Smart Task Manager

Problem Statement:

Develop a **Smart Task Manager** that helps users organize their daily tasks efficiently. The system should allow users to create, categorize, prioritize, and mark tasks as complete or pending. It should provide a **graphical user interface (GUI)** using **Swing**, implement **OOP principles**, and manage tasks using **collections**.

Core Features:

1. **Task Management:** Add, edit, delete, and view tasks.
2. **Categorization:** Tasks can be categorized (e.g., Work, Personal, Shopping, etc.).
3. **Prioritization:** Assign priority levels (High, Medium, Low).
4. **Sorting and Filtering:** Sort tasks based on priority, deadline, or category.
5. **Status Tracking:** Mark tasks as "Pending" or "Completed."
6. **Exception Handling:** Handle invalid inputs, empty lists, etc.
7. **Basic Multithreading:** Auto-save tasks periodically without freezing the UI.
8. **GUI (Swing):** Display tasks in a **JTable**, with buttons to manage them.
9. **MVC Architecture:** Separate UI, logic, and data.

Class Schema:

- **Task (Encapsulation, Basic Data Storage)**
    - Attributes: `title`, `category`, `priority`, `isCompleted`, `dueDate`
    - Methods: `getters and setters`, `toString()`
- **TaskManager (Business Logic, Uses Collections)**
    - Attributes: `List<Task> tasks`
    - Methods: `addTask()`, `removeTask()`, `updateTask()`, `sortTasks()`, `filterTasks()`
- **TaskUI (Swing, Inherits from JFrame, Implements ActionListener)**
    - Attributes: `JTable taskTable`, `JButton addTaskButton`, `deleteTaskButton`, `updateTaskButton`
    - Methods: `actionPerformed()`, `updateTable()`

Functionalities:

- **Adding Tasks:** User inputs task details, which are stored in a collection.
- **Editing Tasks:** Modify an existing task's details.
- **Deleting Tasks:** Remove a task from the list.
- **Sorting & Filtering:** Allow sorting by priority, category, or completion status.
- **Auto-Save:** Every 10 seconds, save tasks in a local file (Multithreading).

- **User-friendly GUI:** Display tasks in a structured table with buttons for actions.
- **Exception Handling:** Prevent invalid inputs and handle errors gracefully.

Extra Credit Opportunities:

Students can **improve** the project by adding:

1. **Custom Themes:** Users can switch between light/dark mode.
2. **Reminders:** Notify users about approaching deadlines.
3. **Undo Feature:** Restore deleted tasks within a session.
4. **Keyboard Shortcuts:** Speed up task management.
5. **Bulk Operations:** Edit multiple tasks at once.

Submission Format:

1. **Source Code:** Submit all Java files inside a ZIP file.
2. **Project Report (Typed, PDF Format):**
    - **Software Requirements Specification (SRS)**
    - **UML Diagrams:** Class Diagram, Use-Case Diagram
    - **Code Explanation:** Describe the implementation details
    - **Screenshots of Execution:** Show GUI & core functionalities
    - **Extra Credit Features (if added):** Justify their addition

---

Expense Tracker

Problem Statement:

Develop an **Expense Tracker** to help users manage their daily expenses. The system should allow users to add expenses, categorize them, and generate basic reports.

Core Features:

1. **Expense Management:** Add, edit, delete, and view expenses.
2. **Categorization:** Users can categorize expenses (e.g., Food, Transport, Bills, etc.).
3. **Monthly Summary:** Display total expenses by category for a selected month.
4. **Sorting & Filtering:** Sort by amount, date, or category.
5. **Graphical Report:** Display bar charts using **Swing components.**
6. **Exception Handling:** Handle invalid inputs and calculation errors.
7. **GUI (Swing):** Display transactions in a structured table.
8. **MVC Architecture:** Maintain separation between UI, logic, and data.

