

TODO APPLICATION
A MINI PROJECT REPORT

Submitted by

GIRIDHARAN E 230701091

HARISANKAR S 230701100

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2023 - 24

BONAFIDE CERTIFICATE

Certified that this project report “ TO-DO LIST APPLICATION” is the bonafide work of

“GIRIDHARAN E(230701091) , HARI SANKAR S(230701100)”

who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____

SIGNATURE

Mrs.Divya.M
Assistant Professor,
Computer Science and Engineering,
Rajalakshmi Engineering College,
Thandalam,Chennai 602105

SIGNATURE

INTERNAL EXAMINER

EXTERNAL EXAMIER

ABSTRACT:

The ToDo Application is a comprehensive and user-friendly task management tool that helps users organize and manage daily tasks effectively. It provides a structured system for creating, updating, and monitoring tasks based on priority and category. By streamlining task management, the application enhances productivity and ensures efficient time management, ultimately improving users' ability to handle their workloads effectively.

This project incorporates SQL, Java Swing, and Java Server Pages (JSP), along with a graphical user interface that makes it intuitive and engaging for users. SQL handles backend data storage and retrieval, while Java provides the core application logic, supporting a robust and dynamic environment for task creation, updating, and deletion. JSP assists in rendering real-time information within the app, creating a fluid and responsive user experience.

This report details the system's design, implementation, and the various technologies integrated into the project, illustrating how the ToDo Application not only meets user requirements but also provides a versatile and scalable solution for task management.

TABLE OF CONTENTS

Chapter 1

1 INTRODUCTION	
1.1 INTRODUCTION	6
1.2 OBJECTIVES	7
1.3 MODULES	7

Chapter 2

2 SURVEY OF TECHNOLOGIES	
2.1 SOFTWARE DESCRIPTION	9
2.2 LANGUAGES	9
2.2.1 PHP	9
2.2.2 SQL	9
2.2.3 HTML	10
2.2.4 CSS	10
2.2.5 JAVASCRIPT	10

Chapter 3

3 REQUIREMENTS AND ANALYSIS	
3.1 REQUIREMENT SPECIFICATION	11
3.1.1 FUNCTIONAL REQUIREMENTS	11
3.1.2 NON FUNCTIONAL REQUIREMENTS	12
3.2 HARDWARE AND SOFTWARE REQUIREMENTS	13
3.3 ARCHITECTURE DIAGRAM	14
3.4 ER DIAGRAM	15
3.5 NORMALIZATION	16

Chapter 4

4 PROGRAM CODE _____

4.1 PROGRAM CODE-----	18
-----------------------	----

Chapter 5

5 RESULTS AND DISCUSSION _____

5.1 RESULTS AND DISCUSSION -----	27
----------------------------------	----

Chapter 6

6 CONCLUSION _____

6.1 CONCLUSION-----	31
---------------------	----

Chapter 7

7 REFERENCES _____

7.1 REFERENCES-----	32
---------------------	----

1.1 INTRODUCTION

Efficient task management is crucial for productivity, particularly for individuals handling multiple responsibilities. The ToDo Application is designed to help users organize their tasks by creating an easy-to-use platform that categorizes tasks, prioritizes based on urgency, and allows seamless updates. This project addresses the core needs of task management: organization, prioritization, and accessibility. Users can sort tasks by category and priority, ensuring that high-priority tasks are easily accessible and manageable.

The system maintains a detailed database of tasks, including information about due dates, priority levels, categories, and statuses. With these functionalities, users can rely on the ToDo Application to keep track of their responsibilities and deadlines efficiently.

1.2 OBJECTIVES

- Develop a centralized task management platform.
- Facilitate task categorization and prioritization.
- Enable real-time updates for effective task monitoring.
- Improve user productivity by offering a streamlined, accessible interface.
- Maintain scalability for future feature expansion.

1.3 MODULES

1. Task Creation and Management Module:

This module allows users to create tasks with specific details such as title, category, priority level, and due date. It provides a structured form to enter task details, ensuring all necessary information is captured and stored in the database.

2. Priority and Category Management:

This feature enables users to categorize tasks (e.g., Work, Personal) and assign priority levels (e.g., High, Medium, Low). With color-coded priority tags, users can visually differentiate between tasks, promoting focus on urgent items.

3. **Database Module:**

This module is responsible for storing, retrieving, and managing task-related data. SQL powers the data storage, providing secure and efficient data management with easy retrieval functionalities.

4. **Real-Time Notifications and Reminders:**

This module sends users reminders for upcoming or overdue tasks, ensuring they stay on top of deadlines and important priorities.

5. **User Interface Module:**

Using Java Swing, the UI module provides a graphical interface that displays tasks in an organized, visually appealing manner. This module includes functionalities for searching and filtering tasks, improving ease of use.

CHAPTER 2: SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

Many task management tools currently exist, but many are either too complex or lack key features like custom prioritization, real-time updates, and flexibility for customization. Additionally, some applications do not provide offline support, making them dependent on internet availability, which can hinder productivity. These limitations in existing systems have driven the development of a more versatile and user-centered ToDo Application.

2.2 PROPOSED SYSTEM

The ToDo Application is designed to address these limitations, offering a lightweight, user-friendly experience for individuals needing a streamlined yet powerful task management system. By combining essential features like categorization, prioritization, and timely reminders, the application enhances productivity and provides users with a tailored experience. Key components include:

- **Intuitive Interface:** Uses a simple layout and color-coded priority labels for ease of navigation.
- **Offline Capability:** Allows users to manage tasks without needing an internet connection.
- **Real-time Updates and Reminders:** Ensures that users are notified of approaching deadlines or overdue tasks promptly.

2.3 SYSTEM REQUIREMENTS

2.3.1 Hardware Requirements

- **Processor:** Intel Core i3 or higher
- **RAM:** 4 GB minimum
- **Storage:** 100 MB minimum
- **Display:** 1024x768 resolution or higher

2.3.2 Software Requirements

- **Operating System:** Windows, macOS, or Linux
- **Development Platform:** Java Development Kit (JDK) 11 or later
- **Database:** MySQL for data storage
- **Additional Libraries:** Java Swing for the GUI, JDBC for database connectivity

CHAPTER 3: SYSTEM DESIGN

3.1 ARCHITECTURE DIAGRAM

The architecture of the ToDo Application includes the following primary components:

1. **User Interface (UI):** Developed using Java Swing, it provides an interactive layout where users can view, add, update, and delete tasks.
2. **Application Logic:** The core functionality of the application is managed here. It includes task creation, updating, deletion, and handling reminders.
3. **Database:** A MySQL database that stores tasks, categories, priorities, and completion statuses, enabling persistent data storage and retrieval.

3.2 DATA FLOW DIAGRAMS

Level 0 DFD: Represents the general flow of data between the user and the system.

Level 1 DFD: Details each interaction, such as task creation, editing, and deletion.

