```
import streamlit as st
import pandas as pd
import os
from datetime import datetime
from io import StringIO


class Task:
    def __init__(self, task_id, task_name, task_status):
        self.task_id = task_id
        self.task_name = task_name
        self.task_status = task_status


def is_task_name_unique(task_list: list, task_name: str) -> bool:
    # Check if task name is unique in the current task list to ensure correct
deletion/modification of tasks
    return all(task[1].lower() != task_name.lower() for task in task_list)


def add_task(task_list: list, new_task: Task) -> list:
    task_list.append([new_task.task_id, new_task.task_name, new_task.task_status])
    st.success("Task added successfully!")
    return task_list


def complete_task(task_list: list, task_name: str) -> list:
    for task in task_list:
        if task[1] == task_name:
            task[2] = "Completed"
            st.success(f"Task '{task_name}' completed successfully!")
            return task_list
    st.error(f"Task '{task_name}' not found!")
    return task_list


def delete_task(task_list: list, task_name: str) -> list:
    for i, task in enumerate(task_list):
        if task[1] == task_name:
            del task_list[i]
            st.success(f"Task '{task_name}' deleted successfully!")
            return task_list
    st.error(f"Task '{task_name}' not found!")
    return task_list


def change_status(task_list: list, task_name: str, new_status: str) -> list:
    for task in task_list:
        if task[1] == task_name:
            task[2] = new_status
            st.success(f"Task '{task_name}' status changed to {new_status}!")
            return task_list
    st.error(f"Task '{task_name}' not found!")
    return task_list


def export_tasks(task_list: list) -> None:
    df = pd.DataFrame(task_list, columns=["Task ID", "Task Name", "Task Status"])
    filename = f"tasks_{datetime.today().strftime('%Y-%m-%d')}.csv"
    df.to_csv(filename, index=False)
    st.success(f"Tasks exported to {filename}!")


def load_tasks() -> list:
    # Get all CSV files in current directory
    csv_files = [f for f in os.listdir('.') if f.endswith('.csv')]

    if not csv_files:
        st.warning("No CSV files found in current directory")
        return []

    # Let user select which CSV to load
    selected_file = st.selectbox(
        "Select a CSV file to load tasks from",
        csv_files,
        key='csv_file_selector'
    )

    try:
        df = pd.read_csv(selected_file)
        # Validate required columns
        if not all(col in df.columns for col in ["Task ID", "Task Name", "Task Status"]):
            st.error("Selected file doesn't have the required columns (Task ID, Task Name, Task
Status)")
            return []
        return df.values.tolist()
    except Exception as e:
        st.error(f"Error loading {selected_file}: {str(e)}")
        return []


def import_tasks(task_list: list) -> list:
    st.header("Import Tasks from CSV")
    st.warning("This will overwrite your current tasks!")

    uploaded_file = st.file_uploader("Choose a CSV file", type="csv") # limit to just csv input
files

    if uploaded_file is not None:
        try:
            stringio = StringIO(uploaded_file.getvalue().decode("utf-8"))
            df = pd.read_csv(stringio)

            if not all(col in df.columns for col in ["Task ID", "Task Name", "Task Status"]): #
ensure that uploaded file is in the right format
                st.error(
                    "Invalid CSV format. Please upload a file with 'Task ID', 'Task Name', and
'Task Status' columns.")
                return task_list

            new_tasks = df.values.tolist()
            st.session_state.tasks = new_tasks
            st.success(f"Successfully imported {len(new_tasks)} tasks!")
