LOVELY PROFESSIONAL UNIVERSITY

Tempus Flow

Software Requirements Specification

INT219
FRONT END WEB DEVELOPER

Prepared By: Aarav Singh Rajput(12301162), Rohith E S(12303701), Prashant mete(12320674), Parth Patidar(12316831)

Prepared for
Continuous Assessment 2
Spring 2025

1. Introduction

1.1 Purpose

The primary objective of this Software Requirements Specification (SRS) document is to meticulously and explicitly outline the core functionality, system interfaces, and inherent design constraints of the software project titled **TempusFlow**. TempusFlow is a **multi-user productivity management system**, envisioned to provide users with comprehensive tools and features to manage, track, and enhance their daily productivity through a unified digital platform.

This SRS serves as a foundational blueprint for all stakeholders involved in the development lifecycle of the project. Its detailed descriptions aim to establish a shared understanding among the following audiences:

- **Developers**, who will utilize this document to guide the implementation of both frontend and backend system components, ensuring alignment with expected system behavior and performance requirements.
- **Testers and Quality Assurance (QA) personnel**, who will refer to this specification to construct test cases, verify functional correctness, and validate the software against the documented requirements.
- **Project stakeholders**, including clients, end-users, instructors, or project supervisors, who require a transparent overview of what the system is expected to do, how it is intended to operate, and what limitations may exist during development and deployment.
- **Future maintainers or contributors**, who may engage with the system post-deployment for the purposes of enhancement, optimization, bug-fixing, or feature expansion. For them, this document will act as a reference to comprehend the original intentions, constraints, and scope of the system.

By compiling all key elements of TempusFlow’s intended capabilities, this SRS ensures that development proceeds in a structured, traceable, and manageable way, minimizing the chances of misinterpretation or scope deviation throughout the lifecycle of the project.

1.2 Scope

TempusFlow is a **web-based** application designed as a comprehensive time and productivity management tool. Its primary purpose is to empower users—whether students, professionals, or anyone seeking better personal organization—with intuitive features that allow for efficient planning, scheduling, and note-keeping. The system will function entirely within a web environment, ensuring platform independence and ease of access across various devices.

The core functionalities of TempusFlow will include the following:

- **User Authentication and Secure Data Storage**:
  The application will implement mechanisms for **user registration, login, and logout**, ensuring that each user has a **secure, private workspace**. All personal data—including tasks, notes, and events—will be securely stored using encryption and protected from unauthorized access through proper session and database management techniques.
- **Personalized Dashboards for Each User**:
  Upon successful authentication, each user will be presented with a **customized dashboard** tailored to display their individual productivity data. This dashboard will serve as the central hub from which users can monitor their schedule, upcoming tasks, recent notes, and other relevant personal information.
- **CRUD Operations on Tasks, Notes, and Events**:
  The system will fully support **Create, Read, Update, and Delete (CRUD)** functionalities for three primary entities:
  - **Tasks**: To-do items or assignments with optional deadlines and priorities.
  - **Events**: Calendar-based scheduled activities with time and date.
  - **Notes**: Freely written content that users may want to store for future reference.

  These operations ensure that users can maintain an evolving and accurate log of their responsibilities and ideas.
- **Cross-Device Accessibility with Responsive Design**:
  TempusFlow will feature a **responsive user interface**, meaning it will automatically adapt to various screen sizes and resolutions, such as those of mobile phones, tablets, laptops, and desktop monitors. This ensures that users can conveniently interact with their productivity tools regardless of the device they are using.

However, it is important to clearly define the **limitations** of the system as well. **TempusFlow will not include**:

- **AI-Based Suggestions or Automatic Time Tracking**:
  The system is designed to be straightforward and manually operated. It will **not use artificial intelligence** to analyze user habits or make automated recommendations. Furthermore, it will not track time spent on activities or provide passive analytics on user behavior.

The **overarching goals** of TempusFlow are summarized as follows:

- **Facilitate Streamlined Task Planning**:
  By offering tools for organizing tasks and events, the system aims to reduce planning complexity and allow users to clearly structure their daily responsibilities.
- **Offer User-Definable Scheduling and Note Organization**:
  Users will have the flexibility to define their own time slots, categorize notes, and arrange their productivity inputs according to personal preferences.
- **Enhance Productivity Through Easy Access to Daily Schedules**:
  The application’s emphasis on visual clarity and ease of navigation will ensure that users can

quickly refer to their agenda, reducing time spent on administrative overhead and helping them focus on what matters most.

