1. **INTRODUCTION**

**1.1 Introduction**

The Task Manager is a web application thoughtfully designed to streamline task organization and enhance productivity. In an increasingly busy world, keeping track of various responsibilities and deadlines can be challenging. This application provides a central platform where users can easily add, view, and manage tasks with due dates and times. When a task’s scheduled time arrives, the Task Manager sends an automated email notification, serving as a timely reminder and helping users stay on top of their priorities without needing to constantly monitor their to-do lists.

The automated email alert system is a standout feature that supports seamless productivity by ensuring tasks are completed on time. As the due date approaches, the Task Manager sends reminders directly to the user’s email, reducing the risk of missed deadlines and helping them focus on high-priority tasks. This feature empowers users to be proactive in managing their tasks, enabling a more organized approach to day-to-day responsibilities. With this system, users can avoid the stress of forgotten tasks and feel confident that they will be reminded precisely when they need to act.

The Task Manager’s interface is designed for simplicity and ease of use, making it accessible for users with diverse needs and technical backgrounds. Each feature is crafted to provide a seamless experience—from adding a new task to editing existing ones. With an intuitive layout, users can quickly navigate through their tasks, prioritize responsibilities, and adjust as needed. The application is adaptable, whether users are managing personal to-dos, work-related tasks, or larger projects that require consistent tracking and reminders.

This tool serves as more than just a task organizer; it acts as a personal productivity assistant. By allowing users to offload the mental burden of remembering deadlines, the Task Manager promotes an organized and stress-free workflow. As a reliable and convenient solution for task tracking and management, the Task Manager helps users achieve their goals by ensuring they are always notified when tasks are due, leading to improved efficiency and focus throughout their day.

**1.2 Motivation**

The Task Manager project addresses the need for effective task management in today’s busy world. It allows users to schedule tasks with specific deadlines and receive automated email reminders, helping them stay organized without constant oversight. Designed for various users, the app features an intuitive interface and timely notifications that enhance focus, reduce stress, and boost productivity. By streamlining task handling, the Task Manager empowers users to manage their time effectively and achieve their goals effortlessly.

**1.3 Scope**

The project also includes a secure login system to protect user data, enabling individuals to manage their tasks privately. Beyond individual task tracking, the Task Manager can be expanded to include features such as priority settings, task categorization, and completion tracking, providing a holistic approach to productivity and time management. In the future, this tool could be scaled to support teams or groups, making it useful for collaborative task management in both personal and professional contexts.

**1.4 Project Outline**

Chapter-1 Introduction

Chapter-2 Literature Survey

Chapter-3 System Study and Analysis

Chapter-4 Software Design

Chapter-5 Technologies

Chapter-6 Implementation

Chapter-7 Testing

Chapter-8 Screenshots

Chapter-9 Conclusion and Future Work

Chapter-10 References

**2. LITERATURE SUREVY**

**Title:** What a To-Do: Studies of Task Management Towards the Design of a Personal Task List Manager

**Authors:**Victoria Bellotti, Brinda Dalal, Nathaniel Good, Peter Flynn, Daniel G. Bobrow, and Nicolas Ducheneaut.

**Published Year:**2004

The paper "What a To-Do: Studies of Task Management Towards the Design of a Personal Task List Manager" by Victoria Bellotti et al. explores effective personal task management strategies. It highlights users' efficient approaches to organizing tasks and suggests design features for task management tools to enhance user engagement and completion rates. The research provides insights for creating better personal task managers that align with user behaviours.

**Title:**The Psychology of Task Management: The Smaller Tasks Trap

**Authors:**Zohar Rusou, Moty Amar, and Shahar Ayal

**Published Year:**2020

The paper "The Psychology of Task Management: The Smaller Tasks Trap" by Zohar Rusou, Moty Amar, and Shahar Ayal examines why people often prioritize smaller, easier tasks over larger, more impactful ones. This behavior, termed the "smaller tasks trap," is linked to lower rational thinking and can reduce overall productivity. The authors suggest that this bias might be addressed by designing task management tools that help users focus on higher-priority tasks, rather than defaulting to the simpler ones​.

**Title:**An Intelligent Personal Assistant for Task and Time Management

**Authors:**Karen Myers, Pauline Berry, Jim Blythe, Ken Conley, Melinda Gervasio, Deborah McGuinness, David Morley, Avi Pfeffer, Martha Pollack, and Milind Tambe

**Published Year:**2007

The paper "An Intelligent Personal Assistant for Task and Time Management" by Karen Myers et al. describes a user-centric AI system designed to manage tasks by automating routine actions and helping users focus on complex tasks. Using the Belief-Desire-Intention framework, the assistant adapts to individual work styles, reducing cognitive load and enhancing productivity.

**3. SYSTEM STUDY AND ANALYSIS**

**3.1 Problem Statement**

Many tools are built for large teams or complex projects, making them too complicated for personal use. They also don’t always offer real-time notifications or reminders, which can lead to missed deadlines and forgotten tasks. Additionally, these tools usually need an internet connection to work, which limits access when offline. Finally, they often lack features like progress tracking or feedback that help users stay motivated. This project aims to create a straightforward task manager with personalized notifications and essential features to support individual productivity.

**3.2 Existing System**

The existing task management systems on the market offer features like task lists, reminders, and basic collaboration options. However, many are designed primarily for teams and large projects, which adds unnecessary complexity for individual users. While they do provide some scheduling and prioritization tools, most lack flexible, real-time notifications that can adapt to a user’s changing needs. Additionally, many tools depend on internet connectivity, limiting offline access. As a result, existing systems can be overwhelming and less effective for users seeking simple, personalized task management.

**3.3 Limitations of the Existing System**

Existing task management systems are often too complex for personal use, as they’re usually designed for team collaboration. Many lack personalization options, so users can’t customize the tool to fit their specific needs. They also don’t offer flexible, real-time notifications, which can lead to missed tasks. Additionally, most require an internet connection, making them hard to use offline. These limitations show the need for a simpler, more personalized task manager that suits individual needs.

