DIEOR Property Inventory System Manual

Antonio Luis Torres
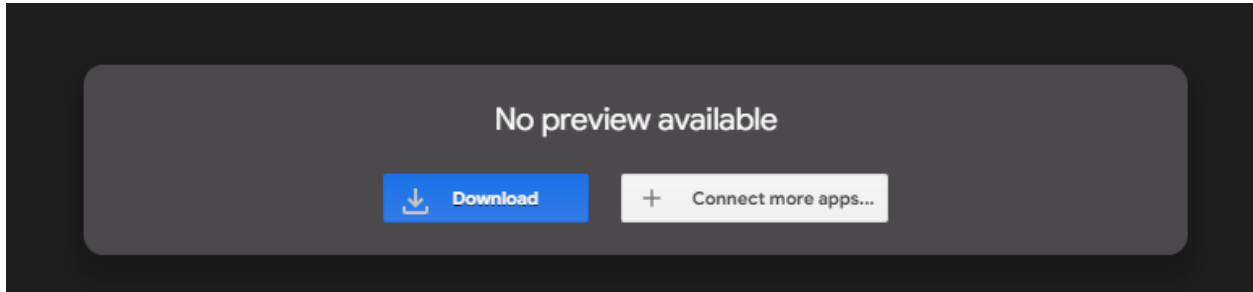
John Joven Juanes

How to download the application?

1. Download the application using this link:

   https://drive.google.com/file/d/1lwiVwfNArTUVCbRYs7_qQrsCUXAJJJ2O/view

2. Click the download button

3. Click "Download anyway"

4. Go to your downloads folder to find the application setup

5. Click "More info:

Windows protected your PC

Microsoft Defender SmartScreen prevented an unrecognized app from starting. Running this app might put your PC at risk.

More info

Don't run

6. Click "Run anyway"

Windows protected your PC

Microsoft Defender SmartScreen prevented an unrecognized app from starting. Running this app might put your PC at risk.

