Avinash Kola

[kola@umd.edu](mailto:kola@umd.edu)

UMD directory ID: kola

**Household Chores Distributor**

# Table of Contents

[Table of Contents 2](#_Toc77487619)

[List of Figures 3](#_Toc77487620)

[1.0. Introduction 4](#_Toc77487621)

[1.1. Purpose 3](#_Toc77487622)

[1.2. Scope of Project 3](#_Toc77487623)

[1.3. Glossary 5](#_Toc77487624)

[1.4. References 5](#_Toc77487625)

[2.0. Overall Description 6](#_Toc77487627)

[2.1 System Environment 6](#_Toc77487628)

[2.2 Requirements Specification Functional 7](#_Toc77487629)

[2.2.1 User Management 8](#_Toc77487630)

[Use case: User 7](#_Toc77487631)

[2.2.3 Chore Management 9](#_Toc77487634)

[Use case: Assign Chore 10](#_Toc77487637)

[Use case: Set Chore priority 11](#_Toc77487638)

[Use case: View Chore List 12](#_Toc77487639)

[Use case: Complete Chore 12](#_Toc77487640)

[Use case: Chore Exchange 13](#_Toc77487641)

Use case: Customize Preferences...........................................................................13

[2.3 User Characteristics 14](#_Toc77487648)

2.4 Non-Functional Requirements 15

[3.0. Requirements Specification 16](#_Toc77487650)

[3.1 External Interface Requirements 17](#_Toc77487651)

[3.2 Functional Requirements 17](#_Toc77487652)

[3.2.1 User 18](#_Toc77487653)

[3.2.2 Create Chore 18](#_Toc77487654)

[3.2.3 Set Chore priority 19](#_Toc77487655)

[3.2.4 Chore List 20](#_Toc77487656)

[3.2.5 Complete Chore 21](#_Toc77487657)

[3.2.6 Delete Chore 21](#_Toc77487658)

[3.2.7 Chore Notification 22](#_Toc77487659)

[3.3 Detailed Non-Functional Requirements 23](#_Toc77487665)

[3.3.1 Logical Structure of the Data 23](#_Toc77487666)

[3.3.2 Security 24](#_Toc77487667)

3.3.3 Misuse Cases..................................................................................................25

# List of Figures

Figure 1 - System Environment………………………………………………......... 5

Figure 3 - Chores Use Cases…………………………………….............................. 8

# Introduction

## **1.1. Purpose**

This document is a detailed guide to the Household Chores Distributor web application. It describes what the application does, how users will use it, and the regulations it adheres to. It's for both those who will use the application and those who will create it. We'll also share this paper with the leaders of the home or organization for their approval to ensure that everyone is on the same page.

## **1.2. Scope of Project**

The Household Chores Distributor is a user-friendly web application aims to provides a user-centric platform that allows individuals to independently manage their household tasks and preferences, promoting autonomy. It will provide an organized platform for assigning, tracking, and completing various household tasks, ensuring that everyone successfully shares responsibility.

This system is designed to share the household work among family or roommates will result in efficient chores management and equal and fair task distribution with Improved communication through E-Mail notification updates. Chores Accountability and reports include stats like history, completions, and overdue tasks will improve the transparency among the family or roommates.

## **1.3. Glossary**

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Chore | A specific household task or responsibility. |
| Assign Chore | The process of assigning a chore to a household member |
| Chore Priority | The level of importance to a chore |
| Chore Notification | Alerts and reminders for chore assignments. |
| Chore Completion | Marking a chore as finished when completed |
| Chore History | A log that records chore assignment and completion history |
| Customization | User-configurable settings for chore preferences and priorities. |
| User | Individuals who interact with the Household Chores Distributor system |
| User autonomy | Users will have independence and control of managing Household Chores system without need of having admin privileges. |
| Software Requirements Specification | A document that completely describes all of the functions of a proposed system and the constraints under which it must operate. |