**3.4 Proposed System**

The proposed system is a user-friendly task manager tailored for individual use, focusing on simplicity and effective task management. It includes a clean, intuitive interface where users can easily add, prioritize, and track tasks. Key features like real-time notifications, customizable reminders, and offline access ensure users stay on top of their tasks without missing deadlines. The system also includes progress tracking to help users monitor their productivity and stay motivated. By addressing the limitations of existing systems, this provides a streamlined and personalized experience for managing daily tasks efficiently**.**

**3.5 Advantages of Proposed System**

The proposed task manager offers a simplified, personalized experience, making it easy for individuals to organize and prioritize tasks. Real-time notifications and reminders help users stay on schedule, while offline access ensures continuous availability. Additionally, progress tracking keeps users motivated and improves productivity, addressing the key limitations found in existing task management systems.

**3.6 Functional Requirements**

* **User Registration and Login:** Users should be able to create accounts and log in securely.
* **Task Management:**Users can add, view, update, and delete tasks
* **Notifications:** The system should send alerts or notifications based on task deadlines or other criteria.

**3.7 Non-Functional Requirements**

* **Usability**: The interface should be intuitive and easy to navigate.
* **Scalability**: The system should be able to handle a growing number of users and tasks.

**3.8 User Interface Requirements**

**3.8.1 System Requirements**

The web application requires a local development environment MySQL. Users should have a modern web browser (such as Chrome, Firefox, or Safari) and a minimum of 4GB RAM and a dual-core processor for optimal performance.

**3.8.1.1 Hardware Requirements**

A dual-core processor (Intel i3 or equivalent) with at least 8GB RAM is recommended to ensure smooth operation of the application. Additionally, a minimum of 20GB of free disk space is necessary for installing MySQL and storing project files, along with a stable internet connection for updates and testing.

**3.8.1.2 Software Requirements**

The application development will utilize HTML, CSS, JavaScript and requiring a code editor (like Visual Studio Code or Sublime Text). Users must have MySQL installed for the web server and database management.

**4. SYSTEM DESIGN**

**4.1 System Architecture Design**

Database

Login page

EnterTask

Time&Date

Add Task

Task

Manager

Remove Task

Mark Complete

EnterTask

Time&Date

**Fig 1: System Architecture Design**

The design features the following components:

1. **Database:** Manages and stores user data, task information, and timestamps.
2. **Login Page:** Provides user authentication and access control to ensure that only authorized users can manage tasks.
3. **Task Manager Module:** The core of the application, where users can:
   * Enter Task: Allows users to add new tasks.
   * Add Task: Another function to create new tasks in the system.
   * Remove Task: Provides users with the ability to delete tasks.
   * Mark Complete: Lets users mark tasks as completed.
4. **Time & Date Tracking:** Adds timestamps for task creation and completion, helping users to manage tasks based on deadlines or timestamps.

This architecture outlines a basic task management system designed for organizing and tracking user tasks with essential CRUD (Create, Read, Update, Delete) operations

**4.2 UML Diagrams**

A UML diagram shows the unified visual presentation of the UML (Unified Modeling Language) system intending to let developers or business owners understand, analyze, and undertake the structure and behaviors of their system.So far, the UML diagram has become one of the most common business process modeling tools, which is also highly significant to the development of object-oriented software.UML diagrams have many benefits for both software developers and businesspeople.

The most key advantages are:

• **Problem-Solving** - Enterprises can improve their product quality and reduce cost especially for complex systems in large scale. Some other real-life problems including physical distribution or security can be solved;

• **Improve Productivity** - By using the UML diagram, everyone in the team is on the same page and lots of time are saved down the line;

• **Easy to Understand** - Since different roles are interested in different aspects of the system, the UML diagram offers non-professional developers, for example, stakeholders, designers, or business researchers, a clear and expressive presentation of requirements, functions and processes of their system.

**⮚Class Diagram:**

**1.User Class**:

* + **Attributes**:
    - id : int: A private attribute representing the user's unique identifier.
    - name : String: A private attribute representing the user's name.
  + **Methods**:
    - login() : boolean: A public method that handles user login and returns a boolean.
    - logout() : void: A public method that handles user logout and returns no value.

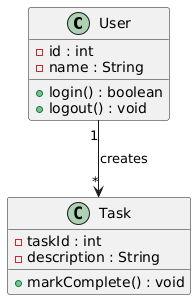
**2.Task Class**:

* + **Attributes**:
    - taskId : int: A private attribute representing the unique identifier for the task.
    - description : String: A private attribute describing the task.
  + **Methods**:
    - markComplete() : void: A public method that marks the task as complete and returns no value.

**3.Relationship**:

* + There is an association between User and Task, labeled**"creates"**.
  + The notation 1 to \* indicates a **one-to-many relationship**; each User can create multiple Task instances.

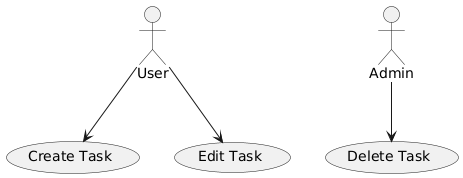
This diagram models a system where users can create and manage multiple tasks, reflecting a common task management application structure.

****

**Fig 2: Class Diagram**

**⮚Use Case Diagram**

* **User Roles:**
* User: A regular user who can create and edit tasks.
* Admin: A special user with the additional ability to delete tasks.
* **Actions:**
* Create Task: The user can add new tasks to the system.
* Edit Task: The user can update or change existing tasks.
* Delete Task: Only the admin has permission to remove tasks.

** Fig 3: Use Case Diagram**

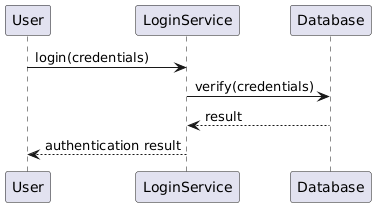
**⮚Sequence Diagram**

**• User initiates login**: The user sends their credentials (username and password) to the LoginService by calling the login(credentials) method.

**• LoginService verifies credentials**: The LoginService then communicates withthe Database to verify the user’s credentials by calling the verify(credentials) method.

**• Database responds with result**: The Database processes the verification request and returns a result (successful or failed authentication) to the LoginService.

**• LoginService returns authentication result**: Based on the response from the Database, the LoginService sends an authentication result back to the User, indicating whether the login was successful or not.

