ZenTime Design Document



### **Table Of Contents**

1. **Introduction**

**1.1** Project Objectives

**1.2** Project Scope

1. **Program Specification**

**2.1** Scheduling System

**2.2** Reminder System

**2.3** Progress Reports

**2.4** Event Handling & Data Processing

**2.5** System Tray & Desktop Notifications

**2.6** Data Management

1. **Graphical User Interface Designs**

**3.1** Scheduling System GUIs

**3.2** Alarm System GUI

**3.3** Progress Reports GUI

1. **Program Structure**

**4.1** Design Pattern Diagram

**4.2** Class & Object Breakdown

1. **Program Requirements & Test Cases**
2. **Project Schedule**

**6.1** Task Breakdown List

**6.2** Kanban Board

**6.3** Gantt Charts

**6.4** Project Milestones

### **Introduction**

This document describes the design of a time and productivity management program named ZenTime, which is to be used by students. The program is intended to be used for personal work, school-work, and other personal tasks to assist the end-users in managing time and boosting productivity.

**1.1 Project Objectives**

ZenTime will provide a number of features and key deliverables to assist its end-users with coordinating their time and completing their tasks. The program’s central feature will be a scheduling system which will allow the user to schedule their tasks and other important work deadlines. Additionally, each task will be able to be accessed and viewed through a list on a visually appealing and intuitive graphical user interface for the user’s ease of access to the product.

The user will be able to enable or disable desktop notification for deadlines or tasks listed on their schedule in advance of their occurrences to assist with completing their tasks in a timely manner. Furthermore, they will be able to use a reminder system which is intended to be used to set an alarm-like scheduled desktop notification which will send the user a desktop notification for their reminder(s). This will assist the user with remembering their deadlines and ensuring they are able to complete their tasks on time.

The user’s task progress will be tracked according to the status of the tasks on their schedule. Specific progress updates and task completion data will be tracked to allow the user to view their progress and productivity through electronically generated charts and graphs. Moreover, their data will be analyzed by ZenTime in order to make accurate estimates (ETAs) for their productivity and task completion progress.

ZenTime will simultaneously integrate each feature into an easily accessible desktop application designed for reusability and long-term use. In conclusion, ZenTime will aim to assist users with the monitoring of their productivity, management and coordination of their time, and scheduling of their tasks.

**1.2 Project Scope**

Key Deliverables:

* Scheduling System
  + Ability to schedule tasks/deadlines at a specific date & time
  + Ability to reschedule tasks
  + Ability to remove tasks from schedule
  + Ability to access and view schedule in a graphical user interface
* Reminder System & Desktop Notifications
  + Ability to enable desktop notifications for certain tasks on the user’s schedule
  + Ability to add new alarms or reminders
  + Receive desktop notifications for reminders
* Progress Reports
  + Progress bar which displays the percentage of tasks completed on a day-by-day basis
  + Automatic tracking of progress data, time taken to complete tasks, and other relevant data regarding user’s productivity
  + Visualizations and displays of progress data through pie-charts, line graphs and other graphical displays of
  + Processing of progress data to provide detailed information, reports and predictions for user’s (e.g. Estimated time to complete tasks and if tasks can be completed on time based on previous data)
  + To-do list styled list of tasks generated and displayed based on user’s schedule showing upcoming deadlines
* Clock/Time System
  + Users will be able to see the current date and time in the GUI of the project through a digital clock which updates in real-time

Assumptions:

* The end-user(s) will meet the system requirements to run Java 8 and will be using the Windows operating system
* Github will be available for use as the project’s remote repository
* System will never be used by multiple users at once (More than one person using the same schedule)
* All tasks are self-scheduled and managed properly by the user

Constraints:

* Limited knowledge and experience using graphical-user-interface APIs/frameworks in Java such as java.AWT, javax.swing, and org.jfreechart
* Limited experience using Github and source control software such as Git
* Limited experience with *.csv* file I/O using CSVReader, FileWriter, and BufferedReader objects in Java
* Time (Deadline of January 21st, 2022)

Exclusions:

* Integration with existing scheduling and calendar systems (e.g Google, Outlook, etc.)
* Ability to export progress analytics
* Ability to share schedules with others
* Custom presets for scheduled tasks

### **Program Specification**

##### **2.1 Scheduling System**

The scheduling system will exist in a *javax.swing* GUI within the main frame and window of the program. The main window will open as a JFrame component when the program is executed by the user, and will act as a portal for other frames and features within the program as well. The main window will contain a digital time and clock as a JLabel continuously being updated overtime, and will also contain two JPanels on each side of the frame. The right panel will contain an *Exit* JButton which closes the program and a *Help* JButton which opens another frame containing instructions and text tutorials on how to use the program. Additionally, the right panel will also have a *Reminders* JButton which will open up a new frame leading the user to the program’s reminder system, as well as a *Progress Reports* JButton which will lead the user to the program’s progress tracking and monitoring system.

The left panel of the main frame will contain the core of the scheduling system. It will have a JTable listing all the tasks on the user’s schedule with three columns which will display the time, task name, and status of the task across a row. The tasks will be sortable by column, and the table will have a JScrollPane allowing the user to scroll through their list of tasks. The program will make use of a custom recursive merge sort algorithm in order to sort by date, alphabetical order, and status. Moreover, below the JTable will reside an *Add Task* JButton, a *View Details* JButton, and an *Edit Task* JButton. The user will be able to select a task through the JTable and click the *View Details* and *Edit Task* buttons to open popup frames which will allow the user to add tasks to their schedule, as well as modify or remove tasks.

The *Add Task* JFrame will contain two JTextFields with JLabels prompting the user to enter their task name and task description. There will also be four JComboBoxes with a JLabel prompting the user to select a date and time for their task’s deadline. Additionally, there will be three JRadioButtons a part of a ButtonGroup object which will allow the user to select one of three task types: Timed task, task with deadline, and event. A timed task will track the time taken for the user to complete the task, as well as provide them the ability to start timing and finish timing the completion of the task at any time. Likewise, a task with a deadline will simply have a deadline, but will not be timed. Lastly, an event will not be timed, have a deadline, or have a status regarding completion, but instead will simply be an event on the user’s schedule which occurs at a specific time such as a meeting. Moreover, the frame will also have a JCheckBox with a JLabel prompting the user if they would like to receive a desktop notification prior to the deadline of the task. Furthermore, there will be an *Add* JButton which will add the user’s task based on their entered details to the scheduling system, as well as a *Cancel* JButton which will close the frame.

The *Edit Task* JFrame will contain the exact same JComponents as the *Add Task* frame, with the exception of the *Add* JButton and the addition of a *Remove* JButton which will allow the user to remove the task from their schedule. In addition, the user will be able to modify the task’s information and update the task’s information with a *Update* JButton.

