# Software Requirements Specification

**for**

# DO IT.
# To-Do List Application

**Version 1.0 approved**

**Prepared by Oona Kintscher and Anhviet Le**

**Purdue University Northwest**

**10/10/2023**

# Table of Contents

**Table of Contents** ..... **ii**
**Revision History** ..... **ii**
**1. Introduction** ..... **1**
1.1 Purpose ..... 1
1.2 Document Conventions ..... 1
1.3 Intended Audience and Reading Suggestions ..... 1
1.4 Product Scope ..... 2
1.5 References ..... 2
**2. Overall Description** ..... **2**
2.1 Product Perspective ..... 2
2.2 Product Functions ..... 2
2.3 User Classes and Characteristics ..... 3
2.4 Operating Environment ..... 3
2.5 Design and Implementation Constraints ..... 3
2.6 User Documentation ..... 3
2.7 Assumptions and Dependencies ..... 4
**3. External Interface Requirements** ..... **4**
3.1 User Interfaces ..... 4
3.2 Hardware Interfaces ..... 4
3.3 Software Interfaces ..... 4
3.4 Communications Interfaces ..... 5
**4. System Features** ..... **5**
4.1 Adding a Task ..... 5
4.2 Removing a Task ..... 7
4.3 Editing a Task ..... 8
4.4 Saving Items to Database ..... 10
**5. Other Nonfunctional Requirements** ..... **11**
5.1 Performance Requirements ..... 11
5.2 Safety Requirements ..... 11
5.3 Security Requirements ..... 11
5.4 Software Quality Attributes ..... 11
5.5 Business Rules ..... 11
**6. Other Requirements** ..... **12**
6.1 Future Deliverables ..... 12
6.2 Planned Features and Functionalities ..... 12
**Appendix A: Glossary** ..... **13**
**Appendix B: Analysis Models** ..... **13**
B.1. Use Case Diagram ..... 13
B.2. UI Design - State Diagram ..... 13
B.3 Class Diagram ..... 14

# Revision History

| Name | Date | Reason For Changes | Version |
|---|---|---|---|
| No Revision | 10/10/2023 | | 1.0 |
| | | | |

# 1. Introduction

## 1.1 Purpose

The purpose of this document is to gather ideas, define constraints, and analyze the requirements for the product to provide a complete understanding of the user's role and the developer's tasks. This document specifies the software requirements for version 0.1 of the to-do list application and lists deliverables for future versions. It outlines the scope of the product, the overall application, and its core functionality. All use cases and features of the product are extensively described alongside the type of audience intended to use the product.

## 1.2 Document Conventions

| | | |
|---|---|---|
| *Italic* | - | Proper Names |
| **Bold** | - | Headings and Subtitles |
| Underline | - | Key Words |

## 1.3 Intended Audience and Reading Suggestions

This document is intended to be read by:

| | | |
|---|---|---|
| Developers | - | This document is used to guide the development process for the described product. |
| Testers | - | The verification of compliance with all requirements and validation of the absence of errors can be achieved using this document. |
| User | - | This document can be seen as a detailed user manual, including the full capabilities of the software and outline of interfaces. |

This product is not meant for release; instead, the development team will keep it until higher versions are developed. The professor in charge of the project will assess the product's performance for this version.

- Developers are expected to possess a robust grasp of the *Java* programming language and a proficient familiarity with *XML*, which is extensively utilized in *Android Studio*. Additionally, prior experience working with *Android Studio* is considered essential.
- Testers should demonstrate a solid understanding of prevalent software glitches and adhere to industry standards related to the constraints of application testing.
- Users are encouraged to have a working knowledge of Android devices and be proficient in the process of installing applications from the Google Play Store.

## 1.4 Product Scope

The software described herein is a task management application called *DO IT*. Its primary function is to enable individuals to manage their tasks more efficiently. This user-friendly application provides all the essential features required to maintain a to-do list. Furthermore, the app helps achieve personal and work objectives by addressing productivity challenges, improving time management, streamlining operations, and enhancing satisfaction. Currently, the software is only accessible in English, up to version 0.1, and to the development team.