**Fig 4: Sequence Diagram**

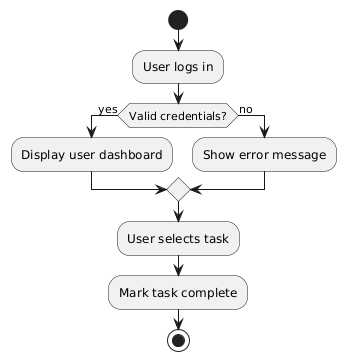
**⮚Activity Diagram**

* **User logs in**: The process begins when the user attempts to log in.
* **Valid credentials**?: A decision point checks whether the user's login credentials are valid.

i. If yes, the flow continues to Display user dashboard.

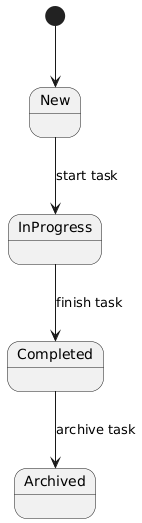
ii. If no, an Error message is shown to the user.

* **User selects task**: Once the user is on their dashboard, they can choose a task.
* **Mark task complete**: After selecting a task, the user can mark it as complete.

The process ends after the task is marked complete, as indicated by the final end marker.

**Fig 5: Activity Diagram**

**⮚State Diagram**

* **New:** The task starts in the "New" state, indicating it has been created but not yet started.
* **Start task:** When the task is started, it moves to the "InProgress" state.
* **InProgress:** The task is now being actively worked on.
* finish task: Once the work on the task is completed, it moves to the "Completed" state.
* **Completed:** The task has been finished.
* **archive task:** After completion, the task can be archived, moving it to the "Archived" state.
* **Archived:** The task is now archived, indicating it has been stored for future reference and is no longer active.

**Fig 6: State Diagram**

**5. TECHNOLOGIES**

**5.1 Python**

**5.1.1 Introduction of Python**

⮚Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

⮚Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

⮚Python is Interactive − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

⮚Python is Object-Oriented − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

⮚Python is a Beginner's Language − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

**5.1.2 History of python**

Python is a widely-used general-purpose, high-level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code. In the late 1980s, history was about to be written. It was that time when working on Python started. Soon after that, Guido Van Rossum began doing its application-based work in December of 1989 at Centrum Wiskunde& Informatica (CWI) which is situated in the Netherlands. It was started firstly as a hobby project because he was looking for an interesting project to keep him occupied during Christmas. The programming language in which Python is said to have succeeded is ABC Programming Language, which had interfacing with the Amoeba Operating System and had the feature of exception handling. He had already helped to create ABC earlier in his career and he had seen some issues with ABC but liked most of the features. After that what he did was really very clever. He had taken the syntax of ABC, and some of its good features. It came with a lot of complaints too, so he fixed those issues completely and had created a good scripting language that had removed all the flaws. The inspiration for the name 17 came from BBC‟s TV Show – „Monty Python‟s Flying Circus‟, as he was a big fan of the TV show and also he wanted a short, unique and slightly mysterious name for his invention and hence he named it Python! He was the “Benevolent dictator for life” (BDFL) until he stepped down from the position as the leader on 12th July 2018. For quite some time he used to work for Google, but currently, he is working at Dropbox. The language was finally released in 1991. When it was released, it used a lot fewer codes to express the concepts, when we compare it with Java, C++ & C. Its design philosophy was quite good too. Its main objective is to provide code readability and advanced developer productivity. When it was released it had more than enough capability to provide classes with inheritance, several core data types exception handling and functions.

**5.1.3 Features of Python**

Python provides many useful features which make it popular and valuable from the other programming languages. It supports object-oriented programming, procedural programming approaches and provides dynamic memory allocation. We have listed below a few essential features.

**1. Easy to learn and use:** Python is easy to learn as compared to other programming languages. Its syntax is straightforward and much the same as the English language. There is no use of the semicolon or curly-bracket, the indentation defines the code block. It is the recommended programming language for beginners.

**2. Expressive Language:** Python can perform complex tasks using a few lines of code. A simple example, the hello world program you simply type print("Hello World"). It will take only one line to execute, while Java or C takes multiple lines.

**3. Interpreted Language:** Python is an interpreted language; it means the Python program is executed one line at a time. The advantage of being interpreted language, it makes debugging easy and portable.

**4. Cross-platform Language:** Python can run equally on different platforms such as Windows, Linux, UNIX, and Macintosh, etc. So, we can say that Python is a portable language. It enables programmers to develop the software for several competing platforms by writing a program only once.

**5. Free and Open Source:** Python is freely available for everyone. It is freely available on its official website www.python.org. It has a large community across the world that is dedicatedly working towards make new python modules and functions. Anyone can contribute to the Python community. The open-source means, "Anyone can download its source code without paying any penny."

**6. Object-Oriented Language:** Python supports object-oriented language and concepts of classes and objects come into existence. It supports inheritance, polymorphism, and encapsulation, etc. The object-oriented procedure helps to programmer to write reusable code and develop applications in less code.

**7. Extensible:** It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our Python code. It converts the program into byte code, and any platform can use that byte code.

**8. Large Standard Library:** It provides a vast range of libraries for the various fields such as machine learning, web developer, and also for the scripting. There are various machine learning libraries, such as Tensor flow, Pandas, Numpy, Keras, and Pytorch, etc. Django, flask, pyramids are the popular framework for Python web development.

**9. GUI Programming:** Supports Graphical User Interface is used for the developing Desktop application. PyQT5, Tkinter, Kivy are the libraries which are used for developing the web application.

**10. Integrated:** It can be easily integrated with languages like C, C++, and JAVA, etc. Python runs code line by line like C,C++ Java. It makes easy to debug the code.

**11. Embeddable:** The code of the other programming language can use in the Python source code. We can use Python source code in another programming language as well. It can embed other language into our code.

**12. Dynamic Memory Allocation:** In Python, we don't need to specify the data-type of the variable. When we assign some value to the variable, it automatically allocates the memory to the variable at run time. Suppose we are assigned integer value 15 to x, then we don't need to write int x = 15. Just write x = 15.5.1.4

**5.1.4 Required Python Libraries:**

**1. Flask:**

• Acts as the main framework for building and running the web app.

• Manages session flow from login to task updates and rendering templates like login.html, taskmanager.html, etc.

**2. mysql-connector-python:**

• Provides a secure interface to connect to MySQL and run SQL queries.

