FINAL PROJECT
OBJECT ORIENTED VISUAL PROGRAMMING

Topic:

Task Manager

LECTURER:

Drs. Nurhadi Sukmana, M.Sc

Conducted by:

STUDENTS OF INFORMATICS

GROUP 10

1. **Abdillah Algifary - 001202400076**
2. **Abdullah Faqih R - 001202400219**
3. **LiyangFan - 001202400222**
4. **Kuang Shawn - 001202400230**

PRESIDENT OF UNIVERSITY

Jl. Ki Hajar Dewantara, Kota Jababeka, Cikarang Baru, Bekasi 17550-Indonesia Phone (021) 8910 9762-6. Fax (021) 8910 9768

www.presuniv.com | Email: presuniv@gmail.com

TABLE OF CONTENT

CHAPTER I

INTRODUCTION

OOP (*Object Oriented Programing*) is a programming paradigm based on the concept of "*object*" which can contain data in the form of fields or also known as attributes and code, in the form of functions in this paradigm wrapped in one class or object.

Object-oriented data models can be said to provide more flexibility, ease of changing programs, and are widely used in large-scale software engineering. Furthermore, OOP supporters claim that OOP is easier to learn for beginners compared to other approaches..

Currently, the use of technology in Indonesia is increasing rapidly. And of course it requires a system that can facilitate all our activities. because it is very ineffective if you monitor applications manually. Because in our opinion it certainly complicates which can result in reduced time efficiency. So to make it easier, our group developed a standalone Task Manager machine by implementing the *OOP* concept which is expected to help users and to fulfill our responsibilities for the final assignment of the *Object Oriented Visual Programming* course.

1.1 Project Background

Task Manager is a utility in *Windows operating systems* that allows users to monitor and manage processes, applications, and overall system performance. It provides detailed information about resource usage, including *CPU*, memory, disk, and network activity. Users can also use Task Manager to end unresponsive applications, adjust process priorities, and manage startup programs.

In this project we use the *Google Gson* library, a Java file used to serialize and deserialize Task Manager objects into JSON format. Google Gson (*Gson*) itself was released on May 22, 2008 with their initial version 1.0, Gson is very suitable for implementation into small projects that require a light scale.

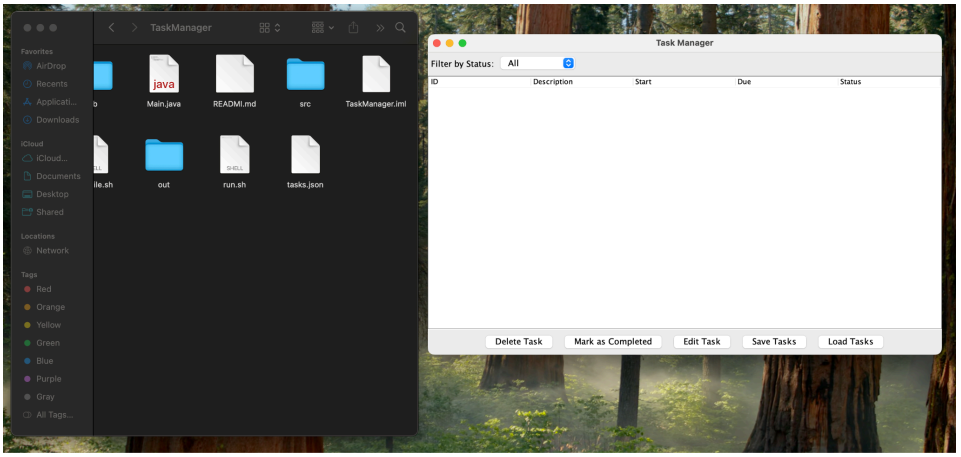
CHAPTER II

CURRENT BUSINESS PROCESS

2.1 Problem Current Process

In Project Task Manager, there are currently several issues that need attention in order to be improved in the next version.

- **Unreadable system**



The current system's ineffectiveness can be attributed to the failure to detect running applications, causing the task manager to not function properly.

- **Difficulty executing compile & run commands**

In our opinion, running the compile and run program commands manually is less efficient. Because with the development of science and technology, we feel that running the command *cd \Users\YourName\FolderName* followed by *javac* FileName.java and *java* FileName.java will waste a lot of time.