App: Dieor Property Inventory App Setup 1.0.0.exe
Publisher: Unknown publisher

Run anyway   Don't run
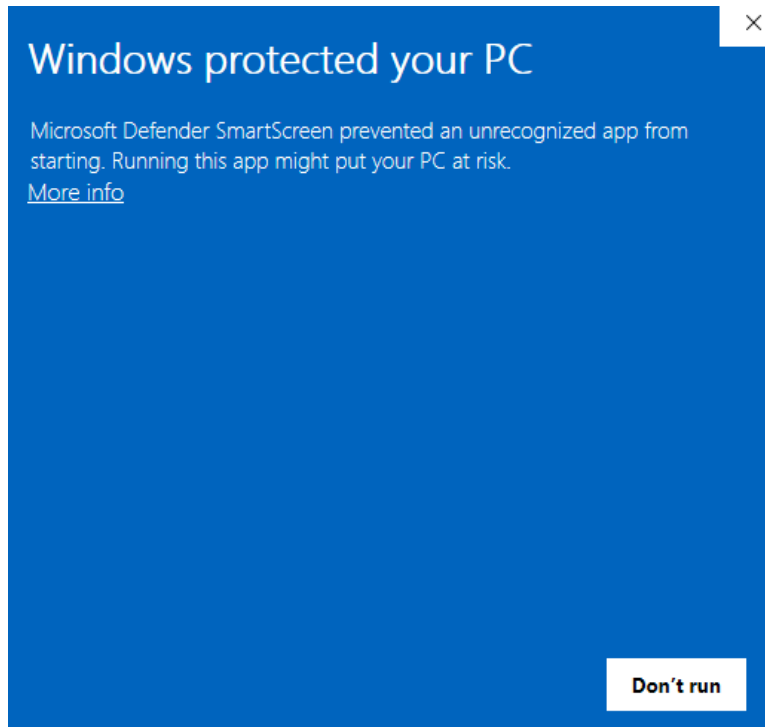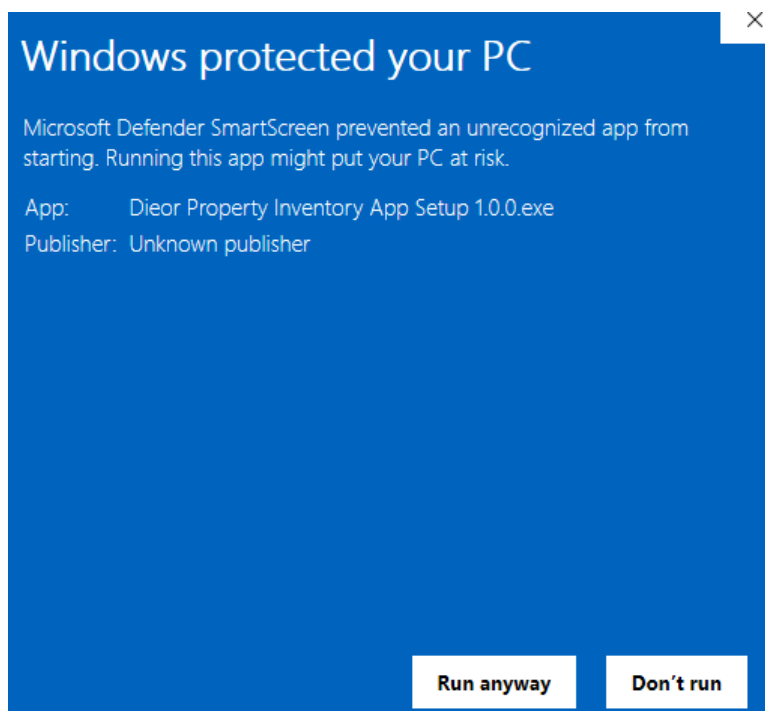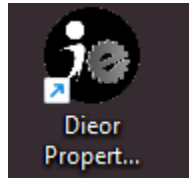
7. The application should open immediately after downloading
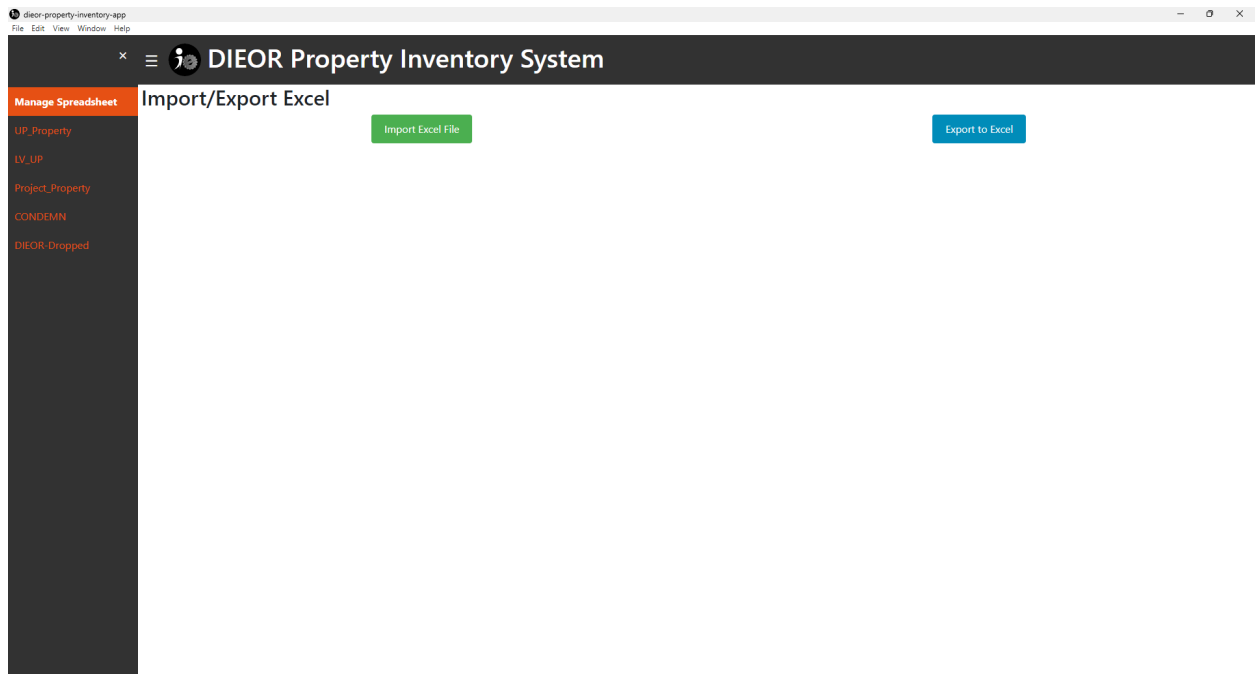
How to open the application?

1. On your desktop, you should find an application with the DIEOR icon

2. Double click on the application icon to open the application

How to Import/Export an Excel File?

1. Go to the Manage Spreadsheet tab

2. Click on the "Import Excel File" button to find and select your excel file to import it to the application.
3. Click on the "Export to Excel" button to export the whole spreadsheet.

How to get the generational report?

1. After importing the excel file, you will see, under the "Manage Spreadsheet" tab, the sheet names of the excel file.
2. Select your desired sheet by clicking

3. Select the filters based on the columns you want to search

4. Type on the search bar the keywords you want to search

5. Click on the "Export Filtered" button to get the generational report

How to add an item?