Class Schema:

- **Expense (Encapsulation, Basic Data Storage)**
  - Attributes: `amount`, `category`, `date`
  - Methods: `getters and setters`, `toString()`
- **ExpenseManager (Business Logic, Uses Collections)**
  - Attributes: `List<Expense> expenses`
  - Methods: `addExpense()`, `removeExpense()`, `generateReport()`, `sortExpenses()`
- **ExpenseUI (Swing, Inherits from JFrame, Implements ActionListener)**
  - Attributes: `JTable expenseTable`, `JButton addExpenseButton`, `deleteExpenseButton`
  - Methods: `actionPerformed()`, `updateTable()`

Functionalities:

- **Adding Expenses:** User inputs details and stores them.
- **Editing Expenses:** Modify an existing expense.
- **Deleting Expenses:** Remove an expense.
- **Sorting & Filtering:** Filter expenses by category, date, or amount.
- **Graphical Summary:** Display expense summary using bar charts.
- **Exception Handling:** Prevent invalid inputs and handle errors gracefully.

Extra Credit Opportunities:

Students can **improve** the project by adding:

1. **Budget Feature:** Allow users to set a monthly spending limit.
2. **Notification Alerts:** Notify when close to budget limit.
3. **Custom Reports:** Generate reports for a custom date range.
4. **Pie Chart Representation:** Show expenses by category visually.
5. **Export to CSV:** Allow users to export expense history.

Submission Format:

3. **Source Code:** Submit all Java files inside a ZIP file.
4. **Project Report (Typed, PDF Format):**
    - **Software Requirements Specification (SRS)**
    - **UML Diagrams:** Class Diagram, Use-Case Diagram
    - **Code Explanation:** Describe the implementation details
    - **Screenshots of Execution:** Show GUI & core functionalities
    - **Extra Credit Features (if added):** Justify their addition

---

Library Management System

Problem Statement:

Develop a **Library Management System** to keep track of books, borrowers, and due dates. Users should be able to borrow, return, and search for books efficiently.

Core Features:

1. **Book Management:** Add, remove, and update books.
2. **Borrowing System:** Borrow and return books with due dates.
3. **Overdue Tracking:** Highlight overdue books.
4. **Search Functionality:** Search books by title, author, or genre.
5. **Sorting & Filtering:** Sort books by availability, author, or genre.
6. **GUI (Swing):** Display book details in a **JTable**.

Class Schema:

- **Book (Encapsulation, Basic Data Storage)**
    - Attributes: `title`, `author`, `genre`, `isAvailable`
    - Methods: `getters and setters`, `toString()`
- **LibraryManager (Business Logic, Uses Collections)**
    - Attributes: `List<Book> books`
    - Methods: `addBook()`, `borrowBook()`, `returnBook()`, `searchBook()`
- **LibraryUI (Swing, Inherits from JFrame, Implements ActionListener)**
    - Attributes: `JTable bookTable`, `JButton borrowButton`, `returnButton`
    - Methods: `actionPerformed()`, `updateTable()`

Functionalities:

- **Borrowing & Returning:** Users can borrow or return books.
- **Search Books:** Search for books by title, author, or genre.
- **Overdue Management:** Highlight overdue books.
- **Sorting & Filtering:** Sort books by availability, author, or genre.

Extra Credit Opportunities:

1. **Automated Late Fee Calculation**
2. **Recommendations Based on Borrowing History**
3. **Book Reservation System**

Submission Format:

5. **Source Code:** Submit all Java files inside a ZIP file.
6. **Project Report (Typed, PDF Format):**

    - **Software Requirements Specification (SRS)**
    - **UML Diagrams:** Class Diagram, Use-Case Diagram
    - **Code Explanation:** Describe the implementation details
    - **Screenshots of Execution:** Show GUI & core functionalities
    - **Extra Credit Features (if added):** Justify their addition

---

Project: To-Do List Application

A user-friendly task management system with reminders and priority levels.

Problem Statement:

Develop a **To-Do List Application** that allows users to manage their daily tasks efficiently. The system should support adding, updating, and deleting tasks while allowing users to set deadlines, priorities, and optional reminders. The application should have a **simple graphical user interface (GUI) using Swing** for better usability.