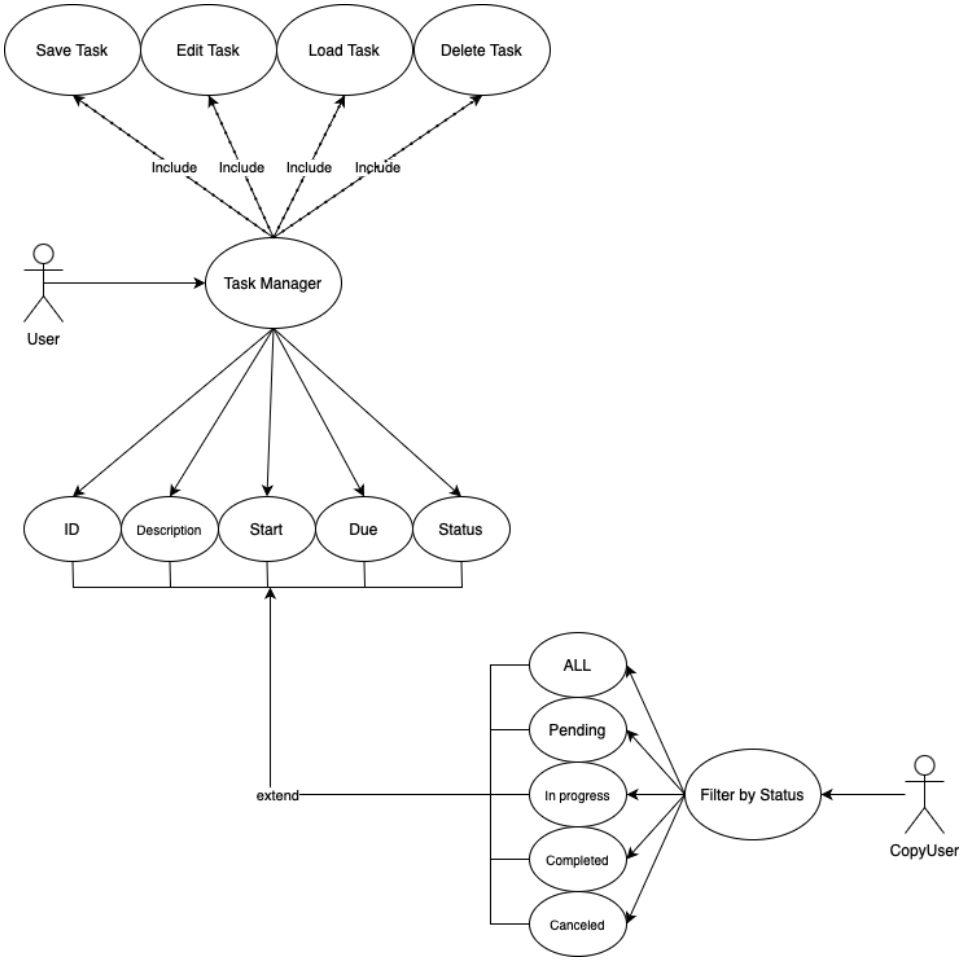
2.2 Solution

- **Creating *shell script* program**

This concerns the commands used to perform automation tasks by creating and running compile.sh and run.sh scripts to create backups and run processes automatically.

2.3 App Overview

- **Diagram**



- **Screenshot**

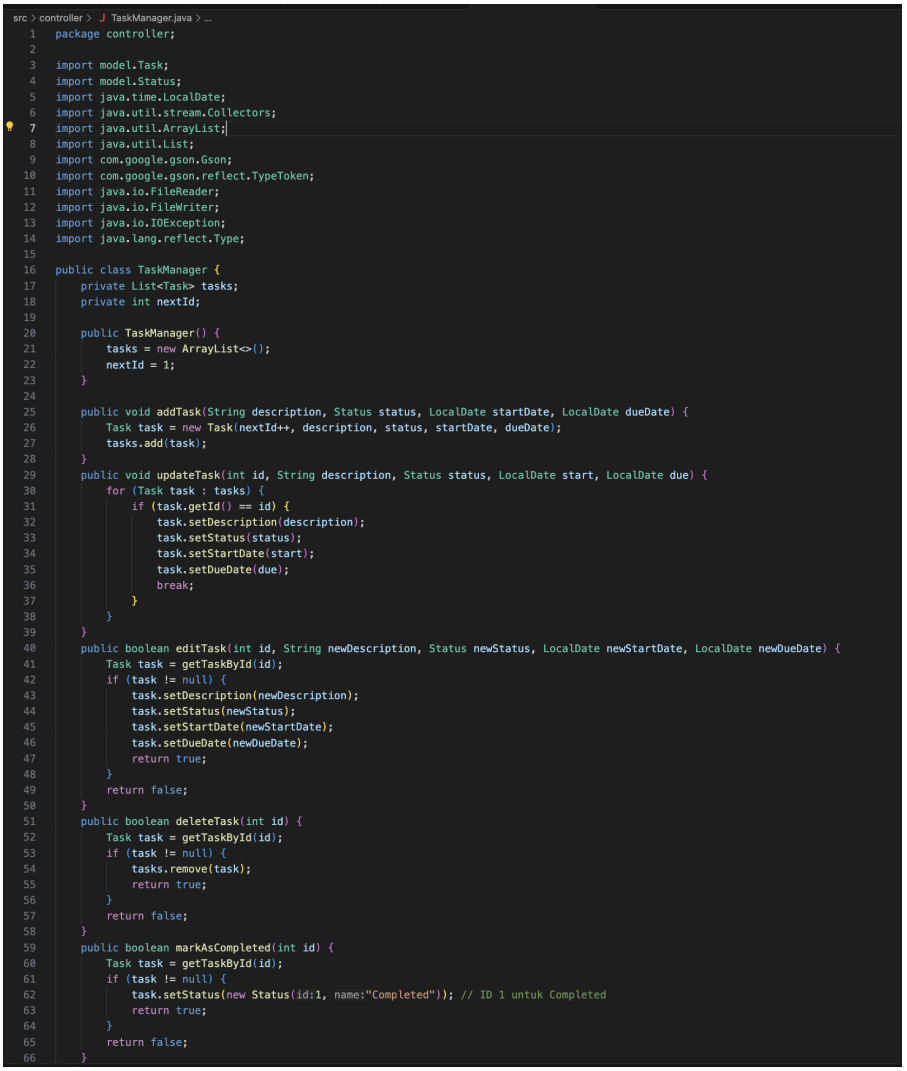
```java
Main.java
import controller.TaskManager;
import view.TaskUI;

import javax.swing.SwingUtilities;

public class Main {
    Run | Debug
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            TaskManager taskManager = new TaskManager();
            new TaskUI(taskManager);
        });
    }
}
```