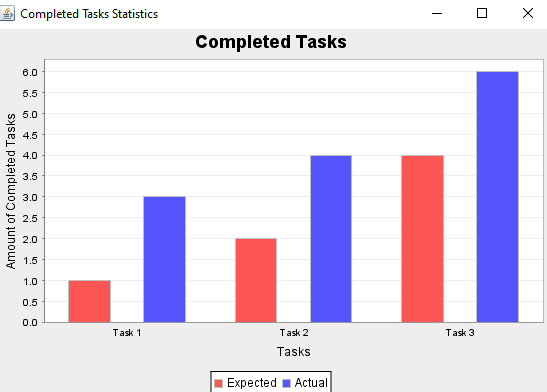
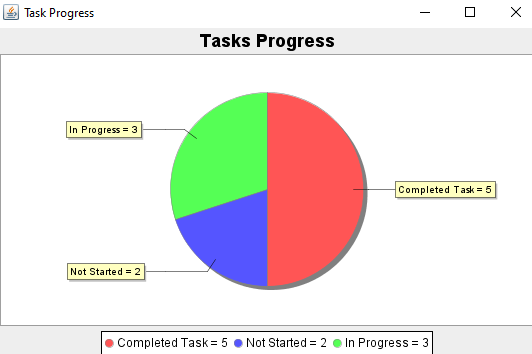
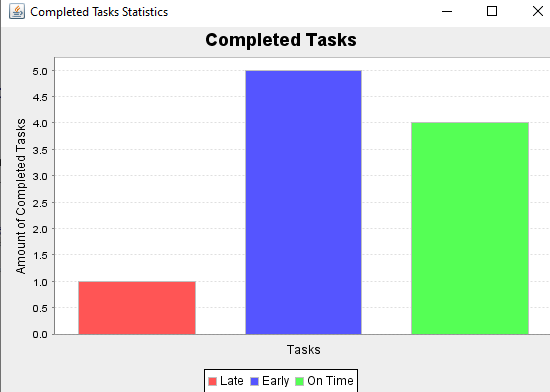
The *View Details* JFrame will contain JLabels displaying the selected task’s information such as its name, description, deadline and other information entered in the *Add Task* frame. Additionally, it will contain a *Start/Finish* JButton in order to update the status of tasks (e.g. Incomplete → In Progress → Complete). Refer to section 3.1 of this document for detailed diagrams of the main frame, add task frame, view details frame, and edit task frame and each of their respective JComponents.

##### **2.2 Reminder System**

The reminder system will contain two JPanels within which will be the two primary features of the system. The first feature of the reminder system will be the ability to add reminders by selecting a time through two JComboBoxes, entering the reminder name, as well as the reminder message. There will be an *Add* JButton which will allow the user to enter the reminder into the system after the reminder details have been entered. This will automatically send the user a desktop notification at their specified time with the reminder message through the system tray and Windows action center. Moreover, the secondary feature of the reminder system will be the ability to edit and remove reminders from the system. The user will be able to select an existing reminder from a JComboBox, and then use the same JComponents from the first JPanel used for adding reminders. After selecting and modifying a reminder, the user can click a *Update* JButton to update the reminder details, as well as a *Remove* JButton which will allow the user to remove the reminder from the system. Refer to section 3.2 for a detailed diagram of the reminder system frame and its JComponents.

##### **2.3 Progress Reports**

The Progress Reports system will contain a JProgressBar at the top of the screen displaying percentage of completed daily tasks. The frame will also have three JPanels, and the first one will contain information regarding the user's progress and productivity. The first panel will have JLabels displaying data regarding progress such as the percentage of tasks which have been completed which place the user ahead of schedule. In addition, the number of outstanding tasks, overdue tasks, and number of incomplete, in progress, and complete tasks are displayed. Moreover, the second JPanel will contain JLabels displaying data regarding productivity such as the least and most productive work weeks, average early and late completion time, and the estimated time for the completion of all outstanding tasks. Lastly, the third JPanel will contain three JButtons which will open new windows displaying the graphs and charts which can be seen below. Refer to section 3.3 for a detailed diagram of the progress reports system frame and its JComponents.



The program will determine estimated times for the completion of timed tasks by calculating the average percentage of extra time taken for the completion of previous timed tasks. If a task’s deadline has not yet been reached, the program will specifically find the average time taken from previous tasks which were completed early. Likewise, if a task is overdue, the program will find the average time taken from previous tasks which were completed late to determine an estimated time of completion for a current task. If there is not enough data available to accurately calculate an estimated time for completion, the program will simply use the deadline as the estimated time of completion.

The *org.jfreechart* package will be used to visualize the data being displayed through the *Task Progress*, *Completion Statistics*, and *Completion Times* buttons. Data stored in the program’s database such as the user’s tasklist, their previous task completion data, and number of tasks completed per week will be used to display the progress and productivity of the user. Refer to section 2.6 for more information regarding the tracking and storage of this data.

##### **2.4 Event Handling & Data Processing**

When the program is executed by the user, the main method of the main program class will instantiate an instance of the scheduling system GUI which functions as the program’s main frame. The GUI class will then instantiate an instance of the scheduling system’s event-handler class as an attribute in its constructor method, and pass itself through as well. By passing the GUI class into the event-handler, the event-handler will be able to reference and update the GUI. This event-handler class will contain all the functionality of the JComponents in the GUI such as the JButtons and JTextFields, and will process all the user’s inputs in the GUI through *ActionListener* objects in *java.awt*. This class will also be responsible for updating the components in the GUI. These *ActionListener* objects will be accessible through the instance of the event handler which will be appended to the JComponents in the GUI class’s constructor method, which separates the user’s view and the processing of their inputs and processing of data. Moreover, this event-handler class will contain a method which reads a *.csv* file storing the user’s tasks, and will create and store them in the program as an arraylist of task objects. After importing all the tasks, it will update the view in the GUI. Likewise, there will be a method so that any time tasks are added or removed, the program will modify the *.csv* file and update its arraylist of task objects, then update the user’s view in the GUI.

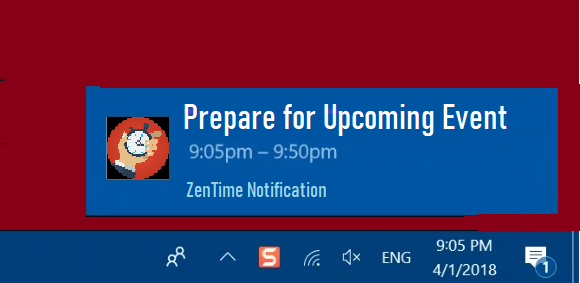
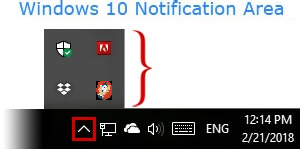
Each frame and GUI class in the program will have a corresponding event handler class which processes the user inputs and user data through reading and writing to *.csv* files. Additionally, each event handler class will have methods which update the GUI and the data it displays, which will use helper methods which read and write to *.csv* files. Moreover, on the main frame if the *Reminders* or *Progress Reports* buttons are clicked by the user, its event-handler will instantiate a new object of the new frame’s GUI class, which will also instantiate an instance of its event handler class within itself. Likewise, in the progress reports frame, if the user clicks any of the buttons leading them to view the visualizations or graphs of their data, a new JFreeChart frame object will be created, however it will not have its own event handler class. The data being graphed and visualized in the JFreeChart frames will be imported and processed through a method in the event-handler class of the *Progress Reports* frame.