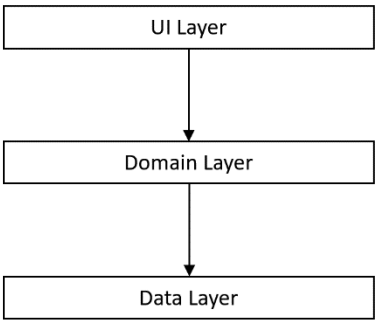
## 1.5 References

| | | |
|---|---|---|
| Project Definition | - | PROJECT_PROPOSAL-ECE354.pdf |
| GitHub Repository | - | https://github.com/oonawith2o/ECE354-PROJECT-I |
| *RecyclerViewSwipeDecorator* Library Repository | - | https://github.com/xabaras/RecyclerViewSwipeDecorator.git |

# 2. Overall Description

## 2.1 Product Perspective

*DO IT* is a free software application suitable for all users interested in organizing their tasks and can be installed on only *Android OS* 7.0 and higher enabled devices. It is an open-source program, which makes it suitable for those who are interested in developing it further by expanding its functions and features. This will also allow any developer to address any bugs that may arise in version 0.1 of the program. In this version, the data is stored on the phone itself and only the phone user can access it. Therefore, the system does not require any connection or communication to other systems, hardware components, or external memory.




## 2.2 Product Functions

Main features of the project:

- add a task to the task list
- remove a task from the task list
- edit a task from the task list
- complete a task from the task list
- task list item provides information on subject, notes, creation time and completion status
- store data to local database
- provide a minimalistic and user-friendly GUI

## 2.3 User Classes and Characteristics

The application is designed to have easy usage. Therefore, there are no restrictions created by the experience or education level of the user. Additionally, the application does not offer different levels of security or privilege. All features will be available to all types of users without any restrictions. The program is built on two types of users:

Professionals - The application will be used in the work environment where work-related events, tasks and assignments can be managed by adding, removing, and editing items in the task list.

Personal - The application is used in everyday life for the organization of common tasks with the help of all the features the application has to offer.

The feature set will be the same for both types of users, but the intent of the user will differ.

## 2.4 Operating Environment

*DO IT* is a fully mobile application that can be used on any *Android* enabled devices with a minimum operating system of *Android* 7.0 and above. However, it is intended to be used on the newest *Android* release. *Android* 14 is developed by *Open Handset Alliance* led by *Google*. It is an open-source software that belongs to the *Android OS* family. The release date of that operating system was the 4th of October 2023.

## 2.5 Design and Implementation Constraints

The development of the *DO IT* application is within *Android Studio* on *Android OS* using the *Java* programming language. We must strictly adhere to the *Google Play Store's* design and programming standards that is required for marketplace acceptance. No specific software development or quality assurance standard provided by the *International Organization of Standardization (ISO)* is followed in this project. The two main requirements for our program are efficient memory management and performance optimization, which are essential to accommodate diverse *Android* devices compatibility. The design of the software's graphical user interface must be developed so that it can adapt to different screen sizes. This will ensure smooth interoperability with other *Android* applications, accommodating various API levels.

## 2.6 User Documentation

A README.md file will suffice as the user documentation which is stored on the application GitHub repository.

## 2.7 Assumptions and Dependencies

This application relies on *Android Studio* for development and has dependencies on *Apache License 2.0*, *Android* packages, and *Google* services, which are integral to its functionality. The following external library is used:

*RecyclerViewSwipeDecorator*
This is a simple utility class allowing the user to swipe left and right on the task item. It was developed by *Paolo Montalto* and *Jasmin Grbo* and is open source on *GitHub*.

# 3. External Interface Requirements

## 3.1 User Interfaces

This product consists of a single graphical user interface with a main view. It displays a current list of the user's tasks. A button is located in the lower right corner of the screen. When clicked, a pop-up window allows the user to add an item to the task list. The add/edit popup consists of input fields for the requested information. At the bottom are two buttons for saving and canceling the operation. The title of the application is displayed at the top of the main screen. Each task is displayed as a small rectangle consisting of the subject, notes, creation date, and a checkbox.