## 

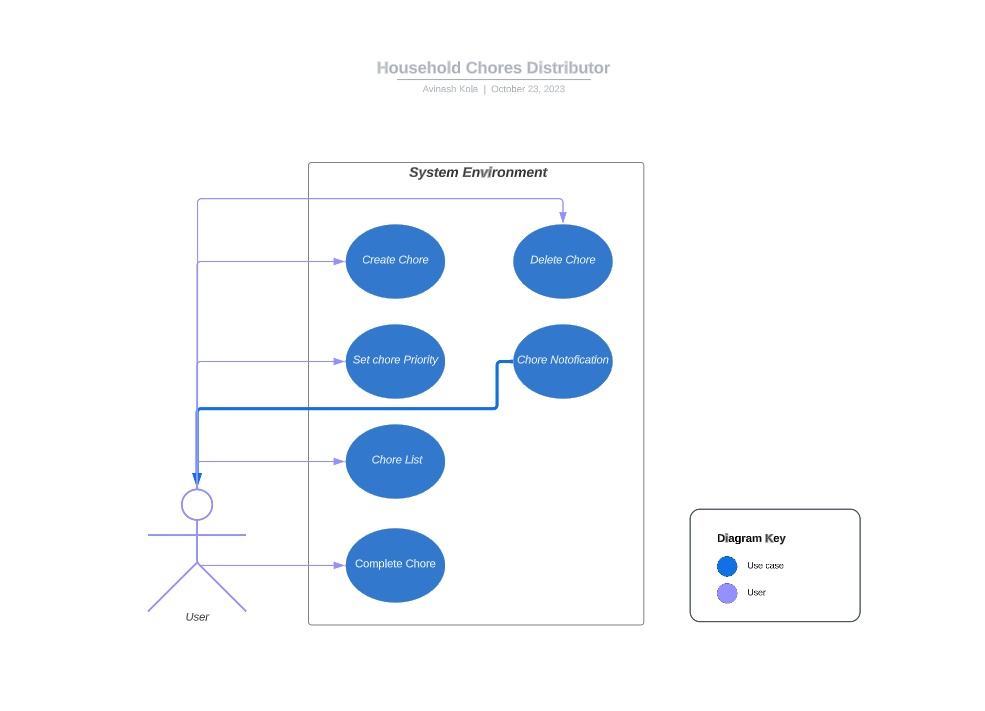
## **1.4. References**

# 1. IEEE Guide to Software Requirements Specifications - <https://ieeexplore.ieee.org/document/278253>

# 2. Lucid Chart - <https://www.lucidchart.com/pages>

# 2.0. Overall Description

# 2.1 System Environment

**Figure 1 – System Environment**

The Household Chores Distributor system has two primary actors that are in charge of accessing various use cases. One is the User, who represents the system's household members. Users interact with the system to complete various household activities.

The "Create Chore," "Chore List," "Set Chore Priority," "Complete Chore," and, "Delete Chore,” use cases can be initiated by the users in the system. Additionally, the system sends Chore notifications to users when chore was assigned to them

## **2.2 Functional Requirements Specification**

This section outlines use cases for user separately in terms of user management and chore management in respective of user autonomy.