Class Schema & Relationships:

- **Task** (Encapsulation, Inheritance)
  - Attributes: `taskId`, `title`, `description`, `dueDate`, `priority (LOW, MEDIUM, HIGH)`, `isCompleted`
  - Methods: `markCompleted()`, `updateTask()`, `getTaskDetails()`
- **Reminder** (Association with Task)
  - Attributes: `reminderId`, `taskId`, `reminderTime`, `isNotified`
  - Methods: `setReminder()`, `triggerNotification()`
- **TaskManager** (Uses **Collection Framework** to manage tasks)
  - Attributes: `taskList (ArrayList<Task>)`
  - Methods: `addTask()`, `removeTask()`, `updateTask()`, `getPendingTasks()`, `sortTasksByPriority()`
- **UserInterface (Swing-based GUI)**
  - Displays task list
  - Allows task addition, deletion, and modifications
  - Provides buttons to mark tasks as completed
  - Shows reminders and alerts
- **ToDoApp (Main Class - Implements MVC Pattern)**
  - Controls the application flow
  - Calls methods from `TaskManager` and updates the UI accordingly

Core Functionalities:

- Add, edit, delete tasks with title, description, due date, and priority

- Mark tasks as completed
- Set and manage reminders for tasks
- Sort tasks by priority (High to Low)
- Filter pending and completed tasks
- Graphical User Interface (Swing-based) for better usability

---

Extra Credit for Improvisation:

- Drag and drop feature for reordering tasks
- Dark mode support in GUI
- Recurring tasks feature (Daily, Weekly, Monthly)
- Custom notification sounds for reminders
- Export task list as a text or PDF file

Students **must justify** any modifications and explain how they improve the user experience.

Submission Format:

- **ZIP file containing:**
  - Source Code (`.java` files)
  - Executable JAR file (if applicable)
  - Report (Typed PDF) including:
    - Software Requirement Specification (SRS)
    - UML Diagrams (Class Diagram, Use Case Diagram, etc.)
    - Implementation Details & Explanation
    - Testing and Sample Outputs
    - Any Extra Features Added

Project: Student Grade Tracker

A system to manage students' grades, calculate performance, and generate reports.

Problem Statement:

Develop a **Student Grade Tracker** that allows teachers or students to input and manage grades for different subjects. The system should support adding, updating, and deleting grades, calculating averages, and generating performance reports. The application should use **Swing for the GUI** to provide an intuitive interface for users.

Class Schema & Relationships:

- **Student** (Encapsulation, Inheritance)
  - Attributes: `studentId`, `name`, `gradeLevel`, `subjectGrades (Map<Subject, Grade>)`
  - Methods: `addGrade()`, `updateGrade()`, `calculateAverage()`, `getReportCard()`
- **Subject** (Simple Class)
  - Attributes: `subjectId`, `subjectName`
- **Grade** (Encapsulation)
  - Attributes: `gradeValue`, `letterGrade`, `remarks`
  - Methods: `convertToLetterGrade()`, `assignRemarks()`
- **GradeManager** (Uses **Collection Framework** to manage students and grades)
  - Attributes: `studentList (ArrayList<Student>)`
  - Methods: `addStudent()`, `removeStudent()`, `updateStudentGrades()`, `generateClassReport()`
- **UserInterface (Swing-based GUI)**
  - Displays student list and their grades
  - Allows adding, modifying, and deleting grades
  - Provides a **button to generate a grade report**
- **GradeTrackerApp (Main Class - Implements MVC Pattern)**
  - Controls application flow
  - Calls methods from `GradeManager` and updates the UI accordingly

Core Functionalities:

- Add, edit, delete student records
- Assign grades for different subjects
- Calculate **average grade and overall performance**
- Convert numeric grades to letter grades (A, B, C, etc.)
- Generate **student performance reports**
- Graphical User Interface (**Swing-based**)

Extra Credit for Improvisation:

- **Graphical Reports** (bar charts or pie charts for grade distribution)
- **Sorting & Filtering** (e.g., list students with top grades)
- **Customizable Grading Scale** (support for different grading systems)
- **Attendance Tracking Integration**
- **Export reports as a PDF file**

Students **must justify** any modifications and explain how they improve the system.

