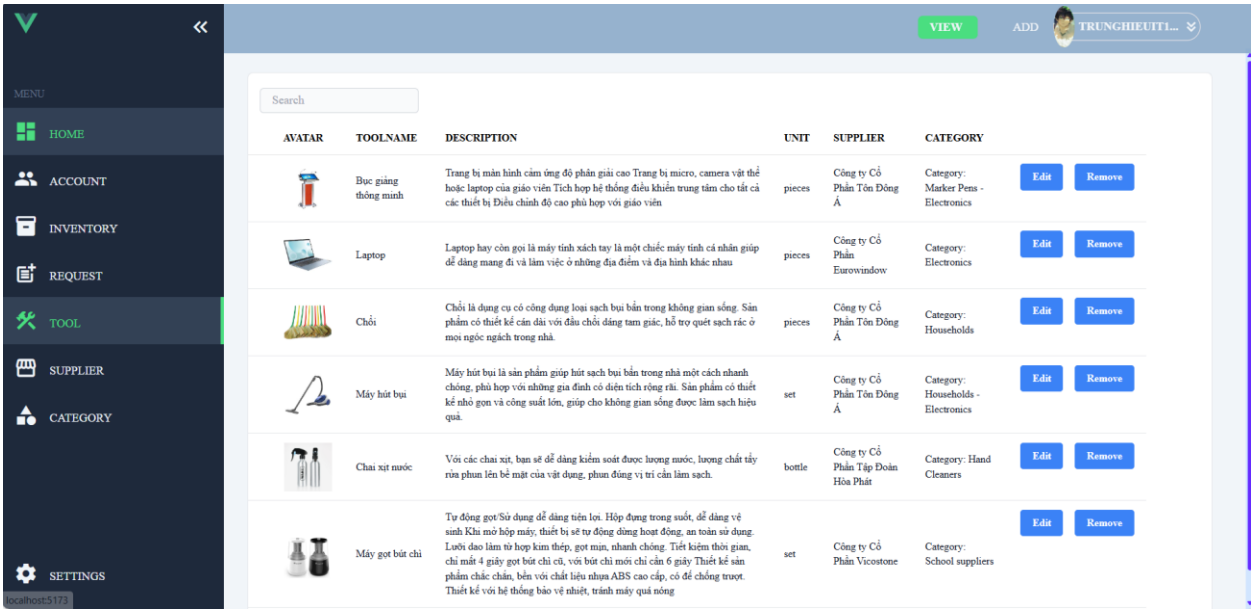

Figure 29 View

4.6.2 Create

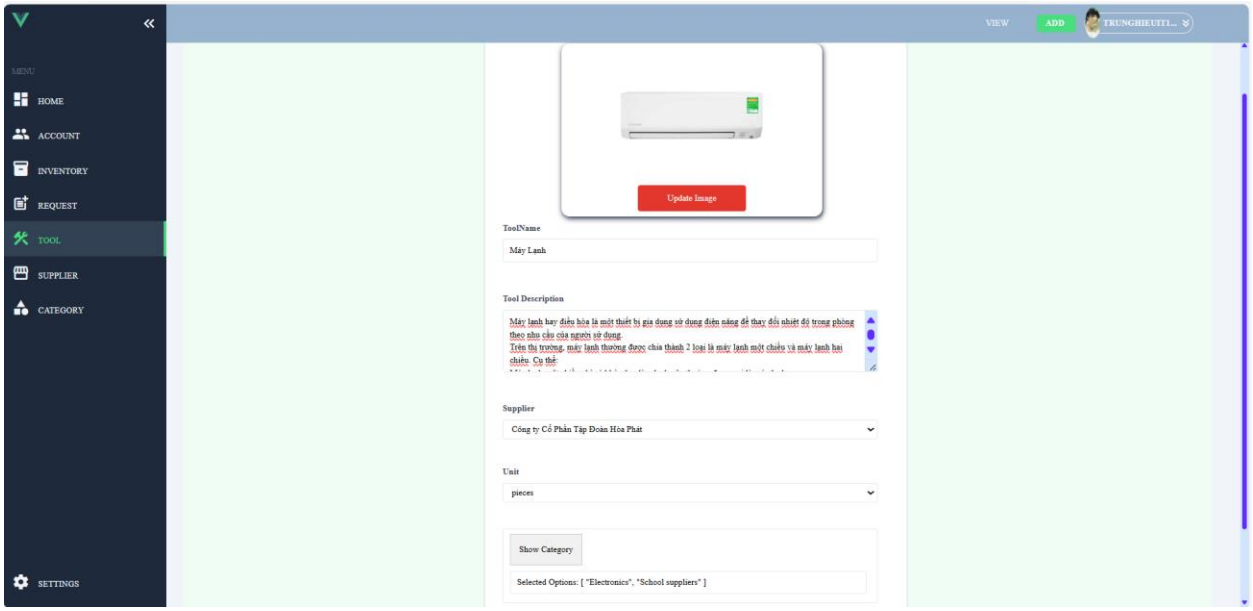

Figure 30 Create

4.6.3 Edit

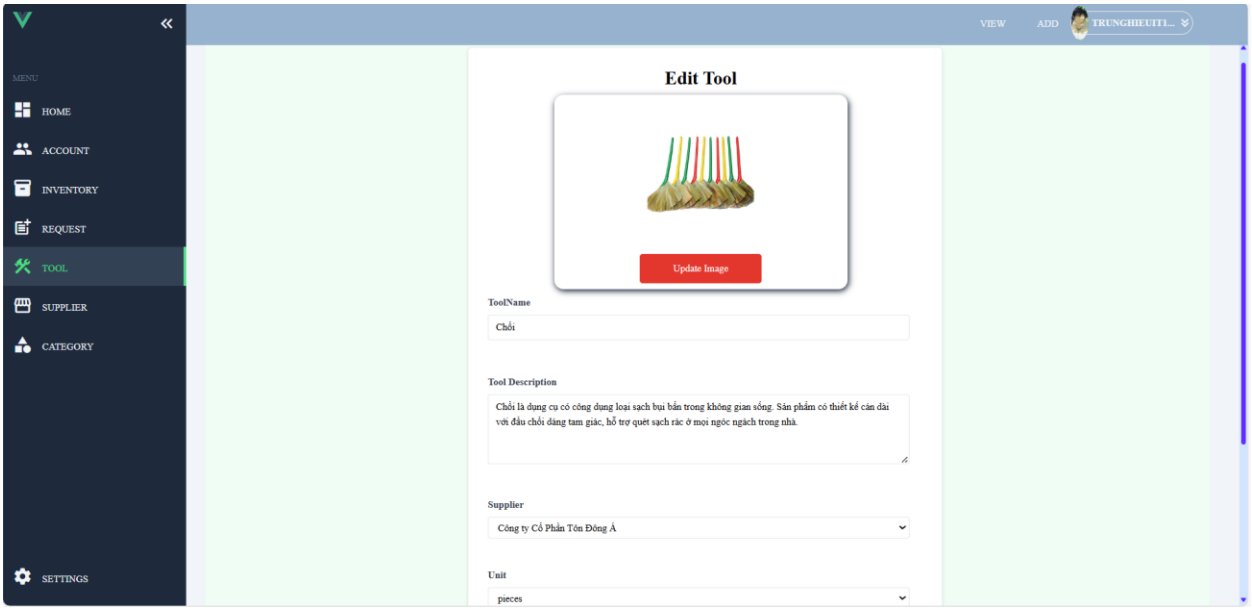