### **2.2.1 Chores**

**Use case: User**

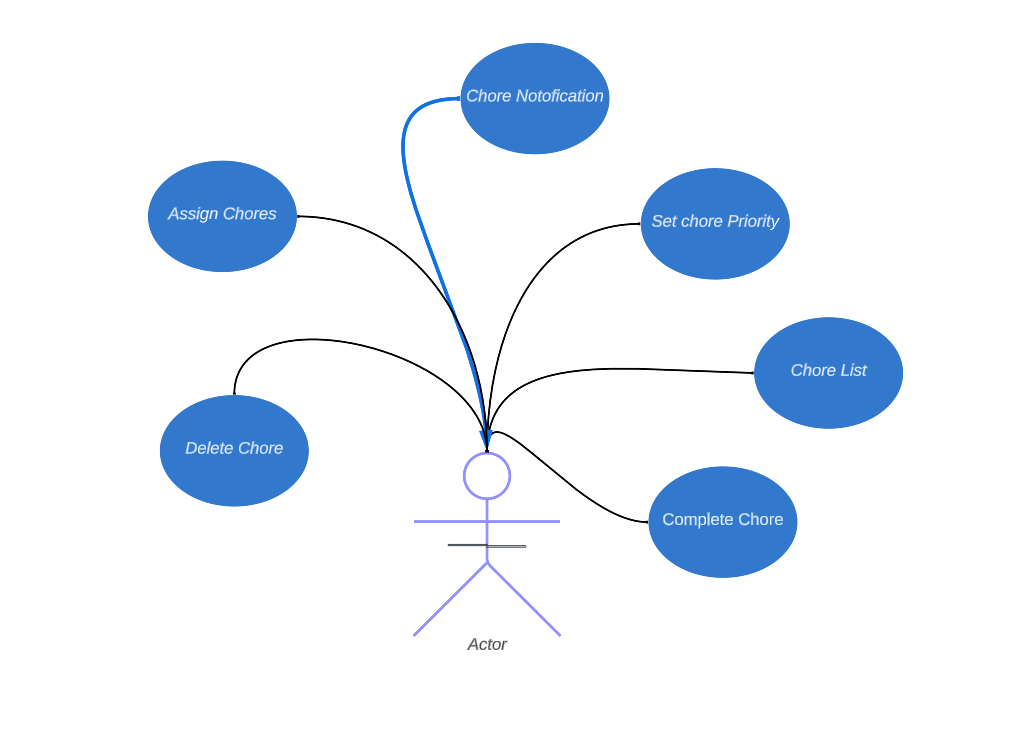


Figure 2 - Chores

**Actor:** User

**Description:**

In the system, Actor is the user who can assign chores, set the chores priority, View the chores list, marking completion of chore, customization and chore exchange. The user will receive the notifications from the system.

**Steps:**

1. The user logs into the system.
2. The user can view a personalized dashboard displaying their assigned chores, priorities, and due dates.
3. The user can create a chore to themselves or other family members.
4. The user can set the priority level (low, required, or urgent) for their assigned chores.
5. The user can view a list of all available chores.
6. The user can mark assigned chores as completed.
7. The user can delete chore assigned by them
8. While creating a chore users can select if they want to send notification to chore assigned user.

**Error Flow 1 - Invalid Login Credentials:**

If a user inputs invalid login credentials (for example, a wrong username or password), the system will display an error saying and the user must retry the login.

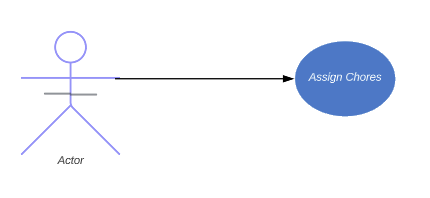
**Error Flow 2 - Error in Chore Assignment:**

If the user attempts to assign a chore to a family member who already has more chores than others, the system will give a warning and recommend assigning chores to others. The user has the option of continuing or make changes

### **2.2.3** **Chores**

**Use case: Create Chore**

**Diagram:**



**Description:**

Actor is the user in the system who can use this use case to assign Chores to household members or themselves.

**Steps:**

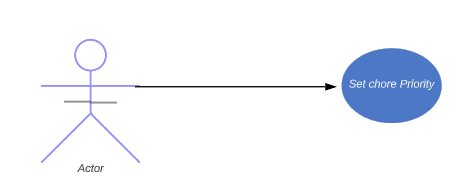
1. Actor will select themselves or other users to assign the chore
2. Then Actor will select the chore to be assigned
3. After Selecting the chore, the Actor will assign that chore to the user.

**Error Flow- Chore unavailability**

If the user trying to assign a chore to user which is already assigned to another household member, then the system will display a message saying chore already assigned to a different user.

**Use Case: Setting Chore Priority**

**Diagram:**



**Description:**

Actor is the user in the system who can use this use case to set Chores Priority to already assigned Chores for household members or themselves.