1. Select the sheet that you want to add an item to
2. Click on the "Add Item" button

3. Fill up the needed information
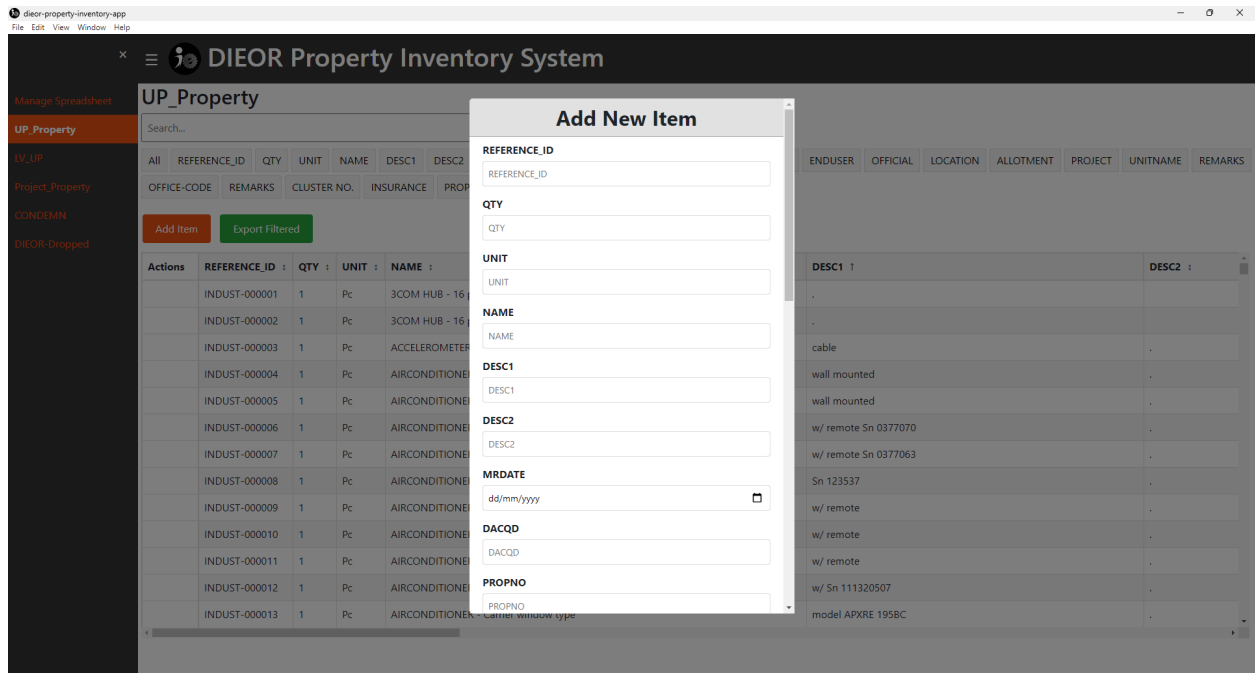
4. Scroll down and select the "add" button to add this entry to the sheet
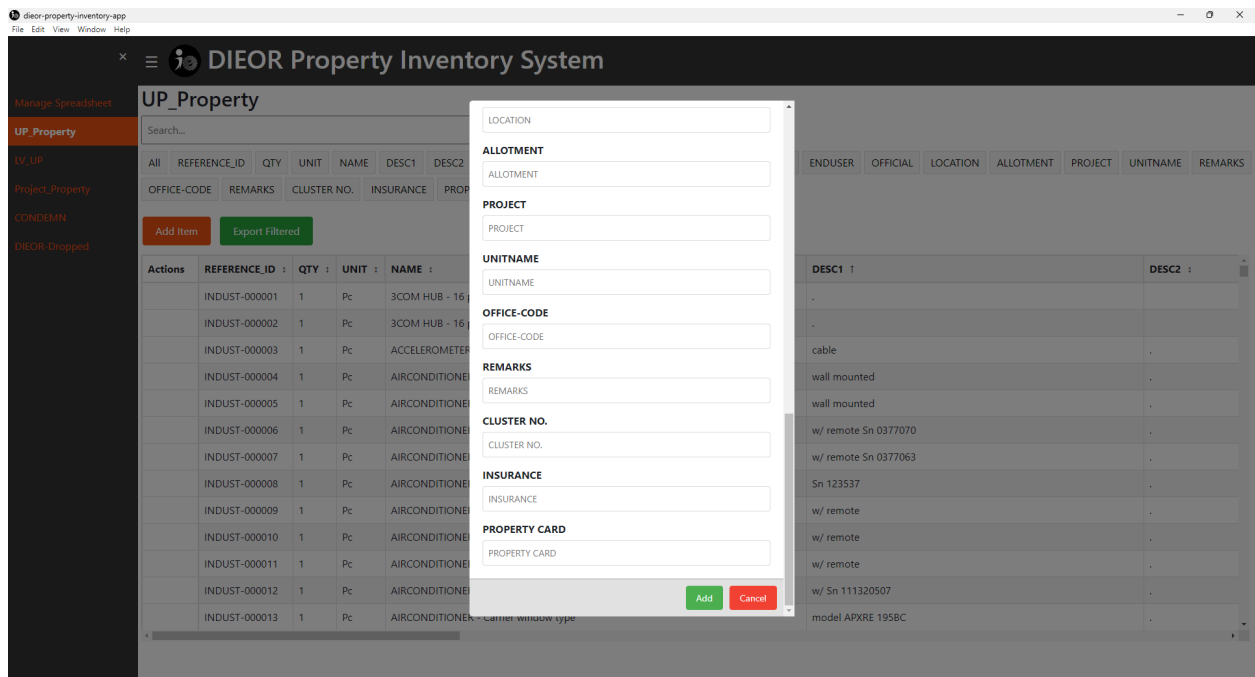
How to edit/delete an entry?