Submission Format:

- **ZIP file containing:**
  - Source Code (`.java` files)
  - Executable JAR file (if applicable)
  - Report (Typed PDF) including:
    - **Software Requirement Specification (SRS)**
    - **UML Diagrams (Class Diagram, Use Case Diagram, etc.)**
    - **Implementation Details & Explanation**
    - **Testing and Sample Outputs**
    - **Any Extra Features Added**

Project: Inventory Management System

A system to manage stock levels, track inventory, and send low-stock alerts.

Problem Statement:

Develop an **Inventory Management System** that allows businesses to track their stock efficiently. The system should support adding, updating, and deleting products, keeping track of stock levels, and providing **alerts when stock is low**. The application should have a **Swing-based GUI** for easy interaction.

Class Schema & Relationships:

- **Product** (Encapsulation, Inheritance)
  - Attributes: `productId`, `name`, `category`, `quantity`, `price`, `reorderLevel`
  - Methods: `updateStock()`, `isLowStock()`, `getProductDetails()`
- **Inventory** (Uses **Collection Framework** to manage products)
  - Attributes: `productList (ArrayList<Product>)`
  - Methods: `addProduct()`, `removeProduct()`, `updateProduct()`, `getLowStockProducts()`
- **AlertSystem** (Observer Pattern)
  - Attributes: `alertList (List<Product>)`
  - Methods: `checkStockLevels()`, `sendLowStockAlert()`
- **UserInterface (Swing-based GUI)**
  - Displays inventory list
  - Allows adding, modifying, and deleting products
  - Provides alerts when products are running low
- **InventoryApp (Main Class - Implements MVC Pattern)**
  - Controls application flow
  - Calls methods from `Inventory` and updates the UI accordingly

Core Functionalities:

- Add, edit, delete product records
- Track stock levels for each product

- Alert system for **low stock warnings**
- Sort and filter products based on **categories or stock levels**
- Graphical User Interface (**Swing-based**)

Extra Credit for Improvisation:

- **Bulk Stock Import** (Allow adding multiple products at once)
- **Search Feature** (Filter products by name or category)
- **Graphical Representation** (Charts for stock levels)
- **Expiry Date Tracking** (For perishable goods)
- **Export Inventory Report to PDF**

Students **must justify** any modifications and explain how they improve the system.

Submission Format:

- **ZIP file containing:**
    - Source Code (`.java` files)
    - Executable JAR file (if applicable)
    - Report (Typed PDF) including:
        - **Software Requirement Specification (SRS)**
        - **UML Diagrams (Class Diagram, Use Case Diagram, etc.)**
        - **Implementation Details & Explanation**
        - **Testing and Sample Outputs**
        - **Any Extra Features Added**

Project: Simple Paint Application

A basic drawing application that allows users to create shapes, change colors, and erase drawings.

Problem Statement:

Develop a **Simple Paint Application** that enables users to draw different shapes (lines, rectangles, ovals, freehand), change colors, and erase parts of the drawing. The application should use **Swing for GUI** and support **basic mouse interactions** to create drawings.

Class Schema & Relationships:

- **Shape (Abstract Class - Demonstrates Abstraction & Inheritance)**
  - Attributes: `x, y, color`
  - Methods: `draw(Graphics g)`, `setColor()`, `resizeShape()`
- **Line, Rectangle, Oval (Extend Shape - Demonstrates Polymorphism)**
  - Methods: Override `draw(Graphics g)` to draw the respective shape
- **Canvas (Manages Drawing & Uses Collections to Store Shapes)**
  - Attributes: `shapeList (ArrayList<Shape>)`
  - Methods: `addShape()`, `removeShape()`, `clearCanvas()`, `repaintCanvas()`
- **PaintToolbox (Handles UI Elements like Buttons & Color Picker)**
  - Attributes: `selectedShape`, `selectedColor`, `eraserMode`
  - Methods: `changeColor()`, `selectShape()`, `toggleEraser()`
- **UserInterface (Swing-based GUI - Implements MVC Pattern)**
  - Provides **buttons for shape selection, color selection, eraser**
  - Displays **canvas where users draw**
- **PaintApp (Main Class - Controls Application Flow)**
  - Calls methods from `Canvas` and `PaintToolbox` to update UI