Main Colors of the GUI

- #272829
- #61677A
- #D8D9DA
- #FFF6E0

## 3.2 Hardware Interfaces

The product is designed to operate on *Android* devices running *Android OS* (version 7.0 and higher). It interfaces with the underlying hardware components of these devices, including touchscreens, processors, memory, and network interfaces. The software interacts with the hardware to facilitate user input via touch gestures, store and retrieve data from the device's storage, and utilize network connectivity for data synchronization.

## 3.3 Software Interfaces

The application interfaces with various software components, prominently utilizing *Android Studi*o as its development environment. It relies on the *Android OS* core functionalities like process management and UI rendering. The application adheres to *Android's* standard API protocols, maintaining compatibility and flexibility within the development environment. The software interface of *DO IT* is intricately tethered to the constraints inherent to the *Java* programming language. The design of the various views of the application is based on *Extensible Markup Language*. The objects on each view are then defined, stored, and organized in *.xml* files. *SQLiteDatabase* is the class used

to create the project's database and store all tasks values. It stores the data to a text file on the device. The *SQLiteOpenHelper* class manages the database creation and the version management.

## 3.4 Communications Interfaces

In this version of the project there is no communication interfaces used other than reading and writing to the database file using *SQLiteDatabaseHelper*. Future versions of the application with group tasks functionality will implement cloud server storage using *SQLiteOnline*. This relies on *HTTP* for communication with remote servers, ensuring efficient data exchange. Strong encryption secures data in transits, and data transfer rates are optimized for quick synchronization. This mechanism ensures seamless data updates between the *Android* device and server, maintaining consistency of the data.

# 4. System Features

## 4.1 Adding a Task

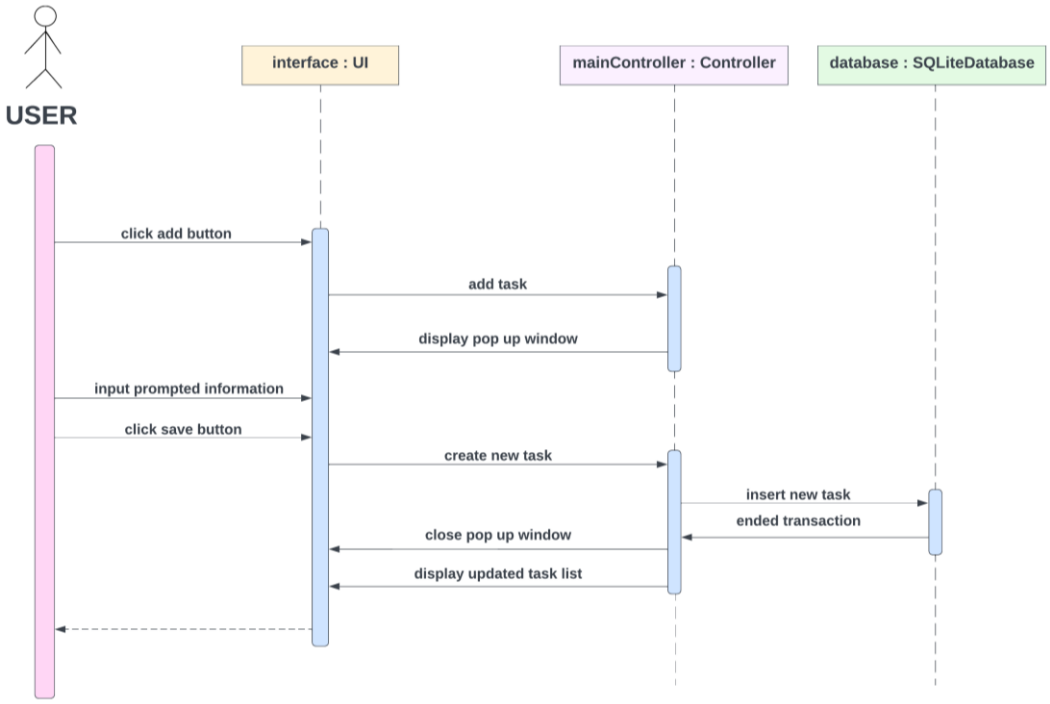
### 4.1.1 Description and Priority

This feature allows the user to add an item to the task list and specify the subject, note, and completion status of the item. It is of high priority because of its necessity for the main purpose of the application to be fulfilled.