1. The first column of each sheet is the "Action" column
2. Hover your mouse over to reveal the "edit" icon and "delete" icon
3. Clicking the edit icon will bring up this window which is similar to the add item window
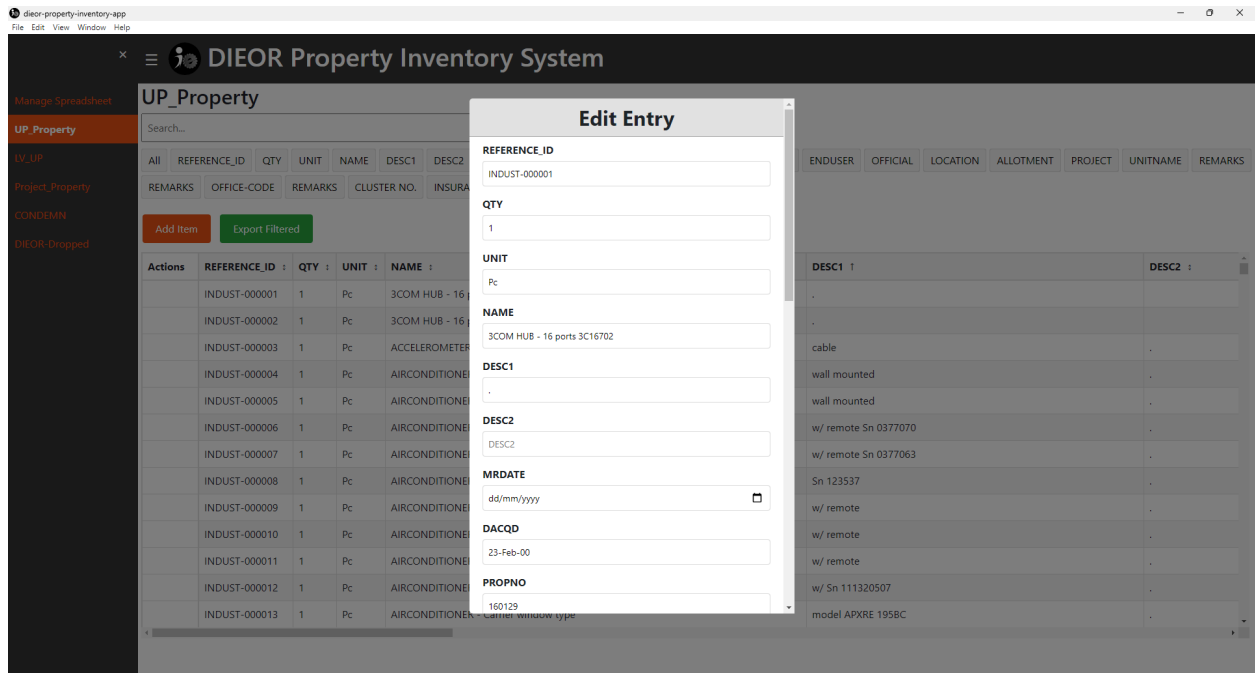