The event-handler classes will function as controllers in a MVC design pattern which can be seen in section 4.1.

The program will make use of the *java.util.datetime* package in order to easily and efficiently process, compare, and generate dates and times used in the scheduling and reminder system. This will allow for a consistent formatting of dates and times, and make it possible to obtain the local time of the user’s machine in order to send desktop notifications at the correct times, run the digital clock, and also track the time the user takes to complete tasks.

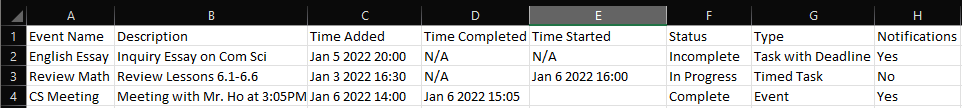
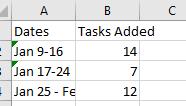
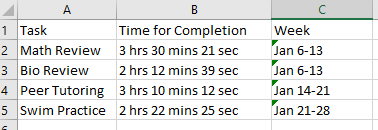
##### **2.5 System Tray & Desktop Notifications**

The main method of the main class of the program will create a TrayIcon object using *java.awt* when the program is executed, which will append itself to the user’s system tray. This will allow the user to hide/show the main frame as well as exit the program completely as the *X* button in the top-right corner of their screen will simply hide the main frame, but continue to run the program in the background through the system tray. The purpose of this is to allow the user to receive desktop notifications through the system tray and Windows action center. Any tasks with notifications enabled or reminders will display a desktop notification at the specified time by continuously checking the local time to check when to send the desktop notification. The system tray icon and desktop notifications will appear as follows:



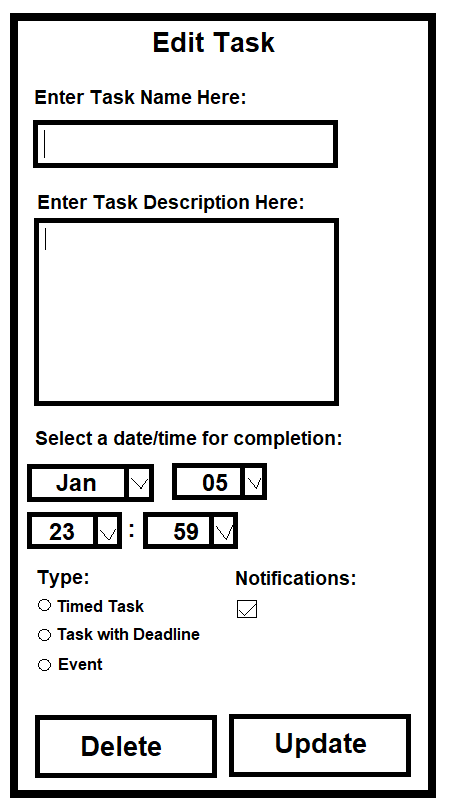
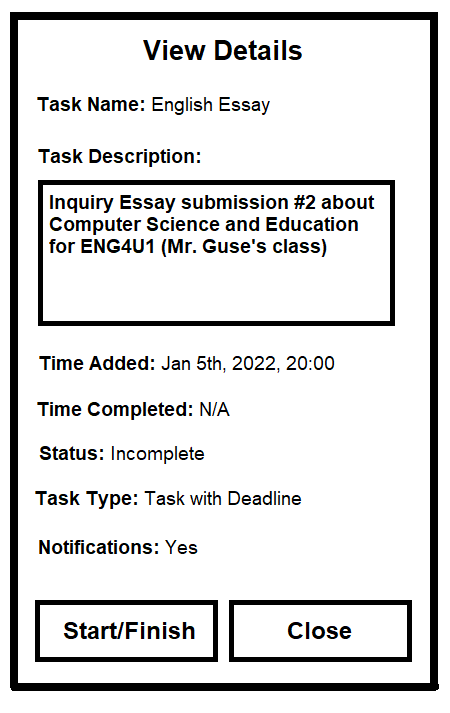
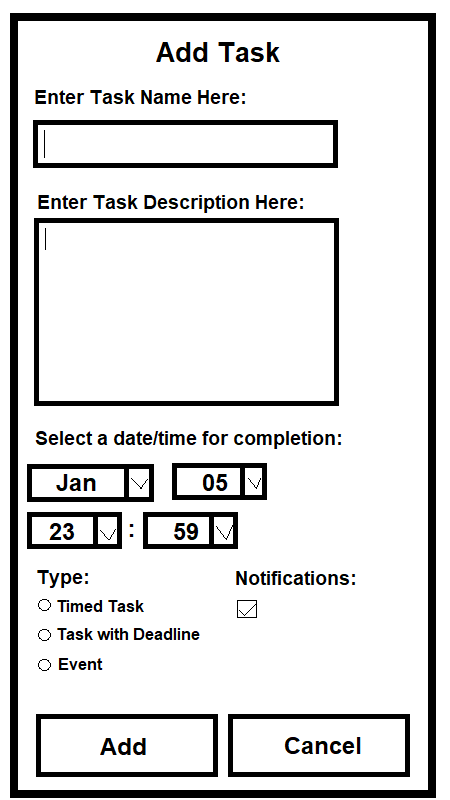
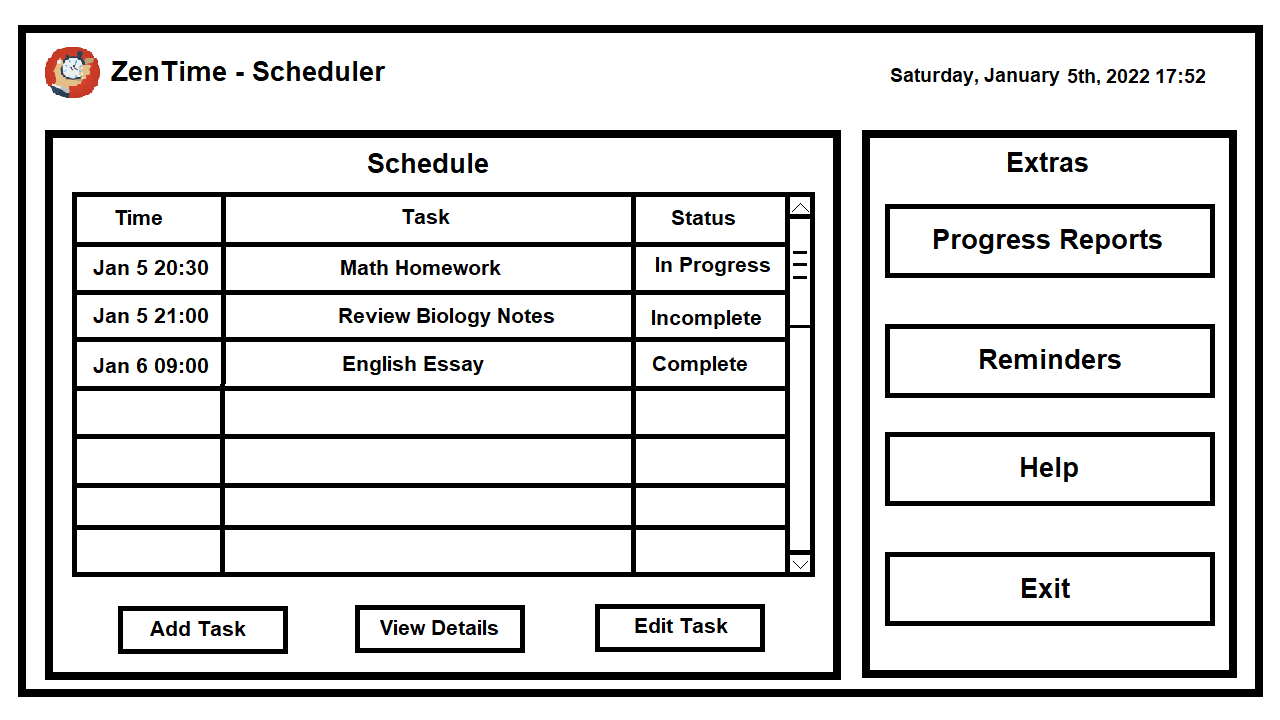
##### **2.6 Data Management**