- *Main.java*

  On lines 1 and 2 to connect the controller from the *TaskManager.java* file and on line 4 to take the utility class in the *Swing* package to provide various helper methods for working with Swing and GUI.



```
src > controller > J TaskManager.java > ...
1   package controller;
2
3   import model.Task;
4   import model.Status;
5   import java.time.LocalDate;
6   import java.util.stream.Collectors;
7   import java.util.ArrayList;
8   import java.util.List;
9   import com.google.gson.Gson;
10  import com.google.gson.reflect.TypeToken;
11  import java.io.FileReader;
12  import java.io.FileWriter;
13  import java.io.IOException;
14  import java.lang.reflect.Type;
15
16  public class TaskManager {
17      private List<Task> tasks;
18      private int nextId;
19
20      public TaskManager() {
21          tasks = new ArrayList<>();
22          nextId = 1;
23      }
24
25      public void addTask(String description, Status status, LocalDate startDate, LocalDate dueDate) {
26          Task task = new Task(nextId++, description, status, startDate, dueDate);
27          tasks.add(task);
28      }
29      public void updateTask(int id, String description, Status status, LocalDate start, LocalDate due) {
30          for (Task task : tasks) {
31              if (task.getId() == id) {
32                  task.setDescription(description);
33                  task.setStatus(status);
34                  task.setStartDate(start);
35                  task.setDueDate(due);
36                  break;
37              }
38          }
39      }
40      public boolean editTask(int id, String newDescription, Status newStatus, LocalDate newStartDate, LocalDate newDueDate) {
41          Task task = getTaskById(id);
42          if (task != null) {
43              task.setDescription(newDescription);
44              task.setStatus(newStatus);
45              task.setStartDate(newStartDate);
46              task.setDueDate(newDueDate);
47              return true;
48          }
49          return false;
50      }
51      public boolean deleteTask(int id) {
52          Task task = getTaskById(id);
53          if (task != null) {
54              tasks.remove(task);
55              return true;
56          }
57          return false;
58      }
59      public boolean markAsCompleted(int id) {
60          Task task = getTaskById(id);
61          if (task != null) {
62              task.setStatus(new Status(id:1, name:"Completed")); // ID 1 untuk Completed
63              return true;
64          }
65          return false;
66      }
```

