

# Practical Assessment Task

## Phase 2 – Design Document

CHINASA NWOSU

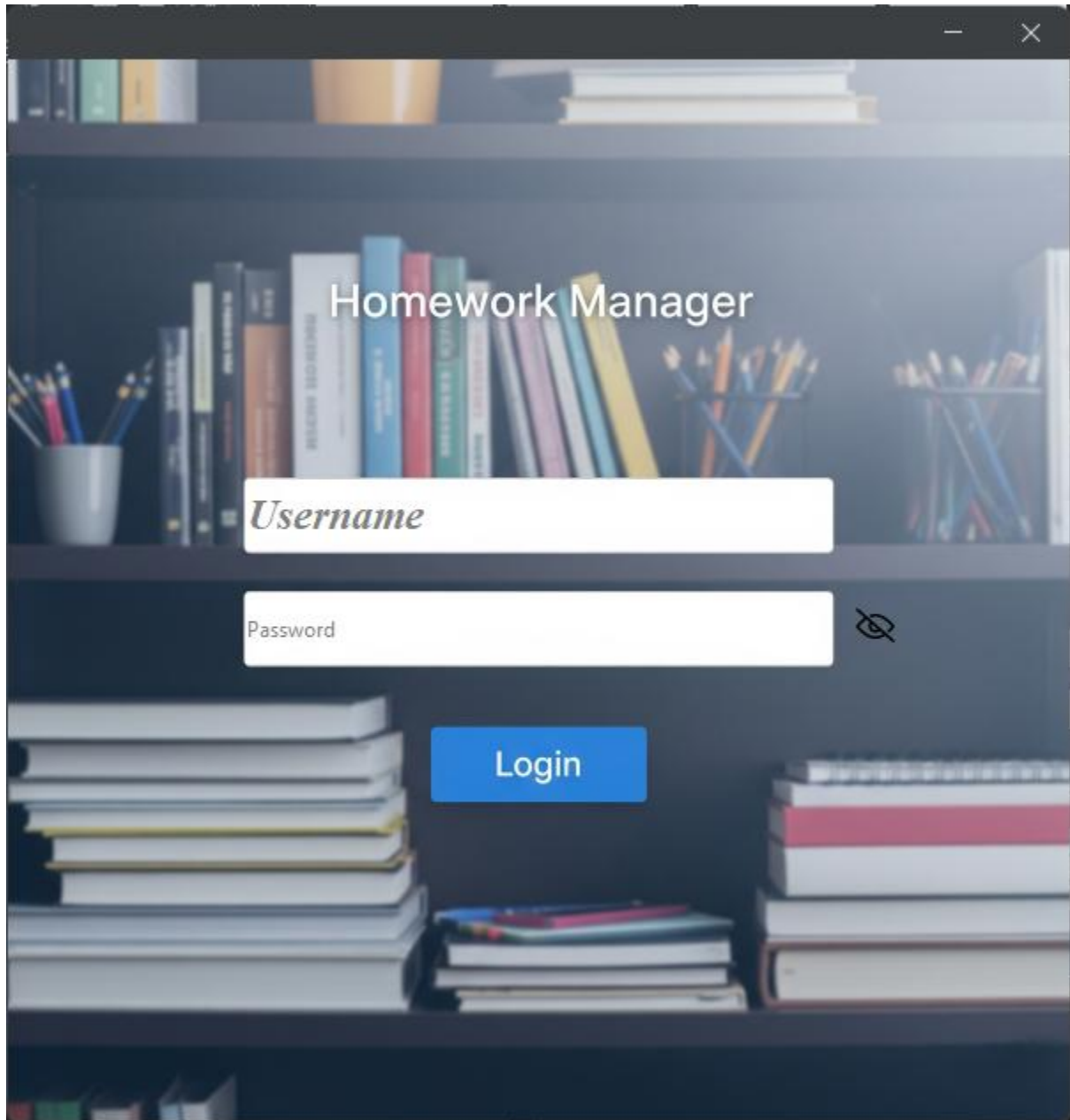
# Table of Contents

Table of Contents.....	2
User Interface Design .....	3
<b>Login Panel</b> .....	3
Login Page .....	3
<b>Main Menu</b> .....	5
Main Menu.....	5
Side Panel .....	6
Dashboard Panel .....	8
Homework Panel .....	10
Progress Panel .....	11
Settings Panel.....	13
<b>Task Frame</b> .....	15
Side Panel .....	15
AddTask Panel.....	17
EditTask Panel.....	19
DeleteTask Panel .....	22
MarkDone Panel .....	24
Program Flow.....	26
Flow Diagram .....	28
Class Design And OOP Principles.....	29
Advanced Data Structures .....	29
OOP Principles .....	30
Secondary Storage Design .....	32
Overview .....	32
logins.txt File Structure .....	32
Tasks.txt File Structure.....	33
Explanation of Secondary Storage Design.....	34
Explanation of how Primary Data Structures relate to Secondary Storage .....	36