The program’s directory will contain each class in a *.java* file whose program structure can be seen in section 4.1. The program directory will additionally store data in *.csv* files locally on the user’s Windows machine. More specifically, the user’s scheduled task list and information, the number of tasks being added per week, reminders, and time spent on each timed task will be tracked and stored in separate files. The program will utilize FileWriter, BufferedReader, and/or CSVWriter objects to read and write to files. Storing this data allows the user to close the program and reopen it without losing any information, as it can be retrieved from these files when the program is reopened. The data stored in *.csv*files will stored as follows:

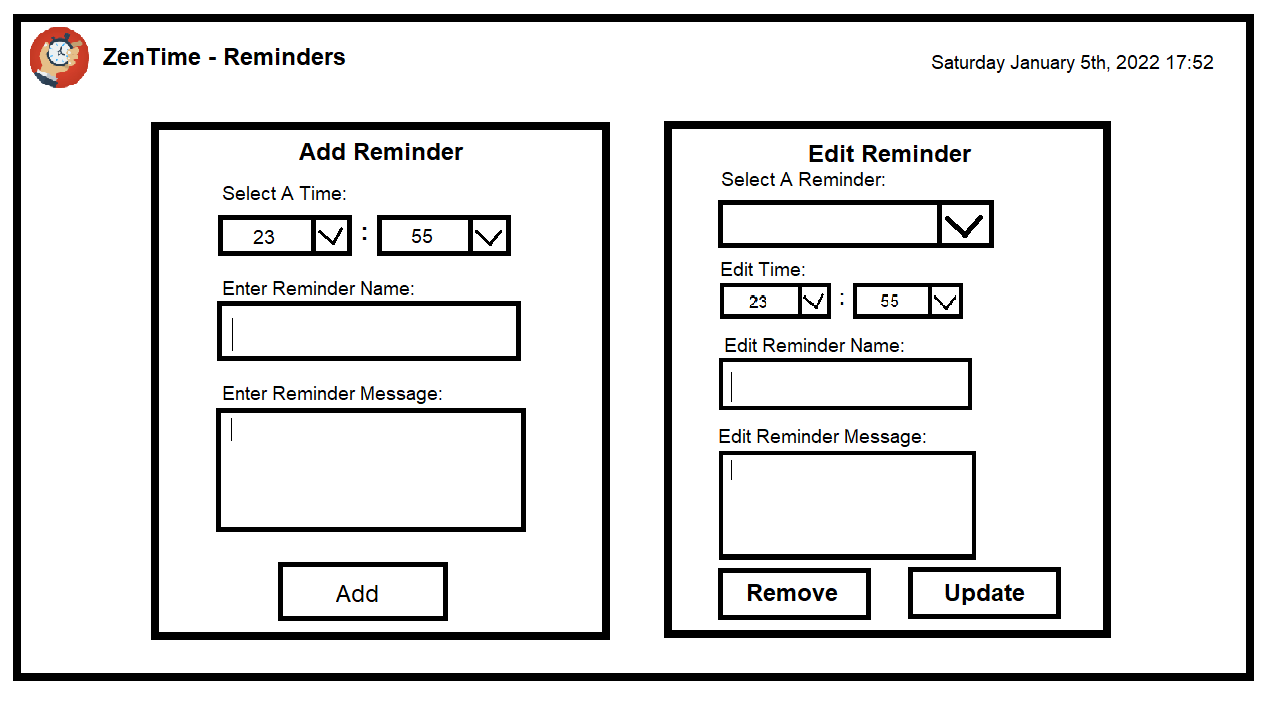


### **Graphical User Interface Designs**

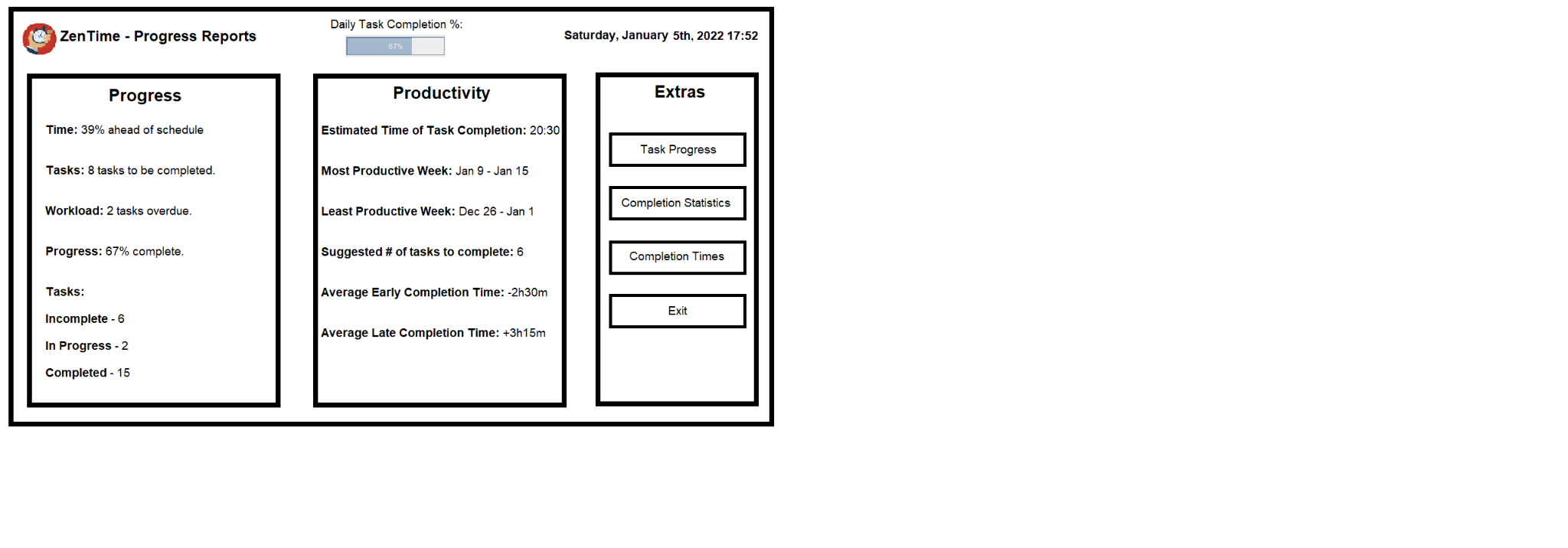
##### **3.1 Scheduling System GUIs**



##### **3.2 Reminder System GUI**

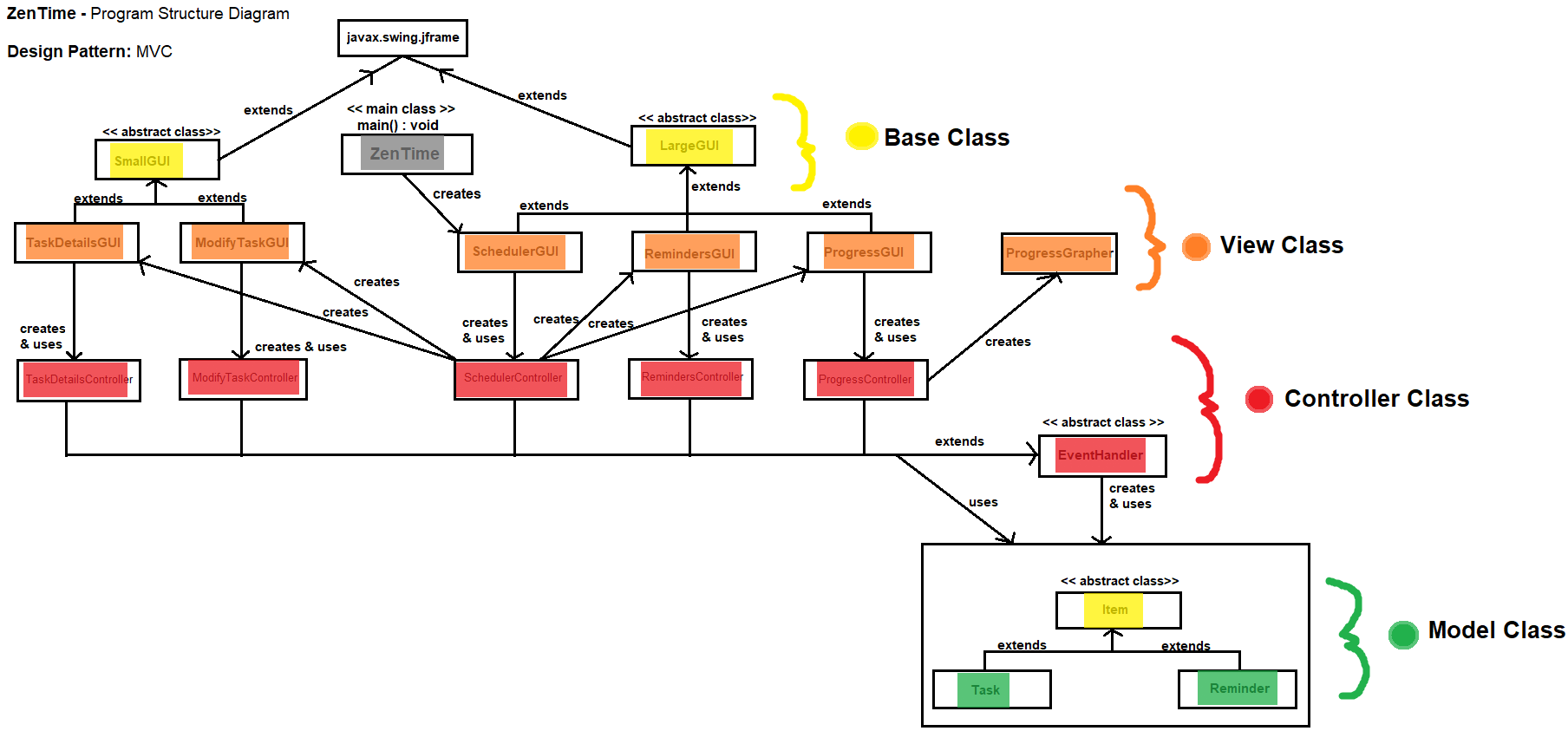


##### **3.3 Progress Reports GUI**



### **Program Structure**

##### **4.1 Structure & Design Pattern Diagram**



([**Link to full-resolution image**](https://i.imgur.com/xDUxANu.png)**)**

##### **4.2 Class & Object Breakdown**

ZenTime’s program structure will function under a Model-View-Controller (MVC) design pattern. Each class in the program will have its attributes encapsulated with getter and setter methods where necessary. The *Task* and *Reminder* classes will inherit from an abstract Item class and will have their own underlying implementations for certain methods such as notification time methods. These classes represent the program’s model classes which are the data models used in the program. Moreover, all frames/windows which open as GUIs in the program are separated into their own classes, each with a name ending with *GUI*. These classes represent the view, and will all either inherit from the *LargeGUI* or *SmallGUI* base classes depending on the type of frame. For example, the view details frame is smaller and has fewer components, so the *TaskDetailsGUI* will inherit from the *SmallGUI* class. Additionally, the base GUI classes will inherit from the JFrame class so that there is no need to instantiate JFrame objects in the program. Every GUI class in the program will *only* contain the components the user sees on-screen, and a controller class will process their inputs, manipulate the model classes, and update the view/GUI classes. However, the *ProgressGrapher* class which generates graphs and charts is an exception to this and will not have its own controller class, and instead will be controlled by the *ProgressController* class which is also the controller class of the *ProgressGUI*. Each controller class will inherit from the abstract *EventHandler* class which will contain methods used to control and manipulate the model objects which will be instantiated in the controllers.