1.3 Definitions, Acronyms, and Abbreviations

This section provides clear definitions and explanations of the abbreviations, acronyms, and technical terms used throughout this Software Requirements Specification (SRS) document. These terms are commonly referenced in the context of software development and are integral to understanding the functionality and scope of the **TempusFlow** system.

- **UI (User Interface):**
  The **User Interface (UI)** refers to the visual elements of the application through which users interact with the system. This includes menus, buttons, forms, icons, layout design, and other graphical components that facilitate communication between the user and the underlying software logic. A well-designed UI enhances usability and accessibility, ensuring that users can intuitively navigate and perform desired actions within TempusFlow.
- **UX (User Experience):**
  User Experience (UX) encompasses the overall interaction, satisfaction, and emotional response a user has when using the application. It considers factors such as system responsiveness, ease of use, flow of interaction, and aesthetic appeal. In TempusFlow, providing a positive UX is a critical objective, achieved through intuitive design, minimal friction in task completion, and responsive behavior across devices.
- **DB (Database):**
  The **Database (DB)** is the structured storage system where all persistent data related to TempusFlow is stored and managed. This includes user credentials, task lists, event schedules, and written notes. The database will be designed to support secure data retrieval and manipulation, enforce data integrity, and ensure confidentiality and availability of user information.
- **CRUD (Create, Read, Update, Delete):**
  CRUD represents the four fundamental operations that can be performed on data entities within the application. In the context of TempusFlow, CRUD operations apply to:
  - **Tasks**: Users can create new tasks, view existing ones, modify task details, or remove them once completed or irrelevant.
  - **Events**: Users can schedule events, review upcoming ones, change details, or delete outdated entries.
  - **Notes**: Users can write, read, edit, or delete personal notes.

These operations are essential for ensuring that users can fully control and manage their productivity data in a dynamic and personalized manner.

1.4 References

This section lists all the external sources, tools, documentation, and standard guidelines that have been consulted or are relevant to the development and documentation of **TempusFlow**. These references serve as authoritative resources for design decisions, technical implementation, and adherence to best practices in software engineering.

- **GitHub Repository: https://github.com/theSolution12/TempusFlow**
  This is the official version-controlled codebase for the TempusFlow project. It contains all source code files, project documentation, commit histories, issues, and collaborative contributions. It serves as the central hub for development tracking, feature integration, and team collaboration.
- **PHP Manual: https://www.php.net/manual/en/index.php**
  The official PHP documentation is referenced throughout the development of the server-side backend logic. It provides comprehensive information on PHP syntax, built-in functions, language constructs, best practices, and security measures. Since TempusFlow uses PHP to handle authentication, database interactions, and business logic, this manual is critical for accurate and efficient development.
- **MySQL Documentation: https://dev.mysql.com/doc/**
  This is the official documentation for MySQL, the relational database management system used in the TempusFlow project. The documentation covers database setup, schema design, SQL syntax, query optimization, data security, and performance tuning. It is instrumental in ensuring the robustness, scalability, and reliability of the data layer within the system.
- **IEEE Software Requirements Specification Guide**
  This standard, provided by the Institute of Electrical and Electronics Engineers (IEEE), outlines the best practices for writing Software Requirements Specification (SRS) documents. The structure and formatting of this document follow the guidelines laid out in the IEEE 830-1998 SRS standard. It ensures that the requirements are complete, clear, and verifiable, and it facilitates communication among all project stakeholders.

1.5 Overview

This Software Requirements Specification (SRS) document provides a **comprehensive overview of the TempusFlow system**, including its expected functionality, major system features, and the various **design constraints** that guide and limit its development.

The document has been meticulously structured to follow a logical progression, allowing all stakeholders—including developers, testers, project managers, and future maintainers—to gain a clear understanding of what the system will accomplish and how it is expected to behave under defined conditions.