4. Scroll down to confirm your changes
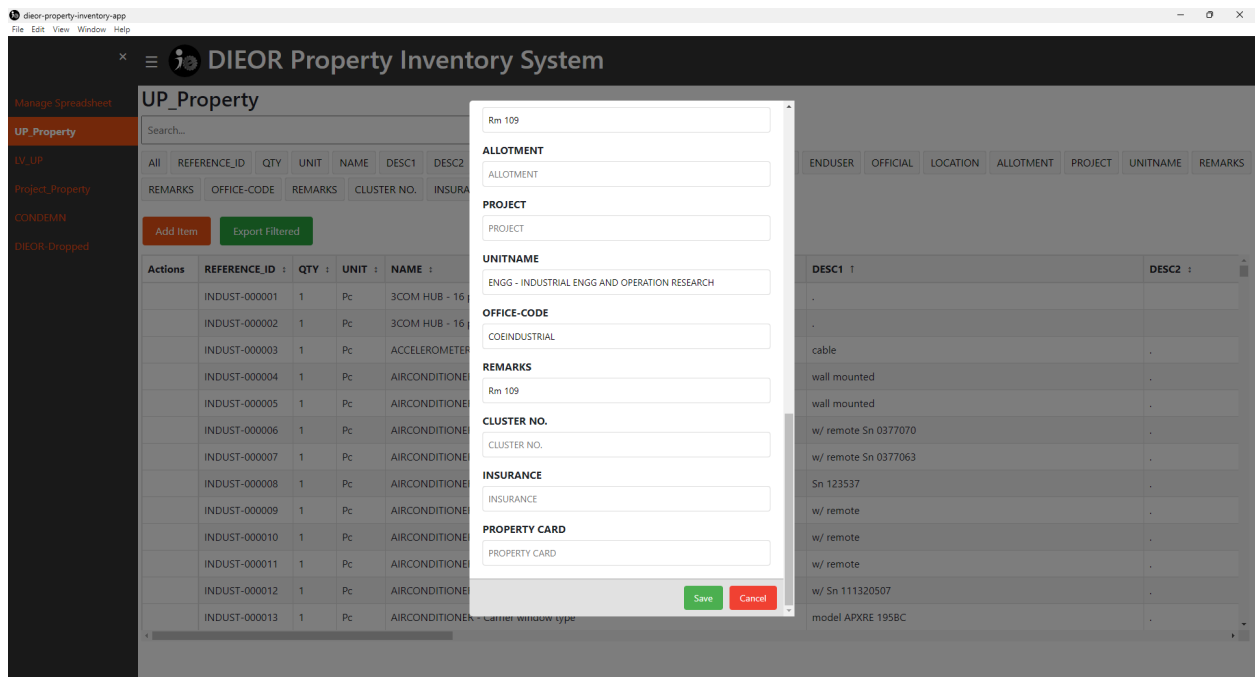
5. While clicking on the delete icon will bring up this window
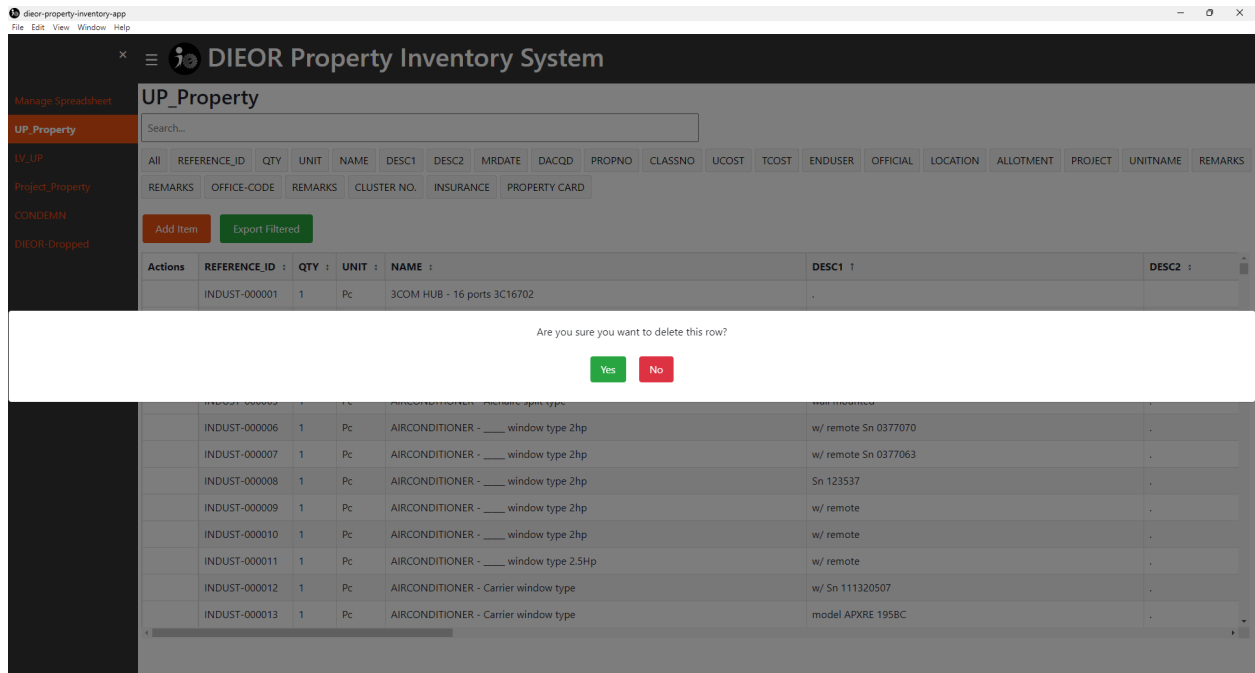
6. Which will confirm whether you want to delete the desired row entry