• Allows CRUD (Create, Read, Update, Delete) operations for user accounts and tasks, making it integral for data persistence.

**3. requests:**

• Makes external API calls for sending task completion emails.

• Manages the HTTP request structure and headers for a successful email API interaction, which is important for notifying users in real-time.

**5.2 Web Development Technologies**

**5.2.1 HTML**

HTML (HyperText Markup Language) is the fundamental markup language used for creating web pages. It provides the structural foundation of a website by utilizing a series of elements and tags that define content such as headings, paragraphs, lists, links, and multimedia. HTML enables the embedding of images and videos, allowing developers to create rich, interactive user experiences. As the backbone of web content, HTML is essential for building the layout 20 and organization of information on the web, ensuring compatibility across different browsers and devices.

**5.2.2 CSS**

CSS (Cascading Style Sheets) is a stylesheet language that enhances the presentation and layout of web pages created with HTML. By separating content from design, CSS allows developers to control the visual appearance of elements, including colors, fonts, spacing, and positioning, resulting in aesthetically pleasing interfaces. CSS enables responsive design, ensuring that web pages adapt seamlessly to different screen sizes and devices, which improves user experience. Additionally, it supports animations and transitions, providing dynamic visual effects that engage users and enhance interactivity.

**5.2.3 Flask**

Flask is a lightweight and flexible web framework for Python that facilitates rapid web application development. Following the WSGI (Web Server Gateway Interface) and MVC (Model-View-Controller) architectural patterns, Flask provides the tools needed to create dynamic web applications and RESTful APIs efficiently. Its minimalist design allows developers to start quickly without the complexities of larger frameworks, making it ideal for small to medium-sized projects. Flask features a built-in development server for testing and debugging, along with a powerful templating engine, Jinja2, which enables the generation of dynamic content while maintaining a clear separation between application logic and presentation.

**5.3 Visual Studio Code**

Visual Studio Code is “a free-editor that helps the programmer write code, helps in debugging and corrects the code using the intelli-sense method ”. In normal terms, it facilitates users to write the code in an easy manner. Many people say that it is half of an IDE and an editor, but the decision is up to to the coders. Any program/software that we see or use works on the code that runs in the background. Traditionally coding was used to do in the traditional editors or even in the basic editors like notepad! These editors used to provide basic support to the coders.

**6. IMPLEMENTATION**

**6.1 Implementation Steps**

⮚**Step-1: Set Up the MySQL Database**

•**Create Database:** Use data.sql to create the taskmanager database and the Accounts table for storing user information.

•**Table Structure:** The Accounts table has fields for Username, Email (as the primary key), and passwords.

⮚**Step-2: Develop the Database Interactions (databases.py)**

•**Database Connection:** Create a create\_connection() function to connect to the MySQL database.

•**User Authentication:** Implement login\_data(email, password) to verify user credentials during login.

•**User Registration:** Implement upload\_data(username, email, password) to add new user data to Accounts and create a unique task table for each user based on their email.

•**Task Management:** Implement functions for adding tasks (add\_record), updating progress (updateprogress), marking tasks as terminated (terminateprogress), and fetching incomplete tasks (incompleted\_tasks).

•**History Tracking:** Implement historytable(email) to retrieve a complete history of tasks for each user.

⮚**Step-3: Set Up the Flask Web Application (main.py)**

Define Routes:

•**Login (/):** Verify user login and render the main task manager page if credentials are valid.

•**Signup (/signup):** Capture new user data, register the user, and handle errors if the user already exists.

•**Add Task (/addtask):** Add a new task for the user.

•**Task Status Update:** Routes like /timeup, /removetask, and /markcomplete update the task status, send email notifications if completed, and reflect changes on the main page.

•**History (/history):** Display the user's task history.

•**Global Variables:** Use main\_email to store the email of the currently logged-in user and access it across different routes.

⮚**Step-4: Implement Email Notifications (sentemail.py)**

•**Send Email:** Use the requests library to call an external API and send email notifications when tasks are marked as complete.

•**Username Extraction:** Retrieve the user’s name with extract\_username(email) to personalize the email content.

⮚**Step-5: Build Frontend Templates**

•**HTML Templates:** Create login.html, signup.html, taskmanager.html, success.html, and history.html to handle user interaction:

•**login.html:** Provides login fields and displays error messages for incorrect credentials.

•**signup.html:** Includes a form for new user registration.

•taskmanager.html: Main page to display tasks, add new ones, and update progress.

•**success.html:** Shows a success message on user registration.

•**history.html:** Displays task history with timestamps and status.

⮚**Step-6: Run and Test the Application**

•**Start the Flask App:** Run main.py with Flask in debug mode to test the application locally.

•**Functionality Testing:** Test all functionalities such as login, signup, task addition, progress updates, and history viewing.

•**Email Notification:** Check email notifications for task completion and ensure they reach the specified inbox.

**style.css**

body{

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background-image: url('https://i.pinimg.com/564x/d3/6b/34/d36b34136da16fb0923a2795dc0583a2.jpg');

background-size: cover;

background-position: center;

background-repeat: no-repeat;

color: #000;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

}

.container{

background-color: rgba(255, 255, 255, 0.9);

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

border-radius: 8px;

padding: 20px;

width: 100%;

}

button {

background-color: #000;

color: #fff;

border: none;

padding: 10px 20px;

border-radius: 4px;

cursor: pointer;

transition: background-color 0.3s ease;

}

**history.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Task History</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 20px;

background-color: #f4f4f4;

}

h1 {

text-align: center;

}

table {

width: 100%;

border-collapse: collapse;

margin: 20px 0;

background: white;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

th, td {

padding: 10px;

text-align: left;

border-bottom: 1px solid #ddd;

}

th {

background-color: #f2f2f2;

}

tr:hover {

background-color: #f1f1f1;