Benefit: 9
Penalty: 4
Cost: 1
Risk: 1

### 4.1.2 Stimulus/Response Sequences

The user will have the option to click an add button on the bottom right corner of the screen. The activation of that button allows the user to access an input screen for the new task. Multiple text input objects are used to get the specific information needed to create a new task item. After the user inputs all required information, a save button can be accessed and used to create the new task. Throughout the process a cancel button is provided to jump out of this process. The system will write the new item into the database, and it will then be displayed on the main view.




*Sequence Diagram of Adding Task Feature*

### 4.1.3 Functional Requirements

First, the designs of both the main view and the popup window must be implemented. The add button on the main view must be linked to an *OnClickListener* for the controller to be notified of input from the user. A function that displays the popup window when the button is clicked must be implemented. This window should include two text input objects and two buttons. A *TextChangedListener* should notify the controller whenever the user inputs or changes text. Using that feedback the save button should only be enabled if the input fields are filled in. The cancel button should not have any restriction. If the user decides to save an item, the String inputs of the text fields must be grabbed. A new instance of the item class must be instantiated and inserted into the database.

REQ-1: Layout of Main View includes an Add Button (*activity_main.xml*)
REQ-2: Layout of Popup Window (*item_popup.xml*)
REQ-3: Ability to Write to Database (*DataBaseHelper.java)*

## 4.2 Removing a Task

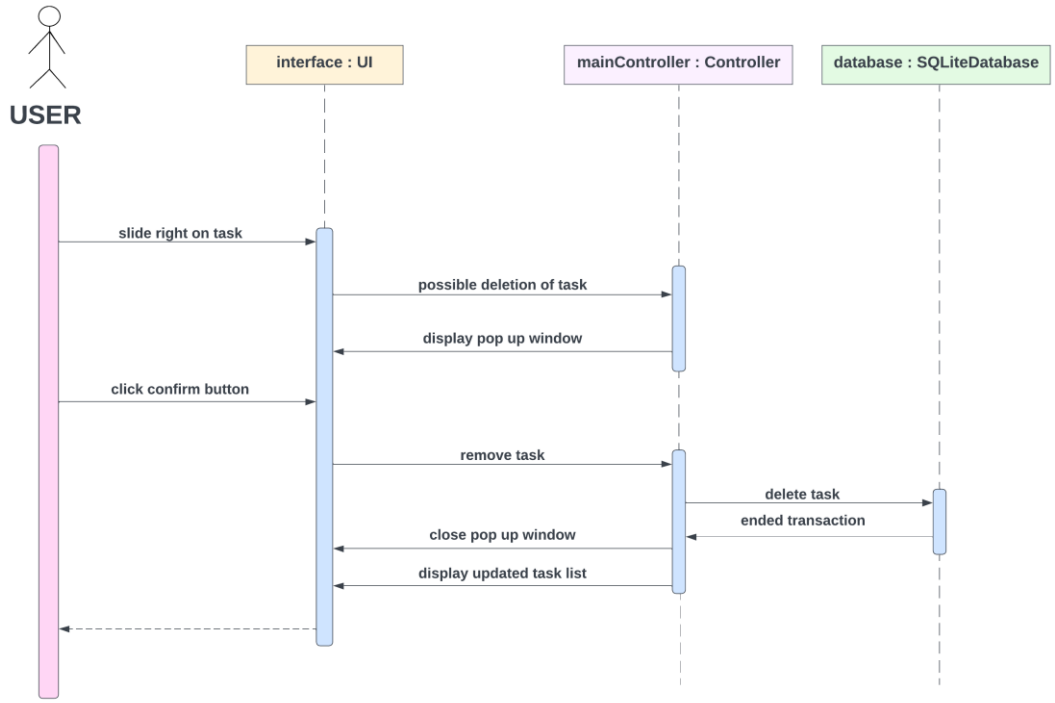
### 4.2.1 Description and Priority

The software should provide the ability to delete items from the task list if their existence is not required by the user. This feature is of high priority and must be delivered for this version of the project.