The **structure of this SRS** is organized into the following key sections:

- **General Descriptions**:
  This section outlines the foundational aspects of the system, including its purpose, overall

scope, and the environment in which it will operate. It also defines essential terms, acronyms, and references that support clarity and consistency throughout the document.

- **Specific Requirements**:
  This core section captures the detailed, measurable, and testable **functional** and **non-functional** requirements of the TempusFlow system. It defines what the system must do (functional behavior) and how well it must perform (performance, usability, reliability, etc.). Each requirement is documented to ensure traceability and to guide implementation and testing activities.
- **Analysis Models**:
  This section includes various visual and textual models used to represent the system's internal workflows, data flow, user interactions, and architectural layout. Diagrams such as use case diagrams, data flow diagrams (DFDs), and sequence diagrams may be included to help visualize the structure and behavior of the system components.
- **Appendices**:
  The appendices provide supplementary information that supports the core content of the document. This may include user interface mockups, database schemas, sample inputs and outputs, glossary of terms, or any additional resources that enhance understanding without disrupting the main flow of the document.

Together, these sections ensure that the SRS serves as a **single source of truth** for TempusFlow's development, facilitating effective communication, alignment, and clarity across all phases of the software development lifecycle.

---

2. General Description

2.1 Product Perspective

TempusFlow is envisioned as a **standalone web application** specifically designed to help users manage their time and productivity more effectively. It operates independently and is self-contained, meaning it does not inherently rely on or require any third-party system to function. Users can access and use the application directly through a web browser without needing to install any additional software or plugins.

Despite being a standalone system, **TempusFlow has been architected with flexibility and interoperability in mind**. This allows it to be **seamlessly integrated into larger digital ecosystems**, especially in **academic or enterprise environments**. In such contexts, it can function as a **productivity enhancement plugin or module**, extending the capabilities of existing systems like:

- **Learning Management Systems (LMS)** in educational institutions, where it could help students manage assignments, lectures, and study plans.
- **Enterprise Resource Planning (ERP)** platforms or internal employee portals, providing professionals with tools to organize meetings, project tasks, and notes.

Such integration would be achieved through standardized web technologies and APIs, ensuring that TempusFlow can complement and enhance existing platforms without causing conflicts or requiring major architectural overhauls.

In summary, TempusFlow is both **self-sufficient for individual use** and **adaptable for institutional integration**, making it a versatile tool capable of serving a wide range of productivity-related needs.

2.2 Product Functions

The core functionality of **TempusFlow** revolves around helping users effectively organize and manage their time, tasks, events, and notes within a centralized, user-friendly web interface. The system is designed to support the following key features:

- **User Registration and Authentication**:
  TempusFlow provides a secure and streamlined process for **new users to register** by creating personal accounts with unique credentials. Once registered, users can **log in through an authentication mechanism** that validates their identity and provides access to their personalized dashboard. This process includes security practices such as password hashing and session management to ensure privacy and data protection.
- **Task Management (Create, Update, Delete Tasks)**:
  Users can perform **CRUD operations** (Create, Read, Update, Delete) on personal tasks. Tasks may include optional fields such as titles, descriptions, due dates, and priority levels. This feature enables users to:
  - Add new tasks as needed
  - Modify existing task information (e.g., deadlines or status)
  - Remove completed or obsolete tasks from their list
    The task manager helps users keep track of their daily to-do items in an organized and structured format.
- **Event Scheduling and Alerts**:
  TempusFlow includes a calendar-based event scheduling system where users can plan and manage upcoming events such as meetings, deadlines, or personal appointments. Events include time, date, title, and optional descriptions.
  Additionally, the system can generate **alert notifications** or **reminders** to ensure users are aware of their upcoming events, helping them avoid missed appointments and stay on track.
- **Notes Storage and Categorization**:
  Users are able to write, edit, and store **text-based notes** within the application. To help with organization, notes can be **categorized by topic or tag**, allowing users to filter or search through their content quickly. This feature is especially useful for saving ideas, project details, class notes, or reminders in a digital notebook format.
- **View Daily Agenda**:
  TempusFlow provides users with a **consolidated daily agenda view**, presenting an overview of their tasks, scheduled events, and relevant notes for the current day. This feature

serves as a productivity snapshot, helping users visualize and prioritize their activities, make informed time management decisions, and stay focused throughout the day.