**Steps:**

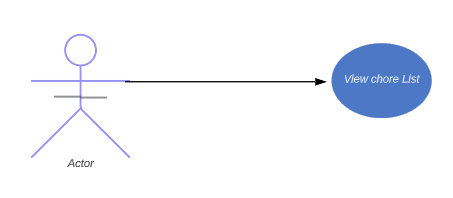
1. The Actor will Assign the chore to themselves or other users
2. Then actor will select the chore to be prioritized
3. After selecting the chore, actor can set the priority level to low, Important, or Urgent.

**Error flow- chore completed**

If the user selected the chore to set a priority level, which is already completed it system will display a error message saying chore is already completed

**Use Case: Chore list**

**Diagram:**



**Description:**

Actor is the user in the system who can use this use case to view all the chores available for the household members.

**Steps:**

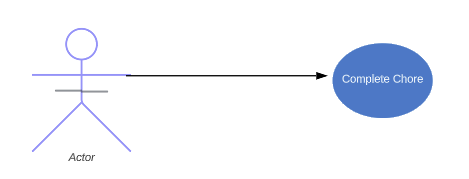
1. First, actor will add the chores to themselves or to the household members
2. Then Actor can select the view chore list option to view all the chores available in the system.

**Alternate Flow - No Chores:**

If the actor or other household members have no chores assigned to them, the system will display a message saying that there are no chores to display.

**Use case: Complete Chore**

**Diagram:**



**Description:**

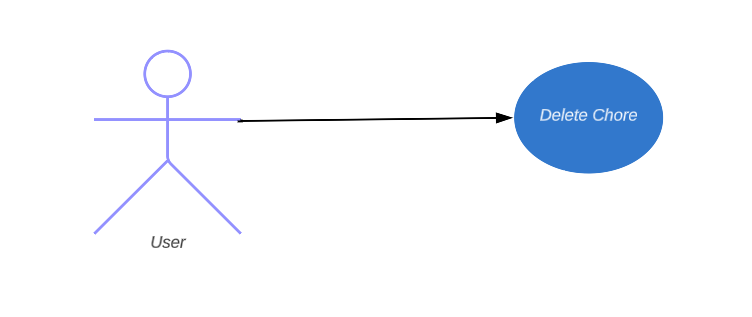
Actor is the user in the system who can use this use case to mark the assigned completed chores for the household members.

**Steps:**

1. To select this use case, we assume that the actor already assigned the chores to themselves or household members.
2. By looking at the priority level the individual will complete the assigned chore
3. The household member will select the complete chore option and marks chore as completed.

**Use case: Delete Chore**

**Diagram:**



**Description:**

Actor is the user in the system who can use this use case to delete the assigned chore by the user

**Steps:**

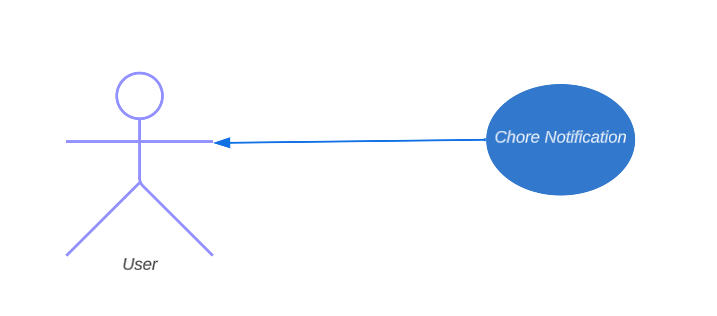
1. To select this use case, we assume that the actor already assigned the chores to themselves or household members.
2. By looking at the status of the chore the individual will check if the task is completed.
3. If completed, the user will delete the chore created by them.

**Alternative Flow – If user not created the Chore:**

If the user attempts to delete a chore not created by them, the system will make the delete option will be disabled.

**Use Case: Chore Notification**

**Diagram:**

****

**Description:**

The system will send Email notifications to users when chore is assigned to them.

**Steps:**

1. To select this use case, we have to select send notification option while creating a chore.
2. If the box is checked then the chore assigned user will be sent a notification about new assigned chore.