3.3 DATABASE DESIGN

The database is designed to store information efficiently, ensuring quick retrieval and updates. It consists of:

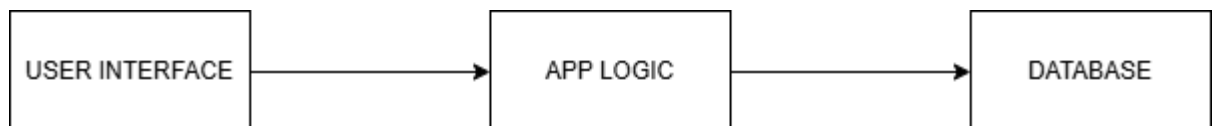
- **Tables:**
 - **Tasks Table:** Stores task ID, description, due date, priority, category, and status.
 - **Categories Table:** Stores different categories available for task assignment.
 - **Priority Levels Table:** Stores various priority levels (High, Medium, Low).
-

CHAPTER 3: SYSTEM DESIGN

3.1 ARCHITECTURE DIAGRAM

The architecture of the ToDo Application includes the following primary components:

1. **User Interface (UI):** Developed using Java Swing, it provides an interactive layout where users can view, add, update, and delete tasks.
2. **Application Logic:** The core functionality of the application is managed here. It includes task creation, updating, deletion, and handling reminders.
3. **Database:** A MySQL database that stores tasks, categories, priorities, and completion statuses, enabling persistent data storage and retrieval.



3.2 DATA FLOW DIAGRAMS

Level 0 DFD: Represents the general flow of data between the user and the system.

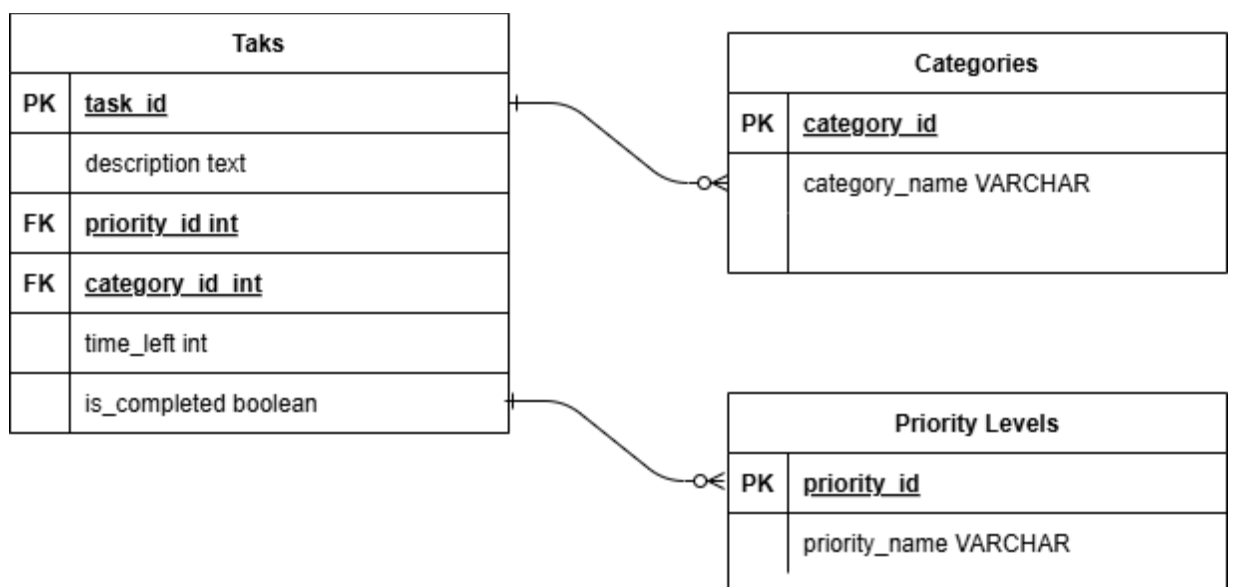
Level 1 DFD: Details each interaction, such as task creation, editing, and deletion.

3.3 DATABASE DESIGN

The database is designed to store information efficiently, ensuring quick retrieval and updates. It consists of:

- **Tables:**

- **Tasks Table:** Stores task ID, description, due date, priority, category, and status.
- **Categories Table:** Stores different categories available for task assignment.
- **Priority Levels Table:** Stores various priority levels (High, Medium, Low).



CHAPTER 4: IMPLEMENTATION

4.1 MODULES AND THEIR IMPLEMENTATION

Each module in the ToDo Application has been implemented to ensure smooth functionality and ease of use. This chapter will describe the backend and frontend implementation, along with the integration of the MySQL database.

4.1.1 Task Management Module

Using Java Swing, this module allows users to add, update, and delete tasks. The GUI provides interactive forms for task entry, ensuring that users can easily navigate between tasks and sort them by priority or category.

4.1.2 Priority and Category Module

This module includes Java-based logic to handle task categorization and priority assignment. Using a combination of JComboBox components and color-coded labels, the UI visually differentiates each task's importance and category.

4.1.3 Database Module

The database module utilizes JDBC for MySQL connectivity, handling SQL operations like insertion, deletion, and updating of tasks. This module ensures data persistence, so tasks remain stored even after the application is closed.

4.1.4 Notifications Module

The notifications module employs Java's Timer class to provide real-time updates and reminders for tasks nearing their deadlines. This module ensures users are alerted to important deadlines and overdue tasks.

CODE:

1.DATABASE

```
package todoapp;
```

```
import java.sql.Connection;
```

```
import java.sql.DriverManager;
```

```
import java.sql.SQLException;
```

```
public class Database {
```

```
    private static final String URL = "jdbc:mysql://localhost:3306/ToDoApp";
```

```
    private static final String USER = "root";
```

```
    private static final String PASSWORD = "Giridharan7#";
```

```
    public static Connection getConnection() throws SQLException {
```

```
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }
}
```