# User Interface Design

## Login Panel

### Login Page

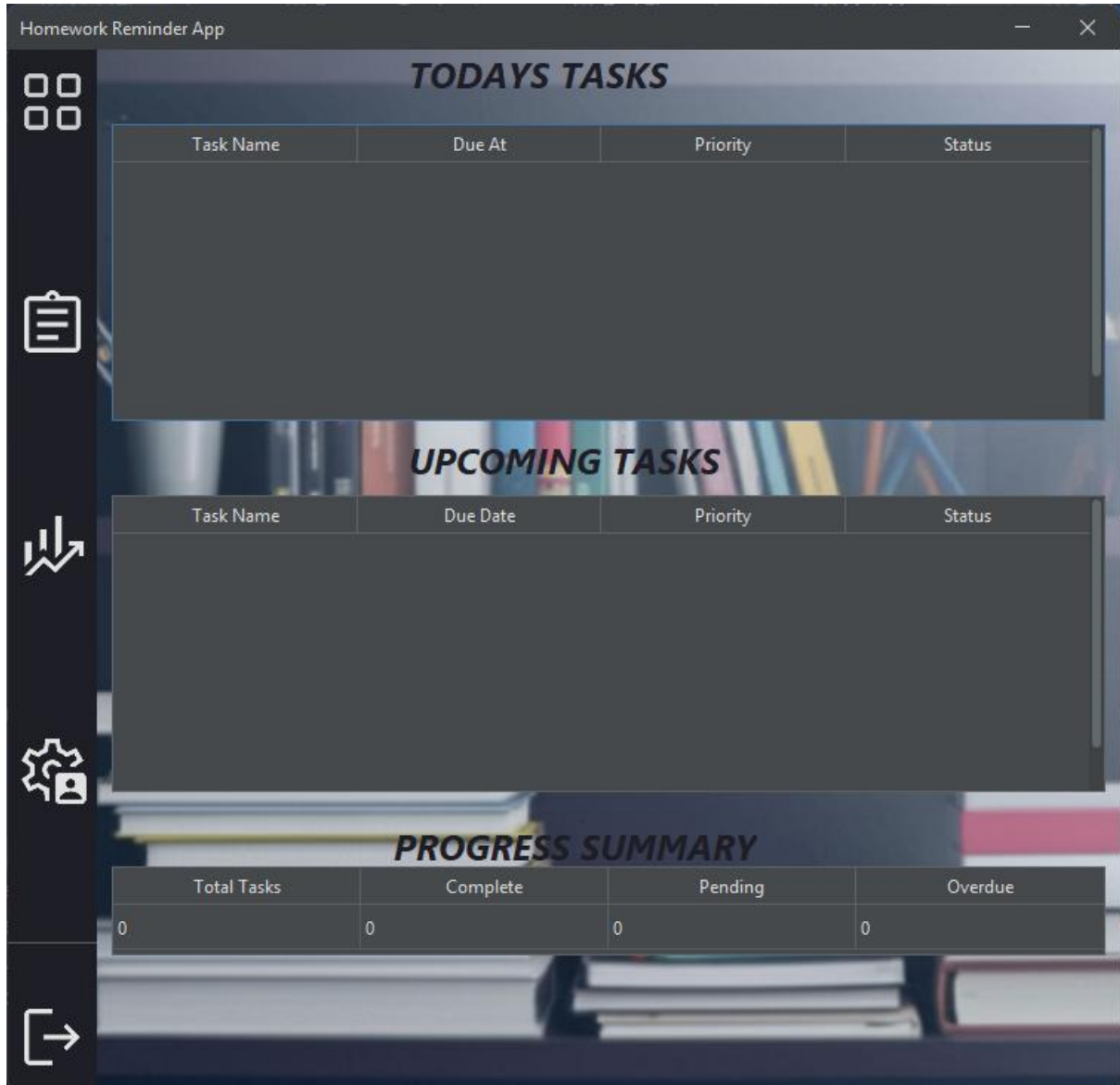


The login page contains 2 input fields where the username and password are input

Component no	Component Type	Component Name	Component Function
1	Text Field	UsernameFld	Field where the user inputs their username
2	Password Field	PasswordFld	Field where the user inputs their password
3	Button	LoginBtn	Buttron logs the user into the app
4	Toggle Button	ToggleBtn	Toggles the visibility of password field
5	Label	Background	Provides the background image for the frame

## Main Menu

### Main Menu



The MainMenu frame is split into 5 panels

Side Panel



Component no	Component Type	Component Name	Component Function
1	Label	DashboardIcon	Provides an icon for the Dashboard Panel, when pressed the Dashboard Panel is made visible
2	Label	HomeworkIcon	Provides an icon for the Homework Panel, when pressed the Homework Panel is made visible
3	Label	ProgressIcon	Provides an icon for the Progress Panel, when pressed the Progress Panel is made visible
4	Label	SettingsIcon	Provides an icon for the Settings Panel, when pressed the Settings Panel is made visible
5	Label	LogoutIcon	Provides an icon for the Logout action, when pressed the user is returned to the Login Page