Benefit: 9
Penalty: 4
Cost: 1
Risk: 1

### 4.2.2 Stimulus/Response Sequences

A right swiping gesture by the user on the specific item of the task list, will allow the user to delete the specified task. The software will indicate that the user is doing a gesture by displaying an animation. Additionally, the software will show an alert dialog window which asks the user to confirm the action taken. It prompts the user to delete or cancel the deletion process. After the button is clicked, the task list on the main screen is updated and displayed.




*Sequence Diagram of Removing Task Feature*

### 4.2.3 Functional Requirements

For a user to delete an item, the user must have added an item into the task list beforehand. In other words, the task list cannot be empty. In addition, the library supporting swiping gestures must be included in the project dependencies section and activated for all items in the task list. The model layer of the software must be notified when the user does the swiping gesture. The controller then must communicate with the UI Layer to display the pop-up window. The buttons provided on the pop-up window must be linked to *OnClickListeners* to receive the user input. Additionally, the deletion of the task must be written into the database using the *DataBaseHelper* class.

REQ-3: Ability to Write to Database (*DataBaseHelper.java)*
REQ-4: Implementation of Addina a Task
REQ-5: Import of Gesture Detection Library

## 4.3 Editing a Task

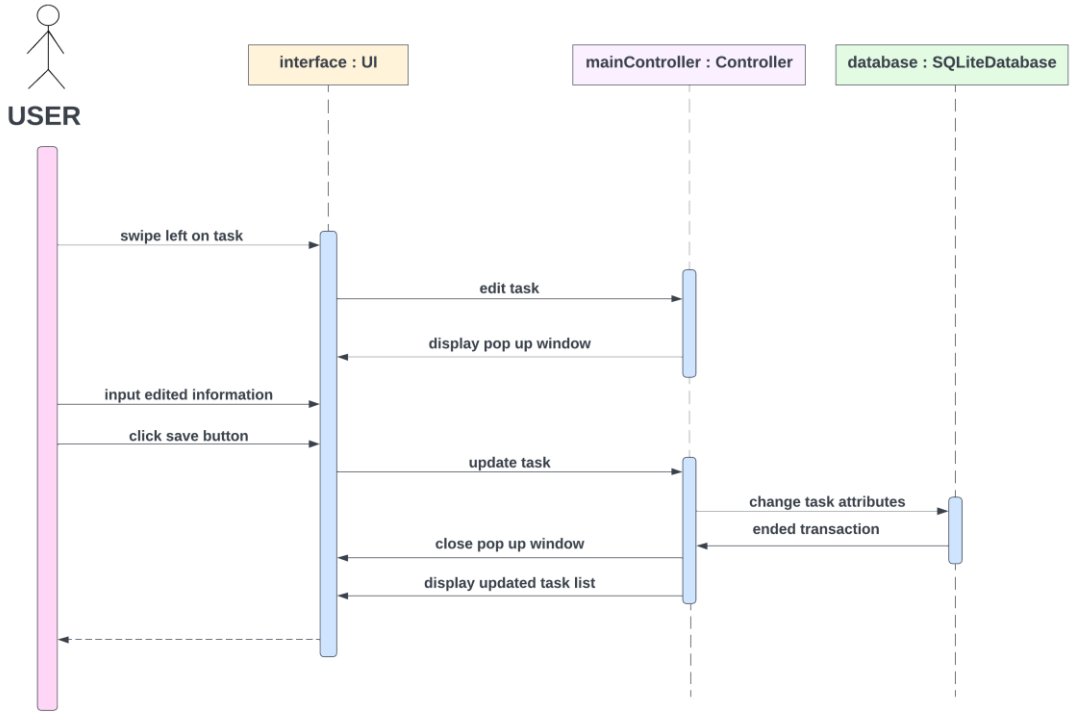
### 4.3.1 Description and Priority

Another main feature of the software is the ability to edit a task. That means the user can change the subject and note information from a selected task. This feature is also of high priority because it enables a smoother handling of tasks.