2.TaskDAO

```
package todoapp;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import javax.swing.JOptionPane;

public class TaskDAO {

    // Add task to the database

    public void addTask(String description, String priority, String category, int
timeLimit) throws SQLException {

        String sql = "INSERT INTO tasks (description, priority, category,
time_limit) VALUES (?, ?, ?, ?)";

        try (Connection conn = Database.getConnection();

            PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, description);

            stmt.setString(2, priority);

            stmt.setString(3, category);
```

```
        stmt.setInt(4, timeLimit);

        stmt.executeUpdate();

    } catch (SQLException e) {

        JOptionPane.showMessageDialog(null, "Error adding task: " +
e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);

    }

}
```

```
// Load all tasks from the database

public ResultSet loadTasks() throws SQLException {

    String query = "SELECT description, time_limit, priority, category,
is_completed FROM tasks";

    Connection conn = null;

    Statement stmt = null;

    ResultSet rs = null;

    try {

        conn = Database.getConnection(); // Open a new connection

        stmt = conn.createStatement();

        rs = stmt.executeQuery(query); // Execute query and retrieve the result
set

        return rs; // Return the result set so you can process it later

    } catch (SQLException e) {
```

```
        JOptionPane.showMessageDialog(null, "Error loading tasks: " +
e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);

        if (rs != null) {

            rs.close(); // Ensure ResultSet is closed if an exception occurs

        }

        if (stmt != null) {

            stmt.close(); // Ensure Statement is closed if an exception occurs

        }

        if (conn != null) {

            conn.close(); // Ensure Connection is closed if an exception occurs

        }

        return null;

    }

}
```

```
// Insert task object into the database
```

```
public void insertTask(Task task) throws SQLException {

    String query = "INSERT INTO tasks (description, time_limit, priority,
category) VALUES (?, ?, ?, ?)";

    try (Connection conn = Database.getConnection();

        PreparedStatement stmt = conn.prepareStatement(query)) {

        stmt.setString(1, task.description);

        stmt.setInt(2, task.timeLeft); // timeLeft is used as the time limit here

        stmt.setString(3, task.priority);

        stmt.setString(4, task.category);

        stmt.executeUpdate();

    } catch (SQLException e) {
```

```
        JOptionPane.showMessageDialog(null, "Error inserting task: " +  
e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);  
    }  
}
```

```
// Remove task from the database by description
```

```
public void removeTask(String description) throws SQLException {  
    String sql = "DELETE FROM tasks WHERE description = ?";  
    try (Connection conn = Database.getConnection();  
        PreparedStatement stmt = conn.prepareStatement(sql)) {  
        stmt.setString(1, description);  
        stmt.executeUpdate();  
    } catch (SQLException e) {  
        JOptionPane.showMessageDialog(null, "Error removing task: " +  
e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);  
    }  
}
```

```
// Mark task as completed by description
```

```
public void markTaskAsCompleted(String description) throws SQLException  
{  
    String sql = "UPDATE tasks SET is_completed = TRUE WHERE  
description = ?";  
    try (Connection conn = Database.getConnection();  
        PreparedStatement stmt = conn.prepareStatement(sql)) {  
        stmt.setString(1, description);  
        stmt.executeUpdate();  
    } catch (SQLException e) {
```

```
        JOptionPane.showMessageDialog(null, "Error marking task as
completed: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
    }
}
```

```
// Update task details in the database
```

```
public void updateTaskDetails(String oldDescription, String newDescription,
int newTimeLimit, boolean isCompleted, String priority) throws SQLException
{
```

```
    String updateQuery = "UPDATE tasks SET description = ?, time_limit = ?,
is_completed = ?, priority = ? WHERE description = ?";
```

```
    try (Connection conn = Database.getConnection();
        PreparedStatement stmt = conn.prepareStatement(updateQuery)) {
        stmt.setString(1, newDescription); // Set the new description
        stmt.setInt(2, newTimeLimit);      // Set the new time limit (in seconds)
        stmt.setBoolean(3, isCompleted);   // Set the task completion status
        stmt.setString(4, priority);       // Set the updated priority
        stmt.setString(5, oldDescription); // Set the old description to locate the
task
        stmt.executeUpdate();
    }
```

```
// Optionally show a confirmation message
```

```
JOptionPane.showMessageDialog(null, "Task updated successfully");
```

```
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Error updating task: " +
e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
    }
```

```
    }  
    }  
}
```

3.TASK

// Task.java

```
package todoapp;
```

```
import javax.swing.*;
```

```
import java.util.Timer;
```

```
import java.util.TimerTask;
```

```
public final class Task {
```

```
    String description;
```

```
    String priority;
```

```
    String category;
```

```
    // Time limit in seconds
```

```
    int timeLeft; // Remaining time in seconds
```

```
    boolean isCompleted;
```

```
    Timer timer;
```

```
    // Label to display the remaining time
```

```
    // Constructor
```

```
    public Task(String description, String priority, String category, int timeLimit)  
    {  
        this.description = description;  
        this.priority = priority;
```

```