Dashboard Panel

### TODAYS TASKS

Task Name	Due At	Priority	Status
-----------	--------	----------	--------

### UPCOMING TASKS

Task Name	Due Date	Priority	Status
-----------	----------	----------	--------

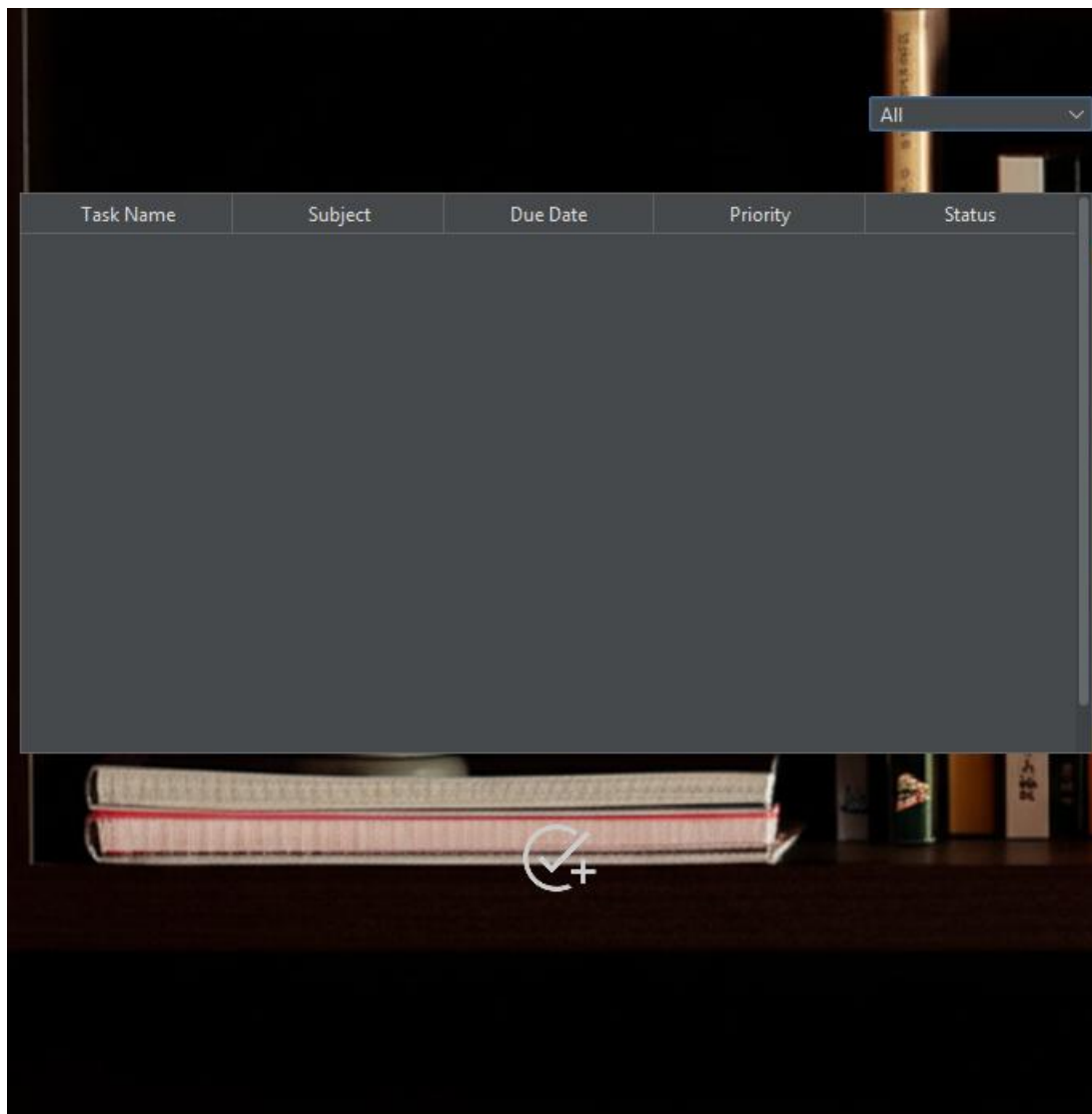
### PROGRESS SUMMARY

Total Tasks	Complete	Pending	Overdue
0	0	0	0



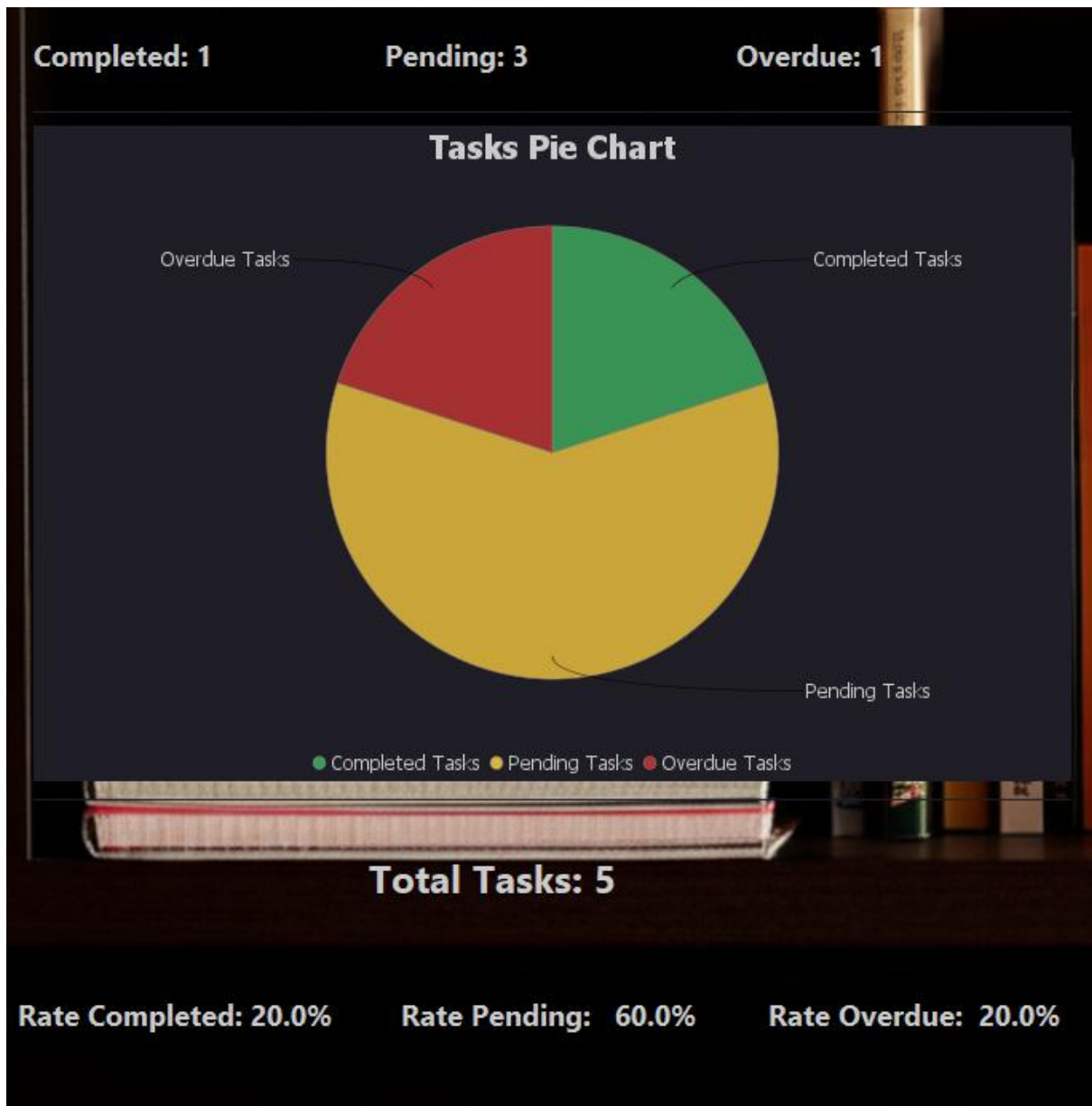
Component no	Component Type	Component Name	Component Function
1	Label	Todays-Tasklbl	Displays The name of the table below
2	Table	Todays-TasksTbl	Displays all the tasks that are due that day
3	Label	Upcoming-Tasklbl	Displays The name of the table below
4	Table	Upcoming-TaskTbl	Displays the rest of the tasks that are not due that day
5	Label	Progress-Sunnarylbl	Displays The name of the table below
6	Table	Progress-SunnaryTbl	Provides a progress summary of the user's tasks

## Homework Panel



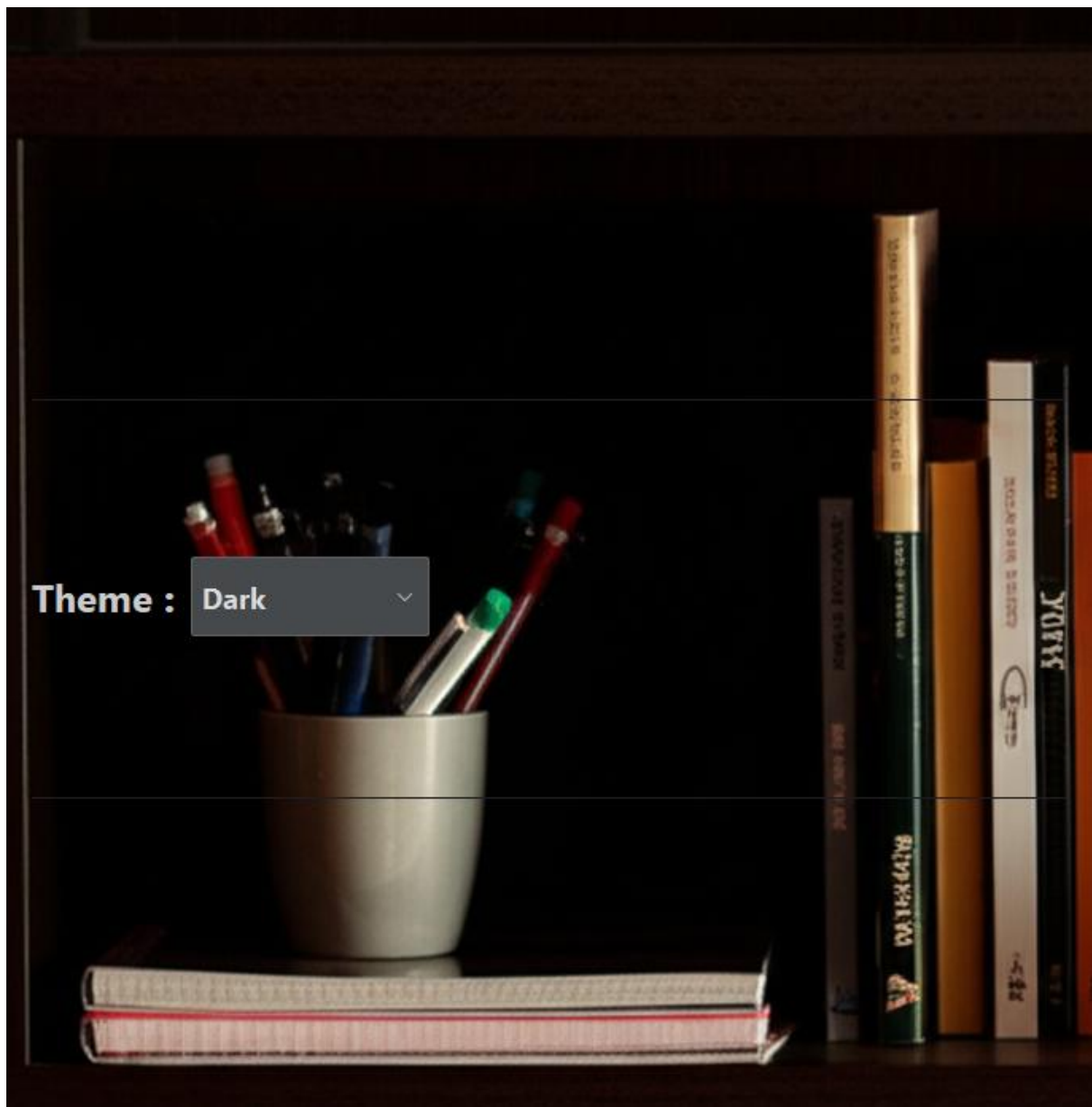
Component no	Component Type	Component Name	Component Function
1	JComboBox	Homework-Filter	Filters the tasks of the user within a time period
2	Table	HomeworkTbl	Displays all the tasks of the user
3	Button	AddTaskBtn	Displays the TaskFrame when pressed