Benefit: 9
Penalty: 5
Cost: 1
Risk: 1

### 4.3.2 Stimulus/Response Sequences

When the user swipes left on an item in the task list the model layer is informed and given a tag indicating what gesture was performed by the user. The controller then changes the main view to display a pop-up window displaying the stored data of the item. This data can be edited by the user. If the save button in the left corner is pressed by the user, the controller is notified which then writes the new data into the correct row of the database. When the transaction with the database is completed the pop-up window is closed and an updated item list is displayed on the main view.




*Sequence Diagram of Editing Task Feature*

### 4.3.3 Functional Requirements

This feature is a combination of the two functional requirements given in the Add Item and Delete Item feature definitions. For this feature to work, there must be a task in the current task list. Additionally, the gesture recognition library must be included in the dependencies section of the Gradle configurations. If these requirements are met, each item must be provided with the gesture design implementation. Furthermore, the same popup window as for adding an item must be displayed, where the input text boxes are filled with the stored data. A save and cancel button with implemented *OnClickListeners* must also be included. Lastly, the updated information must be written into the database by using the *DataBaseHelper* class.

REQ-2: Layout of Popup Window (*item_popup.xml*)
REQ-3: Ability to Write to Database (*DataBaseHelper.java)*
REQ-4: Implementation of Addina a Task
REQ-5: Import of Gesture Detection Library

## 4.4 Saving Items to Database

### 4.4.1 Description and Priority

A high priority feature of this project is storing the task list so that when reopening the application all items and the corresponding information still exist. In other words, a database has to be configured and implemented.

Benefit: 9
Penalty: 5
Cost: 3
Risk: 1

### 4.4.2 Stimulus/Response Sequences

This feature does not require any direct interaction with the user. Therefore, this feature is implemented as a simple class in the software. The detailed interaction with the database file is not included in the project requirements. Objects and functions from libraries are used for easy access and modification. Whenever one of the listed features is completed by the user, it calls the functions provided by the database helper class which then interacts with the actual database.

### 4.4.3 Functional Requirements

The table in which the data is stored must be created, i.e., the table name, its columns and rows must be defined. Specific functions to create, read, and write to the table must be accessed and used. In addition, each of the above functions performs a different action on the database, meaning that a set of steps must be defined to complete each process. The ID of elements is determined by the row number of the element in the database. All this should be summarized in one class which can be instantiated, and its functions can be called.

REQ-6: Import of *SQLiteDatabase* and *SQLiteDataBaseHelper* Libraries
REQ-7: Interfaces for Access by Other Functions

# 5. Other Nonfunctional Requirements

## 5.1 Performance Requirements

The performance requirements for our *DO IT* application revolve around delivering a smooth and efficient user experience. This entails ensuring that interactions within the app, such as adding, editing, or deleting tasks, are responsive and devoid of noticeable delays. To enhance overall device usability, resource optimization is crucial. The app should be designed to minimize memory usage and avoid causing excessive battery drain. These measures will collectively contribute to a user-friendly and reliable application on Android devices.

## 5.2 Safety Requirements

Ensuring safety is a paramount concern in the development of *DO IT*. We are committed to strict adherence to pertinent safety regulations, safeguarding user data privacy, and introducing robust local data backup measures. The app will operate seamlessly without causing interruptions to critical device functions or user distractions.

## 5.3 Security Requirements

*DO IT* will implement robust security measures to protect user data and ensure privacy. This includes compliance with external regulations and industry standards related to security and privacy. This application will provide users with a secure and private experience. In this version of the app all data is stored locally, giving it the safety of not being accessible through an external server. Security errors would arise if individuals got access to the phone, which however then is not in this project's scope.