How to save your progress?

1. Go to the "Manage Spreadsheet" tab
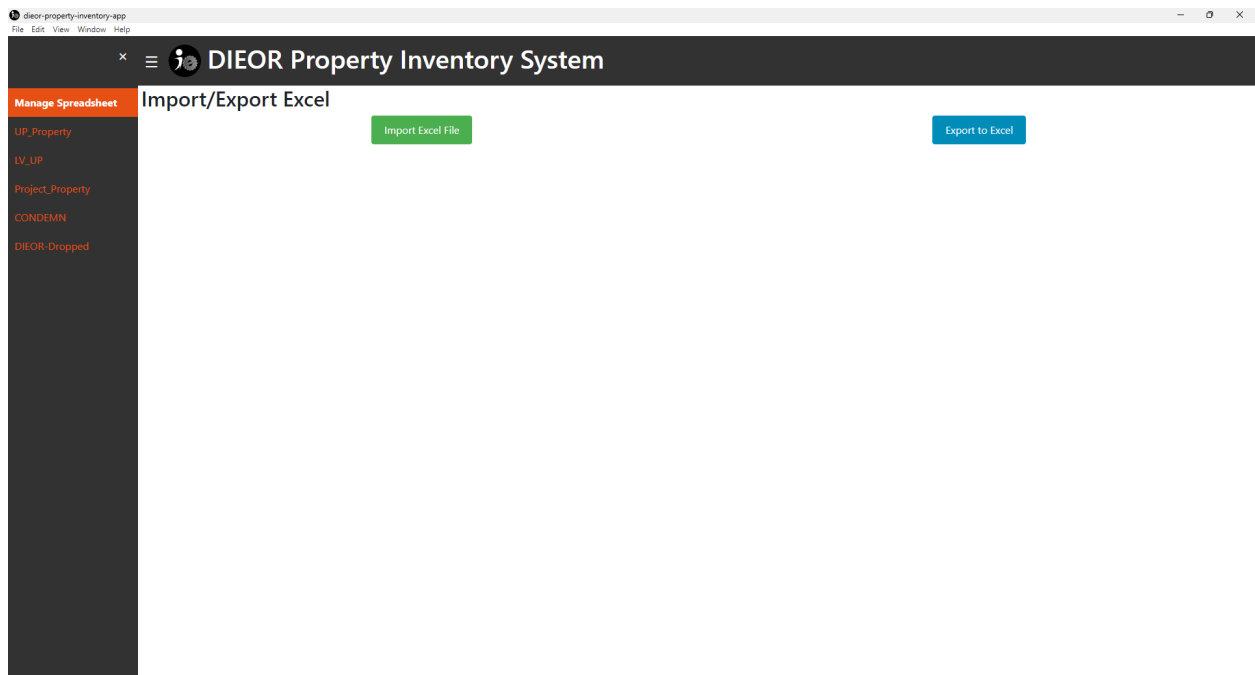
2. Select "Export to Excel"

How to get the source code?

1. Go to our Github link:

   https://github.com/Weejee01/DIEOR-Property-Inventory-System-App/tree/readExcel

   - The most updated version as of today is on the readExcel branch
2. Click the green button "code"
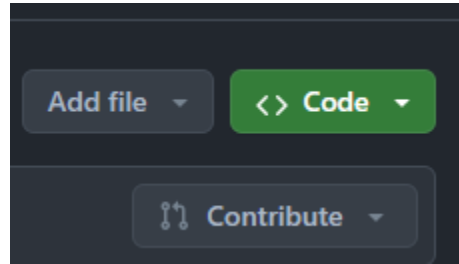
3. Click "Download Zip"
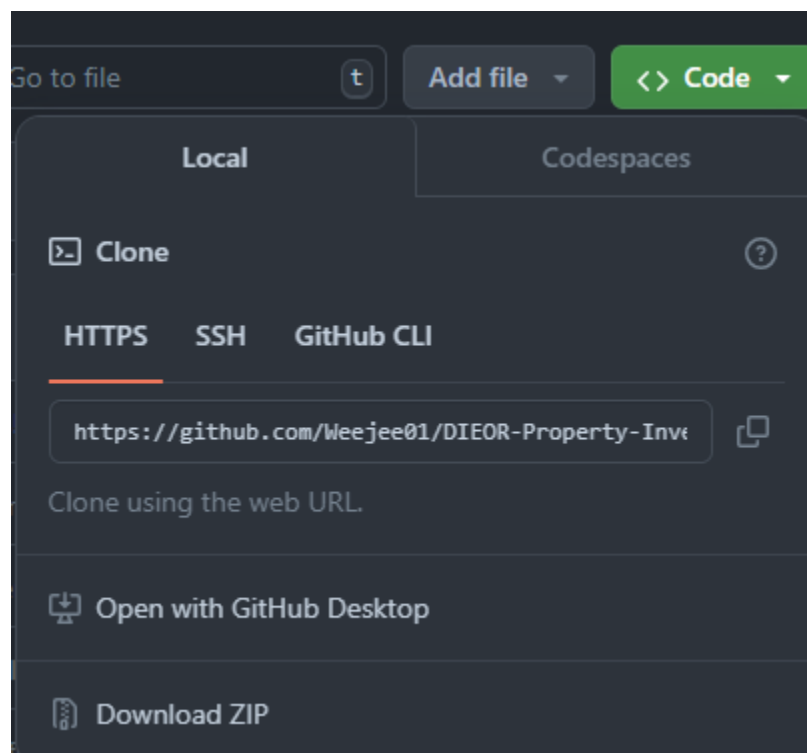
4. Go to your downloads folder to see the zip file