2.3 User Characteristics

The **TempusFlow** system is designed to accommodate a broad and diverse user base, each with unique goals and productivity requirements. Below are the primary categories of users who are expected to interact with the system, along with their specific needs and characteristics:

- **Students**
  This user group includes high school, college, and university students who face a wide array of **academic responsibilities**, such as attending classes, completing assignments, preparing for exams, and participating in extracurricular activities. These users often juggle multiple deadlines and require tools that help them:
    - **Organize coursework**
    - **Track assignment deadlines**
    - **Schedule study sessions and group meetings**
    - **Take and categorize lecture notes**
      TempusFlow caters to these needs by providing a structured, visual, and responsive interface for managing daily academic workloads and boosting time management skills.
- **Professionals**
  Professionals comprise individuals working in various industries who need **structured scheduling systems** to maintain productivity and work-life balance. This group includes managers, developers, designers, consultants, and remote workers. Their requirements often include:
    - **Creating and tracking project tasks**
    - **Scheduling and receiving alerts for meetings**
    - **Maintaining a clear view of daily priorities**
    - **Writing down ideas or notes during work sessions**
      For this demographic, TempusFlow serves as a centralized workspace that helps reduce cognitive overload and ensures efficient planning and execution of professional duties.
- **General Users**
  This category includes **non-student and non-professional individuals** who are looking to improve their **personal productivity**. This may include homemakers, retirees, freelancers, or hobbyists seeking better management of daily chores, routines, goals, or personal projects. Their expectations often include:
    - **Tracking simple to-do lists**
    - **Managing personal events and reminders**

- **Maintaining habit logs or journaling**
- **Storing and retrieving categorized personal notes**
  TempusFlow provides an intuitive and lightweight solution for these users to stay organized and motivated in their daily lives.

2.4 General Constraints

The development and deployment of **TempusFlow** are subject to a number of **technical and environmental constraints** that must be adhered to in order to ensure compatibility, maintainability, and efficient operation of the system. These constraints define the boundaries within which the system must be designed and implemented:

- **Must Support Desktop and Mobile Browsers**
  TempusFlow is a **web-based application** and must be fully functional across both **desktop and mobile platforms**. This requires a **responsive front-end design** that adjusts dynamically to different screen sizes, resolutions, and orientations. The user interface (UI) should offer consistent usability and visual integrity whether accessed from a wide-screen monitor, laptop, tablet, or smartphone. Support for major browsers such as **Google Chrome, Mozilla Firefox, Safari, Microsoft Edge**, and mobile equivalents is required.
- **Backend Must Be Implemented in PHP**
  The server-side logic, including user authentication, session handling, business logic, and communication with the database, **must be implemented using PHP**. PHP is chosen due to its widespread use in web development, seamless integration with MySQL, compatibility with LAMP environments, and its large ecosystem of libraries and documentation.
- **Data Stored in MySQL Database**
  All persistent data—such as user credentials, tasks, events, notes, and preferences—**must be stored in a MySQL relational database**. The data must be structured using normalized schemas, and integrity must be enforced through the use of appropriate constraints (e.g., primary keys, foreign keys). The database should also support efficient querying, indexing, and scalability considerations for future growth.
- **Deployment Requires a LAMP Stack Environment**
  TempusFlow is to be deployed within a **LAMP (Linux, Apache, MySQL, PHP)** stack environment. This widely used server architecture ensures compatibility with the chosen technologies, simplifies the deployment process, and provides a reliable, secure, and scalable platform. The system must be configured to run on a server that meets these criteria, with the necessary versions of each component installed and properly configured.