**Alternative Flow – Email not valid:**

If the user provided an invalid email address, then the system will return an error saying invalid email address.

## **2.3 User Characteristics**

Household Members(users) may have varying levels of technological knowledge and some people are more at ease with digital platforms than others but are expected to be Internet literate. To provide accessibility for all members, the system should have an easy-to-use interface similar to that of common smartphone apps.

Administrators must be capable of performing basic system management operations. They should be able to manage user accounts efficiently, similar to how user profiles are managed in regular computer applications.

## **2.4 Non-Functional Requirements**

The household Chores distributor system will function on a server linked to high-speed internet, guaranteeing that it operates smoothly for users. Usability is an essential preference, with a user interface built for easy navigation that suits users with a wide range of technological backgrounds. As the number of users increases, the system's scalability ensures that it can handle greater loads without compromising performance. Also, Compatibility broadens the system's reach by ensuring that it works flawlessly across numerous devices and browsers, improving accessibility. Also, User efficiency not solely depends on the system but their own device hardware and internet speed connection.

# Requirements Specification

## **3.1 External Interface Requirements**

External interfaces will be added to the Household Chores Distributor system to improve user experience and system functionality. At first, it will use with ordinary email services to send users chore notifications and updates. This email interface will ensure that the system and users' email accounts communicate seamlessly, improving chore management efficiency. Furthermore, the system will support web browsers as the primary user interface, ensuring compatibility across a wide range of devices and platforms and making it easily available to users regardless of web browser usage.

## **3.2 Functional Requirements**

**3.2.1 User**

|  |  |
| --- | --- |
| **Use Case Name** | User |
| **XRef** | Use Case: 2.2.1 |
| **Trigger** | User logs into the system |
| **Precondition** | User has a valid account and is logged out |
| **Basic Path** | 1. User logs into the system. 2. User views a personalized dashboard displaying their assigned chores, priorities, and due dates. 3. User assigns a chore to themselves or other family members. 4. User sets the priority level (low, important, or urgent) for their assigned chores. 5. User views a list of all available chores. 6. User marks assigned chores as completed. 7. User requests a chore exchange with another family member. 8. User customizes their chore preferences and system settings, including chore types, work frequency, and notification preferences. |
| **Alternative Paths** | Invalid Login Credentials: If a user inputs invalid login credentials (for example, a wrong username or password), the system will display an error saying and the user must retry the login.  Error in create chore: If the user attempts to assign a chore to a family member who already has more chores than others, the system will give a warning and recommend assigning chores to others. The user has the option of continuing or make changes |
| **Postcondition** | User successfully manages chores and preferences in the system. |
| **Other** | Data: Username, password, chore details, priority levels, chore list, customization preferences. |

* + 1. **Create Chore**

|  |  |
| --- | --- |
| **Use Case Name** | Assign Chore |
| **XRef** | Use Case: 2.2.3 - Chores |
| **Trigger** | User selects "Create Chore" in the system |
| **Precondition** | User is logged in and has not already assigned the selected chore |
| **Basic Path** | 1. User selects "Create Chore" in the system. 2. User chooses a household member to assign the chore to. 3. User selects the chore to be assigned. 4. User assigns the selected chore to the chosen household member. |
| **Alternative Paths** | If the user trying to create a chore to user which is already assigned to another household member, then the system will display a message saying chore already assigned to a different user. |
| **Postcondition** | The selected chore is successfully assigned to the chosen household member. |
| **Other** | Data: Chore details, assigned household member. |

* + 1. **Set Chore Priority**

|  |  |
| --- | --- |
| **Use Case Name** | Set Chore Priority |
| **XRef** | Use Case: 2.2.3 - Chores |
| **Trigger** | User selects "Set Chore Priority" in the system |
| **Precondition** | User is logged in and has assigned chores to themselves or household members |
| **Basic Path** | 1. User selects "Set Chore Priority" in the system. 2. User chooses a chore for which to set the priority. 3. User sets the priority level (low, important, or urgent) for the selected chore. |
| **Alternative Paths** | If the user selected the chore to set a priority level, which is already completed it system will display a error message saying chore is already completed. |
| **Postcondition** | The priority level for the selected chore is successfully set. |
| **Other** | Data: Chore details, priority level. |