}

</style>

</head>

<body>

<h1>Task History</h1>

<table>

<thead>

<tr>

<th>S.No</th>

<th>Task</th>

<th>Task Entered Date</th>

<th>Task Entered Time</th>

<th>Task Completion Date</th>

<th>Task Completion Time</th>

<th>Progress</th>

</tr>

</thead>

<tbody id="taskHistoryList">

{% for his in history %}

<tr>

<td>{{loop.index}}</td>

<td>{{his[0]}}</td>

<td>{{his[1]}}</td>

<td>{{his[2]}}</td>

<td>{{his[3]}}</td>

<td>{{his[4]}}</td>

<td>{{his[5]}}</td>

</tr>

{% endfor %}

</tbody>

</table>

</body>

</html>

**login.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Login</title>

<link rel="stylesheet" href="{{url\_for('static',filename='styles.css')}}">

</head>

<body>

<form action="/" method="post">

<div><font color="white">{{error}}</font></div>

<div class="container">

<table border="0" align="center" cellpadding="15px" cellspacing="1px">

<tr><tdalign="center" colspan="3"><h1>LOGIN</h1></td></tr>

<tr>

<tdalign="center">GMAIL</td>

<td>:</td>

<tdalign="left"><input type="email" name="email" required></td>

</tr>

<tr>

<td>PASSWORD</td>

<td>:</td>

<tdalign="left"><input type="password" name="password" required></td>

</tr>

<tr>

<td colspan="3" align="center"><button type="submit">LOGIN</button></td>

</tr>

<tr>

<td colspan="3"><p>You don't have an account? <a href="/signup" style="text-decoration:none;">Sign up</a></p></td>

</tr>

</table>

</div>

</form>

</body>

</html>

**signup.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>SIGN UP</title>

<link rel="stylesheet" href="{{url\_for('static',filename='styles.css')}}">

</head>

<body>

<form action="/signup" method="post">

<div><font color="white">{{error}}</font></div>

<div class="container" id="createaccount">

<table border="0" align="center" cellpadding="15px" cellspacing="1px">

<tr><tdalign="center" colspan="3"><h1>SIGN UP</h1></td></tr>

<tr>

<tdalign="center">ENTER FULL NAME</td>

<td>:</td>

<tdalign="left"><input type="text" name="username" required></td>

</tr>

<tr>

<tdalign="center">ENTER GMAIL</td>

<td>:</td>

<tdalign="left"><input type="email" name="email" required></td>

</tr>

<tr>

<td>ENTER PASSWORD</td>

<td>:</td>

<tdalign="left"><input type="password" name="password" required></td>

</tr>

<tr>

<td colspan="3" align="center"><table border="0" width="75%" align="right">

<tr>

<td><button type="submit">Create Account</button></td>

<td><a href="{{url\_for('login')}}"><button type="button">Cancel</button></a></td>

</tr>

</table>

</td>

</tr>

</table>

</div>

</form>

</body>

</html>

**success.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Account created successfully</title>

<link rel="stylesheet" href="{{url\_for('static',filename='styles.css')}}">

</head>

<body>

<form action="/signup" method="post">

<div class="container" id="createdaccount">

<h1>Account Created Successfully</h1>

<center><ahref="{{url\_for('login')}}"><button type="button">Login</button></a></center>

</div>

</form>

</body>

</html>

**taskmanager.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Personal Task Manager</title>

<link rel="stylesheet" href="{{url\_for('static',filename='styles.css')}}">

<style>

.container {

background-color: rgba(255, 255, 255, 0.9);

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

border-radius: 8px;

padding: 20px;

width: 100%;

max-width: 600px;

}

h1 {

text-align: center;

color: #000;

}

.task-list {

margin: 20px 0;

padding: 0;

list-style: none;

}

.task-list li {

display: flex;

justify-content: space-between;

align-items: center;

margin-bottom: 10px;

padding: 10px;

background-color: #f9f9f9;

border: 1px solid #ddd;

border-radius: 4px;

transition: background-color 0.3s ease;

}

.task-list li:hover{

background-color: #f1f1f1;

border-color:black;

}

.buttons {

margin-top: 20px;

display: flex;

justify-content: center;

gap: 10px;

}

.buttons button {

background-color: #000;

color: #fff;

border: none;

padding: 10px 20px;

border-radius: 4px;

cursor: pointer;

transition: background-color 0.3s ease;

}

.buttonsbutton:hover {

background-color: #333;

}

.buttonsbutton:focus {

outline: none;

box-shadow: 0 0 0 3px rgba(0, 0, 0, 0.25);

}

.buttonsbutton:active {

background-color: #111;

}

.selected {

background-color: #e2e6ea;

}

.notification {

color: red;

text-align: center;

margin-top: 20px;

}

</style>

</head>

<body>

<div class="container">

<h1>Personal Task Manager</h1>

<ul id="taskList" class="task-list">

{% for record in records %}

<li>

<span>{{record[0]}}</span>

<span>{{record[1]}}</span>

<span>{{record[2]}}</span>

<span></span>

</li>

{% endfor %}

</ul>

<div class="buttons">

<button onclick="addTask()" id="addtask">Add Task</button>

<button id="removetask">Remove Task</button>

<button id="markcomplete">Mark complete</button>

<a href="{{url\_for('history')}}"><button id="showhistory">History</button></a>

</div>

<div class="notification" id="notification"></div>

<p style="text-align: center; margin-top: 20px;">

<a href="{{url\_for('login')}}" style="color: red;">Logout</a>

</p>

</div>

<script>

function startCountdown(taskName,dateSpan,timeSpan,endTime, spanElement) {

const countdownInterval = setInterval(() => {

const now = new Date();

const remainingTime = endTime - now;

if (remainingTime<= 0) {

clearInterval(countdownInterval);

spanElement.textContent = "Task completed!";

// Send value to Flask

fetch('/timeup', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ task: ${taskName},date: ${dateSpan},time:${timeSpan} }),

})

.then(response =>response.json())

.then(data => {

console.log('Response from server:', data);

});

alert(Time's up for: ${taskName});

location.reload();

} else {

const seconds = Math.floor((remainingTime / 1000) % 60);

const minutes = Math.floor((remainingTime / (1000 \* 60)) % 60);

const hours = Math.floor((remainingTime / (1000 \* 60 \* 60)) % 24);

const days = Math.floor(remainingTime / (1000 \* 60 \* 60 \* 24));

spanElement.textContent = ${days}d ${hours}h ${minutes}m ${seconds}s;

}

}, 1000);