Core Functionalities:

- **Draw basic shapes** (lines, rectangles, ovals, freehand)
- **Color selection tool** to change shape colors

- **Eraser tool** to remove parts of the drawing
- **Undo & Clear Canvas options**
- **Graphical User Interface (Swing-based)**

Extra Credit for Improvisation:

- **Layering system** (Move shapes forward or backward)
- **Save & Load drawings as image files**
- **Add text annotation feature**
- **Gradient and texture filling**
- **Custom brush styles (dotted, dashed, etc.)**

Students **must justify** any modifications and explain how they improve the system.

Submission Format:

- **ZIP file containing:**
    - Source Code (`.java` files)
    - Executable JAR file (if applicable)
    - Report (Typed PDF) including:
        - **Software Requirement Specification (SRS)**
        - **UML Diagrams (Class Diagram, Use Case Diagram, etc.)**
        - **Implementation Details & Explanation**
        - **Testing and Sample Outputs**
        - **Any Extra Features Added**

Project: Fitness Tracker

A simple fitness tracking application to log workouts, track calories, and generate fitness reports.

Problem Statement:

Develop a **Fitness Tracker** that allows users to log their workouts, track calories burned, and generate fitness reports. The application should provide a **Swing-based GUI** where users can input exercises, set goals, and review progress.

Class Schema & Relationships:

- **User** (Encapsulation, Inheritance)
  - Attributes: `userId`, `name`, `age`, `weight`, `height`, `goalCalories`, `workoutHistory (List<Workout>)`
  - Methods: `updateProfile()`, `calculateBMI()`, `getWorkoutSummary()`
- **Workout** (Encapsulation)
  - Attributes: `workoutId`, `exerciseType`, `duration`, `caloriesBurned`
  - Methods: `calculateCalories()`, `getWorkoutDetails()`
- **WorkoutManager** (Uses **Collection Framework** to store workouts)
  - Attributes: `workoutList (ArrayList<Workout>)`
  - Methods: `addWorkout()`, `removeWorkout()`, `getTotalCaloriesBurned()`, `generateFitnessReport()`
- **UserInterface (Swing-based GUI)**
  - Displays workout history and calorie stats
  - Allows users to **log, modify, and delete workouts**
  - Provides a **fitness report generation feature**
- **FitnessApp (Main Class - Implements MVC Pattern)**
  - Controls application flow
  - Calls methods from `WorkoutManager` and updates the UI accordingly

Core Functionalities:

- **Add, edit, and delete workouts**
- **Track calories burned based on workout type & duration**

- **Calculate BMI based on user profile**
- **Generate fitness reports (total workouts, calories burned, progress tracking)**
- **Graphical User Interface (Swing-based)**

Extra Credit for Improvisation:

- **Graphical Progress Chart** (Calories burned vs. time)
- **Exercise Recommendations** (Suggest workouts based on fitness goals)
- **Customizable Workout Goals** (Daily/Weekly goals)
- **Dark Mode & UI Themes** for better user experience
- **Export fitness reports as a PDF file**

Students **must justify** any modifications and explain how they improve the system.

Submission Format:

- **ZIP file containing:**
  - Source Code (`.java` files)
  - Executable JAR file (if applicable)
  - Report (Typed PDF) including:
    - **Software Requirement Specification (SRS)**
    - **UML Diagrams (Class Diagram, Use Case Diagram, etc.)**
    - **Implementation Details & Explanation**
    - **Testing and Sample Outputs**
    - **Any Extra Features Added**

Project: Job Application Manager

A system to track job applications, interview dates, and application statuses.

Problem Statement:

Develop a **Job Application Manager** that helps users track their job applications, manage interview schedules, and monitor application statuses. The system should allow users to log details about **applied jobs**, set reminders for interviews, and update application statuses. The application should use a **Swing-based GUI** for easy interaction.