- *TaskManager.java*

  In line 9 and 10 we try to import *Gson* using the command com.google.Gson. We also create component *ID*, *Description*, *Status*, *Start* to *Due* (you can see in line 25 to 39), After that you will see the CRUD system in our program (see in line 25 to 56). And yes, as you can see we added some features such as *markAsCompleted*, *save TasktoFile* and *loadTaskFromFile*.

```
67
68      public Task getTaskById(int id) {
69          for (Task task : tasks) {
70              if (task.getId() == id) {
71                  return task;
72              }
73          }
74          return null;
75      }
76      public List<Task> getAllTasks() {
77          return tasks;
78      }
79      public List<Task> filterByStatus(String statusName) {
80          return tasks.stream()
81                  .filter(task -> task.getStatus().getName().equalsIgnoreCase(statusName))
82                  .collect(Collectors.toList());
83      }
84      public List<Task> sortByDate(String type) {
85          return tasks.stream()
86                  .sorted((t1, t2) -> {
87                      if (type.equalsIgnoreCase(anotherString:"start")) {
88                          return t1.getStartDate().compareTo(t2.getStartDate());
89                      } else {
90                          return t1.getDueDate().compareTo(t2.getDueDate());
91                      }
92                  })
93                  .collect(Collectors.toList());
94      }
95
96      public void saveTasksToFile(String filename) {
97          try (FileWriter writer = new FileWriter(filename)) {
98              Gson gson = new Gson();
99              gson.toJson(tasks, writer);
100         } catch (IOException e) {
101             e.printStackTrace();
102         }
103     }
104     public void loadTasksFromFile(String filename) {
105         try (FileReader reader = new FileReader(filename)) {
106             Gson gson = new Gson();
107             Type taskListType = new TypeToken<List<Task>>() {}.getType();
108             List<Task> loadedTasks = gson.fromJson(reader, taskListType);
109             if (loadedTasks != null) {
110                 tasks.clear();
111                 tasks.addAll(loadedTasks);
112                 nextId = tasks.stream().mapToInt(Task::getId).max().orElse(other:0) + 1;
113             }
114         } catch (IOException e) {
115             e.printStackTrace();
116         }
117     }
118 }
```

```
67
68      public Task getTaskById(int id) {
69          for (Task task : tasks) {
70              if (task.getId() == id) {
71                  return task;
72              }
73          }
74          return null;
75      }
76      public List<Task> getAllTasks() {
77          return tasks;
78      }
79      public List<Task> filterByStatus(String statusName) {
80          return tasks.stream()
81                  .filter(task -> task.getStatus().getName().equalsIgnoreCase(statusName))
82                  .collect(Collectors.toList());
83      }
84      public List<Task> sortByDate(String type) {
85          return tasks.stream()
86                  .sorted((t1, t2) -> {
87                      if (type.equalsIgnoreCase(anotherString:"start")) {
88                          return t1.getStartDate().compareTo(t2.getStartDate());
89                      } else {
90                          return t1.getDueDate().compareTo(t2.getDueDate());
91                      }
92                  })
93                  .collect(Collectors.toList());
94      }
95
96      public void saveTasksToFile(String filename) {
97          try (FileWriter writer = new FileWriter(filename)) {
98              Gson gson = new Gson();
99              gson.toJson(tasks, writer);
100         } catch (IOException e) {
101             e.printStackTrace();
102         }
103     }
104     public void loadTasksFromFile(String filename) {
105         try (FileReader reader = new FileReader(filename)) {
106             Gson gson = new Gson();
107             Type taskListType = new TypeToken<List<Task>>() {}.getType();
108             List<Task> loadedTasks = gson.fromJson(reader, taskListType);
109             if (loadedTasks != null) {
110                 tasks.clear();
111                 tasks.addAll(loadedTasks);
112                 nextId = tasks.stream().mapToInt(Task::getId).max().orElse(other:0) + 1;
113             }
114         } catch (IOException e) {
115             e.printStackTrace();
116         }
117     }
118 }
```

- *TaskUI.java*

  In this file focused on the user interface, we try to use a simple design to make it easy for anyone to use. But it is likely that we will continue to update again in patch 1.1.0

- *Task.java*

  There may be many questions, how are ID, description, status, etc. managed? Yes, that's right, in this file we manage the system that connects between the view and the controller work.

src > model > J Status.java > ...

```java
package model;

public class Status {
    private int id;
    private String name;
    public Status(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }
    public String getName() {
        return name;
    }

    public void setId(int id) {
        this.id = id;
    }
    public void setName(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }
}
```

- *Status.java*

  As we explained earlier, we have a system for users to filter by status to make it easier for them to see the applications they are looking for.

- **Code**
  - ➢ *Main.java*

```
import controller.TaskManager;
import view.TaskUI;

import javax.swing.SwingUtilities;

public class Main {
   public static void main(String[] args) {
       SwingUtilities.invokeLater(() -> {
           TaskManager taskManager = new TaskManager();
           new TaskUI(taskManager);
       });
   }
}
```