Figure 31 Edit

4.7 Supplier

4.7.1 View

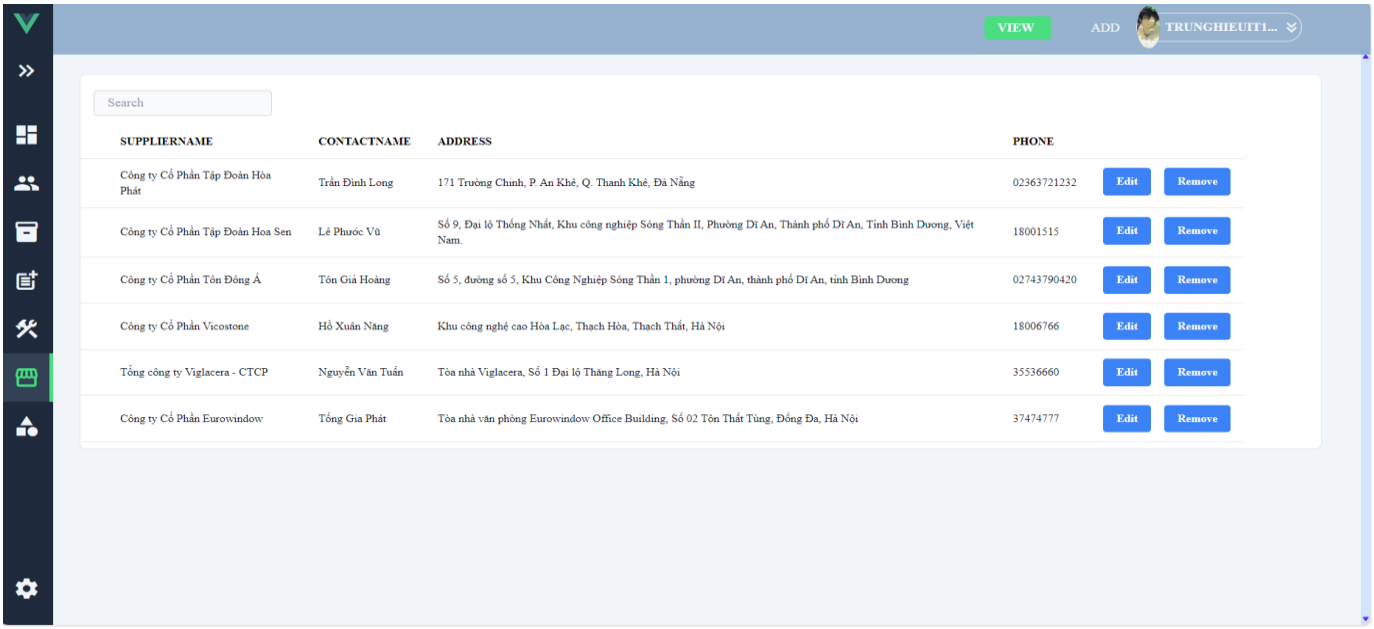

Figure 32 View

4.7.2 Create

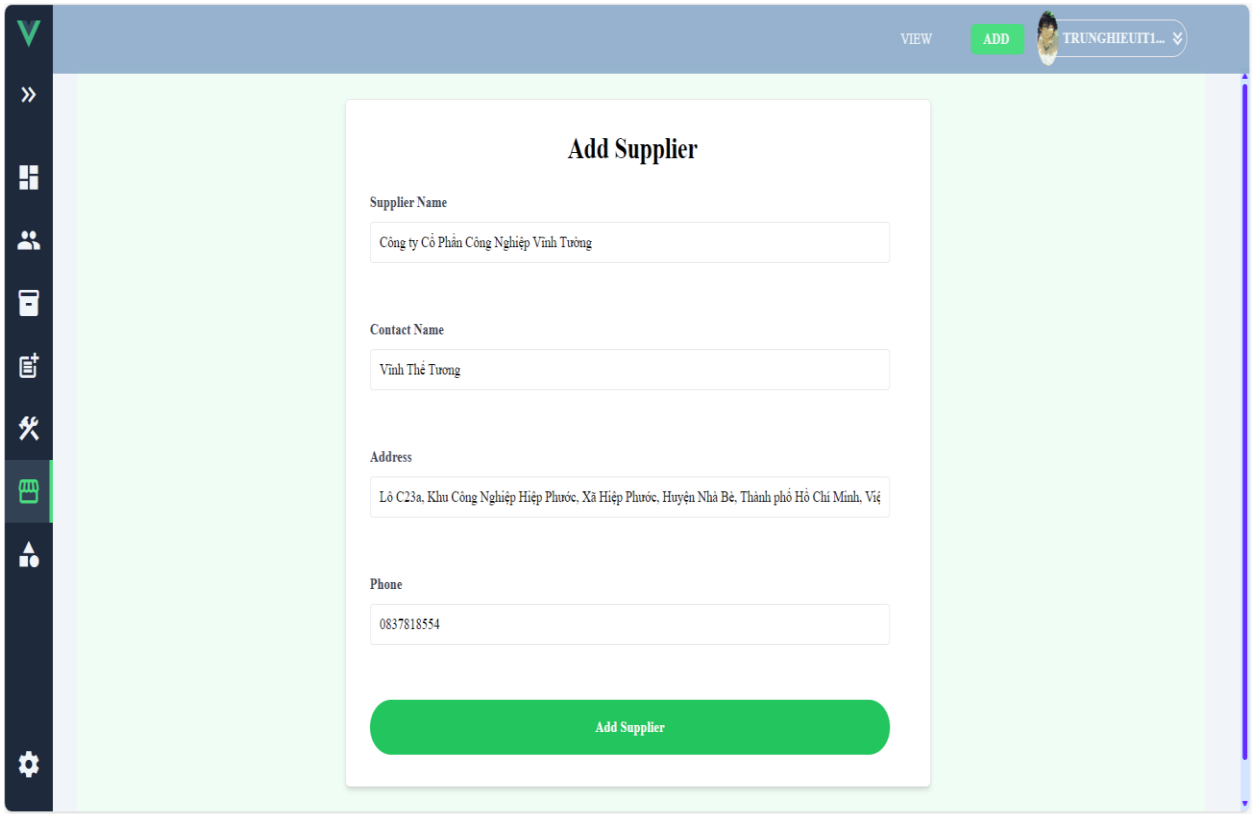

Figure 33 Create