* + 1. **Chore list**

|  |  |
| --- | --- |
| **Use Case Name** | Chore List |
| **XRef** | Use Case: 2.2.3 - Chores |
| **Trigger** | User can view chores listed in the system |
| **Precondition** | User is logged in and has assigned chores to themselves or household members |
| **Basic Path** | 1. User will log into the system. 2. User views a list of all available chores in the system. |
| **Alternative Paths** | If the actor or other household members have no chores assigned to them, the system will display a message saying that there are no chores to display. |
| **Postcondition** | The user can see a list of available chores. |
| **Other** | Data: List of available chores. |

* + 1. **Complete Chore**

|  |  |
| --- | --- |
| **Use Case Name** | Complete Chore |
| **XRef** | Use Case: 2.2.3 - Chores |
| **Trigger** | User selects "Complete Chore" in the system |
| **Precondition** | User is logged in and has assigned chores to themselves or household members |
| **Basic Path** | 1. User selects "Complete Chore" in the system. 2. User looks at the priority level and completes the assigned chore accordingly. 3. The household member marks the chore as completed. |
| **Alternative Paths** | If the user attempts to complete a lower-priority chore when there are still higher-priority activities waiting, the system will display a warning, advising them to do higher-priority chores first. The user can either continue or change their selection. |
| **Postcondition** | The chore is marked as completed in the system. |
| **Other** | Data: Completed chore details. |

* + 1. **Delete Chore**

|  |  |
| --- | --- |
| **Use Case Name** | Delete Chore |
| **XRef** | Use Case: 2.2.1 (Chores) |
| **Trigger** | User chooses "Delete Chore" option in the system |
| **Precondition** | User is logged in and has appropriate permissions |
| **Basic Path** | 1. User selects "Delete Chore" in the system. 2. User looks at the priority level and completes the assigned chore accordingly. 3. The Chore will be deleted from the database. |
| **Alternative Paths** | N/A |
| **Postcondition** | The chore will be deleted |
| **Other** | - |

* + 1. **Chore Notification**

|  |  |
| --- | --- |
| **Use Case Name** | Chore Notification |
| **XRef** | Use Case: 2.2.1 (Chores) |
| **Trigger** | User chooses "chore notification" option in the system |
| **Precondition** | User is logged in and has appropriate permissions |
| **Basic Path** | 1. User selects "create chore" in the system. 2. After assigning the chore to himself or other household members user will select option to send notification. 3. The notification will be sent to the chore assigned user. |
| **Alternative Paths** | N/A |
| **Postcondition** | The Chore assigned user will receive the Email notification |
| **Other** | - |

## **3.3 Detailed Non-Functional Requirements**

### **3.3.1 Logical Structure of the Data**

**User Entity**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Item** | **Type** | **Description** | **Comment** |
| User ID | Integer | Unique identifier for users | Used as primary key |
| Username | Text | User's username |  |
| Password | Text | User's password (hashed and securely stored) |  |
| Email | Text | User's email address |  |
| Is superuser | Text | Type of user (e.g., family member, roommate) |  |
| Date Joined | Text | Date when the user account was created |  |
| Last login | Date | Shows user last login information |  |

**Chore Entity**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Item** | **Type** | **Description** | **Comment** |
| Chore ID | Integer | Unique identifier for chores | Used as primary key |
| Assigned date | Date | Assigned date of the Chore |  |
| Chore Priority | Category | Priority level of chore (low, important, urgent) |  |
| Due Date | Date | Due date for completing the chore |  |
| Chore Assigner | Text | Name of the user who assigned the chore |  |
| Chore done by | Text | Name of the user who completed the chore |  |
| Chore done Date | Date | Date when the chore was completed | Null if not completed |
| Status | Text | Status of the chore (assigned, completed, overdue) |  |
| Assigned To | Integer | User to whom the chore is assigned |  |