  - ➢ *TaskManager.java*

```
package controller;

import model.Task;
import model.Status;
import java.time.LocalDate;
import java.util.stream.Collectors;
import java.util.ArrayList;
import java.util.List;
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.lang.reflect.Type;

public class TaskManager {
   private List<Task> tasks;
   private int nextId;

   public TaskManager() {
       tasks = new ArrayList<>();
       nextId = 1;
   }
```

```
    public void addTask(String description, Status status, LocalDate
startDate, LocalDate dueDate) {
        Task task = new Task(nextId++, description, status, startDate,
dueDate);
        tasks.add(task);
    }
    public void updateTask(int id, String description, Status status,
LocalDate start, LocalDate due) {
        for (Task task : tasks) {
            if (task.getId() == id) {
                task.setDescription(description);
                task.setStatus(status);
                task.setStartDate(start);
                task.setDueDate(due);
                break;
            }
        }
    }
    public boolean editTask(int id, String newDescription, Status newStatus,
LocalDate newStartDate, LocalDate newDueDate) {
        Task task = getTaskById(id);
        if (task != null) {
            task.setDescription(newDescription);
            task.setStatus(newStatus);
            task.setStartDate(newStartDate);
            task.setDueDate(newDueDate);
            return true;
        }
        return false;
    }
    public boolean deleteTask(int id) {
        Task task = getTaskById(id);
        if (task != null) {
            tasks.remove(task);
            return true;
        }
        return false;
    }
    public boolean markAsCompleted(int id) {
        Task task = getTaskById(id);
        if (task != null) {
            task.setStatus(new Status(1, "Completed")); // ID 1 untuk
Completed
```

```
            return true;
        }
        return false;
    }

    public Task getTaskById(int id) {
        for (Task task : tasks) {
            if (task.getId() == id) {
                return task;
            }
        }
        return null;
    }
    public List<Task> getAllTasks() {
        return tasks;
    }
    public List<Task> filterByStatus(String statusName) {
        return tasks.stream()
                .filter(task ->
task.getStatus().getName().equalsIgnoreCase(statusName))
                .collect(Collectors.toList());
    }
    public List<Task> sortByDate(String type) {
        return tasks.stream()
                .sorted((t1, t2) -> {
                    if (type.equalsIgnoreCase("start")) {
                        return t1.getStartDate().compareTo(t2.getStartDate());
                    } else {
                        return t1.getDueDate().compareTo(t2.getDueDate());
                    }
                })
                .collect(Collectors.toList());
    }

    public void saveTasksToFile(String filename) {
        try (FileWriter writer = new FileWriter(filename)) {
            Gson gson = new Gson();
            gson.toJson(tasks, writer);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public void loadTasksFromFile(String filename) {
```

```
        try (FileReader reader = new FileReader(filename)) {
            Gson gson = new Gson();
            Type taskListType = new TypeToken<List<Task>>() {}.getType();
            List<Task> loadedTasks = gson.fromJson(reader, taskListType);
            if (loadedTasks != null) {
                tasks.clear();
                tasks.addAll(loadedTasks);
                nextId = tasks.stream().mapToInt(Task::getId).max().orElse(0)
+ 1;
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

➢ *Task.java*

.

```
package model;

import java.time.LocalDate;

public class Task {
    private int id;
    private String description;
    private Status status;
    private LocalDate startDate;
    private LocalDate dueDate;

    public Task(int id, String description, Status status, LocalDate
startDate, LocalDate dueDate) {
        this.id = id;
        this.description = description;
        this.status = status;
        this.startDate = startDate;
        this.dueDate = dueDate;
    }

    public int getId() {
```

```
        return id;
    }
    public String getDescription() {
        return description;
    }
    public Status getStatus() {
        return status;
    }
    public LocalDate getStartDate() {
        return startDate;
    }
    public LocalDate getDueDate() {
        return dueDate;
    }

    public void setId(int id) {
        this.id = id;
    }
    public void setDescription(String description) {
        this.description = description;
    }
    public void setStatus(Status status) {
        this.status = status;
    }
    public void setStartDate(LocalDate startDate) {
        this.startDate = startDate;
    }
    public void setDueDate(LocalDate dueDate) {
        this.dueDate = dueDate;
    }

    @Override
    public String toString() {
        return description + " (" + status.getName() + ")";
    }
}
```

➢ *TaskUI.java*

```
package view;

import controller.TaskManager;
import model.Status;
import model.Task;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.time.LocalDate;
import java.util.List;

public class TaskUI extends JFrame {
   private TaskManager taskManager;
   private JTable taskTable;
   private DefaultTableModel tableModel;
   private JTextField descriptionField;
   private JComboBox<Status> statusComboBox;
   private JTextField startDateField;
   private JTextField dueDateField;

   public TaskUI(TaskManager taskManager) {
       this.taskManager = taskManager;
       setTitle("Task Manager");
       setSize(800, 500);
       setLocationRelativeTo(null);
       setDefaultCloseOperation(EXIT_ON_CLOSE);

       initComponents();
       JPanel filterPanel = new JPanel(new
FlowLayout(FlowLayout.LEFT)); // Panel Filter
       JComboBox<String> filterComboBox = new JComboBox<>(new String[]{
           "All", "Pending", "In progress", "Completed", "Canceled"
       });
       filterPanel.add(new JLabel("Filter by Status:"));
       filterPanel.add(filterComboBox);
       filterComboBox.addActionListener(e -> refreshTable((String)
filterComboBox.getSelectedItem())); // Add a listener to a ComboBox
```

```
filter

        add(filterPanel, BorderLayout.BEFORE_FIRST_LINE); // Add to
Layout
        JButton deleteButton = new JButton("Delete Task"); // Delete and
complete buttons
        JButton completeButton = new JButton("Mark as Completed");
        JButton saveButton = new JButton("Save Tasks");
        JButton loadButton = new JButton("Load Tasks");
        JButton editButton = new JButton("Edit Task");
        JPanel buttonPanel = new JPanel();

        saveButton.addActionListener(e -> {
            taskManager.saveTasksToFile("tasks.json");
            JOptionPane.showMessageDialog(this, "Tasks saved!");
        });

        loadButton.addActionListener(e -> {
            taskManager.loadTasksFromFile("tasks.json");
            refreshTable();
            JOptionPane.showMessageDialog(this, "Tasks loaded!");
        });