4.7.3 Edit

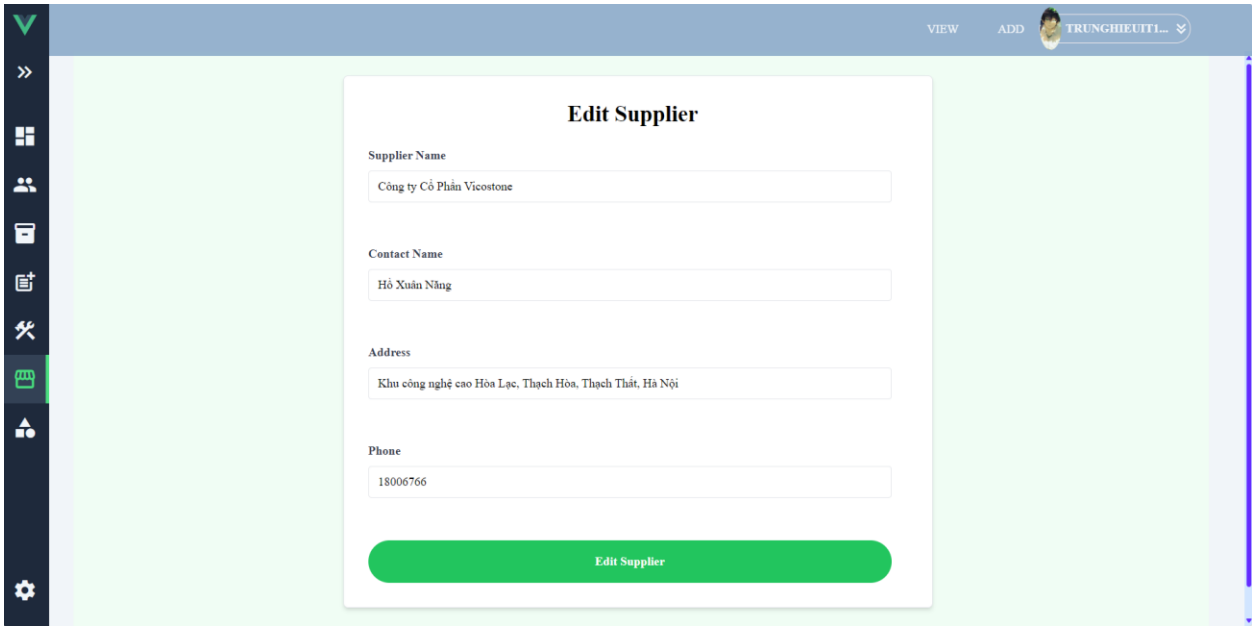

Figure 34 Edit

4.8 Category

4.8.1 View

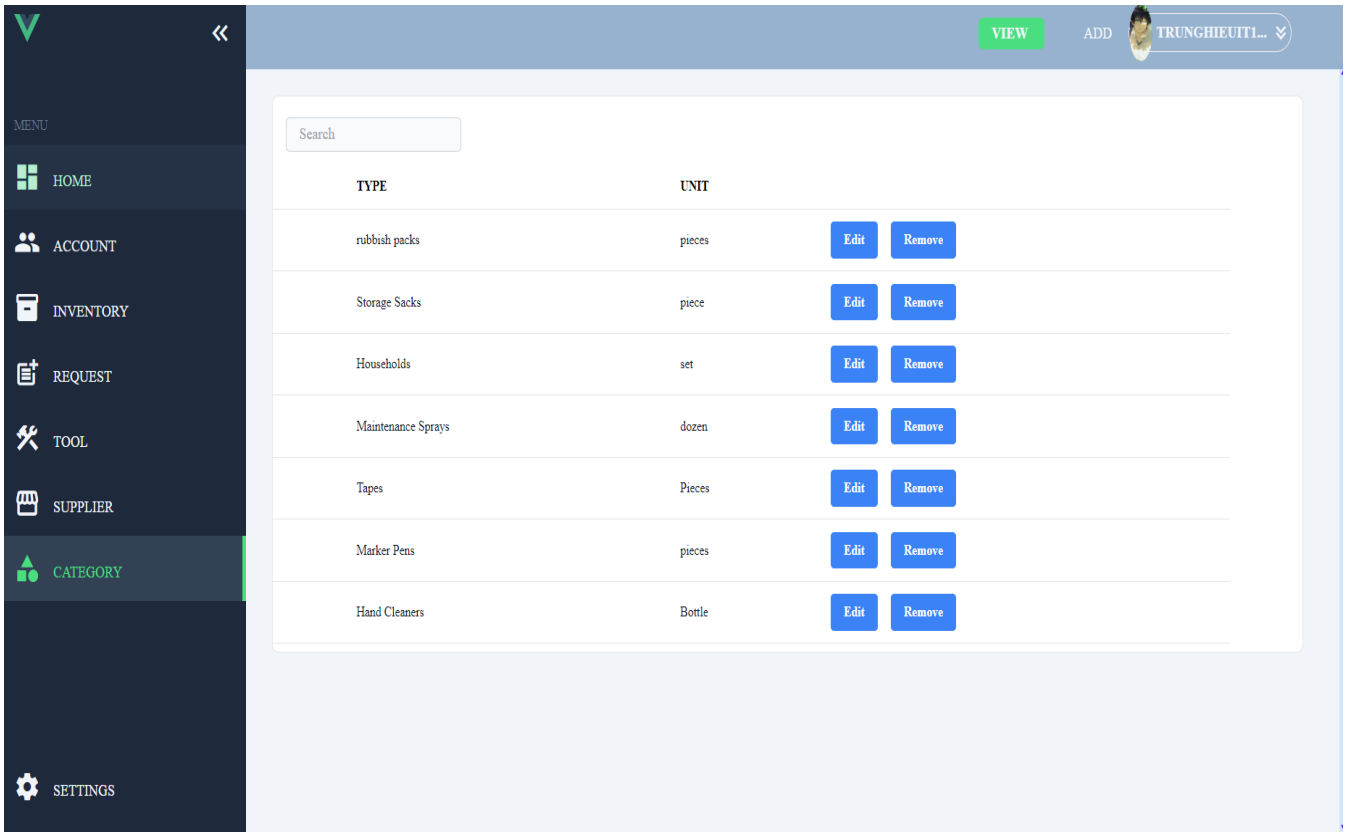

Figure 35. View

4.8.2 Create