DIEOR-Property-I
nventory-System
-App-readExcel

5. Right click and Extract the DIEOR-Property-Inventory-System-App-readExcel folder

Structure and Initialization of the Frontend

main.js

- **Imports**:
  - Vue's `createApp` to create a new Vue application instance.
  - The main `App` component from `App.vue`.
  - The `router` for handling route navigation.
  - Bootstrap CSS and JS for styling and components.
- **App Initialization**:
  - The Vue app is created and associated with the `App` component.
  - The router is used in the app.
  - The app is mounted to the DOM element with the id `#app`.

App.vue

- **Template**:
  - Contains the main structure of the app with a header, sidebar, and main content area.
  - The header includes a menu icon, a logo, and a title.
  - The sidebar contains the `SideMenu` component and handles the display of the sidebar based on the state of `isSidebarOpen`.

- ○ The main content area uses `router-view` to render the matched component for the active route.
- **Script**:
  - ○ Imports the `SideMenu` component.
  - ○ Defines data properties such as `isSidebarOpen` and `sheets`.
  - ○ Methods include:
    - ■ `toggleSidebar` to open/close the sidebar
    - ■ `handleImportComplete` to handle imported data
    - ■ `selectSheet` to switch between different sheets
    - ■ `loadExistingData` to load data from a JSON file using Electron.
- **Style**:
  - ○ Styles for the header, sidebar, and main content area are provided. It includes transitions for smooth opening/closing of the sidebar and other elements.

Application Structure and Initialization

- **Main Process (`background.js`)**:
  - ○ Manages the lifecycle and behavior of the application windows.
  - ○ Sets up the main `BrowserWindow` with configurations like size, web preferences, and icon.
  - ○ Handles window creation, opening URLs, and app events like `ready`, `window-all-closed`, and `activate`.
  - ○ Uses `@electron/remote/main` to manage remote module usage.
- **Renderer Process (`preload.js`)**:
  - ○ Uses `contextBridge` and `ipcRenderer` to securely expose specific functionality to the renderer process.
  - ○ Allows the renderer to interact with Electron APIs, like file dialogs, file reading, and JSON file handling, without exposing the full Electron API surface.

Key Features and Functionality

- **File Handling**:
  - Provides functionality to read, write, and copy files, as well as save and load JSON data. This includes methods like `save-json-file`, `load-json-file`, and `writeFile`.
  - Handles files through the main process using `fs.promises` for asynchronous file operations.
- **Dialog Interaction**:
  - Exposes methods for showing open and save dialogs via IPC communication, allowing the renderer to request file paths from the user.
- **Environment Handling**:
  - Differentiates between development and production modes using the `isDevelopment` flag. This includes handling the loading of URLs and enabling developer tools in development mode only.
- **Security Considerations**:
  - Implements `contextIsolation` to enhance security by isolating the renderer context from Node.js and Electron APIs.
  - Disables `nodeIntegration` and `enableRemoteModule` (though `@electron/remote` is used), which are critical for maintaining security boundaries.
- **Event Handling**:
  - Handles various application events such as `window-all-closed` (to quit the app on non-macOS platforms) and `activate` (to recreate a window on macOS when the dock icon is clicked).

Components

1. AddItem.vue

- **Template**:

- ○ A modal dialog with a form to add new items.
- ○ It dynamically creates form inputs based on the provided `headers` prop.
- ○ The form inputs are bound to the `newRow` data property, which is initialized with empty values.
- **Script**:
  - ○ Props: `headers` (array of headers to create form fields).
  - ○ Data: `newRow` (an object representing the new item being added).
  - ○ Methods:
    - ■ `initializeNewRow`: Initializes the `newRow` object with empty values based on the headers.
    - ■ `save`: Emits an `add` event with the new row data when the form is submitted.
    - ■ `cancel`: Emits a `cancel` event to close the modal without saving.
    - ■ `getInputType`: Returns the appropriate input type (`text`, `date`, `number`) based on the header name.
- **Style**:
  - ○ Scoped styles for the modal, form, buttons, and other elements.