2.5 Assumptions and Dependencies

The successful operation, deployment, and usage of the **TempusFlow** system depend on certain external conditions and prerequisites. These **assumptions** and **dependencies** are not controlled by the system itself but are critical to its functionality and performance. It is expected that these conditions will be met in the intended deployment and usage environment:

- **Users Have Internet Access**
  TempusFlow is a **web-based application** that requires users to be **connected to the internet** to access and interact with the system. All functionalities—such as user login, data retrieval, task updates, and agenda viewing—are performed through live HTTP requests to the web server. Therefore, it is assumed that end users will have a stable internet connection for uninterrupted access to the platform.
- **PHP and MySQL Are Pre-installed on the Host**
  Since the backend of TempusFlow is built using **PHP**, and all persistent data is stored in a **MySQL database**, it is assumed that the **target server environment** has both **PHP (with required extensions)** and **MySQL installed and properly configured** prior to deployment. Any missing packages or incorrect configuration may hinder the system's operation, so these dependencies must be satisfied as part of the server setup process.
- **Web Browsers Are Up-to-date and Compatible**
  TempusFlow utilizes **modern web technologies** (e.g., HTML5, CSS3, JavaScript, responsive design) to ensure a rich and dynamic user experience. As such, it is assumed that users are accessing the system through **modern, standards-compliant web browsers** that are **regularly updated**. The system is tested and optimized for browsers such as:
  - **Google Chrome**
  - **Mozilla Firefox**
  - **Microsoft Edge**
  - **Safari**
    Outdated browsers or those with disabled JavaScript/CSS features may not support full functionality or render the interface correctly.

---

3. Specific Requirements

This section outlines the **detailed, functional, and non-functional requirements** of the **TempusFlow** system. Each requirement is designed to be:

- **Correct**: Each requirement reflects the intended behavior of the system and accurately addresses the problem it is meant to solve.
- **Traceable**: The origin of each requirement is clearly defined, and it can be traced throughout the development lifecycle. This ensures that every requirement can be verified and validated against design, implementation, and testing.
- **Unambiguous**: Each requirement is clearly stated with no room for multiple interpretations, ensuring that all stakeholders have a consistent understanding of what the system must do.
- **Verifiable**: Every requirement can be tested and validated through various means, such as system testing, user acceptance testing, or review of source code and documentation.
- **Prioritized**: Requirements are classified based on their importance and urgency, helping the development team focus on delivering the most critical features first.

- **Complete**: All aspects of the system functionality are covered, ensuring there are no gaps in the specification.
- **Consistent**: There are no conflicting requirements. All requirements align with each other and with the overall system objectives.
- **Uniquely Identifiable**: Each requirement has a unique identifier to ensure easy tracking and referencing throughout the project lifecycle.

These characteristics ensure that the requirements are **robust, clear**, and **actionable**, providing a strong foundation for system development, testing, and deployment.

3.1 External Interface Requirements

The following section outlines the **external interface requirements** for the **TempusFlow** system, which define the interactions between the system and its users. These requirements include the design and functionality of the **user interfaces (UI)** that provide users with access to system features. The interfaces must be **intuitive**, **responsive**, and **user-friendly** to ensure a positive **User Experience (UX)** across devices and platforms.

3.1.1 User Interfaces

The system will include the following user interfaces, each of which is critical to the core functionality of TempusFlow:

- **Login Page**
  The **Login Page** is the entry point for users to access their personalized dashboard. The page must include:
  - **Username/Email and Password fields** for users to authenticate themselves.
  - A **"Login" button** that submits credentials for validation.
  - A **"Forgot Password" link** to allow users to recover their accounts if they forget their login credentials.
  - A **"Register" link** to redirect new users to the registration page.
    The page should also feature clear error messages if the login attempt is unsuccessful (e.g., invalid credentials or locked account) and display a loading indicator while authentication is in progress.
- **Registration Page**
  The **Registration Page** will allow new users to create an account. This page will include:
  - Fields for **name, email address, and password** (and possibly a **confirm password** field).
  - An option for users to accept the **terms and conditions** and **privacy policy**.
  - A **"Register" button** to submit the information and create an account.
  - A **"Back to Login" link** for users who want to return to the login page.

    - **Validation checks** to ensure the email is unique and the password meets security requirements (e.g., minimum length, special characters).
      The page must provide real-time feedback on form input, such as indicating missing or incorrect fields.