        this.category = category;

        this.timeLeft = timeLimit;

        this.isCompleted = false;

        new JLabel("Time Left: " + formatTime(timeLeft)); // Initialize the label
with formatted time

    }
```

```
// Format the time into HH:MM:SS

public String formatTime(int seconds) {

    int hours = seconds / 3600;

    int minutes = (seconds % 3600) / 60;

    int remainingSeconds = seconds % 60;

    return String.format("%02d:%02d:%02d", hours, minutes,
remainingSeconds);

}
```

```
public boolean isCompleted() {

    return isCompleted;

}
```

```
public void setCompleted(boolean isCompleted) {

    this.isCompleted = isCompleted;

}
```

```
public String getPriority() {

    return priority;

}
```

```
public String getDescription() {
    return description;
}

public int getTimeLeft() {
    return timeLeft;
}

// Method to set the remaining time left for the task
public void setTimeLeft(int timeLeft) {
    this.timeLeft = timeLeft;
}

public void setDescription(String description) {
    this.description = description;
}

public void markAsCompleted() {
    this.isCompleted = true;
    if (this.timer != null) {
        this.timer.cancel();
    }
}
}
```

4.UpdateTaskDialog

```
package todoapp;

import javax.swing.*.*;
import java.awt.*.*;
import java.sql.SQLException;
```

```
public class UpdateTaskDialog extends JDialog {  
    private final JTextField descriptionField;  
    private final JTextField timeField;  
    private final JCheckBox completedCheckBox;  
    private final JComboBox<String> priorityComboBox;  
    private final Task task;  
    private final MainScreen mainScreen; // Reference to MainScreen  
  
    public UpdateTaskDialog(MainScreen mainScreen, Task task) {  
        super(mainScreen, "Update Task", true);  
        this.mainScreen = mainScreen; // Store reference to MainScreen  
        this.task = task;  
  
        // Initialize components with current task data  
        descriptionField = new JTextField(task.description);  
        timeField = new JTextField(String.valueOf(task.timeLeft / 3600.0));  
        completedCheckBox = new JCheckBox("Completed", task.isCompleted);  
        priorityComboBox = new JComboBox<>(new String[] {"Low", "Medium",  
"High"});  
        priorityComboBox.setSelectedItem(task.priority);  
  
        setupLayout();  
        setupSaveButton();  
  
        setSize(300, 250);  
    }  
}
```

```
        setLocationRelativeTo(mainScreen);  
        setVisible(true);  
    }
```

```
// Setup layout using GridBagLayout
```

```
private void setupLayout() {  
    setLayout(new GridBagLayout());  
    GridBagConstraints gbc = new GridBagConstraints();  
    gbc.fill = GridBagConstraints.HORIZONTAL;  
    gbc.insets = new Insets(5, 5, 5, 5);  
  
    addComponent("Task Description:", descriptionField, 0, gbc);  
    addComponent("Time (hours):", timeField, 1, gbc);  
    addComponent("Completed:", completedCheckBox, 2, gbc);  
    addComponent("Priority:", priorityComboBox, 3, gbc);  
}
```

```
// Helper method to add components to the layout
```

```
private void addComponent(String label, Component component, int row,  
GridBagConstraints gbc) {  
    gbc.gridy = row;  
  
    gbc.gridx = 0;  
    gbc.weightx = 0.2;  
    add(new JLabel(label), gbc);  
  
    gbc.gridx = 1;
```

```
        gbc.weightx = 0.8;
        add(component, gbc);
    }

    // Set up save button with action listener
    private void setupSaveButton() {
        JButton saveButton = new JButton("Save");
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = 4;
        gbc.gridwidth = 2;
        gbc.insets = new Insets(10, 5, 5, 5);
        add(saveButton, gbc);

        saveButton.addActionListener(e -> handleSave());
    }

    // Handle the save operation and update task details
    private void handleSave() {
        try {
            // Get updated values from form fields
            String newDescription = descriptionField.getText().trim();
            double newTimeLimitInHours = validateTimeInput();
            int newTimeLimitInSeconds = (int) (newTimeLimitInHours * 3600);
            boolean isCompleted = completedCheckBox.isSelected();
            String priority = (String) priorityComboBox.getSelectedItem();
```

```
// Check if description is not empty
if (newDescription.isEmpty()) {
    showError("Task description cannot be empty.");
    return;
}
```

```
// Update the database with new task details
TaskDAO taskDAO = new TaskDAO();
taskDAO.updateTaskDetails(
    task.description,
    newDescription,
    newTimeLimitInSeconds,
    isCompleted,
    priority
);
```

```
// Update the in-memory task object
task.description = newDescription;
task.timeLeft = newTimeLimitInSeconds;
task.isCompleted = isCompleted;
task.priority = priority;
```

```
// Refresh the MainScreen task list and close dialog
mainScreen.refreshTasks();
dispose();
```

```
    } catch (NumberFormatException ex) {  
        showError("Invalid time input. Please enter a valid number for time.");  
    } catch (SQLException ex) {  
        showError("Failed to update task: " + ex.getMessage());  
    }  
}
```

```
// Validates time input and returns the value in hours
```

```
private double validateTimeInput() throws NumberFormatException {  
    String timeText = timeField.getText().trim();  
    if (timeText.isEmpty()) {  
        throw new NumberFormatException("Time field cannot be empty.");  
    }  
    double time = Double.parseDouble(timeText);  
    if (time <= 0) {  
        throw new NumberFormatException("Time must be greater than 0.");  
    }  
    return time;  
}
```

```
// Display error messages
```

```
private void showError(String message) {  
    JOptionPane.showMessageDialog(this,  
        message,  
        "Error",  
        JOptionPane.ERROR_MESSAGE);  
}
```

```
}  
}
```