## 5.4 Software Quality Attributes

Beyond core functionality, our product places significant importance on two key quality attributes: robustness and scalability. Robustness ensures that the application remains stable and resilient, minimizing crashes or unexpected errors. This focus guarantees a reliable and frustration-free user experience. Simultaneously, we prioritize scalability to accommodate future growth and evolving users' needs. By emphasizing these attributes, *DO IT* aims to provide users with a dependable and adaptable product while also simplifying the development process for our team.

## 5.5 Business Rules

*DO IT* users have complete control over their tasks. Each user can create, edit, and delete their own tasks, with no access to tasks created by other users. By default, tasks are private and can only be seen by the task creator, ensuring user data privacy. Our app simplifies task management, giving users the autonomy to manage their tasks efficiently without administrative roles.

# 6. Other Requirements

## 6.1 Future Deliverables

| | | |
|---|---|---|
| Database Migration | - | Transition to an online cloud database using *SQLOnline*. |
| Language Expansion | - | Currently in English with plans for international language support. |
| Legal Compliance | - | Presently US-based, with expansion into international legal requirements. |
| Code Reusability | - | Focus on maintaining organized, clean, and easily understandable code for seamless developer collaboration and future additions. |

## 6.2 Planned Features and Functionalities

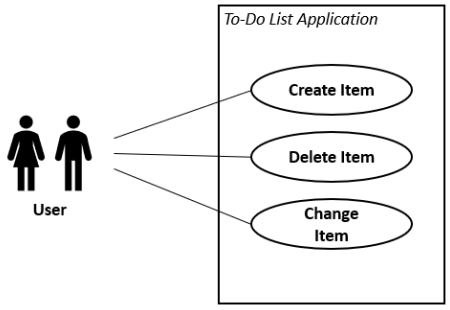
- Task Management Features
    - Due Date and Time: Users can assign due dates and times to tasks or events for better time management.
    - Priority: The application allows users to prioritize tasks, ensuring they can focus on what's most important.
    - Schedule View: Users can view their tasks and events in a calendar format for efficient planning.
    - Notification: Users receive reminders and notifications for upcoming tasks and events.
    - Hide Completed Items: Users can hide or archive completed tasks to maintain a clutter-free task list.
- User Account Features
    - Username, Password, and Email: Users can create and manage their accounts using a username and password and provide an email address for account recovery and communication.
    - Account Management: Users have access to account settings, including profile updates and security configurations.
- Tag and Categorization Features: Users can categorize tasks and events by adding tags, making it easier to organize and search for items.
- Recurring Tasks Feature: Users can set tasks or events to repeat at specified intervals, automating routine activities.
- Collaboration Features: The application enables multiple users to collaborate on tasks or events, promoting teamwork and shared responsibilities.

# Appendix A: Glossary

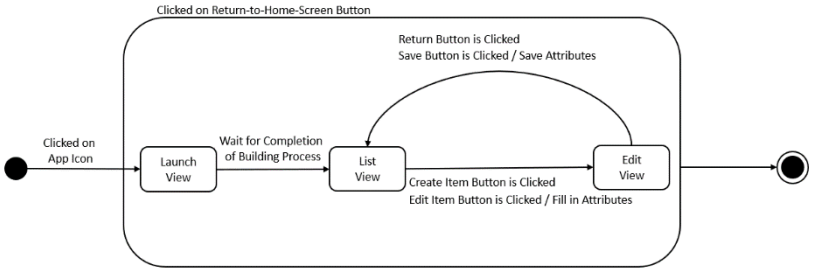
*API: Application Programming Interface*
*GUI: Graphical User Interface*
*OS: Operating System*
*REQ: Requirement*
*SQ: Structured Query*
*UI: User Interface*

# Appendix B: Analysis Models

## B.1. Use Case Diagram




## B.2. UI Design - State Diagram



## B.3 Class Diagram