## Progress Panel



3	Label	Overduelbl	Displays the number of tasks overdue by the user
5	Panel	PieChartPnl	Provides a panel in which the pie chart can be displayed
6	Pie Chart	Chart	Provides a graphical interpretation of the user's tasks in the form of a pie chart
7	Label	TotalTasksbl	Displays the total number of tasks the user has
8	Label	RatePending-lbl	Displays the rate at which the tasks are completed as a percentage
9	Label	RatePending-lbl	Displays the rate at which the users tasks are pending as a percentage
10	Label	RateOverdue-lbl	Displays the rate at which the users tasks are overdue as a percentage
11	Label	ProgBG	Provides the Progress Panel with a ackground

## Settings Panel



Component no	Component Type	Component Name	Component Function
1	Label	Themelbl	Displays the name of the JComboBox next to it
2	JComboBox	ThemeSelector	Changes the backgrounds, colour of the icons and text to match the selected theme
3	Label	SettingsBG	Provides a background for the settings panel

## Task Frame

The Task Frame is split into 5 panels

Side Panel



Component no	Component Type	Component Name	Component Function
1	Label	AddTaskIcn	Provides an icon for the AddTask Panel, when pressed the AddTask Panel is made visible
2	Label	EditTaskIcn	Provides an icon for the EditTask Panel, when pressed the EditTask Panel is made visible
3	Label	Delete - TaskIcn	Provides an icon for the DeleteTask Panel, when pressed the DeleteTask Panel is made visible
4	Label	MarkDone-TaskIcn	Provides an icon for the MarkDone Panel, when pressed the MarkDone Panel is made visible



## AddTask Panel

**Task Name :**

**Subject :**

**Due Date :**

**Due Time :**

**Priority :**

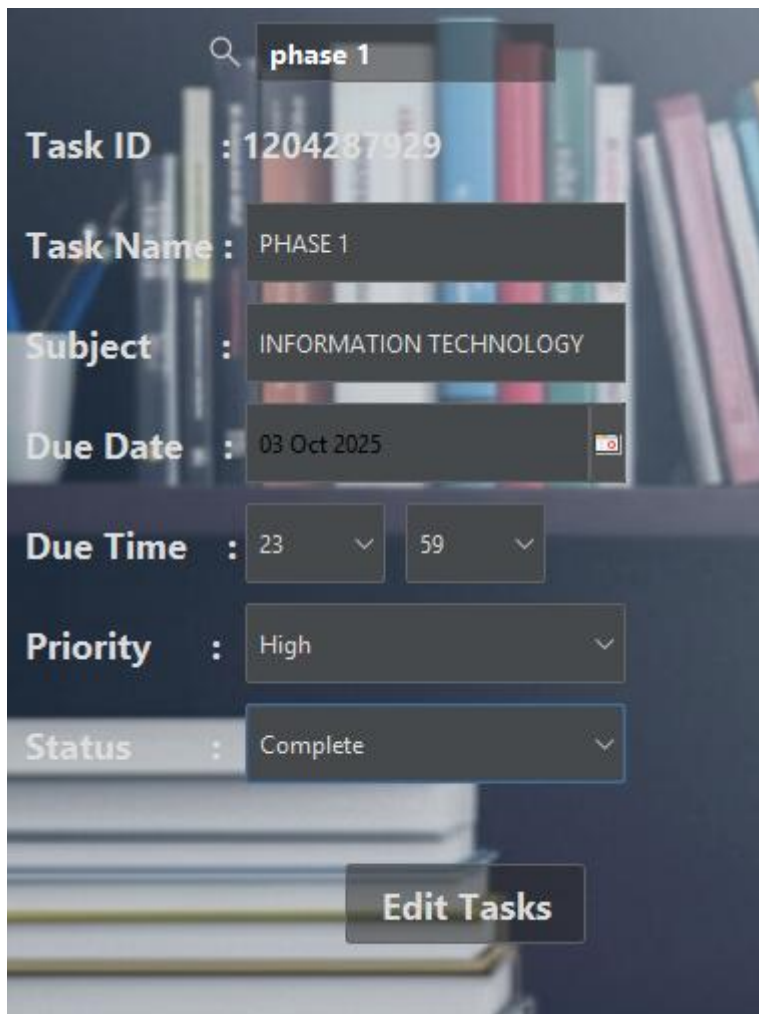
**Status :**

**Add Task**

Component no	Component Type	Component Name	Component Function
1	Label	TaskNameLbl	Displays the name of the text field next to it
2	Text Field	TaskName	Provides a space for the user to input the name of their task
3	Label	SubjectLbl	Displays the name of the text field next to it
4	Text Field	Subject	Displays the name of the text field next to it