}

document.addEventListener("DOMContentLoaded", () => {

const items = document.querySelectorAll("li");

items.forEach(item => {

const taskName = item.querySelector("span:nth-child(1)").textContent;

const dateSpan = item.querySelector("span:nth-child(2)").textContent;

const timeSpan = item.querySelector("span:nth-child(3)").textContent;

const endTime = new Date(${dateSpan} ${timeSpan}).getTime();

const timerSpan = item.querySelector("span:nth-child(4)");

startCountdown(taskName,dateSpan,timeSpan,new Date(endTime), timerSpan);

});

});

function addTask() {

const task = prompt("Enter task:");

const date = prompt("Enter date(YYYY-MM-DD):");

const time = prompt("(24Hours Format)Enter time(HH:MM):");

const newItem = document.createElement('li');

const span1 = document.createElement('span');

span1.textContent = task;

const span2 = document.createElement('span');

span2.textContent = date;

const span3 = document.createElement('span');

span3.textContent = time;

const span4 = document.createElement('span');

newItem.appendChild(span1);newItem.appendChild(span2);

newItem.appendChild(span3);newItem.appendChild(span4);

document.getElementById('taskList').appendChild(newItem);

const endTime = new Date(${date} ${time}).getTime();

startCountdown(new Date(endTime), span4);

location.reload();

if (task && date && time) {

fetch('/addtask', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ task, date, time }),

})

.then(response =>response.json())

.then(data => {

location.reload(); // Refresh the page to see the new task

});

}

}

let selectedItem = null;

// Function to select an item

document.querySelectorAll('li').forEach(item => {

item.addEventListener('click', function() {

// Remove the 'selected' class from any previously selected item

if (selectedItem) {

selectedItem.classList.remove('selected');

}

// Mark the currently clicked item as selected

selectedItem = this;

selectedItem.classList.add('selected');

});

});

// Function to remove the selected item

document.addEventListener("DOMContentLoaded", () => {

document.getElementById('removetask').addEventListener('click', function() {

if (selectedItem) {

// Get the spans from the selected item

const spans = selectedItem.getElementsByTagName("span");

const taskName = spans[0].textContent;

const dateSpan = spans[1].textContent;

const timeSpan = spans[2].textContent;

selectedItem.remove(); // Remove the selected <li>

removeTask(taskName,dateSpan,timeSpan);

location.reload();

selectedItem = null; // Reset the selected item

} else {

alert('Please select an item to remove.');

}

});

document.getElementById('markcomplete').addEventListener('click', function() {

if (selectedItem) {

// Get the spans from the selected item

const spans = selectedItem.getElementsByTagName("span");

const taskName = spans[0].textContent;

const dateSpan = spans[1].textContent;

const timeSpan = spans[2].textContent;

selectedItem.remove(); // Remove the selected <li>

markListed(taskName,dateSpan,timeSpan);

location.reload();

selectedItem = null; // Reset the selected item

} else {

alert('Please select an item to mark completed.');

}

});

});

function removeTask(task,date,time) {

if (task) {

fetch('/removetask', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ task,date,time }),

})

.then(response =>response.json())

.then(data => {

location.reload(); // Refresh the page to see the updated tasks

});

}

}

function markListed(task,date,time) {

// This function can be expanded based on your needs

if(task){

fetch('/markcomplete', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ task,date,time }),

})

.then(response =>response.json())

.then(data => {

location.reload(); // Refresh the page to see the updated tasks

});

}

}

</script>

</body>

</html>

**data.sql**

create database taskmanager;

use taskmanager;

create table Accounts(

Username varchar(50) not null,

Email varchar(80) primary key,

passwords varchar(80) not null

);

select \* from Accounts;

**data.sql**

import mysql.connector

# Establish a connection to the database

def create\_connection():

return mysql.connector.connect(

host='localhost',

user='root',

password='root',

database='taskmanager'

)

**databases.py**

def login\_data(email, password):

result = False

connection = create\_connection()

cursor = connection.cursor()

query = f"""SELECT passwords FROM Accounts WHERE Email='{email}';"""

cursor.execute(query)

results = cursor.fetchall()

if results:

for passwd in results:

if passwd[0] == password:

result = True

else:

result = False

else:

result ="no user"

cursor.close()

connection.close()

return result

def upload\_data(username,email, password):

connection = create\_connection()

cursor = connection.cursor()

query1 = """

insert into Accounts(Username,Email,passwords) values(%s,%s,%s);

"""

query2 = f"""create table {email}(

TASK VARCHAR(500),

systemdate DATE DEFAULT(CURRENT\_DATE),

systemtime TIME DEFAULT(CURRENT\_TIME),

userdate DATE,

usertime TIME,

PROGRESS VARCHAR(20) DEFAULT 'Not Complete'

);"""

cursor.execute(query1,(username,email, password))

cursor.execute(query2)

cursor.close()

connection.commit()

connection.close()

def incompleted\_tasks(email):

connection = create\_connection()

cursor = connection.cursor()

query1 = f""" select TASK,userdate,usertime from {email} where PROGRESS='not complete'; """

cursor.execute(query1)

records = cursor.fetchall()

formatted\_records = [

(task, str(userdate), str(usertime))

for task, userdate, usertime in records

]

cursor.close()

connection.commit()

connection.close()

return formatted\_records

def updateprogress(task,date,time,email):

connection = create\_connection()

cursor = connection.cursor()

query1 = f""" update {email} set PROGRESS='completed' where TASK='{task}' and userdate='{date}' and usertime='{time}'; """

cursor.execute(query1)

cursor.close()

connection.commit()

connection.close()

def add\_record(task,date,time,email):

connection = create\_connection()

cursor = connection.cursor()

query1 = f""" insert into {email}(TASK,userdate,usertime) values('{task}','{date}','{time}'); """

cursor.execute(query1)

cursor.close()

connection.commit()

connection.close()

def terminateprogress(task,date,time,email):

connection = create\_connection()

cursor = connection.cursor()

query1 = f""" update {email} set PROGRESS='Terminated' where TASK='{task}' and userdate='{date}' and usertime='{time}'; """

cursor.execute(query1)

cursor.close()

connection.commit()

connection.close()

def historytable(email):

connection = create\_connection()

cursor = connection.cursor()

query1 = f""" select \* from {email}; """

cursor.execute(query1)

records = cursor.fetchall()

formatted\_records = [

(task,str(sysdate), str(systime), str(userdate), str(usertime),progress)

for task,sysdate,systime,userdate,usertime,progress in records

]

cursor.close()

connection.commit()

connection.close()

return formatted\_records

def extract\_username(email):

connection = create\_connection()

cursor = connection.cursor()

query1 = f""" select Username from Accounts where Email='{email}'; """

cursor.execute(query1)

record = cursor.fetchall()

cursor.close()

connection.commit()

connection.close()

return record

**main.py**

from flask import \*

from databases import login\_data,upload\_data,add\_record,incompleted\_tasks,updateprogress,terminateprogress,historytable

from sentemail import sentmail

app = Flask(\_name\_)

main\_email = "abc"