5. CustomDialog

```
package todoapp;
```

```
import javax.swing.*;
```

```
import java.awt.*;
```

```
class CustomDialog extends JDialog {
```

```
    private final JLabel taskDescriptionLabel;
```

```
    private final JLabel timerLabel;
```

```
    // Constructor updated to accept taskDescription and timerString
```

```
    public CustomDialog(JFrame parentFrame, String title, String  
taskDescription, String timerString) {
```

```
        super(parentFrame, title, true);
```

```
        setSize(300, 200);
```

```
        setLocationRelativeTo(parentFrame);
```

```
        setLayout(new GridLayout(2, 1));
```

```
        taskDescriptionLabel = new JLabel("Task: " + taskDescription);
```

```
        taskDescriptionLabel.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
        timerLabel = new JLabel("Time Remaining: " + timerString);
```

```
        timerLabel.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
        add(taskDescriptionLabel);
        add(timerLabel);

        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    }

    // Method to update the timer label (in case the time changes while the dialog
    // is open)
    public void updateTimer(String timerString) {
        timerLabel.setText("Time Remaining: " + timerString);
    }
}
```

6.MAINSCREEN

```
package todoapp;

import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.HashMap;
import java.util.Map;
import java.util.Timer;
import java.util.TimerTask;
```

```
public class MainScreen extends JFrame {

    private final DefaultListModel<Task> taskModel = new
DefaultListModel<>();

    private final JList<Task> taskList = new JList<>(taskModel);

    private final TaskDAO taskDAO = new TaskDAO();

    private final Map<String, Task> tasks = new HashMap<>();


    public MainScreen() {

        setTitle("To-Do List");

        setSize(600, 400);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setLayout(new BorderLayout());

        setLocationRelativeTo(null); // Center the window on screen


        // Gradient panel for background

        KGradientPanel gradientPanel = new KGradientPanel();

        gradientPanel.setLayout(new BorderLayout());

        gradientPanel.setkStartColor(new Color(33, 150, 243)); // Start color of the
gradient

        gradientPanel.setkEndColor(new Color(255, 87, 34)); // End color of the
gradient


        // Create and style buttons with specified solid colors

        JButton addButton = createSmoothButton("Add Task", new Color(76, 175,
80), Color.WHITE);

        JButton removeButton = createSmoothButton("Remove Task", new
Color(244, 67, 54), Color.WHITE);

        JButton completeButton = createSmoothButton("Mark as Completed", new
Color(255, 193, 7), Color.WHITE);
```

```
        JButton getDetailsButton = createSmoothButton("Get Details", new
Color(33, 150, 243), Color.WHITE);

        JButton updateButton = createSmoothButton("Update Task", new
Color(156, 39, 176), Color.WHITE);


// Panel for buttons

JPanel buttonPanel = new JPanel(new GridLayout(1, 5, 10, 10));

buttonPanel.add(addButton);

buttonPanel.add(removeButton);

buttonPanel.add(updateButton);

buttonPanel.add(getDetailsButton);

buttonPanel.add(completeButton);


// Styling and positioning

taskList.setCellRenderer(new TaskRenderer());

gradientPanel.add(new JScrollPane(taskList), BorderLayout.CENTER);

gradientPanel.add(buttonPanel, BorderLayout.SOUTH);

setContentPane(gradientPanel);


// Load tasks from database

loadTasks();


// Button actions

addButton.addActionListener(e -> new AddTaskScreen(this));

removeButton.addActionListener(e -> removeTask());

updateButton.addActionListener(e -> updateTask());

getDetailsButton.addActionListener(e -> showTaskDetails());
```

```
completeButton.addActionListener(e -> markTaskAsCompleted());

setVisible(true);
}

// Create and style a button with solid color background and hover effect
private JButton createSmoothButton(String text, Color backgroundColor,
Color textColor) {
    JButton button = new JButton(text) {
        @Override
        protected void paintComponent(Graphics g) {
            Graphics2D g2d = (Graphics2D) g;

            g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

            if (getModel().isPressed()) {
                g2d.setColor(backgroundColor.darker());
            } else if (getModel().isRollover()) {
                g2d.setColor(backgroundColor.brighter());
            } else {
                g2d.setColor(backgroundColor);
            }

            g2d.fillRoundRect(0, 0, getWidth(), getHeight(), 20, 20);
            super.paintComponent(g);
        }
    };
};
```

```
button.addMouseListener(new MouseAdapter() {  
    @Override  
    public void mouseEntered(MouseEvent evt) {  
        button.setBackground(background-color.darker());  
    }  
  
    @Override  
    public void mouseExited(MouseEvent evt) {  
        button.setBackground(background-color);  
    }  
});  
  
return button;  
}  
  
public void loadTasks() {  
    taskModel.clear();  
    tasks.clear();  
    try (ResultSet rs = taskDAO.loadTasks()) {  
        while (rs.next()) {  
            String description = rs.getString("description");  
            int timeLimit = rs.getInt("time_limit");  
            String priority = rs.getString("priority");  
            String category = rs.getString("category");  
            boolean isCompleted = rs.getBoolean("is_completed");
```

```
Task task = new Task(description, priority, category, timeLimit);
if (isCompleted) {
    task.markAsCompleted(); // Mark as completed when loading from
DB
}
taskModel.addElement(task);
tasks.put(description, task);
if (!isCompleted) {
    startTaskTimer(task);
}
}
} catch (SQLException e) {
    JOptionPane.showMessageDialog(this, "Failed to load tasks: " +
e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
}
}
```

```
private void removeTask() {
    Task selectedTask = taskList.getSelectedValue();
    if (selectedTask != null) {
        try {
            taskDAO.removeTask(selectedTask.description);

            if (selectedTask.timer != null) {
                selectedTask.timer.cancel();
            }
        }
    }
}
```

```
        tasks.remove(selectedTask.description);
        taskModel.removeElement(selectedTask);
        JOptionPane.showMessageDialog(this, "Task removed successfully.");
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(this, "Failed to remove task: " +
e.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
    }
} else {
    JOptionPane.showMessageDialog(this, "Please select a task to remove.",
"Warning", JOptionPane.WARNING_MESSAGE);
}
}
```

```
private void markTaskAsCompleted() {
    Task selectedTask = taskList.getSelectedValue();
    if (selectedTask != null) {
        try {
            taskDAO.markTaskAsCompleted(selectedTask.description);
            selectedTask.markAsCompleted();
            taskDAO.removeTask(selectedTask.description);

            if (selectedTask.timer != null) {
                selectedTask.timer.cancel();
            }
        }
    }
}
```

```
tasks.remove(selectedTask.description);

// Create a SwingWorker to handle the sliding animation and task removal
new SwingWorker<Void, Void>() {

    @Override

    protected Void doInBackground() throws Exception {

        // Get the original position of the task in the list

        int originalY = taskList.getCellBounds(taskList.getSelectedIndex(),
taskList.getSelectedIndex()).y;

        // Animate the task sliding out (move it off the screen vertically)
        for (int i = 0; i < 50; i++) { // Adjust the iteration count for speed

            final int offsetY = i * 2; // Move 2 pixels per iteration

            SwingUtilities.invokeLater(() -> {

                // Move the task off the screen by adjusting the vertical position

                taskList.scrollRectToVisible(new Rectangle(0, originalY +
offsetY, taskList.getWidth(), taskList.getHeight()));

            });

            Thread.sleep(10); // Adjust speed

        }

        // Short delay before removing the task

        Thread.sleep(500);

        return null;

    }

}
```

```
@Override
protected void done() {
    // Remove the task from the list after the animation completes
    taskModel.removeElement(selectedTask);
    taskList.repaint();

    // Ensure the rendering for other tasks is intact by revalidating and
    repainting the task list
    taskList.revalidate(); // Revalidate the list to refresh its layout
    taskList.repaint(); // Repaint the list to restore the priority label

    // Show the success message after removing the task
    showCongratulations();
}

}.execute();

} catch (SQLException e) {
    JOptionPane.showMessageDialog(this, "Failed to mark task as
    completed: " + e.getMessage(),
        "Error", JOptionPane.ERROR_MESSAGE);
}

} else {
    JOptionPane.showMessageDialog(this, "Please select a task to mark as
    completed.", "Warning", JOptionPane.WARNING_MESSAGE);
}
}
```

```
public void startTaskTimer(Task task) {
    task.timer = new Timer();
    task.timer.schedule(new TimerTask() {
        @Override
        public void run() {
            task.timeLeft--;
            taskList.repaint();

            if (task.timeLeft <= 0) {
                task.timer.cancel();

                JOptionPane.showMessageDialog(MainScreen.this, "Task \"\" +
task.description + "\" has expired!", "Time Up",
JOptionPane.WARNING_MESSAGE);

                removeTask();
            }
        }
    }, 1000, 1000);
}

private void showCongratulations() {
    JOptionPane.showMessageDialog(this, "Congratulations! Task is marked as
completed.");
}

private void showTaskDetails() {
    Task selectedTask = taskList.getSelectedValue();
    if (selectedTask != null) {
```

```

        int hours = selectedTask.timeLeft / 3600;

        int minutes = (selectedTask.timeLeft % 3600) / 60;

        int seconds = selectedTask.timeLeft % 60;


        String timerString = String.format("%02d:%02d:%02d", hours, minutes,
seconds);

        String status = selectedTask.isCompleted ? "Completed" : "Pending";


        String message = String.format(

            "Task Name: %s\nPriority: %s\nCategory: %s\nTime Left:
%s\nStatus: %s",

            selectedTask.description, selectedTask.priority,
selectedTask.category, timerString, status

        );


        JOptionPane.showMessageDialog(this, message, "Task Details",
JOptionPane.INFORMATION_MESSAGE);

    } else {

        JOptionPane.showMessageDialog(this, "Please select a task to view
details.", "Warning", JOptionPane.WARNING_MESSAGE);

    }

}


private void updateTask() {

    Task selectedTask = taskList.getSelectedValue();

    if (selectedTask != null) {

        new UpdateTaskDialog(this, selectedTask);

    } else {

```

```
        JOptionPane.showMessageDialog(this, "Please select a task to update.",
"Warning", JOptionPane.WARNING_MESSAGE);

    }

}
```

```
public void refreshTasks() {

    loadTasks();

    taskList.repaint();

}

}
```