- **Task Dashboard**
  The **Task Dashboard** is the central interface where users can view, manage, and organize their tasks. It must feature:
    - A **task list** displaying tasks with titles, due dates, and status indicators.
    - A **search and filter feature** to allow users to quickly find tasks by name, due date, or priority.
    - A **Create New Task** button for adding new tasks to the system.
    - An **edit and delete option** for managing existing tasks.
    - A clear **task categorization** system (e.g., Personal, Work, etc.) for easy filtering.
    - A **priority indication** (e.g., color-coding or flags) to differentiate high-priority tasks.
    - A visual indicator of task **completion status** (e.g., checkboxes or strikethrough text).
      The Task Dashboard should also update in real-time if changes are made to tasks, allowing users to track their progress efficiently.
- **Event Calendar View**
  The **Event Calendar View** will display a monthly or weekly calendar layout where users can see their scheduled events. The interface should include:
    - A **calendar grid** showing the days of the month or week, with each day displaying a summary of scheduled events.
    - The ability to **click on a day** to view more details or add new events for that day.
    - An **event creation** option where users can define event titles, times, locations, and descriptions.
    - Visual **alerts or reminders** for upcoming events.
    - The ability to **edit or delete existing events** by interacting with them directly in the calendar view.
      The calendar must support both **desktop and mobile views** with a responsive design that adjusts to different screen sizes.
- **Notes Section**
  The **Notes Section** provides users with a space to create and organize their personal notes. Key features of the Notes Section include:
    - A **create new note** button to allow users to write new notes.
    - The ability to **edit and delete** existing notes.
    - **Categorization options** (e.g., by tag or subject) to help users organize their notes.
    - A **search function** to quickly find specific notes.

- Option to **pin important notes** for easy access.
- A simple and clean **text editor** for writing and formatting notes (bold, italics, lists, etc.).
- The ability to **attach dates and reminders** to specific notes for task integration. Notes should be displayed in an organized and easy-to-read layout, with the option to collapse or expand them for a cleaner interface.

3.1.2 Hardware Interfaces

The **TempusFlow** system is designed to be accessed through standard computing devices, and as such, it does not require any specialized or custom hardware. The system operates on general-purpose hardware platforms that support web browsing capabilities. The hardware interface requirements are as follows:

- **Keyboard, Mouse, and Display for Input/Output**
  The system assumes users will interact with **TempusFlow** through standard input and output devices:
  - **Keyboard**: Users will input text, navigate forms, and interact with the interface using a standard keyboard. This includes tasks such as entering login credentials, creating tasks, typing notes, and scheduling events.
  - **Mouse**: The mouse will be used for interacting with clickable elements on the user interface, such as buttons, links, and task/event items. The mouse will also be used for drag-and-drop actions (if applicable) and for adjusting selections within the interface.
  - **Display**: The system assumes a display of sufficient resolution for web browsing. While there is no minimum screen size requirement, **TempusFlow** is optimized for responsive design and will adjust its layout depending on the device's screen size (e.g., desktop, tablet, mobile).
- **No Specialized Hardware Required**
  TempusFlow does not require any **specialized hardware** beyond the standard computing devices used by the user. The system is designed to run on typical **desktop computers, laptops, smartphones**, and **tablets** with internet access. No additional hardware, such as specialized input devices, sensors, or proprietary peripherals, is necessary for full system functionality.

3.1.3 Software Interfaces

The **TempusFlow** system interfaces with several software components, including **server-side programming languages**, **database management systems**, and **client-side web browsers**. The following outlines the required software interfaces for the system:

- **PHP 8+**
  The backend of TempusFlow is built using **PHP** as the server-side scripting language. The system requires **PHP version 8 or later** for:

    - Handling user authentication and session management.
    - Processing server-side logic for tasks, events, and notes.
    - Communicating with the MySQL database to store, retrieve, and update user data.
    - Implementing business logic related to user interactions with the application. PHP 8+ ensures the use of modern features like **type declarations**, **match expressions**, **JIT compilation**, and improved performance compared to earlier versions of PHP.
- **MySQL 5.7+**
  The database for TempusFlow is implemented using **MySQL**, a widely-used relational database management system. The system requires **MySQL version 5.7 or later** to:
    - Store user data, including login credentials, task details, event schedules, and notes.
    - Support SQL queries and transactions to interact with data efficiently and reliably.
    - Use database features such as **foreign keys**, **indexes**, and **views** for optimal data retrieval and integrity. MySQL 5.7+ ensures compatibility with modern SQL features and enhanced performance for handling growing datasets.
- **Web Browsers: Chrome, Firefox, Edge**
  TempusFlow is designed to be used via **modern web browsers** that support the latest web standards. The system is compatible with the following browsers:
    - **Google Chrome**: A widely-used, fast, and secure browser, ideal for accessing web applications.
    - **Mozilla Firefox**: Known for its developer-friendly tools and privacy features, Firefox ensures smooth operation of web applications.
    - **Microsoft Edge**: A modern browser from Microsoft based on Chromium, ensuring full compatibility with web standards.