@app.route('/',methods=['GET','POST'])

def login():

global main\_email

if request.method == 'POST':

email = request.form.get('email')

password = request.form['password']

value = login\_data(email, password)

if value == True:

main\_email = email.replace("abc",email)

records = incompleted\_tasks(main\_email)

return render\_template("taskmanager.html",records=records)

else:

return render\_template('login.html',error="Invalid Details!")

return render\_template('login.html')

@app.route('/signup',methods=['GET','POST'])

def signup():

if request.method == 'POST':

try:

gmail = request.form.get('email')

passwd = request.form['password']

username = request.form.get('username')

upload\_data(username,gmail, passwd)

return render\_template('success.html')

except:

return render\_template('signup.html',error="User Already Exist!")

return render\_template('signup.html')

@app.route('/addtask',methods=['GET','POST'])

def addtask():

data = request.get\_json()

task = data.get('task')

date = data.get('date')

time = data.get('time')

add\_record(task,date,time,main\_email)

return render\_template("taskmanager.html")

@app.route('/timeup',methods=['GET','POST'])

def timeup():

data = request.get\_json()

task = data.get('task')

date = data.get('date')

time = data.get('time')

updateprogress(task,date,time,main\_email)

sentmail(task,date,time,main\_email)

return render\_template("taskmanager.html")

@app.route('/removetask',methods=['GET','POST'])

def removetask():

data = request.get\_json()

task = data.get('task')

date = data.get('date')

time = data.get('time')

terminateprogress(task,date,time,main\_email)

return render\_template("taskmanager.html")

@app.route('/markcomplete',methods=['GET','POST'])

def markcompleted():

data = request.get\_json()

task = data.get('task')

date = data.get('date')

time = data.get('time')

updateprogress(task,date,time,main\_email)

return render\_template("taskmanager.html")

@app.route('/history',methods=['GET','POST'])

def history():

history = historytable(main\_email)

return render\_template("history.html",history=history)

if \_name=='main\_':

app.run(debug=True)

**sentemail.py**

import requests

from databases import extract\_username

def sentmail(task,date,time,email):

name = extract\_username(email)

url = "https://mail-sender-api1.p.rapidapi.com/"

content = f"Dear {name}, Your Task '{task}' is completed in {date} at {time}"

payload = {

"sendto": email,

"name": "PersonalTaskManger Team",

"replyTo": "hr2946352@gmail.com",

"ishtml": "false",

"title": "Task Completion Notification",

"body": content

}

headers = {

"x-rapidapi-key": "a576603aa2msh6e3b4855bc70b31p1c5001jsnb2573c21c4ed",

"x-rapidapi-host": "mail-sender-api1.p.rapidapi.com",

"Content-Type": "application/json"

}

response = requests.post(url, json=payload, headers=headers)

print(response.json())

return response.json()

**7. TESTING**

**7.1 Introduction to Testing**

Testing is a critical phase in the development of the Personal Task Manager application. It ensures the reliability, functionality, and performance of the app, providing users with a smooth and seamless experience. This section outlines the objectives and strategies employed in testing the Personal Task Manager.

**7.2 Purpose of Testing**

The primary purpose of testing is to identify and rectify any issues or bugs within the application before it is deployed. This helps in delivering a high-quality product that meets user requirements and business goals. Testing helps ensure that all features function as intended, the application performs well under various conditions, and users can interact with the app effortlessly and securely.

**7.3 Testing Objectives**

⮚**Functional Testing:** Verify functionalities like user authentication, task management, and email notifications.

⮚**Performance Testing:** Test application performance under various loads and response times for tasks.

⮚**Security Testing:** Ensure secure password storage and user data protection.

⮚**Integration Testing:** Validate third-party integrations and database interactions.

⮚**Regression Testing:** Ensure new updates don’t break existing functionality.

⮚**Accessibility Testing:** Ensure the application is accessible to users with disabilities.

By implementing these testing strategies, we ensure that the Personal Task Manager is robust, user-friendly, and ready for deployment. This comprehensive approach to testing helps in delivering a reliable and high-performing application that meets the needs of its users.

**7.4 Test Case Design**

**Test Case 1: User Login**

•Objective: Validate the login functionality.

•Preconditions: User is not logged into the application.

1. Test Steps:

1. Navigate to the login page.
2. Input valid email and password.
3. Click on the login button.

•Expected Result: Successful login and redirection to the task manager page.

2. Test Steps:

i. Input invalid email or password.

ii. Click on the login button.

iii. Expected Result: Display an "Invalid Details!" error message.

**Test Case 2: User Signup**

•Objective: Validate the signup functionality.

•Preconditions: User is not registered

1. Test Steps:

1. Navigate to the signup page.
2. Input unique email, password, and username.
3. Click on the signup button.

•Expected Result: Successful signup and redirection to the success page.

2. Test Steps:

1. Input an existing email.
2. Click on the signup button.

•Expected Result: Display "User Already Exist!" error message.

**Test Case 3: Adding a Task**

•Objective: Validate the functionality of adding a new task.

•Preconditions: User is logged into the application.

1. Test Steps:

i. Navigate to the add task page.

ii. Input task details including name, date, and time.

1. Click on the add button.

•Expected Result: Task is added successfully&visible on task manager page.

**Test Case 4: Updating a Task**

•Objective: Validate the task update functionality.

•Preconditions: User has an existing task.

1. Test Steps:

i. Send a POST request to update the task's status.

ii. Check the task's updated status.

•Expected Result: Task is updated successfully&an email notification is sent.

**Test Case 5: Removing a Task**

•Objective: Validate the task removal functionality.

•Preconditions: User has an existing task.