7.AddTaskScreen

// AddTaskScreen.java

```
package todoapp;
```

```
import javax.swing.*;
```

```
import java.awt.*;
```

```
import java.sql.SQLException;
```

```
import java.util.TimerTask;
```

```
public class AddTaskScreen extends JFrame {

    private final JTextField taskField = new JTextField();

    private final JComboBox<String> categoryBox;

    private final JComboBox<String> priorityComboBox;

    private final TaskDAO taskDAO = new TaskDAO();

    private final MainScreen mainScreen;

    private final JLabel timerLabel = new JLabel("Time Left: 00:00:00",
SwingConstants.CENTER); // Timer label
```

```
public AddTaskScreen(MainScreen mainScreen) {  
    this.mainScreen = mainScreen;  
    setTitle("Add New Task");  
    setSize(400, 350);  
    setLayout(new BorderLayout());  
  
    // Gradient Background  
    JPanel contentPanel = new JPanel(new BorderLayout()) {  
        @Override  
        protected void paintComponent(Graphics g) {  
            Graphics2D g2d = (Graphics2D) g;  
            g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,  
RenderingHints.VALUE_ANTIALIAS_ON);  
            Color gradientStart = new Color(200, 162, 200); // Light Purple  
            Color gradientEnd = new Color(128, 0, 128);  
            GradientPaint gradientPaint = new GradientPaint(0, 0, gradientStart, 0,  
getHeight(), gradientEnd);  
            g2d.setPaint(gradientPaint);  
            g2d.fillRect(0, 0, getWidth(), getHeight());  
            super.paintComponent(g);  
        }  
    };  
    contentPanel.setOpaque(false);  
    contentPanel.setLayout(new GridLayout(6, 1, 10, 10)); // Added extra space  
for the timer label
```

```
// Create and add components

JLabel taskLabel = new JLabel("Task Description:");
taskLabel.setForeground(Color.WHITE);
taskField.setPreferredSize(new Dimension(200, 30));
taskField.setToolTipText("Enter the description of the task");


JLabel priorityLabel = new JLabel("Priority:");
priorityLabel.setForeground(Color.WHITE);
String[] priorities = {"Low", "Medium", "High"};
priorityComboBox = new JComboBox<>(priorities);
priorityComboBox.setSelectedIndex(0);
priorityComboBox.setToolTipText("Select a priority for the task");// Set the
default selection to "Low"


JLabel categoryLabel = new JLabel("Category:");
categoryLabel.setForeground(Color.WHITE);
String[] categories = {"Work", "Personal", "Shopping", "Fitness"};
categoryBox = new JComboBox<>(categories);
categoryBox.setToolTipText("Select a category for the task");


// Save Button

JButton saveButton = new JButton("Save Task");
saveButton.setBackground(new Color(0, 128, 0));
saveButton.setForeground(Color.WHITE);
saveButton.setFocusPainted(false);
```

```
// Add timer label to the panel

timerLabel.setForeground(Color.WHITE);
timerLabel.setFont(new Font("Arial", Font.BOLD, 14));


// Adding components to the panel
contentPanel.add(taskLabel);
contentPanel.add(taskField);
contentPanel.add(priorityLabel);
contentPanel.add(priorityComboBox);
contentPanel.add(categoryLabel);
contentPanel.add(categoryBox);
contentPanel.add(timerLabel); // Added timer label at the bottom


add(contentPanel, BorderLayout.CENTER);
add(saveButton, BorderLayout.SOUTH);


// Action Listeners
saveButton.addActionListener(e -> saveTask());
taskField.addActionListener(e -> saveTask());


setVisible(true);
}


private void saveTask() {
    String description = taskField.getText().trim();
    String priority = (String) priorityComboBox.getSelectedItem();
```

```
String category = (String) categoryBox.getSelectedItem();

// Validate inputs
if (description.isEmpty()) {
    JOptionPane.showMessageDialog(this, "Task description cannot be
empty.", "Error", JOptionPane.ERROR_MESSAGE);
    return;
}

// Add this JComboBox for selecting priority

// Replace this line in your saveTask method

// Old validation part (removing the old numeric check)
if (priority == null) {
    JOptionPane.showMessageDialog(this, "Priority must be selected.", "Error",
JOptionPane.ERROR_MESSAGE);
    return;
}

// You can directly use the priority value (Low, Medium, High) now instead of
checking numeric ranges

try {
    String timeLimitHoursStr = JOptionPane.showInputDialog(this, "Enter
time limit in hours:");
```

```
if (timeLimitHoursStr == null) return;

double timeLimitHours = Double.parseDouble(timeLimitHoursStr);
int timeLimitInSeconds = (int) (timeLimitHours * 3600);

if (timeLimitInSeconds <= 0) {
    JOptionPane.showMessageDialog(this, "Time limit must be a positive
number.", "Error", JOptionPane.ERROR_MESSAGE);
    return;
}

// Add task to database
taskDAO.addTask(description, priority, category, timeLimitInSeconds);

// Create a new Task object and update MainScreen
Task newTask = new Task(description, priority, category,
timeLimitInSeconds);
mainScreen.loadTasks();
mainScreen.startTaskTimer(newTask);

// Start the timer on the task
startTimer(newTask);

dispose();
} catch (NumberFormatException e) {

    JOptionPane.showMessageDialog(this, "Please enter a valid number for
the time limit.", "Error", JOptionPane.ERROR_MESSAGE);
```

```
    } catch (SQLException e) {

        JOptionPane.showMessageDialog(this, "Failed to save task: " +
e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);

    }

}

private void startTimer(Task task) {

    // Start a timer for this specific task and update the label every second

    task.timer.scheduleAtFixedRate(new TimerTask() {

        @Override

        public void run() {

            if (task.timeLeft > 0) {

                task.timeLeft--;

                updateTimerLabel(task);

            } else {

                task.timer.cancel();

            }

        }

    }, 1000, 1000);

}

private void updateTimerLabel(Task task) {

    // Convert timeLeft from seconds to HH:MM:SS format and update the
label

    int hours = task.timeLeft / 3600;

    int minutes = (task.timeLeft % 3600) / 60;
```

```
        int seconds = task.timeLeft % 60;

        String timeString = String.format("%02d:%02d:%02d", hours, minutes,
seconds);

        timerLabel.setText("Time Left: " + timeString);

    }
}
```

8.SplashScreen

```
package todoapp;

import javax.swing.*;
import java.awt.*;
import java.util.Timer;
import java.util.TimerTask;

// Splash Screen class
class SplashScreen extends JWindow {
    private JProgressBar progressBar;

    public SplashScreen() {

        // Example gradient colors
        // Load splash image

        ImageIcon splashIcon = new
ImageIcon(getClass().getResource("/images/todo.png"));

        Image splashImage = splashIcon.getImage();

        // Get original image width and height
```

```
int imgWidth = splashImage.getWidth(null);
int imgHeight = splashImage.getHeight(null);

// Calculate scale factor to fit the splash screen dimensions (400x300)
double scale = Math.min(400.0 / imgWidth, 300.0 / imgHeight);

// Calculate new width and height while maintaining aspect ratio
int newWidth = (int) (imgWidth * scale);
int newHeight = (int) (imgHeight * scale);

// Scale the image
Image scaledImage = splashImage.getScaledInstance(newWidth,
newHeight, Image.SCALE_SMOOTH);

// Create a label to hold the scaled image
JLabel label = new JLabel(new ImageIcon(scaledImage));

// Set the splash screen size based on the scaled image and progress bar
height
setSize(newWidth, newHeight + 30); // Adding extra space for progress bar

// Center the splash screen in the middle of the screen
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
int x = (screenSize.width - getWidth()) / 2;
int y = (screenSize.height - getHeight()) / 2;
```

```
setLocation(x, y);
```

```
// Create and configure the progress bar  
progressBar = new JProgressBar();  
progressBar.setMinimum(0);  
progressBar.setMaximum(100);  
progressBar.setStringPainted(true);  
progressBar.setPreferredSize(new Dimension(newWidth, 20));  
progressBar.setForeground(Color.GREEN); // Set progress bar color to  
green
```

```
// Add the components to the splash screen  
getContentPane().setLayout(new BorderLayout());  
getContentPane().add(label, BorderLayout.CENTER);  
getContentPane().add(progressBar, BorderLayout.SOUTH);  
setVisible(true);
```

```
// Update the progress bar over 3 seconds  
Timer timer = new Timer();  
timer.schedule(new TimerTask() {  
    int progress = 0;
```

```
@Override
```

```
public void run() {  
    if (progress < 100) {  
        progress += 30;  
        progressBar.setValue(progress);  
    } else {  
        timer.cancel();  
        setVisible(false);  
        dispose();  
        new MainScreen(); // Launch the main application  
    }  
}  
}, 0, 300); // Updates every 300ms  
}
```