            return new_tasks

        except Exception as e:
            st.error(f"Error importing tasks: {str(e)}")
            return task_list

    return task_list


def main():
    st.title("Task Management Application")

    if 'tasks' not in st.session_state:
        st.session_state.tasks = []

    if st.button("Load Tasks from CSV"):
        st.session_state.tasks = load_tasks()

    with st.sidebar:
        st.header("Actions")
        action = st.selectbox(
            "Choose an action",
            ["Add Task", "Complete Task", "Delete Task", "Change Status", "View Tasks", "Import
Tasks", "Export Tasks"]
        )
    if action == "Add Task":
        st.header("Add New Task")
        with st.form("add_task_form"):
            task_id = str(int(datetime.now().timestamp()))  # Using timestamp as unique ID
            task_name = st.text_input("Task Name", key="task_name_input")
            task_status = st.selectbox(
                "Status",
                ["Not Started", "In Progress", "Completed", "On Hold", "Cancelled"]
            )

            submitted = st.form_submit_button("Add Task")
            if submitted:
                if not task_name:
                    st.error("Task name cannot be empty!")
                elif not is_task_name_unique(st.session_state.tasks, task_name):
                    st.error("Task name must be unique! Please choose a different name.")
                else:
                    new_task = Task(task_id, task_name.strip(), task_status)
                    st.session_state.tasks = add_task(st.session_state.tasks, new_task)

    elif action == "Complete Task":
        st.header("Complete a Task")
        if st.session_state.tasks:
            incomplete_tasks = [task[1] for task in st.session_state.tasks if task[2] !=
"Completed"]

            if incomplete_tasks:
                task_name = st.selectbox(
                    "Select Task to Complete",
                    incomplete_tasks,
                    key="complete_task_select"
                )
                if st.button("Complete Task"):
                    st.session_state.tasks = complete_task(st.session_state.tasks, task_name)
            else:
                st.success("All tasks are already completed!")
        else:
            st.warning("No tasks available to complete!")

    elif action == "Delete Task":
        st.header("Delete a Task")
        if st.session_state.tasks:
            task_name = st.selectbox(
                "Select Task to Delete",
                [task[1] for task in st.session_state.tasks],
                key="delete_task_select"
            )
            if st.button("Delete Task"):
                st.session_state.tasks = delete_task(st.session_state.tasks, task_name)
        else:
            st.warning("No tasks available to delete!")

    elif action == "Change Status":
        st.header("Change Task Status")
        if st.session_state.tasks:
            task_name = st.selectbox(
                "Select Task",
                [task[1] for task in st.session_state.tasks],
                key="change_status_select"
            )
            current_status = next(task[2] for task in st.session_state.tasks if task[1] ==
task_name)

            new_status = st.selectbox(
                "New Status",
                ["Not Started", "In Progress", "Completed", "On Hold", "Cancelled"],
                index=["Not Started", "In Progress", "Completed", "On Hold",
"Cancelled"].index(current_status)
```

```
            )
            if st.button("Update Status"):
                st.session_state.tasks = change_status(st.session_state.tasks, task_name,
new_status)
        else:
            st.warning("No tasks available to update!")

    elif action == "View Tasks":
        st.header("Current Tasks")
        if st.session_state.tasks:
            df = pd.DataFrame(
                st.session_state.tasks,
                columns=["Task ID", "Task Name", "Status"]
            )
            st.dataframe(df[["Task Name", "Status"]].sort_values(by="Status"))

            # Show stats about total, completed and pending tasks at a glance
            col1, col2, col3 = st.columns(3)
            total_tasks = len(df)
            completed_tasks = len(df[df["Status"] == "Completed"])

            col1.metric("Total Tasks", total_tasks)
            col2.metric("Completed", completed_tasks)
            col3.metric("Pending", total_tasks - completed_tasks)
        else:
            st.info("No tasks available. Add some tasks to get started!")

    elif action == "Import Tasks":
        st.session_state.tasks = import_tasks(st.session_state.tasks)

    elif action == "Export Tasks":
        st.header("Export Tasks")
        if st.session_state.tasks:
            if st.button("Export to CSV"):
                export_tasks(st.session_state.tasks)
        else:
            st.warning("No tasks available to export!")


if __name__ == "__main__":
    main()
```