        deleteButton.addActionListener(e -> deleteSelectedTask());
        completeButton.addActionListener(e -> completeSelectedTask());
        editButton.addActionListener(e -> editSelectedTask());

        buttonPanel.add(deleteButton);
        buttonPanel.add(completeButton);
        buttonPanel.add(editButton);
        buttonPanel.add(saveButton);
        buttonPanel.add(loadButton);

        add(buttonPanel, BorderLayout.SOUTH);



        setVisible(true);
    }
```

```
   private void initComponents() {
        JPanel inputPanel = new JPanel(new GridLayout(2, 5, 10, 10)); //
Input panel

        descriptionField = new JTextField();
        startDateField = new JTextField("2025-05-05");
        dueDateField = new JTextField("2025-05-10");

        statusComboBox = new JComboBox<>(); // ComboBox Status
        statusComboBox.addItem(new Status(1, "Pending"));
        statusComboBox.addItem(new Status(2, "In progress"));
        statusComboBox.addItem(new Status(3, "Completed"));
        statusComboBox.addItem(new Status(4, "Canceled"));

        JButton addButton = new JButton("Add Task");

        inputPanel.add(new JLabel("Description"));
        inputPanel.add(new JLabel("Start Date (YYYY-MM-DD)"));
        inputPanel.add(new JLabel("Due Date (YYYY-MM-DD)"));
        inputPanel.add(new JLabel("Status"));
        inputPanel.add(new JLabel(""));
        inputPanel.add(descriptionField);
        inputPanel.add(startDateField);
        inputPanel.add(dueDateField);
        inputPanel.add(statusComboBox);
        inputPanel.add(addButton);

        // Table
        tableModel = new DefaultTableModel(new String[]{"ID",
"Description", "Start", "Due", "Status"}, 0) {
            @Override
            public boolean isCellEditable(int row, int column) {
                return false; // This means that all table cells cannot
be edited directly
            }
        };
        taskTable = new JTable(tableModel);
        JScrollPane tableScrollPane = new JScrollPane(taskTable);
```

```
        addButton.addActionListener(e -> addTask()); // Add task action

        // Layout
        setLayout(new BorderLayout());
        add(inputPanel, BorderLayout.NORTH);
        add(tableScrollPane, BorderLayout.CENTER);
    }

    private void addTask() {
        try {
            String description = descriptionField.getText();
            LocalDate start = LocalDate.parse(startDateField.getText());
            LocalDate due = LocalDate.parse(dueDateField.getText());
            Status status = (Status) statusComboBox.getSelectedItem();

            taskManager.addTask(description, status, start, due);
            refreshTable();
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this, "Invalid input format!",
"Error", JOptionPane.ERROR_MESSAGE);
        }
    }
    private void deleteSelectedTask() {
        int selectedRow = taskTable.getSelectedRow();
        if (selectedRow >= 0) {
            int taskId = (int) tableModel.getValueAt(selectedRow, 0);
            int confirm = JOptionPane.showConfirmDialog(this, "Are you
sure you want to delete this task?");
            if (confirm == JOptionPane.YES_OPTION) {
                taskManager.deleteTask(taskId);
                refreshTable();
            }
        } else {
            JOptionPane.showMessageDialog(this, "Please select a task to
delete.");
        }
    }
    private void completeSelectedTask() {
        int selectedRow = taskTable.getSelectedRow();
```

```
        if (selectedRow >= 0) {
            int taskId = (int) tableModel.getValueAt(selectedRow, 0);
            taskManager.markAsCompleted(taskId);
            refreshTable();
        } else {
            JOptionPane.showMessageDialog(this, "Please select a task to
mark as completed.");
        }
    }
    private void editSelectedTask() {
        int selectedRow = taskTable.getSelectedRow();
        if (selectedRow >= 0) {
            int taskId = (int) tableModel.getValueAt(selectedRow, 0);

            String newDescription = JOptionPane.showInputDialog(this,
"New Description:");
            String newStartDate = JOptionPane.showInputDialog(this, "New
Start Date (YYYY-MM-DD):");
            String newDueDate = JOptionPane.showInputDialog(this, "New
Due Date (YYYY-MM-DD):");

            Status[] statusOptions = {
                new Status(1, "Pending"),
                new Status(2, "In progress"),
                new Status(3, "Completed"),
                new Status(4, "Canceled")
            };
            Status newStatus = (Status)
JOptionPane.showInputDialog(this, "New Status:",
                    "Edit Status", JOptionPane.QUESTION_MESSAGE, null,
statusOptions, statusOptions[0]);

            try {
                LocalDate start = LocalDate.parse(newStartDate);
                LocalDate due = LocalDate.parse(newDueDate);

                taskManager.updateTask(taskId, newDescription,
newStatus, start, due);
                refreshTable();
```