  The web application is expected to work across the latest versions of these browsers, offering users a consistent and responsive experience. It is assumed that users will keep their browsers updated to avoid compatibility issues.
-

3.1.4 Communications Interfaces

The **TempusFlow** system communicates with users and other external systems through standard web communication protocols. The following outlines the communication interfaces used by the system:

- **Standard HTTP/HTTPS Protocol for Client-Server Interaction**
  TempusFlow will use the **HTTP (Hypertext Transfer Protocol)** and **HTTPS (Hypertext Transfer Protocol Secure)** to facilitate communication between the client (user's web browser) and the server (where the backend is hosted). Key points include:
    - **HTTP/HTTPS Requests**: The system uses standard HTTP/HTTPS methods such as GET, POST, PUT, and DELETE to interact with the server. For example, GET requests will retrieve data (e.g., user tasks or notes), POST requests will send new

data (e.g., new task creation or note addition), PUT requests will update existing data, and DELETE requests will remove data.

- **Secure Transmission**: HTTPS will be used to ensure that all data exchanged between the client and the server is encrypted and secure, protecting sensitive user information such as login credentials, tasks, and notes.
- **RESTful Architecture**: The system will adhere to a **RESTful** (Representational State Transfer) API design, allowing for easy and efficient communication between the frontend and backend. REST principles will be used for structuring the URLs, responses, and interactions between the client and the server.
- **Session Management**: The system will use **cookies** and **sessions** over HTTP/HTTPS to manage user logins and maintain the user's state across multiple requests during their session.

This communication protocol ensures **standardized, secure**, and **reliable** client-server interactions, enabling smooth and consistent functionality across all supported browsers.

3.2 Functional Requirements

3.2.1 User Authentication

User authentication is a critical feature of **TempusFlow**, allowing users to securely register and log in to the system. This section outlines the specific requirements related to the **user authentication** process, including inputs, processing, and outputs.

3.2.1.1 Introduction

The **user authentication** system provides a secure way for users to register and log into their accounts, protecting sensitive user data and ensuring that only authorized individuals can access their personalized dashboards. Authentication involves validating the user’s credentials and managing their session to maintain access during their interaction with the system.

3.2.1.2 Inputs

The authentication system will accept the following inputs from the user:

- **Username**: A unique identifier chosen by the user during registration. This can be an email address or a custom username.
- **Email** (for registration): A valid, unique email address used for account creation and communication.
- **Password**: A secret string of characters entered by the user to verify their identity. Passwords must be securely hashed before storage to prevent unauthorized access.

The inputs will be gathered through form fields on the **Login Page** (for logging in) and the **Registration Page** (for creating new accounts).

3.2.1.3 Processing

The processing of the authentication system involves several steps to ensure both security and proper functioning:

- **Input Validation**: Both the username/email and password will be validated to ensure they are properly formatted. This includes checking:
    - Email format (for registration).
    - Password strength (e.g., minimum length, character diversity).
- **Password Hashing**: When a user registers or changes their password, the system will hash the password using a secure hashing algorithm (e.g., **bcrypt** or **argon2**) before storing it in the database. This ensures that even if the database is compromised, user passwords are not exposed in plain text.
- **Session Creation**: Upon successful login, the system will generate a unique session ID and store it in a session cookie. This session ID will be used to authenticate the user on subsequent requests, allowing them to stay logged in until they log out or their session expires.

The authentication system must also include **rate limiting** and **brute force protection** to prevent unauthorized attempts at accessing user accounts through multiple failed login attempts.