* + 1. **Security**

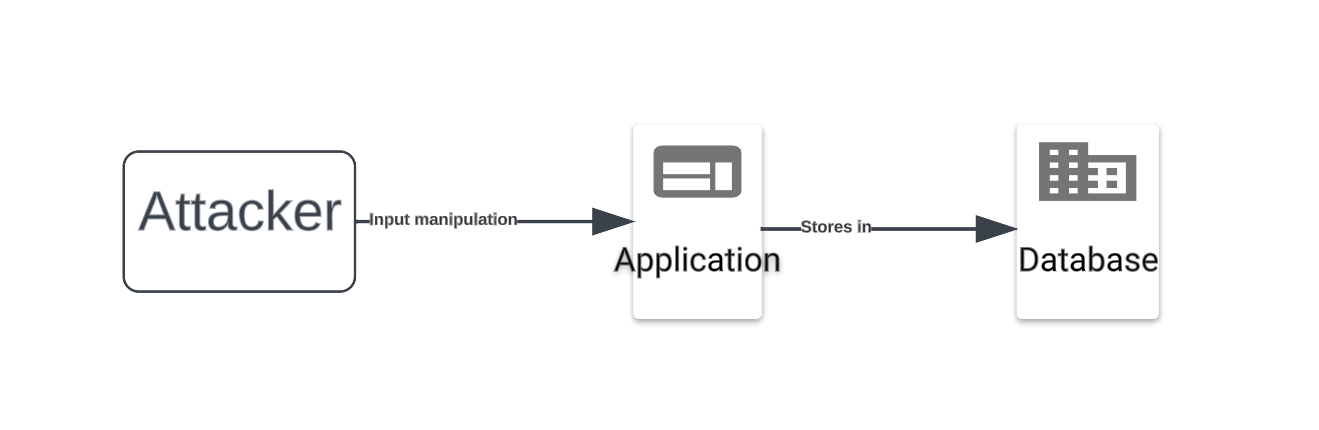
Ensuring strong security measures for the Household Chores Distributor application is of utmost importance to prevent potential vulnerabilities. The system must have robust input validation and parameterized queries to protect against SQL injection. This ensures that user inputs interacting with the database are thoroughly sanitized. To avoid Cross-Site Scripting (XSS) threats, user-generated content must be automatically escaped and sanitized before rendering on web pages. The system must enforce stringent controls for strong authentication and authorization, requiring users to authenticate before accessing restricted areas and ensuring proper authorization checks. The system will limit repeated login attempts within a specific timeframe to prevent rate-limiting and brute-force attacks. During registration, email verification and validation checks ensure that only valid email addresses are accepted. Users will be required to verify their emails via confirmation links. The system will limit the number of registration attempts to prevent account flooding.

Furthermore, data access controls will be enforced to restrict write/delete access to authorized personnel, while read access will be permitted for all authorized users. Physical access to the server will also be restricted to authorized personnel to prevent unauthorized physical access. These security measures are essential to protect user data and maintain system integrity.