5	Label	DueDateLbl	Displays the name of the Date field next to it
6	JDate-Chooser	DueDate	Provides a way for the user to input a due date for the task
7	Label	DueTime	Displays the name of the JComboBoxes next to it
8	JComboBox	HourChooser	Provides a way for the user to enter a due hour for the task
9	JComboBox	Minute-Chooser	Provides a way for the user to enter a due minute for the task
10	Label	PriorityLbl	Displays the name of the JComboBox next to it
11	JComboBox	Priority-Chooser	Provides a way for the user to enter a level of priority for the task
12	Label	StatusLbl	Displays the name of the JComboBox next to it
13	JComboBox	Status-Chooser	Provides a way for the user to enter the status for the task
14	Button	AddTaskBtn	Button adds the task to the text file when pressed


## EditTask Panel

The image shows a web form titled "EditTask Panel" overlaid on a background of a bookshelf. The form contains several input fields and a button. At the top, there is a search bar with a magnifying glass icon and the text "phase 1". Below this, the form displays the following fields: "Task ID" with the value "1204287929", "Task Name" with the value "PHASE 1", "Subject" with the value "INFORMATION TECHNOLOGY", "Due Date" with the value "03 Oct 2025" and a calendar icon, "Due Time" with two dropdown menus showing "23" and "59", "Priority" with a dropdown menu showing "High", and "Status" with a dropdown menu showing "Complete". At the bottom of the form is a button labeled "Edit Tasks".

**Task ID** : 1204287929

**Task Name** : PHASE 1

**Subject** : INFORMATION TECHNOLOGY

**Due Date** : 03 Oct 2025 

**Due Time** :

**Priority** : High

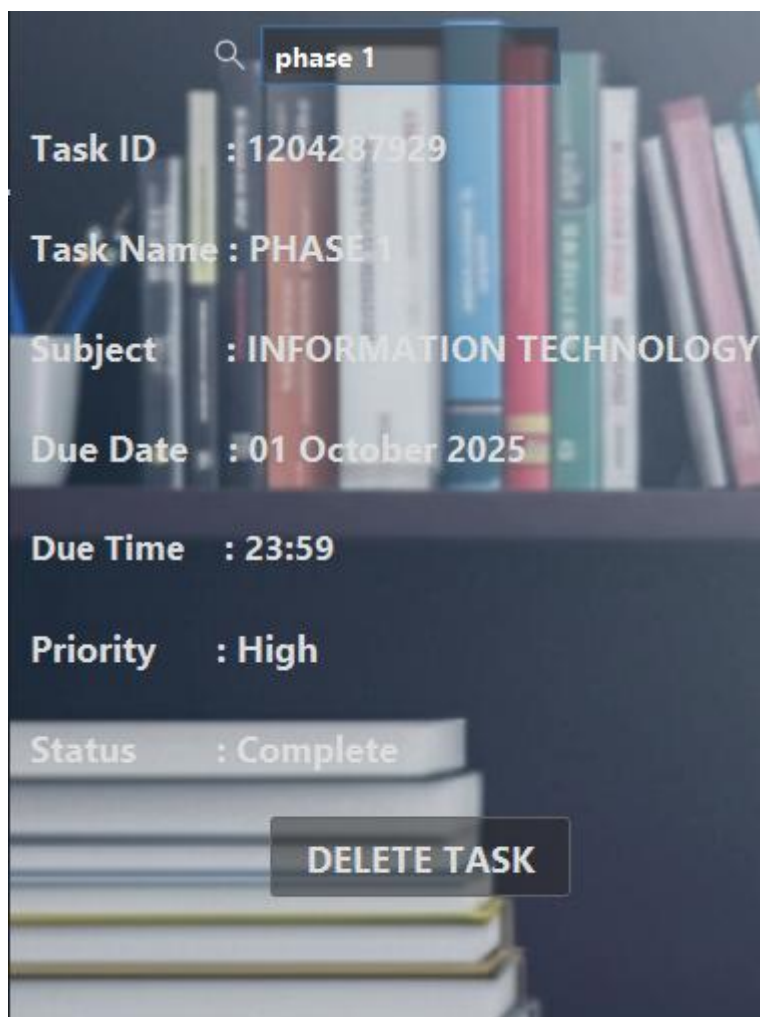
**Status** : Complete

**Edit Tasks**

Component no	Component Type	Component Name	Component Function
1	Button	EditTask-SearchBtn	When pressed the task is searched for, and the text, dates and times are set to that of the task
2	TextField	EditTask-SearchFld	Provides a field for the user to search the name of the task
3	Label	EditTask-IDlbl	Displays the taskID of the task
4	Label	TaskNamelbl	Displays the name of the text field next to it
5	Text Field	TaskName	Provides a space for the user to input the name of their task
6	Label	Subjectlbl	Displays the name of the text field next to it
7	Text Field	Subject	Displays the name of the text field next to it
8	Label	DueDatelbl	Displays the name of the Date field next to it
9	JDate-Chooser	DueDate	Provides a way for the user to input a due date for the task
10	Label	DueTime	Displays the name of the JComboBoxes next to it
11	JComboBox	HourChooser	Provides a way for the user to enter a due hour for the task

12	JComboBox	Minute-Chooser	Provides a way for the user to enter a due minute for the task
13	Label	Prioritylbl	Displays the name of the JComboBox next to it
14	JComboBox	Priority-Chooser	Provides a way for the user to enter a level of priority for the task
15	Label	Statuslbl	Displays the name of the JComboBox next to it
16	JComboBox	Status-Chooser	Provides a way for the user to enter the status for the task
17	Button	EditTaskBtn	Button edits the task to the text file when pressed

## DeleteTask Panel

A screenshot of a mobile application interface for deleting a task. The background is a blurred image of a bookshelf. At the top, there is a search bar with a magnifying glass icon and the text 'phase 1'. Below the search bar, the task details are listed: Task ID : 1204287929, Task Name : PHASE 1, Subject : INFORMATION TECHNOLOGY, Due Date : 01 October 2025, Due Time : 23:59, Priority : High, and Status : Complete. At the bottom, there is a dark grey button with the text 'DELETE TASK' in white capital letters.