9.TaskRenderer

```
package todoapp;
```

```
import javax.swing.*;
```

```
import java.awt.*;
```

```
import java.awt.geom.RoundRectangle2D;
```

```
class TaskRenderer extends DefaultListCellRenderer {
```

```
    @Override
```

```
    public Component getListCellRendererComponent(JList<?> list, Object value,  
int index, boolean isSelected, boolean cellHasFocus) {
```

```
        Task task = (Task) value;
```

```
// Create a JPanel to organize the components

JPanel panel = new JPanel() {

    @Override

    protected void paintComponent(Graphics g) {

        super.paintComponent(g);

        Graphics2D g2d = (Graphics2D) g;

        // Apply gradient background

        GradientPaint gradient = new GradientPaint(0, 0, new Color(200, 162,
200), getWidth(), getHeight(), new Color(128, 0, 128), true);

        g2d.setPaint(gradient);

        g2d.fillRect(0, 0, getWidth(), getHeight());

        // If the task is selected, add a glowing border

        if (isSelected) {

            g2d.setColor(new Color(255, 255, 255, 150)); // Yellow glow

            g2d.setStroke(new BasicStroke(2));

            g2d.draw(new RoundRectangle2D.Double(0, 0, getWidth() - 1,
getHeight() - 1, 15, 15)); // Rounded border for glow effect

        }

    }

};

panel.setLayout(new BorderLayout(15, 15));

panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10)); //
Padding around content
```

```
// Task Description Label

JLabel descriptionLabel = new JLabel(task.description);
descriptionLabel.setFont(new Font("Montserrat", Font.BOLD, 20));

if (isSelected) {
    descriptionLabel.setForeground(Color.YELLOW); // Brighter yellow for
selected description
} else {
    descriptionLabel.setForeground(new Color(255, 255, 255, 100)); //
Further reduced opacity for unselected tasks (alpha = 100)
}

panel.add(descriptionLabel, BorderLayout.NORTH);

// Category Label (different background colors for each category)
JLabel categoryLabel = new JLabel(task.category);
categoryLabel.setFont(new Font("Arial", Font.ITALIC, 12));
categoryLabel.setOpaque(true);
categoryLabel.setHorizontalAlignment(SwingConstants.CENTER);
categoryLabel.setPreferredSize(new Dimension(100, 25));

switch (task.category.toLowerCase()) {
    case "work":
        categoryLabel.setBackground(new Color(0, 123, 255)); // Blue for
Work
        break;
    case "personal":
        categoryLabel.setBackground(new Color(40, 167, 69)); // Green for
Personal
```

```
        break;

        case "shopping":

            categoryLabel.setBackground(new Color(255, 193, 7)); // Yellow for
Shopping
            break;

        case "fitness":

            categoryLabel.setBackground(new Color(220, 53, 69)); // Red for
Fitness
            break;

        default:

            categoryLabel.setBackground(Color.GRAY); // Default for
unrecognized categories
            break;
    }

    if (!isSelected) {

        categoryLabel.setForeground(new Color(255, 255, 255, 100)); // Further
reduced opacity for unselected category label (alpha = 100)

    } else {

        categoryLabel.setForeground(Color.WHITE); // Brighter white for
selected category label

    }

    panel.add(categoryLabel, BorderLayout.WEST);

    // Priority Label (smaller and positioned next to category)
    JLabel priorityLabel = new JLabel(task.priority);

    priorityLabel.setFont(new Font("Arial", Font.ITALIC, 12));
```

```
priorityLabel.setOpaque(true);

priorityLabel.setPreferredSize(new Dimension(80, 25));

priorityLabel.setHorizontalAlignment(SwingConstants.CENTER);

// Color coding for priority levels
switch (task.priority.toLowerCase()) {
    case "high":
        priorityLabel.setBackground(new Color(220, 53, 69)); // Red for high
        priority
        break;
    case "medium":
        priorityLabel.setBackground(new Color(255, 193, 7)); // Yellow for
        medium priority
        break;
    case "low":
        priorityLabel.setBackground(new Color(108, 117, 125)); // Gray for
        low priority
        break;
    default:
        priorityLabel.setBackground(Color.LIGHT_GRAY); // Default color
        break;
}

if (!isSelected) {
    priorityLabel.setForeground(new Color(255, 255, 255, 100)); // Further
    reduced opacity for unselected priority label (alpha = 100)
} else {
```

```
        priorityLabel.setForeground(Color.WHITE); // Brighter white for
selected priority label
    }

    panel.add(priorityLabel, BorderLayout.EAST);

    // Timer Label positioned at the bottom
    JLabel timerLabel = new JLabel(formatTime(task.timeLeft));
    timerLabel.setFont(new Font("Arial", Font.PLAIN, 14));
    timerLabel.setHorizontalAlignment(SwingConstants.CENTER);

    if (!isSelected) {
        timerLabel.setForeground(new Color(255, 223, 186, 100)); // Further
reduced opacity for unselected task timer (alpha = 100)
    } else {
        timerLabel.setForeground(new Color(255, 223, 186)); // Light orange for
selected task timer
    }

    panel.add(timerLabel, BorderLayout.SOUTH);

    // Set selected background color (brighter for selected)
    if (isSelected) {
        panel.setBackground(new Color(90, 0, 90)); // Brighter purple for
selected task
    } else {
        panel.setBackground(new Color(100, 100, 100)); // Gray background for
unselected tasks
    }
}
```

```
    }

    return panel;
}

// Helper method to format the remaining time in HH:MM:SS format
private String formatTime(int timeLeft) {
    int hours = timeLeft / 3600;
    int minutes = (timeLeft % 3600) / 60;
    int seconds = timeLeft % 60;
    return String.format("%02d:%02d:%02d", hours, minutes, seconds);
}
}
```

CHAPTER 5: TESTING

5.1 TESTING STRATEGIES

Testing was conducted on multiple platforms and focused on various aspects, including functionality, usability, and compatibility. Each module underwent unit testing, while integration testing ensured the modules work seamlessly together.

5.2 TEST CASES

Test cases covered include:

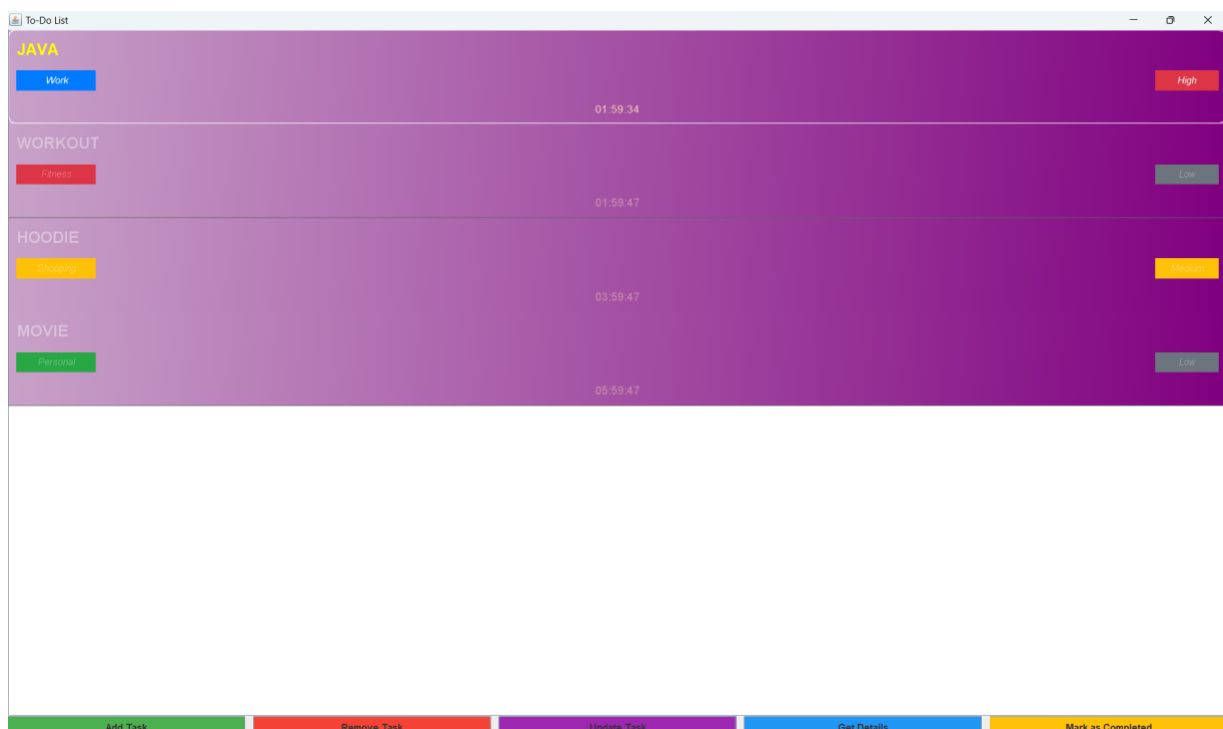
- **Task Addition:** Ensures tasks are added correctly and displayed in the list view.
- **Task Deletion:** Confirms that deleted tasks are removed from both the UI and the database.
- **Database Connectivity:** Verifies the connection to MySQL and data retrieval.

- **Reminder Notifications:** Checks that notifications appear for tasks close to their deadlines.

5.3 RESULTS AND DISCUSSION

Testing results demonstrated that the ToDo Application effectively meets its objectives, with each module performing as expected. Minor bugs identified during testing were resolved, and the application was validated for release.

MAINSCREEN:



ADD TASK SCREEN:

Add New Task

Task Description:

Priority:

Category:

Time Left: 00:00:00

Low

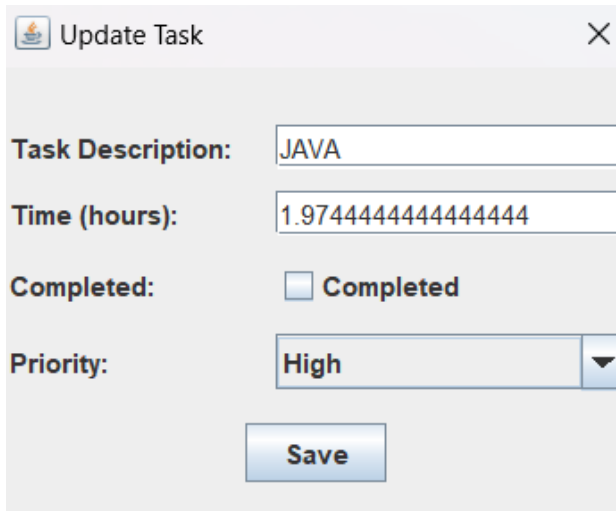
Work

Save Task

SPLASHSCREEN:



UPDATETASK:

A dialog box titled "Update Task" with a close button (X) in the top right corner. It contains four input fields: "Task Description:" with the text "JAVA", "Time (hours):" with the value "1.9744444444444444", "Completed:" with an unchecked checkbox and the label "Completed", and "Priority:" with a dropdown menu showing "High". A "Save" button is located at the bottom center.

Update Task X

Task Description: JAVA

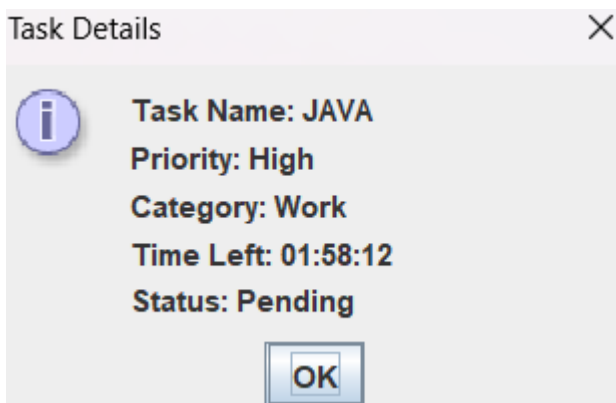
Time (hours): 1.9744444444444444

Completed: ☐ Completed

Priority: High ▼

Save

GET DETAILS:

A dialog box titled "Task Details" with a close button (X) in the top right corner. It features an information icon (i) on the left and a list of task details on the right: "Task Name: JAVA", "Priority: High", "Category: Work", "Time Left: 01:58:12", and "Status: Pending". An "OK" button is at the bottom center.

Task Details X

i **Task Name: JAVA**
Priority: High
Category: Work
Time Left: 01:58:12
Status: Pending

OK

CHAPTER 6: CONCLUSION

The ToDo Application provides a streamlined solution for managing daily tasks, offering users an efficient way to stay organized and prioritize effectively. By enabling real-time notifications, easy categorization, and intuitive priority settings, the application meets the fundamental needs of task management. Future development may include additional features such as cloud synchronization, collaborative task sharing, and advanced analytics for productivity tracking.

Chapter 7

REFERENCES

7.1 REFERENCES

1. **Java Swing Documentation**

Java Swing provides the necessary components for building GUI applications in Java. The official documentation covers essential components, layout managers, and event handling:

- [Java Swing Tutorial](#)

2. **JDBC (Java Database Connectivity)**

JDBC is critical for managing database connections, and the Oracle documentation offers a thorough guide on using JDBC with MySQL and other databases:

- [JDBC Basics](#)

3. **MySQL and Database Design for Java**

For designing and managing databases for Java applications, MySQL provides an accessible platform for creating relational databases:

- [MySQL Documentation](#)
- [Database Design Tutorial](#) – Covers the basics of designing relational databases.

4. **Java Swing Layout Management**

Layout management is vital for building responsive and user-friendly interfaces in Java:

- [Java Swing Layout Management](#)

5. **GitHub Examples for ToDo Applications**

Examining open-source examples can provide insight into building task-based applications in Java:

- [GitHub - Java ToDo Application](#)

6. **Java Timer Class for Scheduling**

For handling reminders or notifications, Java's Timer class documentation explains how to schedule tasks:

- [Java Timer Documentation](#)

7. JavaFX as an Alternative to Swing

If you want to explore other UI options, JavaFX offers advanced features and styling capabilities:

- [JavaFX Documentation](#)

These links provide a solid foundation for implementing a ToDo Application in Java with database support and a graphical interface.