* + 1. **Misuse Cases**

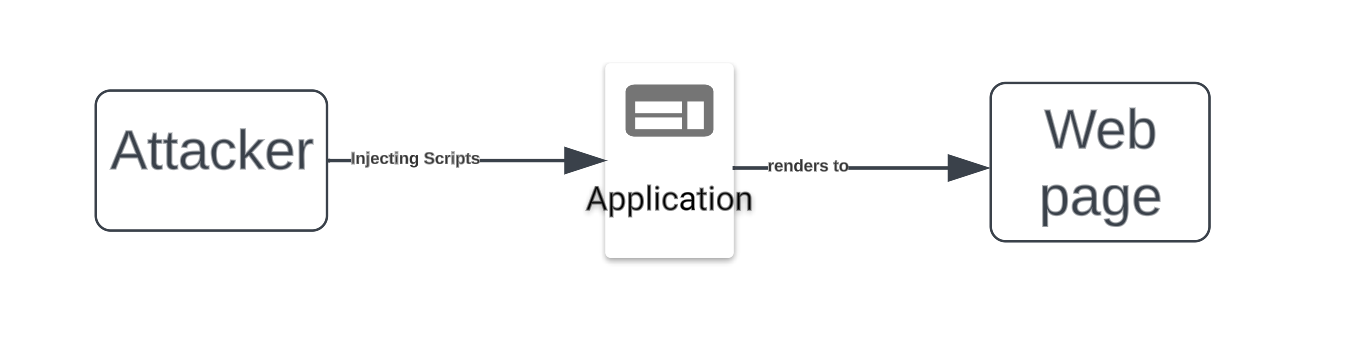
**Misuse Case 1: SQL Injection**

* Threat (STRIDE): Spoofing and Tampering
* Description: An attacker manipulates user input to inject malicious SQL queries into the application.
* Attack Surface: Any form or input field interacting with the database.
* Attack Vector: The attacker submits malicious input that can potentially exploit unsanitized queries.
* Mitigation: We use parameterized queries or Django's ORM to sanitize input to prevent SQL injection.



**Misuse Case 2: Cross-Site Scripting (XSS)**

* Threat (STRIDE): Tampering
* Description: An attacker injects malicious scripts into user-generated content, leading to unauthorized script execution.
* Attack Surface: Any input or template rendering user-generated content.
* Attack Vector: The attacker submits scripts that get rendered on a page.
* Mitigation: To mitigate XSS attacks, we will properly escape user-generated content using Django's auto-escaping.



**Misuse Case 3: Lack of Authentication and Authorization Checks**

* Threat (STRIDE): Elevation of Privilege
* Description: Unauthorized users gain access to restricted areas of the application.
* Attack Surface: Any views or resources that should be protected by authentication and authorization.
* Attack Vector: Unauthorized users attempt to access protected views.
* Mitigation: To prevent unauthorized access, we will implement authentication and authorization checks using Django's built-in features, such as decorators like `@login\_required` and `@permission\_required`.

**Misuse Case 4: Lack of Data Validation and Sanitization**

* Threat (STRIDE): Tampering
* Description: An attacker submits malicious or malformed data that can lead to security vulnerabilities.
* Attack Surface: User input and forms, especially in registration and login processes.
* Attack Vector: The attacker submits data with malicious intent.
* Mitigation: To ensure data integrity and prevent security issues, we will implement data validation and sanitization using Django's built-in form validation.

**Misuse Case 5: Lack of Rate Limiting and Brute-Force Protection**

* Threat (STRIDE): Denial of Service
* Description: Attackers launch brute-force attacks on user accounts, causing service disruption.
* Attack Surface: Login and authentication processes.
* Attack Vector: Repeated login attempts from unauthorized sources.
* Mitigation: To prevent brute-force attacks, we will implement rate limiting and brute-force protection mechanisms using Django security middleware or third-party packages like Django Ratelimit.

**Misuse Case 6: Email Account Verification**

* Threat (STRIDE): Tampering
* Description: An attacker may attempt to tamper with the email verification process by submitting fake or malicious email addresses.
* Attack Surface: The email verification process in the registration workflow.
* Attack Vector: The attacker submits fake or malicious email addresses during registration.
* Mitigation: we will Implement email validation checks to ensure that only valid and legitimate email addresses are accepted during registration. Additionally, verify email addresses by sending a verification link to the provided email and require users to confirm their email addresses before activating their accounts.

**Misuse Case 7: Account Flooding**

* Threat (STRIDE): Denial of Service
* Description: Attackers could attempt to flood your system by repeatedly submitting registration forms, creating a large number of accounts in a short period.
* Attack Surface: The registration form and associated registration process.
* Attack Vector: Attackers submit a high volume of registration requests in a short time frame.
* Mitigation: we will Implement rate limiting to restrict the number of registration attempts from a single source or IP address within a specific time frame. This can help mitigate the risk of account flooding and prevent system overload.