### **Program Requirements & Test Cases**

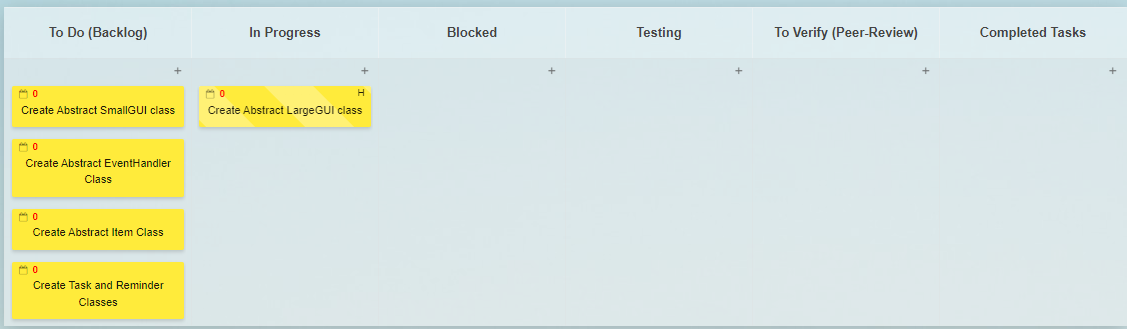
|  |  |
| --- | --- |
| Program Requirements | Test Cases |
| The user can view the program’s main graphical user interface (Scheduler GUI) when the program is executed and use its components | * User can view title, program icon, and buttons in a 1280x720 window * User can press ‘X’ button in top-right corner of screen to close the program * User’s tasks appear in a table * Scrollbar can be used to scroll up and down the list of the user’s tasks in the table * User can press the ‘Progress Reports’ button to open up a 1280x720 window with the progress reports system * User can press the ‘Reminders’ button to open up a 1280x720 window with the reminders system * Program closes when the user presses the ‘Exit’ button * User can view a digital clock which updates in realtime |
| The user is able to add timed tasks, tasks with deadlines, and events to their schedule. They will be able to select a deadline for their task, and be able to enter their task’s name and description into the scheduling system. | * User can click the ‘Add Task’ button to open up a 450x800 window (Add Task GUI) * User can enter their task name into its textbox, task description into its textbox, can select a date/time for completion, task type, and if they wish to receive notifications for the task * User can press the ‘cancel’ JButton in the “Add Task’ GUI if they no longer require a task to be added to their schedule * User cannot add a task with a deadline date/time in the past * User cannot add a task with the same name as a pre-existing task * User can click the ‘Cancel’ button to close the window * User cannot add more than 1 task at the exact same time (cannot open multiple ‘Add Task’ windows) * User can successfully add their task to the schedule and exit the ‘Add Task’ GUI if they’ve entered valid task details (based on red test cases above) * User is prompted through a JDialogPane to re-enter their task details when they click ‘Add’ and their details are invalid (based on red test cases above) |
| The user is able to select a task from the table in the scheduling system and edit its details such as its name, description, deadline, type, and if they wish to receive notifications for the task. | * User can select ONLY ONE task from the table at a time * User can click the ‘Edit Task’ button to open up a 450x800 window IF they have selected a task to edit (Edit Task GUI) * The user’s selected task’s details appear in the textboxes in the GUI when it opens * Users can edit their task name, description and date/time for completion via text boxes present in the window. * User cannot edit task to have a deadline or date/time in the past * Users cannot edit a task to have the same name as a pre-existing task. * User cannot edit more than 1 task at the exact same time (cannot open multiple ‘Edit Task’ windows) * Users cannot edit tasks to occur at the exact same time as another event or task. * User can completely delete tasks from their schedule if they press the ‘delete’ JButton * User can successfully edit their task to match their schedule and exit the ‘Edit Task’ GUI if they’ve entered valid task details (based on red test cases above) * User is prompted through a JDialogPane to re-enter their task details when they click ‘Add’ and their details are invalid (based on red test cases above) |
| The user is able to select a task from the table in the scheduling system and view its details such as its name, description, deadline, type and if they wish to receive notifications for the task. | * User can select ONLY ONE task from the table at a time * User can click the ‘View Task’ button to open up a 450x800 window IF they have selected a task to edit (View Task GUI) * The user’s selected task’s details appear in the un-editable JTextFields in the GUI when it opens * Users cannot make changes to any details of their task via this window. |
| The user is able to add reminders. They will be able to select a time for the reminder to be set off. They will also be able to enter a reminder’s name, and message. | * User can click the ‘Reminders’ button to open up a 1280x700 window (Reminders GUI) * User can select the time for the reminder to be set off with 2 JComboBoxes, one which will contain the hours of the day and another which will contain the minutes. * User can enter the reminder name into the JTextbox offered. When the notification appears in the system tray, this name will be the main message of the notification. * User can enter a reminder message into the JTextbox offered. When the notification appears in the system tray, this message will be the secondary message of the notification. * User cannot enter a reminder for a time previous to current date * User CAN enter the exact same name for more than 1 reminder. (It makes sense for the user to have more than 1 reminder for the same action each day.) * User CAN have more than 1 reminder occur at the same time (For example, a reminder of ‘buy groceries’ may be vague, but having 2 reminders of ‘buy carrots’, and ‘buy apples’ is more specific) * User can successfully add their reminder to their schedule and exit the ‘Reminder’ GUI if they’ve entered valid task details (based on red test cases above) * User is prompted through a JDialogPane to re-enter their task details when they click ‘Add’ and their details are invalid (based on red test cases above) |
| The user is able to select a reminder from the JComboBox on the right JPanel offered in the GUI. Once selected, the user will be able to edit reminder details, such as time, name and message. | * User can click the ‘Reminder’ task button to open up a 1280x700 window (Reminder GUI) * User can select ONLY ONE task from the JComboBox at one time * The user’s selected reminder’s details will appear in the JComboBox offered for time, and in the JTextFields for the reminder name and message * Users can edit the reminder name, message, and time for the reminder to be set off. * User cannot edit reminder to have occurred in the past * User CAN edit the name to have the exact same name as another reminder. (It makes sense for the user to have more than 1 reminder for the same action each day.) * User CAN have more than 1 reminder occur at the same time (For example, a reminder of ‘buy groceries’ may be vague, but having 2 reminders of ‘buy carrots’, and ‘buy apples’ is more specific) * User can completely remove reminders from their schedule and JComboBox, by selected the reminder and pressing the ‘Remove’ button * User can successfully edit their reminder to match their schedule and exit the ‘Reminder’ GUI if they’ve entered valid reminder details (based on red test cases above) * User is prompted through a JDialogPane to re-enter their task details when they click ‘Update’ and their reminder details are invalid (based on the red test cases) |
| Users can view progress in the ‘Progress Reports’ GUI. The progress JPanel will include whether the user is ahead or behind schedule, remaining tasks, workload of tasks (overdue or not), progress (% of tasks completed) and will show number of tasks in each status (Incomplete, IP, Completed) | * User can click the ‘Progress’ button to open up a 1280x700 window (Progress Reports GUI) * User can view how ahead or behind schedule they are (in %) * User can view how many tasks they have remaining on their schedule * User can view what their workload is (how many tasks are overdue) * User can view their progress (% of tasks completed) * User cannot edit, change or in any other way modify the information displayed on this page. * User can press exit to leave the page once they have viewed the information they would want to see |
| User can view productivity reports in the ‘Progress Reports’ GUI. The Productivity JPanel will contain: Estimated time of task completion, most productive week, least productive week, suggested # of tasks to complete, average early completion time, and average late completion time. | * User can click the ‘Progress’ JButton to open up a 1280x700 window (Progress GUI) * User can view their estimated time of task completion (in hr: min format) * User can view their most productive week (7-day range) * User can view their least productive week (7-day range) * User can view suggested # of tasks to complete * User can view average early completion time * User can view average late completion time * User cannot edit, change, or in any other way modify the information displayed on this page * User can press exit to leave the page once they have viewed the information they would want to see |
| User can view Completed Task Statistics in a Bar graph, in the ‘Completion Statistics’ GUI. The entire window will be the data visualized as a bar graph. The bar graph will be a representation of how many completed tasks were 1. Late, 2. Early, and 3. On time. | * User can click the ‘Completion Statistics’ JButton to open up a new 1000x563 window * The window will contain a bar graph, which will display Amount of completed tasks vs time of completion (early, late, or on time) * User cannot modify any data visualized in the bar graph shown (in this window) * User can close window by pressing ‘x’ button in corner of window |
| User can view Tasks Progress in a Pie Chart, in the ‘Tasks Progress’ GUI. The window will be the data visualized as a pie chart. The pie chart will be a representation of what status each task added to the user’s schedule is in, such as, completed, in progress, or Not started | * User can click the ‘Tasks Progress’ JButton to open up a new 1000x563 window. * The window will contain a pie chart containing the status of each task in a schedule. * User cannot modify any data visualized in the pie chart shown * User can close window by pressing ‘x’ button in corner of window |
| User can view Completion Times as a bar graph in the ‘Completion Times’ GUI. The window will be the data visualized as a bar graph. The bar graph will be a representation of what time the user is expected to finish a task in hours, vs what time they really finish at. | * User can click the ‘Completion Times’ JButton to open up a new 1000x563 window * The window will contain a bar graph, with one bar representing the estimated time of completion in hours, and the other bar representing the actual time of completion in hours. * User cannot modify any data visualized in the bar graph shown (in this window) * User can close window by pressing ‘x’ button in corner of window |
| User will be able to receive desktop notifications, and will have an icon for the system tray. | * User can receive, and clear notifications from ZenTime * User can see, and use the icon in the system tray to open/close the app * If all windows are closed the user can right click the icon in the system tray and reopen the frames of the window |
| User will be able to close and reopen the program, but data will still be stored in a csv file. | * User will be able to close all the windows of the program, but will still have access due to the csv files that the program will be writing data too * User can view info from the csv file, even if they completely close the program (not just if they close the window, but completely close the program) * User can edit the details of the csv file by using the program |