Class Schema & Relationships:

- **JobApplication** (Encapsulation)
  - Attributes: `jobId`, `companyName`, `position`, `applicationDate`, `status`, `interviewDate`, `notes`
  - Methods: `updateStatus()`, `rescheduleInterview()`, `getApplicationDetails()`
- **JobManager** (Uses **Collection Framework** to store applications)
  - Attributes: `jobList (ArrayList<JobApplication>)`
  - Methods: `addJob()`, `removeJob()`, `updateJobStatus()`, `getUpcomingInterviews()`
- **ReminderSystem** (Implements **Multithreading** for Alerts)
  - Attributes: `reminderList (List<JobApplication>)`
  - Methods: `scheduleReminder()`, `sendInterviewAlert()`
- **UserInterface (Swing-based GUI)**
  - Displays **job applications list**
  - Allows users to **add, modify, and delete job applications**
  - Provides **interview reminders and status tracking**
- **JobApplicationApp (Main Class - Implements MVC Pattern)**
  - Controls application flow
  - Calls methods from `JobManager` and updates the UI accordingly

Core Functionalities:

- **Add, edit, delete job applications**

- **Track application status (Pending, Interview Scheduled, Accepted, Rejected, etc.)**
- **Set reminders for upcoming interviews**
- **Sort and filter job applications by status, company, or position**
- **Graphical User Interface (Swing-based)**

Extra Credit for Improvisation:

- **Priority System** (Mark high-priority jobs)
- **Export job application report as a PDF file**
- **Customizable notification settings for reminders**
- **Progress Tracking** (Visual indicator for application success rate)
- **Dark Mode & UI Themes** for better user experience

Students **must justify** any modifications and explain how they improve the system.

Submission Format:

- **ZIP file containing:**
  - Source Code (`.java` files)
  - Executable JAR file (if applicable)
  - Report (Typed PDF) including:
    - **Software Requirement Specification (SRS)**
    - **UML Diagrams (Class Diagram, Use Case Diagram, etc.)**
    - **Implementation Details & Explanation**
    - **Testing and Sample Outputs**
    - **Any Extra Features Added**

Project: Car Service Tracker

A system to manage car maintenance records, schedule service reminders, and track service history.

Problem Statement:

Develop a **Car Service Tracker** that helps users keep track of their vehicle maintenance, set reminders for upcoming services, and maintain a history of past services. The system should provide a **Swing-based GUI** for easy interaction, allowing users to log service details, track costs, and receive reminders for scheduled maintenance.

Class Schema & Relationships:

- **Car** (Encapsulation)
  - Attributes: `carId`, `ownerName`, `brand`, `model`, `year`, `lastServiceDate`, `nextServiceDue`
  - Methods: `updateServiceDate()`, `calculateNextServiceDate()`, `getCarDetails()`
- **ServiceRecord** (Encapsulation)
  - Attributes: `serviceId`, `carId`, `serviceType`, `serviceDate`, `cost`, `mechanicNotes`
  - Methods: `getServiceDetails()`, `updateServiceCost()`
- **ServiceManager** (Uses **Collection Framework** to store service records)
  - Attributes: `serviceList (ArrayList<ServiceRecord>)`
  - Methods: `addServiceRecord()`, `removeServiceRecord()`, `getServiceHistory()`
- **ReminderSystem** (Implements **Multithreading** for Alerts)
  - Attributes: `reminderList (List<Car>)`
  - Methods: `scheduleReminder()`, `sendServiceAlert()`
- **UserInterface (Swing-based GUI)**
  - Displays **car and service records**
  - Allows users to **add, modify, and delete service records**
  - Provides **service reminders and cost tracking**
- **CarServiceApp (Main Class - Implements MVC Pattern)**
  - Controls application flow

    - Calls methods from `ServiceManager` and updates the UI accordingly

---

Core Functionalities:

- **Register and manage multiple cars**
- **Log and track maintenance records**
- **Set reminders for next service due date**
- **Calculate service expenses over time**
- **Graphical User Interface (Swing-based)**

Extra Credit for Improvisation:

- **Service Cost Analysis** (Charts for cost trends over time)
- **Predictive Maintenance Alerts** (Based on service history patterns)
- **Multiple Reminder Options** (Email, pop-up notifications)
- **Dark Mode & UI Themes** for better user experience
- **Export service history as a PDF file**

Students **must justify** any modifications and explain how they improve the system.