2. EditExcel.vue

- **Template**:
  - ○ Similar to `AddItem.vue`, but tailored for editing an existing entry.
  - ○ The form inputs are bound to the `editedRow` data property, which is initialized with the values of the `row` prop.
- **Script**:
  - ○ Props: `row` (the existing data row to be edited), `headers` (array of headers for form fields).
  - ○ Data: `editedRow` (a copy of the `row` prop to be modified).
  - ○ Methods:
    - ■ `saveEdit`: Emits a `save` event with the modified row data when the form is submitted.
    - ■ `cancel`: Emits a `cancel` event to close the modal without saving.

- - - **getInputType**: Similar to `AddItem.vue`, returns the appropriate input type based on the header name.
- **Style**:
  - Scoped styles for the modal, form, buttons, and other elements.

3. ImportExcel.vue

- **Template**:
  - Displays buttons for importing and exporting Excel files.
  - Shows a status message based on the current operation.
- **Script**:
  - Imports and parses Excel files using the `XLSX` library.
  - Checks for hidden sheets and only processes visible ones.
  - Converts the data into JSON format and saves it.
  - Provides a method to export the data back to an Excel file, including handling date formatting and column widths.
  - Methods:
    - **selectFile**: Opens a file dialog to allow the user to select an Excel file for import.
    - **parseExcelData**: Parses the contents of the selected Excel file.
    - **saveToJsonFile**: Saves the parsed Excel data as a JSON file.
    - **exportToExcel**: Exports the stored data back to an Excel file.
    - **isDateNumeric**: Checks if a value is a numeric representation of a date.
- **Style**:
  - Provides styling for buttons and status messages.

4. SideMenu.vue

- **Template**:
  - Provides a navigation menu with links to manage spreadsheets and view individual sheets.
- **Script**:

- ○ Accepts `sheets` and `currentSheet` as props to dynamically render the list of sheets and highlight the current one.
- **Style**:
  - ○ Basic styling for navigation links, including hover effects and active state.

5. ViewExcel.vue

- **Template**:
  - ○ Displays the current sheet's data, including options for searching, sorting, and column selection.
  - ○ Provides buttons to add new items and export the filtered data.
  - ○ Includes a table with sortable headers and actions (edit, delete) for each row.
  - ○ Displays a modal for delete confirmation.
- **Script**:
  - ○ Imports components for editing and adding items.
  - ○ Loads sheet data, manages state for the current sheet, and handles user interactions like searching and sorting.
  - ○ Provides methods for editing, deleting, and adding rows.
  - ○ Saves changes to the JSON file and handles exporting filtered data to Excel.
  - ○ Methods:
    - ■ `loadSheetData()`: Loads data for the current sheet from a JSON file and updates the headers and rows.
    - ■ `editRow(rowIndex)`: Sets the row at the given index to be editable.
    - ■ `saveEdit(editedRow)`: Saves the edited row, updates the JSON file, and refreshes the table.
    - ■ `refreshTable()`: Refreshes the table by reassigning the `rows` data property.
    - ■ `cancelEdit()`: Cancels the edit operation, resetting the `editingRow` property.
    - ■ `sortTable(header)`: Sorts the table rows based on the specified header. It toggles between ascending and descending sort order.
    - ■ `getSortIcon(header)`: Returns the sort icon (arrow) for the specified header based on the current sort configuration.

- ■ `confirmDelete(rowIndex)`: Opens a confirmation modal for deleting the row at the given index.
- ■ `deleteRow()`: Deletes the row confirmed for deletion, updates the JSON file, and refreshes the table.
- ■ `cancelDelete()`: Cancels the delete operation and closes the confirmation modal.
- ■ `showAddItemForm()`: Opens the form to add a new item.
- ■ `addItem(newRow)`: Adds a new row to the table, updates the JSON file, and refreshes the table.
- ■ `cancelAdd()`: Cancels the add item operation and closes the form.
- ■ `saveToJsonFile()`: Saves the current headers and rows to the JSON file.
- ■ `performSearch()`: Triggers the search, which is actually handled by the `filteredRows` computed property.
- ■ `toggleColumn(header)`: Toggles the selection of a column header for filtering.
- ■ `toggleAllColumns()`: Selects or deselects all columns for filtering.
- ■ `clearSearch()`: Clears the search query.
- ■ `exportFilteredToExcel()`: Exports the currently filtered rows to an Excel file.
- **Style**:
  - ○ Provides styling for the search bar, column select buttons, table, and modals.
  - ○ Includes features like sticky headers and columns, sortable headers, and action buttons.

Helpful Terminal Commands

- npm install
- npm install v16
- npm use v16
- npm install -g @vue/cli

- npm install @vue/cli-plugin-babel
- npm run serve
- npm run build
- npm run lint
- npm run electron:build
- npm run electron:serve
- npm run postinstall
- npm run postuninstall