```
            } catch (Exception e) {
                JOptionPane.showMessageDialog(this, "Invalid input!",
"Error", JOptionPane.ERROR_MESSAGE);
            }
        } else {
            JOptionPane.showMessageDialog(this, "Please select a task to
edit.");
        }
    }

    private void refreshTable() {
        tableModel.setRowCount(0); // clear table
        List<Task> tasks = taskManager.getAllTasks();
        for (Task task : tasks) {
            tableModel.addRow(new Object[]{
                task.getId(),
                task.getDescription(),
                task.getStartDate(),
                task.getDueDate(),
                task.getStatus().getName()
            });
        }
    }
    private void refreshTable(String filterStatus) {
        tableModel.setRowCount(0); // clear table
        List<Task> tasks = taskManager.getAllTasks();
        for (Task task : tasks) {
            String statusName = task.getStatus().getName();
            if (filterStatus.equals("All") ||
statusName.equals(filterStatus)) {
                tableModel.addRow(new Object[]{
                    task.getId(),
                    task.getDescription(),
                    task.getStartDate(),
                    task.getDueDate(),
                    statusName
                });
            }
        }
```

```
    }
    public static void main(String[] args) {
        TaskManager manager = new TaskManager();
        new TaskUI(manager);
    }
}
```

➢ *Status.java*

```
package view;

import controller.TaskManager;
import model.Status;
import model.Task;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.time.LocalDate;
import java.util.List;

public class TaskUI extends JFrame {
    private TaskManager taskManager;
    private JTable taskTable;
    private DefaultTableModel tableModel;
    private JTextField descriptionField;
    private JComboBox<Status> statusComboBox;
    private JTextField startDateField;
    private JTextField dueDateField;

    public TaskUI(TaskManager taskManager) {
        this.taskManager = taskManager;
        setTitle("Task Manager");
        setSize(800, 500);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);

        initComponents();
```

```
        JPanel filterPanel = new JPanel(new
FlowLayout(FlowLayout.LEFT)); // Panel Filter
        JComboBox<String> filterComboBox = new JComboBox<>(new String[]{
            "All", "Pending", "In progress", "Completed", "Canceled"
        });
        filterPanel.add(new JLabel("Filter by Status:"));
        filterPanel.add(filterComboBox);
        filterComboBox.addActionListener(e -> refreshTable((String)
filterComboBox.getSelectedItem())); // Add a listener to a ComboBox
filter

        add(filterPanel, BorderLayout.BEFORE_FIRST_LINE); // Add to
Layout
        JButton deleteButton = new JButton("Delete Task"); // Delete and
complete buttons
        JButton completeButton = new JButton("Mark as Completed");
        JButton saveButton = new JButton("Save Tasks");
        JButton loadButton = new JButton("Load Tasks");
        JButton editButton = new JButton("Edit Task");
        JPanel buttonPanel = new JPanel();

        saveButton.addActionListener(e -> {
            taskManager.saveTasksToFile("tasks.json");
            JOptionPane.showMessageDialog(this, "Tasks saved!");
        });

        loadButton.addActionListener(e -> {
            taskManager.loadTasksFromFile("tasks.json");
            refreshTable();
            JOptionPane.showMessageDialog(this, "Tasks loaded!");
        });

        deleteButton.addActionListener(e -> deleteSelectedTask());
        completeButton.addActionListener(e -> completeSelectedTask());
        editButton.addActionListener(e -> editSelectedTask());

        buttonPanel.add(deleteButton);
        buttonPanel.add(completeButton);
        buttonPanel.add(editButton);
```

```
        buttonPanel.add(saveButton);
        buttonPanel.add(loadButton);

        add(buttonPanel, BorderLayout.SOUTH);




        setVisible(true);
    }

    private void initComponents() {
        JPanel inputPanel = new JPanel(new GridLayout(2, 5, 10, 10)); //
Input panel

        descriptionField = new JTextField();
        startDateField = new JTextField("2025-05-05");
        dueDateField = new JTextField("2025-05-10");

        statusComboBox = new JComboBox<>(); // ComboBox Status
        statusComboBox.addItem(new Status(1, "Pending"));
        statusComboBox.addItem(new Status(2, "In progress"));
        statusComboBox.addItem(new Status(3, "Completed"));
        statusComboBox.addItem(new Status(4, "Canceled"));

        JButton addButton = new JButton("Add Task");

        inputPanel.add(new JLabel("Description"));
        inputPanel.add(new JLabel("Start Date (YYYY-MM-DD)"));
        inputPanel.add(new JLabel("Due Date (YYYY-MM-DD)"));
        inputPanel.add(new JLabel("Status"));
        inputPanel.add(new JLabel(""));
        inputPanel.add(descriptionField);
        inputPanel.add(startDateField);
        inputPanel.add(dueDateField);
        inputPanel.add(statusComboBox);
        inputPanel.add(addButton);