Submission Format:

- **ZIP file containing:**
    - Source Code (`.java` files)
    - Executable JAR file (if applicable)
    - Report (Typed PDF) including:
        - **Software Requirement Specification (SRS)**
        - **UML Diagrams (Class Diagram, Use Case Diagram, etc.)**
        - **Implementation Details & Explanation**
        - **Testing and Sample Outputs**
        - **Any Extra Features Added**

More Project Ideas to Choose:

If you are choosing any of the following, create a Requirements file as all the previous ones before, and get it verified from the faculty.

1. **Personal Expense Tracker** – A tool to record daily expenses, categorize them, and generate monthly spending reports with budget tracking.

2. **Daily Habit Tracker** – A habit-building app where users can log daily activities, set goals, and track progress with visual reports.

3. **Pet Care Manager** – A system to manage pet health records, vaccinations, feeding schedules, and veterinary appointments with reminders.

4. **Book Reading Tracker** – A personal library app that allows users to log books, track reading progress, and set reading goals.

5. **Event Planning Assistant** – A scheduling tool that helps users manage events, send invitations, and track RSVP responses.

6. **Freelance Work Organizer** – A task and payment tracking system for freelancers, managing client projects, deadlines, and earnings.

7. **Smart Grocery List** – An application to create and manage grocery lists, suggest items based on past purchases, and track inventory at home.

8. **Personal Fitness Coach** – A fitness app that generates customized workout plans and tracks progress based on user-defined goals.

9. **Recipe Organizer** – A digital cookbook where users can save, categorize, and modify recipes with a meal planning feature.

10. **Student Attendance Manager** – A system for teachers or students to log attendance, generate reports, and track attendance trends.

11. **Home Chore Scheduler** – A shared task manager that assigns and tracks household chores for roommates or family members.

12. **Gardening Planner** – A tool to track plant growth, watering schedules, and gardening activities with reminders.

13. **Mood & Emotion Tracker** – A daily journal where users can log moods, describe feelings, and analyze emotional trends over time.

14. **Parking Lot Manager** – A car parking assistant that tracks available parking spots and logs vehicle entries and exits.

15. **Bill Payment Reminder** – A finance app to track due dates for utility bills, loans, and subscriptions with alert notifications.

16. **Time Management Tracker** – A productivity tool that tracks how users spend time on various tasks and suggests improvements.

17. **Waste Management Logger** – A system to track household waste and suggest recycling habits to reduce waste generation.

18. **Local Business Directory** – A catalog for small businesses where users can find nearby services and contact vendors easily.

19. **Music Playlist Organizer** – A tool to create and manage personal playlists with sorting based on mood, genre, or artist.

20. **Volunteer Activity Tracker** – A system for individuals or organizations to log volunteer work, track hours, and generate reports.

21. **Personal Loan Tracker** – An app that helps users track borrowed or lent money, with repayment schedules and reminders.

22. **Smart Diet Planner** – A nutrition assistant that suggests meal plans based on dietary goals and calorie intake.

23. **Online Course Progress Tracker** – A system to manage online course enrollments, track completion status, and set learning goals.

24. **Home Workout Planner** – A virtual personal trainer that generates daily workout routines and tracks fitness progress.

25. **Art Portfolio Manager** – An application for artists to store and showcase their digital artwork with categorization options.

26. **Shopping Wish List** – A tool to manage shopping goals, compare product prices, and track discounts.

27. **Vehicle Fuel Log** – A fuel tracking system to log fuel consumption, calculate mileage, and track expenses.

28. **Sleep Pattern Monitor** – A health tracking app that logs sleep duration, suggests improvements, and tracks sleep quality.

29. **Gift Reminder & Organizer** – A planner that tracks gift ideas for upcoming birthdays, anniversaries, and special occasions.

30. **Home Inventory Manager** – A tool to track home assets, appliances, warranties, and maintenance schedules.