### **Project Schedule**

##### **6.1 Task Breakdown List**

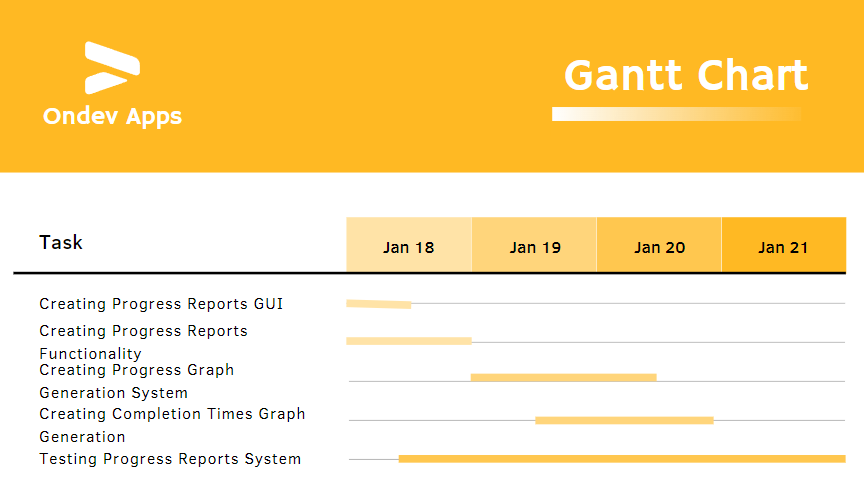
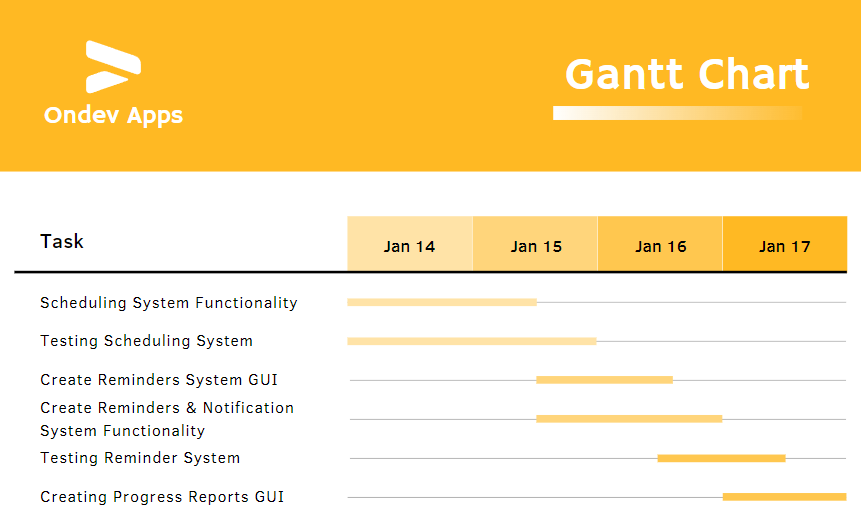
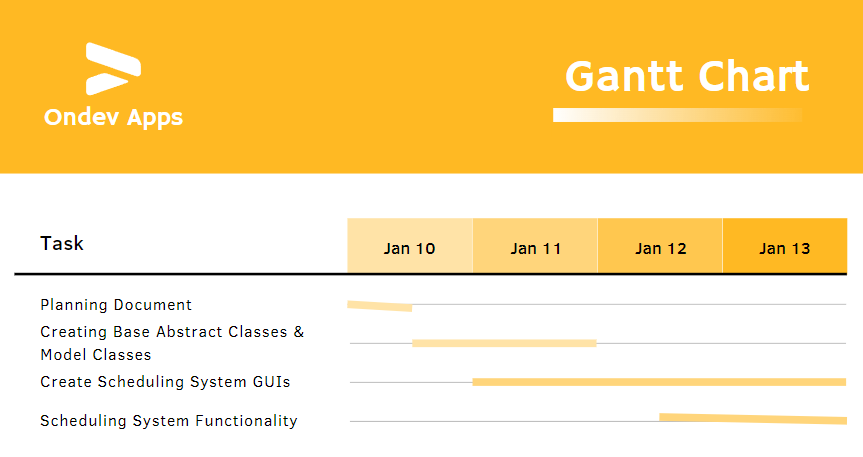
|  |  |
| --- | --- |
| Task | Timeline  (Date to finish by) |
| * Create abstract LargeGUI base class (extends JFrame)   + Constructor method   + Initialize JComponents and JPanels   + Add ZenTime logo/icon to JFrame   + Digital time through run() method (Implement Runnable interface) | Jan 11 |
| * Create abstract SmallGUI base class (extends JFrame)   + Constructor method   + Initialize JComponents and JPanel | Jan 11 |
| * Create abstract Item base class   + Initialize attributes   + Constructor method   + Create getter/setter methods for attributes | Jan 11 |
| * Create Task model class (extends Item)   + Initialize attributes   + Constructor method   + Create getter/setter methods for attributes | Jan 11 |
| * Create Reminder model class (extends Item)   + Initialize attributes   + Constructor method   + Create getter/setter methods for attributes | Jan 11 |
| * Create abstract EventHandler base class   + Constructor method   + Initialize attributes (Arraylist of task objects)   + Create method to read tasks.csv file   + Create method to modify tasks.csv file   + Create method which retrieves the local date/time | Jan 11 |
| * Create main class ZenTime   + Main method | Jan 13 |
| * Create SchedulingGUI class (extends LargeGUI)   + Constructor method   + Create title JLabels   + Create tasks JTable and JScrollBar   + Create Progress Reports, Reminders, Help and Exit button   + Create Add Task, View Details, and Edit Task Button | Jan 13 |
| * Create TaskDetailsGUI class (extends SmallGUI)   + Constructor method   + Create title JLabel   + Create Task Description JTextArea   + Create JLabels for time added, time completed, status, task type, and notifications information   + Create Close button   + Create Start/Finish button | Jan 13 |
| * Create ModifyTaskGUI class (extends SmallGUI)   + Constructor method   + Create title JLabel   + Create TextFields for task name and description   + Create JComboBoxes for date/time   + Create JRadioButtons for task type   + Create JCheckBox for notifications   + Create Cancel button   + Create Add & Update button | Jan 13 |
| * Create SchedulerController class (extends EventHandler)   + Constructor method   + Create ActionListener object/method for Add Task button   + Create ActionListener object/method for View Details button   + Create ActionListener object/method for Edit Task button   + Create ActionListener object/method for JScrollBar   + Create ActionListener object/method for Progress Reports button   + Create ActionListener object/method for Reminders button   + Create ActionListener object/method for Help Button   + Create ActionListener object/method for Exit button | Jan 15 |
| * Create TaskDetailsController class (extends EventHandler)   + Constructor method   + Create ActionListener object/method for Close button   + Create ActionListener object/method for Start/Finish button using inherited methods from EventHandler (File I/O) | Jan 15 |
| * Create ModifyTaskController class (extends EventHandler)   + Constructor method   + Create ActionListener object/method for add button     - Input validation using JDialog/JOptionPanes   + Create ActionListener object/method for edit button     - Input validation using JDialog/JOptionPanes   + Create ActionListener object/method for Cancel button | Jan 15 |
| * Create RemindersGUI class (extends LargeGUI)   + Constructor method   + Create Add Reminder JPanel with its JComponents (Based on section 3.2)   + Create Edit Reminder JPanel with its JComponents (Based on section 3.2) | Jan 16 |