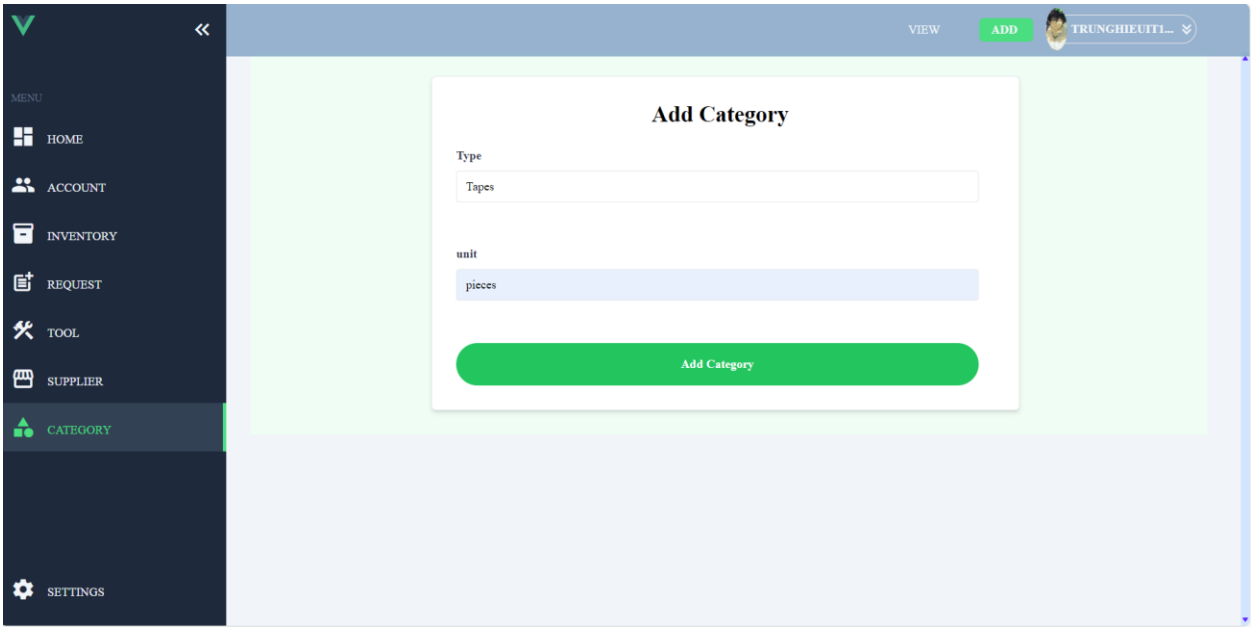

Figure 36. Create

4.8.3 Edit

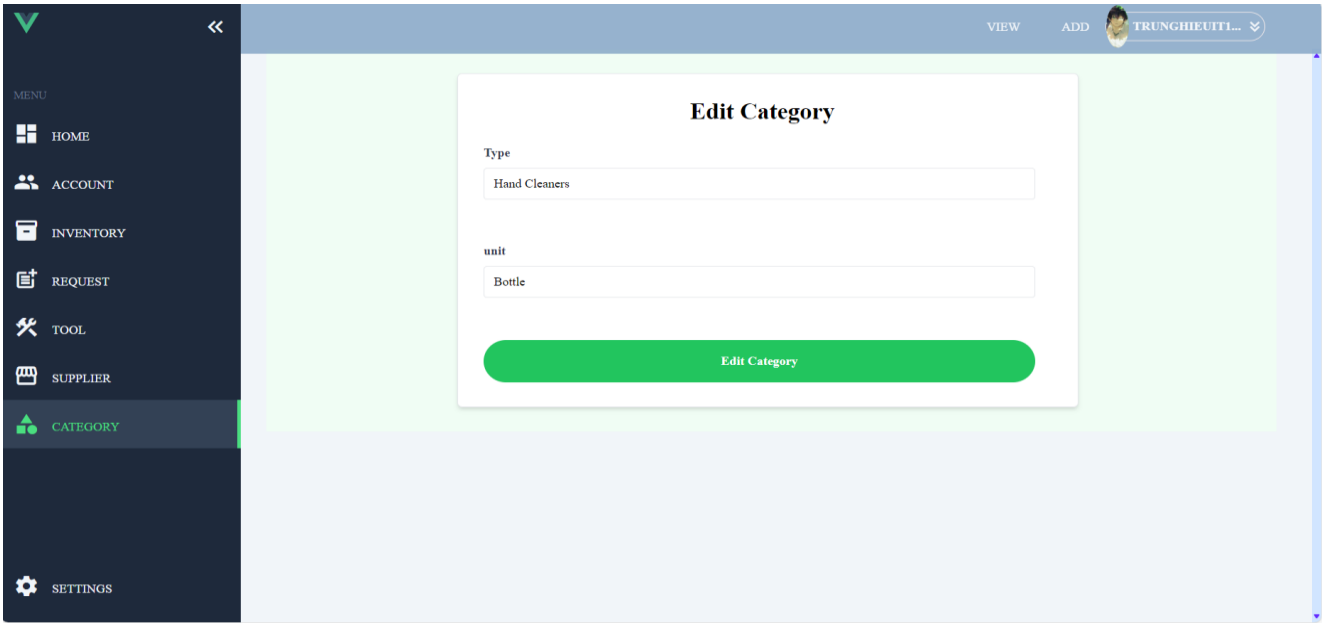

Figure 37 Edit

4.9 Setting

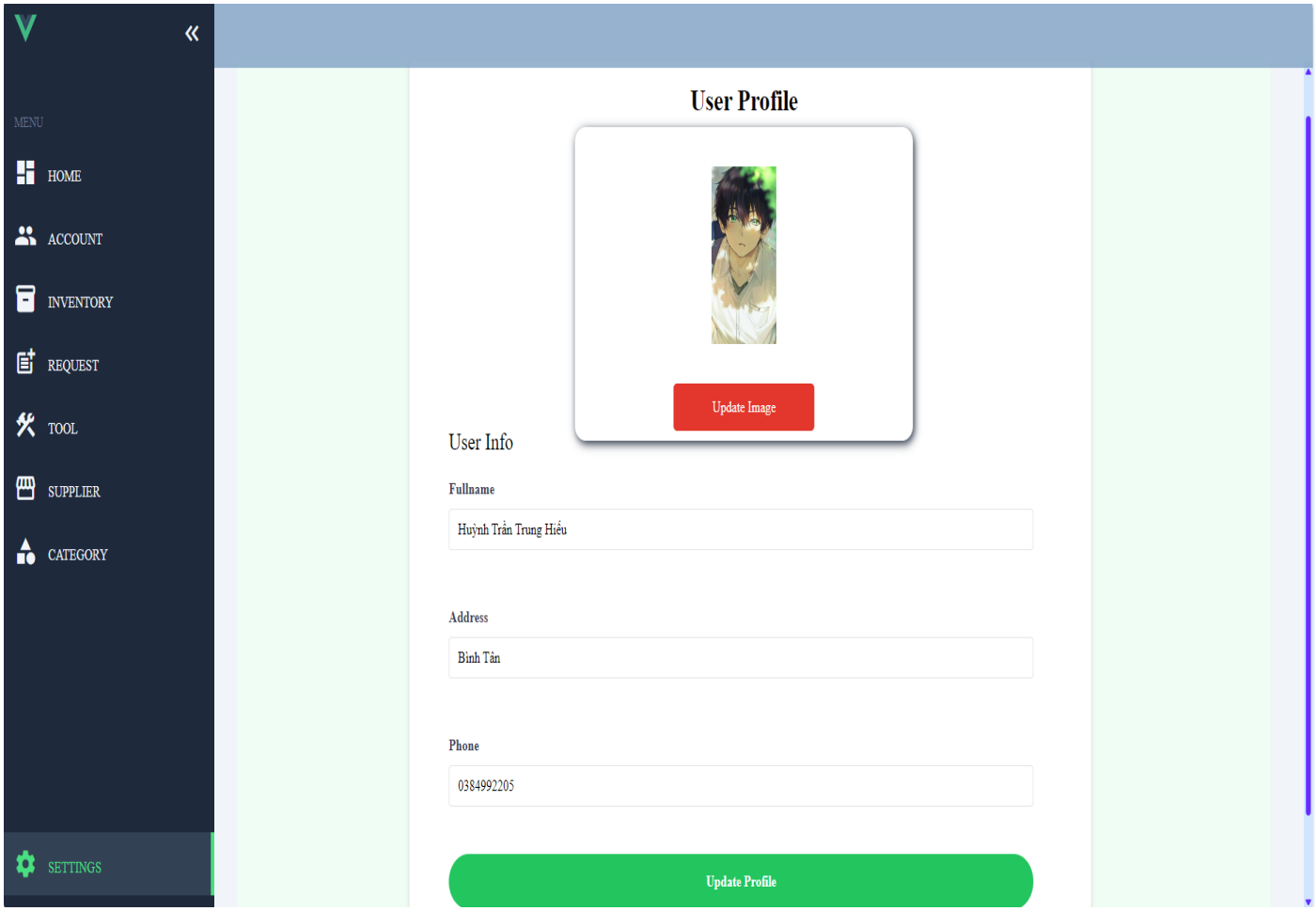

Figure 38 Setting

REFERENCES

1. Microsoft Graph REST API v1.0 endpoint reference - Microsoft Graph v1.0 | Microsoft Learn
2. Components Basics | Vue.js (vuejs.org)
3. How to Draw 5 Types of Architectural Diagrams | Lucidchart Blog
4. Install Tailwind CSS with Vue 3 and Vite - Tailwind CSS
5. Configure One-to-Many Relationship in Entity Framework 6 (entityframeworktutorial.net)
6. Unit Of Work in Repository Pattern - Dot Net Tutorials
7. Generic Repository Pattern in C# with Examples - Dot Net Tutorials
8. FluentValidation — FluentValidation documentation