phase 1

**Task ID** : 1204287929

**Task Name** : PHASE 1

**Subject** : INFORMATION TECHNOLOGY

**Due Date** : 01 October 2025

**Due Time** : 23:59

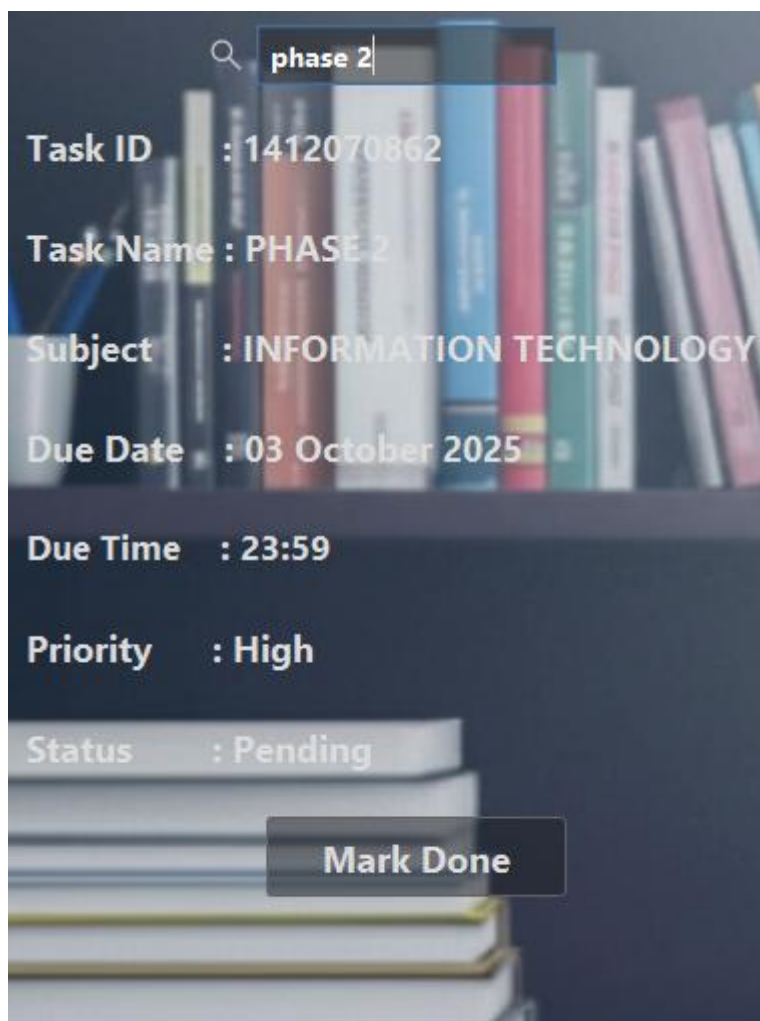
**Priority** : High


**Status** : Complete

**DELETE TASK**

Component no	Component Type	Component Name	Component Function
1	Button	DeleteTask-SearchBtn	When pressed the task is searched for, and the text, dates and times are set to that of the task
2	TextField	DeleteTask-SearchFld	Provides a field for the user to search the name of the task
3	Label	TaskID1	Displays the taskID of the task
4	Label	TaskNamelbl1	Displays the name of the task
5	Label	Subjectlbl1	Displays the subject of the task
6	Label	DueDatelbl1	Displays the due date if the task
7	Label	DueTime1	Displays the due time of the task
8	Label	Prioritylbl1	Displays the priority of the task
9	Label	Statuslbl1	Displays the status of the task
10	Button	Delete-TaskBtn	Button deletes the task to the text file when pressed

## MarkDone Panel

The image shows a mobile application interface for a task management system. At the top, there is a search bar with a magnifying glass icon and the text 'phase 2'. Below the search bar, the task details are displayed in a list format: 'Task ID : 1412070862', 'Task Name : PHASE 2', 'Subject : INFORMATION TECHNOLOGY', 'Due Date : 03 October 2025', 'Due Time : 23:59', 'Priority : High', and 'Status : Pending'. At the bottom of the panel, there is a dark grey button with the text 'Mark Done' in white. The background of the panel is a blurred image of a bookshelf.

 phase 2

**Task ID** : 1412070862

**Task Name** : PHASE 2

**Subject** : INFORMATION TECHNOLOGY

**Due Date** : 03 October 2025

**Due Time** : 23:59

**Priority** : High

**Status** : Pending

**Mark Done**



Component no	Component Type	Component Name	Component Function
1	Button	MarkDone-SearchBtn	When pressed the task is searched for, and the text, dates and times are set to that of the task
2	TextField	MarkDone - SearchFld	Provides a field for the user to search the name of the task
3	Label	TaskID2	Displays the taskID of the task
4	Label	TaskNamelbl2	Displays the name of the task
5	Label	Subjectlbl2	Displays the subject of the task
6	Label	DueDatelbl2	Displays the due date if the task
7	Label	DueTime2	Displays the due time of the task
8	Label	Prioritylbl2	Displays the priority of the task
9	Label	Statuslbl2	Displays the status of the task
10	Button	MarkDone-Btn	Button marks the task done on the text file when pressed

# Program Flow

## Explanation

The program begins at the LoginPage

- The user inputs 'admin' for the username and password
- 

After a successful login, the user is taken to the MainMenu where they can navigate the applications core functions through a sidebar menu **Dashboard, Homework, Progress, and Settings.**

## Main Menu Navigation & Features:

The MainMenu acts as a central hub, using the **Managers.ThemeManager's** panel-switching methods (showDashboardPanel, etc.) to switch between the main views.

## Dashboard

- View a quick summary of tasks
- Uses **Managers.TaskManager** to retrieve **Today's** and **Upcoming Tasks** for display in separate JTables.

## Homework

- View, Search, and Create/Manage all tasks.
- Displays all tasks in a main JTable. Includes a **Search** function using **TaskManager.searchTask()** and a **Task Frame** button to open the **UI.TaskFrame** for task manipulation.

## Progress

- Visualize task completion statistics.
- Calls **Managers.ProgressTracker** to calculate the number of **Completed**, **Pending**, and **Overdue** tasks. Generates and displays a **JFreeChart Pie Chart** based on these stats.

## Settings

- Configure application themes.
- Features a **Theme Selector** (JComboBox) that uses **Managers.DarkTheme** or **Managers.LightTheme** (invoked via `MainMenu.applyTheme()`) to instantly change the application's appearance.