| * Create RemindersController class (extends EventHandler)   + Constructor method   + Create ActionListener object/method for Add button   + Create ActionListener object/method for Remove button   + Create ActionListener object/method for Update button   + Create method which reads and writes to the csv file the reminders are stored in | Jan 16 |
| * Implement Desktop notifications through the system tray in the main method of the main class of the program (ZenTime class) as well as the ability to show/hide the program through menu items in the system tray | Jan 16 |
| * Create ProgressGUI class (extends LargeGUI)   + Constructor method   + Create the daily task completion percentage progress bar   + Create Progress JPanel with JLabels displaying progress info   + Create Productivity JPanel with JLabels displaying productivity info   + Create Extras JPanel with the Task Progress, Completion Statistics, and Completion Times JButtons   + Create the Exit JButton | Jan 18 |
| * Create ProgressController class (extends EventHandler)   + Constructor method   + Create ActionListener object/method for Task Progress button   + Create ActionListener object/method for Completion Statistics button   + Create ActionListener object/method for Completion Times button   + Create ActionListener object/method for Exit button | Jan 18 |
| * Create method in ProgressController class to read the data stored in .csv files and retrieve the specific information from it being graphed   + Timed task completion data file   + Weekly task data file | Jan 18 |
| * Create methods in ProgressController class which calculate productivity data and estimated completion times:   + Estimated time of daily task completion   + Most productive and least productive work weeks   + Suggest # of tasks to complete for daily task completion   + Average early completion time   + Average late completion time   + Expected/Estimated date of completion for an incomplete timed task | Jan 18 |
| * Create the ProgressGrapher class (using JFreeChart)   + Constructor method   + Method for task progress graph generation   + Method for completion statistics graph generation | Jan 19 |
| * Create a method in the ProgressGrapher class which acts as an interface which generates the graph of a specified chart/set of data | Jan 19 |
| * Create a method in the ProgressController class which uses a helper method to retrieve estimated timed task completion data in order to generate a graph of the data through the ProgressGrapher’s interface method | Jan 20 |
| * Create a method in the ProgressGrapher class which uses data passed through from the ProgressController class to graph estimated task completion data | Jan 20 |
| * Create test-data and store into .csv files to rigorously test and improve each data-analysis feature in the program using the program requirements and test cases in section 5 | Jan 20-21 |

##### **6.2 KanBan Board**



**Note:** Task details can be viewed through the task description (Name is condensed to consume less space on KanBan board)

##### **6.3 Gantt Charts**



##### 

##### **6.4 Project Milestones**

Official Start Time: Jan 10, 2022

Milestone 1 - Finish Creating Base Abstract Classes & Model Classes: Jan 11, 2022

Milestone 2 - Create Scheduling System GUIs: Jan 13, 2022

Milestone 3 - Finish Scheduling System Functionality: Jan 15, 2022

Milestone 4 - Create Reminders System GUI:Jan 16, 2022

Milestone 5 - Finish Reminders & Notification System Functionality: Jan 16, 2022

Milestone 6 - Finish Creating Progress Reports GUI & Functionality: Jan 18, 2022

Milestone 7 - Finish Creating Progress Graph Generation System: Jan 19, 2022

Milestone 8 - Finish Creating Completion Times Graph Generation:Jan 20, 2022

End of Project: Jan 21, 2022