1. Test Steps:

i. Send a POST request to remove the task.

ii. Verify the task's absence in the task list.

•Expected Result:Task’s removed successfully, no longer visible on task manager page.

**Test Case 6: Viewing Task History**

•Objective: Validate the history functionality.

•Preconditions: User has completed tasks.

1. Test Steps:

i. Navigate to the history page.

•Expected Result: History of completed tasks is displayed correctly.

**Test Case 7: Email Notification**

•Objective: Validate that email notifications are sent correctly.

•Preconditions: User is logged in and performs an action that triggers an email.

1. Test Steps:

i. Complete or update a task.

•Expected Result: An email is sent with the correct task details.

**Expected Result:** Application handles load without crashing and maintains acceptable response times.

**7.5 Testing Strategies**

**7.5.1 Unit Testing**

This involves testing individual components or functions of the software in isolation to ensure they work correctly. It's like checking each brick in a wall to make sure it's solid before you build the whole wall.

Importance: Ensures each part functions correctly. Identifies issues early and simplifies debugging.

Application: Test functions like login\_data, add\_record, and upload\_data to ensure they perform as expected independently.

**7.5.2 Integration Testing**

This focuses on testing the interactions between different modules or components to ensure they work together as expected. Think of it as making sure the plumbing and electrical systems in a house work together without issues.

Importance: Validates interaction between integrated units. Detects interface issues and ensures seamless communication.

Application: Test the interaction between the login function and the task retrieval function to ensure they work together seamlessly.

**7.5.3 Functional Testing**

This ensures that each function of the software operates in conformance with the requirement specification. It's like verifying that each feature in a car, like the brakes or the lights, works as intended.

Importance: Ensures each function of the application works as expected.

Application: Validate user authentication, task creation, updates, and deletions to ensure they function correctly.

**7.5.4 System Testing**

This involves testing the complete and integrated software system to ensure it meets the specified requirements. It's like inspecting a finished house to ensure everything works together perfectly.

Importance: Tests the entire system to ensure it meets specified requirements. Verifies end-to-end scenarios and overall behavior.

Application: Test the complete workflow from user signup, to task management, to viewing task history, ensuring the entire system functions as intended.

**7.5.5 Acceptance Testing**

This type of testing is done to determine if the software is ready for deployment by verifying it meets business needs and user requirements. Imagine a homeowner checking if their newly built house meets all their expectations before moving in.

Importance: Confirms the system fulfills business needs and user requirements. Determines readiness for deployment.

Application: Have stakeholders verify that all features like signup, task management, and email notifications work as expected and meet business objectives.

**7.6 Test Cases**

A test case is a specific set of conditions, inputs, actions, and expected results used to verify a particular function or feature of an application. It’s like a detailed plan for testing one particular aspect of your application.

**Test Case 1: Adding a Task**

•Objective: Verify that a task is successfully added with valid data.

•Preconditions: User is logged into the application.

•Test Steps:

i. Navigate to the task manager page.

ii. Click on the "Add Task" button.

iii. Fill in the task details (e.g., "Finish documentation", date: "2024-11-03", time: "14:00").

1. Click on the "Save" button.
2. Validate the response.

•Expected Result: Task is added successfully and appears in the task list.

**Test Case 2: Updating a Task**

•Objective: Verify that a task's details can be updated.

•Preconditions: User is logged in and has an existing task.

•Test Steps:

i. Navigate to the task manager page.

ii. Select an existing task.

iii. Click on the "Edit Task" button.

1. Update the task details (e.g., change time to "15:00").

V. Click on the "Save" button.

vi. Validate the response.

•Expected Result: Task details are updated successfully and the updated task appears in the task list

**Test Case 3: Removing a Task**

•Objective: Verify that a task can be removed.

•Preconditions: User is logged in and has an existing task.

•Test Steps:

i. Navigate to the task manager page.

ii. Select an existing task.

iii. Click on the "Remove Task" button.

iv. Confirm the removal.

v. Validate the response.

•Expected Result: Task is removed successfully and no longer appears in the task list.

**Test Case 4: Marking a Task as Complete**

•Objective: Verify that a task can be marked as complete.

•Preconditions: User is logged in and has an existing task.

•Test Steps:

i. Navigate to the task manager page.

ii. Select an existing task.

iii. Click on the "Mark as Complete" button.

iv. Validate the response.

•Expected Result: Task is marked as complete and appears in the completed tasks list.

**9. CONCLUSION**

**9.1 Conclusion**

The Task Manager project successfully achieves its goal of helping users manage and track their tasks by sending timely email notifications. This system not only ensures that tasks are completed on time but also enhances productivity by automating reminders. By using scheduled notifications, users can focus on their priorities without the need for constant manual tracking.

Through the integration of email notifications, this application serves as an efficient and reliable reminder system for individuals and teams. The implementation demonstrates the potential of task management tools in organizing daily workflows and reducing missed deadlines. Future enhancements could include support for SMS notifications, custom scheduling, and task priority sorting, making this Task Manager a robust solution for personal and professional task management needs.

**9.2 Future Scope Of Project**

* **Offline Access:**Enable offline access to the Task Manager, ensuring that users can view and update their tasks even without an internet connection. Changes would synchronize once the device reconnects to the internet.
* **Enhanced Security Features**:Implement advanced security measures such as two-factor authentication (2FA), end-to-end encryption for data, and secure data backups to ensure that user information is always protected.

**10. REFERENCES**

**[1]** Victoria Bellotti, Brinda Dalal, Nathaniel Good, Peter Flynn, Daniel G. Bobrow, Nicolas Ducheneaut,What a To-Do: Studies of Task Management Towards the Design of a Personal Task List Manager,2004.

**[2]**Zohar Rusou, Moty Amar, and Shahar Ayal,ThePsychology of Task Management: The Smaller Tasks Trap.2020.

**[3]**Karen Myers, Pauline Berry, Jim Blythe, Ken Conley, Melinda Gervasio, Deborah McGuinness, David Morley, Avi Pfeffer, Martha Pollack, and Milind Tambe,Karen Myers, Pauline Berry, Jim Blythe, Ken Conley, Melinda Gervasio, Deborah McGuinness, David Morley, Avi Pfeffer, Martha Pollack, and Milind Tambe,2007.