## Task Management Flow (via TaskFrame)

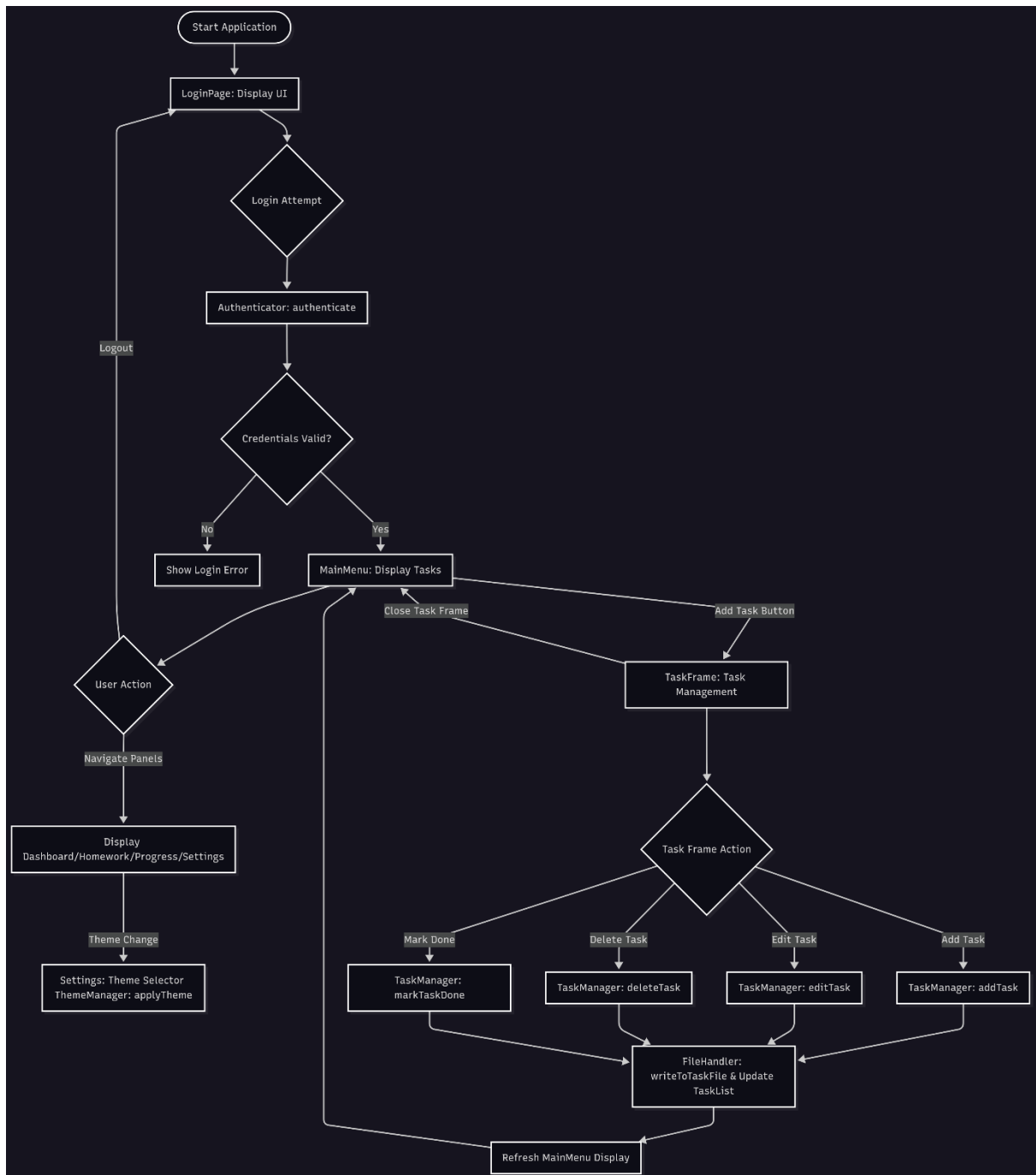
When the user clicks the "Task Frame" button in the **Homework** panel, the **UI.TaskFrame** window opens. This frame allows for all CRUD (Create, Read, Update, Delete) operations on individual tasks and contains dedicated panels for each operation:

TaskFrame Panel	User Action	Manager Method Used
Create Task	Enters task name, subject, date, time, priority, and status.	<code>TaskManager.addTask()</code> is used to save a new Task object
Edit Task	Searches for a task (by ID or name), then updates its details	<code>TaskManager.searchTask()</code> and <code>TaskManager.editTask()</code>
Delete Task	Searches for a task and confirms deletion	<code>TaskManager.deleteTask()</code>
Mark Done	Searches for a task and flags it as complete	<code>TaskManager.markTaskDone()</code>

## Signout

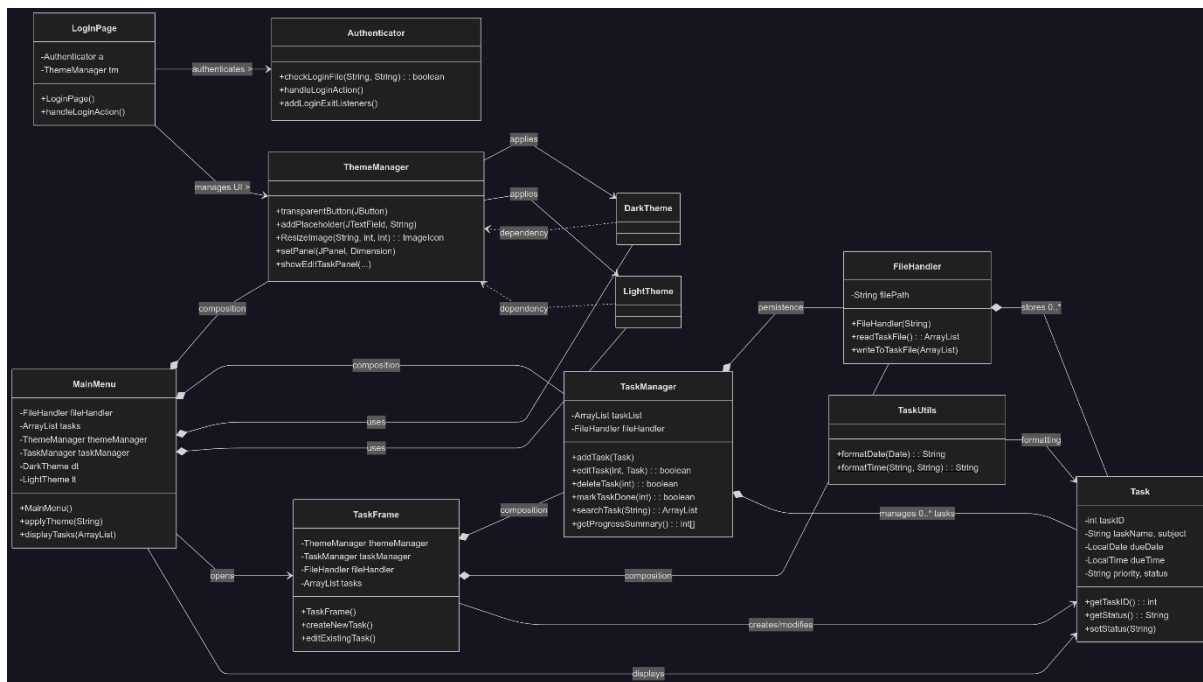
- At any point, the user can click the **Logout icon** in the sidebar.
- This closes the UI.MainMenu and returns the user to the **UI.LoginPage**.

## Flow Diagram



Crated using [Mermaid Chart](#)

# Class Design And OOP Principles



Crated using [Mermaid Chart](#)

# Advanced Data Structures

## 'ArrayList' Usage in Methods

- Some key classes in this design return or manage collections of objects, specifically using the **ArrayList<Task>** data structure.
  - The **FileHandler** class's readTaskFile() method returns an ArrayList<Task>.
  - The **TaskManager** class uses an ArrayList<Task> internally (taskList) and returns an ArrayList<Task> from its searchTask() method.
- While the concept of ArrayList (a resizable-array implementation of the List interface) might not have been formally covered, I have researched and applied it to ensure the program can handle multiple tasks efficiently.
- These methods are designed to **handle multiple records** (tasks) loaded from a file or found during a search in a structured and reusable way, which is a fundamental requirement for a task management application.