        // Table
        tableModel = new DefaultTableModel(new String[]{"ID",
```

```
"Description", "Start", "Due", "Status"}, 0) {
            @Override
            public boolean isCellEditable(int row, int column) {
                return false; // This means that all table cells cannot
be edited directly
            }
        };
        taskTable = new JTable(tableModel);
        JScrollPane tableScrollPane = new JScrollPane(taskTable);
        addButton.addActionListener(e -> addTask()); // Add task action

        // Layout
        setLayout(new BorderLayout());
        add(inputPanel, BorderLayout.NORTH);
        add(tableScrollPane, BorderLayout.CENTER);
    }

    private void addTask() {
        try {
            String description = descriptionField.getText();
            LocalDate start = LocalDate.parse(startDateField.getText());
            LocalDate due = LocalDate.parse(dueDateField.getText());
            Status status = (Status) statusComboBox.getSelectedItem();

            taskManager.addTask(description, status, start, due);
            refreshTable();
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this, "Invalid input format!",
"Error", JOptionPane.ERROR_MESSAGE);
        }
    }
    private void deleteSelectedTask() {
        int selectedRow = taskTable.getSelectedRow();
        if (selectedRow >= 0) {
            int taskId = (int) tableModel.getValueAt(selectedRow, 0);
            int confirm = JOptionPane.showConfirmDialog(this, "Are you
sure you want to delete this task?");
            if (confirm == JOptionPane.YES_OPTION) {
                taskManager.deleteTask(taskId);
```

```
                refreshTable();
            }
        } else {
            JOptionPane.showMessageDialog(this, "Please select a task to
delete.");
        }
    }
    private void completeSelectedTask() {
        int selectedRow = taskTable.getSelectedRow();
        if (selectedRow >= 0) {
            int taskId = (int) tableModel.getValueAt(selectedRow, 0);
            taskManager.markAsCompleted(taskId);
            refreshTable();
        } else {
            JOptionPane.showMessageDialog(this, "Please select a task to
mark as completed.");
        }
    }
    private void editSelectedTask() {
        int selectedRow = taskTable.getSelectedRow();
        if (selectedRow >= 0) {
            int taskId = (int) tableModel.getValueAt(selectedRow, 0);

            String newDescription = JOptionPane.showInputDialog(this,
"New Description:");
            String newStartDate = JOptionPane.showInputDialog(this, "New
Start Date (YYYY-MM-DD):");
            String newDueDate = JOptionPane.showInputDialog(this, "New
Due Date (YYYY-MM-DD):");

            Status[] statusOptions = {
                new Status(1, "Pending"),
                new Status(2, "In progress"),
                new Status(3, "Completed"),
                new Status(4, "Canceled")
            };
            Status newStatus = (Status)
JOptionPane.showInputDialog(this, "New Status:",
                    "Edit Status", JOptionPane.QUESTION_MESSAGE, null,
```

```
statusOptions, statusOptions[0]);

            try {
                LocalDate start = LocalDate.parse(newStartDate);
                LocalDate due = LocalDate.parse(newDueDate);

                taskManager.updateTask(taskId, newDescription,
newStatus, start, due);
                refreshTable();
            } catch (Exception e) {
                JOptionPane.showMessageDialog(this, "Invalid input!",
"Error", JOptionPane.ERROR_MESSAGE);
            }
        } else {
            JOptionPane.showMessageDialog(this, "Please select a task to
edit.");
        }
    }

    private void refreshTable() {
        tableModel.setRowCount(0); // clear table
        List<Task> tasks = taskManager.getAllTasks();
        for (Task task : tasks) {
            tableModel.addRow(new Object[]{
                task.getId(),
                task.getDescription(),
                task.getStartDate(),
                task.getDueDate(),
                task.getStatus().getName()
            });
        }
    }
    private void refreshTable(String filterStatus) {
        tableModel.setRowCount(0); // clear table
        List<Task> tasks = taskManager.getAllTasks();
        for (Task task : tasks) {
            String statusName = task.getStatus().getName();
            if (filterStatus.equals("All") ||
statusName.equals(filterStatus)) {
```

```
                tableModel.addRow(new Object[]{
                    task.getId(),
                    task.getDescription(),
                    task.getStartDate(),
                    task.getDueDate(),
                    statusName
                });
            }
        }
    }
    public static void main(String[] args) {
        TaskManager manager = new TaskManager();
        new TaskUI(manager);
    }
}
```

CHAPTER III

CONCLUSION

The increasing cases of other Task Managers failing to display Activity on the NPU (Neural Processing Unit), **Task Manager 1.0** is here to provide a solution for users.

Easy installation and a system that is continuously tested and developed. We really hope that this project can continue to be developed and continue to be a stepping stone for technological progress in Indonesia.

The malware uses sophisticated tactics to evade detection, including debugger detection and runtime environment inspection. It is also capable of spreading like a worm through connected devices and network shares, meaning that a single infected computer can quickly compromise an entire network. Cybersecurity experts recommend regular software updates, a strong backup strategy, and employee education on cybersecurity hygiene to minimize the risk of ransomware attacks.