# OOP Principles

## Encapsulation

- **All internal fields** in the core classes, such as **Task** (e.g., taskID, taskName, dueDate) and **FileHandler** (e.g., filePath), are declared as **private**.
- This prevents external code from directly modifying internal data.
- The data is accessed and updated exclusively through **public accessor methods** (getters and setters), such as Task.getTaskID() and Task.setStatus(String inStatus), which allows for controlled access and ensures that data remains valid (e.g., the Task constructor validates priority and status strings).

### 3.2.2 Abstraction

- Classes expose only the necessary functions, **hiding the complex internal logic** to simplify usage for other parts of the application.
  - The **Authenticator** class exposes simple methods like handleLoginAction() and checkLoginFile(String, String), hiding the underlying file I/O operations and string parsing logic.
  - The **TaskManager** class exposes high-level methods like addTask(), editTask(), and markTaskDone(). A UI class like MainMenu simply calls taskManager.addTask(t) without needing to know *how* the task is saved to the text file (which is the job of FileHandler).
  - The **TaskUtils** class provides a static abstraction (formatDate(), formatTime()) for common formatting tasks, which is used by UI classes without them needing to know the low-level java.time.format.DateTimeFormatter details.

### 3.2.3 Inheritance

- The current design **does not use inheritance** because each class represents a unique, distinct entity:
  - Task is a data model.
  - Authenticator handles login logic.
  - TaskManager handles task CRUD logic.
  - ThemeManager handles UI appearance.

- **Potential for Extension:** If the project were extended, a common **BaseManager** class could be introduced. All current Manager classes (Authenticator, TaskManager, ThemeManager) could inherit from it to enforce a standard structure for things like logging, configuration file paths, or common utility object instantiation.
- **Current Theme Structure:** The DarkTheme and LightTheme classes are separate and used directly by MainMenu and ThemeManager via composition, which works perfectly for a simple theme switch, but an ITheme interface could be introduced for a more flexible structure if more themes were added in the future.

# Secondary Storage Design

## Overview

This system uses **Plain Text (.txt) files** (Tasks.txt and logins.txt) to permanently store all data. The files are stored locally within the application's file path (txtFiles/). All data is accessed through the **FileHandler** class, which uses string parsing and splitting to read and write records.

The file structure is **flat**, where each line represents a single record. Fields within a record are separated by a **delimiter**. Relationships between data points (like a task's status and its due date) are managed entirely by the **Java application logic** (e.g., within the TaskManager class), as the file itself has no built-in relational features.

## logins.txt File Structure

Field Name	Data Type	Description
Username	Short Text	Login username (String).
Password	Short Text	The corresponding password (String).

**Delimiter:** Fields are separated by a **comma (,)**.

**Usage:** The Authenticator reads this file sequentially to validate credentials during login (checkLoginFile()).



## Tasks.txt File Structure

Field Name	Data Type	Description
TaskID	Integer	Unique integer identifier for the task.
TaskName	Short Text	Name/description of the homework/task
Subject	Short Text	The school subject/module the task relates to.
DueDate	Date	Due date of the task, stored as dd MM yyyy.
DueTime	Time	Due time of the task, stored as HH:mm.
Priority	Short Text	Task urgency (e.g., "High," "Medium," "Low").
Status	Short Text	Current state (e.g., "Pending," "Complete," "Overdue").

**Delimiter:** Fields are separated by a **hash symbol (#)** as defined in FileHandler.java (scLine.split("#")).

**Usage:** The file is fully loaded into memory (ArrayList<Task>) at startup by FileHandler and overwritten entirely when changes are saved by TaskManager.

## Explanation of Secondary Storage Design

The **Plain Text File** design was chosen for multiple reasons, considering the scope of a small, single-user application:

- **Simplicity and Portability:** TXT files require **no external database server, SQL, or complex driver configuration** (unlike MS Access or MySQL), simplifying setup and ensuring the application can run anywhere with minimal dependencies.
- **Ease of Implementation:** It directly aligns with the object-oriented structure of Java, where the data (a line of text) is easily **read, parsed, and mapped** to a simple Task object. Using a simple delimiter (#) is straightforward to manage in the FileHandler class.
- **Sufficient Scale:** For a single user with a few hundred tasks, the overhead of loading and saving the entire file at once is negligible, making it an acceptable and performant solution for a project of this size.

Feature	Plain Text File (Tasks.txt)	Relational Database (e.g., MS Access)
Data Integrity	<b>Low.</b> No built-in validation or constraint checking (e.g., guaranteeing a <b>TaskID</b> is unique). All validation must be done in Java code.	<b>High.</b> Provides primary keys, foreign keys, and data type checks to enforce consistency.
Querying	<b>Difficult.</b> Requires manually iterating through the entire list in Java and writing complex filtering loops (e.g., in <code>TaskManager.searchTask()</code> ).	<b>Easy/Powerful.</b> Uses SQL to retrieve specific, filtered, and aggregated data efficiently.
Relationships	<b>Non-existent (Flat).</b> All "relationships" (like marking a task as "Overdue" based on the DueDate) are logic that must be coded explicitly within the TaskManager.	<b>Built-in.</b> Uses foreign keys to link data across separate tables (normalization).
Implementation Effort	<b>Minimal.</b>	<b>High.</b> Requires configuration, drivers, and writing SQL statements.

The design chosen ensures reliable access to the data through the FileHandler class and suits the needs of this application by prioritizing simplicity and self-containment.

# Explanation of how Primary Data Structures relate to Secondary Storage

Each Java backend class is designed to manage the data flow between the in-memory objects and the file system, ensuring persistence.

Class	Related File(s)	Data Flow
<b>Task</b>	Tasks.txt	A <b>Model Class</b> that represents a single line (record) from the Tasks.txt file in memory. Its fields (e.g., taskID, taskName, dueDate) directly map to the delimited tokens.
<b>FileHandler</b>	Tasks.txt, logins.txt	<b>Dedicated I/O Interface.</b> Fetches raw data from the file, <b>parses</b> the delimited strings, and constructs the corresponding Java objects (Task). It also serializes the objects back into the delimited string format for writing to the file.
<b>TaskManager</b>	Tasks.txt	Holds the entire contents of Tasks.txt as an <b>ArrayList&lt;Task&gt;</b> . It performs all CRUD (Create, Read, Update, Delete) operations on the list and delegates the final <b>Save</b> operation to the FileHandler.
<b>Authenticator</b>	logins.txt	Reads from and writes to logins.txt. It fetches login records and compares them to user input using string comparison.

## Model Classes vs. Manager Classes

- **Model Classes (Task):** Represent the **structure** of the data. The fields in the Task object directly correspond to the order and data type of the tokens in the Tasks.txt record.
- **Manager Classes (TaskManager, FileHandler):** Handle the **logic**. They are responsible for retrieving data from the file, converting it into a structured object (Task), and performing all necessary business logic (e.g., calculating task progress in. This separation ensures a clean and maintainable architecture.