3.2.1.4 Outputs

Upon processing the user's input, the system will produce the following outputs:

- **Dashboard Redirect on Success**: After a successful login or registration, the user will be redirected to their personalized **Task Dashboard**. This page will display the user's tasks, notes, and events, allowing them to interact with the system immediately.
- **Welcome Message**: Upon successful login, a message such as "Welcome back, [Username]" may be displayed on the dashboard to acknowledge the successful authentication.

For new users who successfully register, they will be taken to their personalized dashboard where they can start managing tasks, events, and notes.

3.2.1.5 Error Handling

The system will provide clear, user-friendly error messages if the user's login or registration attempt is unsuccessful. These errors could include:

- **Invalid Login**: If the username/email or password is incorrect, the system will display an error message such as: "Invalid username or password. Please try again."
- **Account Locked**: After a number of failed login attempts, the system will temporarily lock the account and display a message such as: "Your account has been temporarily locked due to multiple failed login attempts. Please try again later."
- **Username/Email Already Exists**: If a user attempts to register with an email or username that is already in use, the system will display: "This email/username is already taken. Please choose another one."

- **Weak Password**: During registration, if the password does not meet the strength criteria (e.g., minimum length, combination of characters), the system will display: "Your password must be at least 8 characters long and contain a mix of letters and numbers."

Error messages will be displayed in a **non-intrusive** manner, guiding users to correct their inputs and retry as needed.

3.2.2 Task Management

Task management is a core feature of **TempusFlow**, enabling users to create, view, and organize their tasks. This section outlines the specific requirements for the **task management** functionality, including the inputs, processing, outputs, and error handling involved in managing tasks.

3.2.2.1 Introduction

The **Task Management** feature allows users to add, update, and remove tasks from their personalized dashboard. Tasks can be associated with deadlines, descriptions, and other relevant information to help users stay organized and on track with their productivity goals. Users will be able to view tasks in a list format, with the ability to update or delete them as needed.

3.2.2.2 Inputs

Users will provide the following inputs when creating or updating tasks:

- **Task Title**: A short and concise description of the task. This field is required for task creation.
- **Task Description**: A more detailed explanation of the task's purpose or steps. This field is optional but recommended to give users more context about the task.
- **Due Date**: The date by which the task should be completed. This field is optional, but if provided, it allows users to prioritize and manage tasks based on deadlines.

These inputs will be collected through form fields in the **Task Management Interface**. If a user is updating an existing task, they will have the ability to modify any of these fields.

3.2.2.3 Processing

Once the user submits the task, the following processing steps will occur:

- **Store Tasks in Database**: The task details (title, description, due date) will be saved to the **MySQL database**. Each task will be associated with the user who created it using a **foreign key** reference to the user's account.
    - The database schema will ensure tasks are properly organized and associated with the correct user.
    - If a due date is specified, it will be stored in the database in a standard date format (e.g., **YYYY-MM-DD**).
- **Associate with User**: Each task will be linked to the user's account. This association ensures that users can only view and manage their own tasks. The system will check the user's session to ensure they are authorized to perform operations on their tasks.

Additional processing can be implemented to allow for future updates to tasks, such as marking a task as completed or updating the due date. This will require operations like updating the task status and re-saving the task details in the database.

3.2.2.4 Outputs

Once the task has been created or updated, the following outputs will be generated:

- **Task List View**: After submitting a task, the user will be redirected to the **Task Dashboard**, where they will see an updated list of tasks. This list will display each task's title, description, due date, and any associated status (e.g., completed, pending). Users will be able to view tasks in a chronological order or filter tasks by status or due date.
    - Each task will have options to **edit** or **delete** it.
    - If tasks are associated with due dates, tasks may be visually prioritized (e.g., tasks with earlier due dates displayed at the top of the list).

The task list view will update dynamically based on user actions, such as adding new tasks or modifying existing ones. If tasks are deleted, the list will reflect this change immediately.

3.2.2.5 Error Handling

The task management system will include appropriate error handling to ensure a smooth user experience:

- **Missing Task Fields**: If the user tries to submit a task with missing required fields (such as the task title), the system will display an error message like: "Please provide a title for your task."
    - The title is a required field, and the system will prevent submission until this field is populated.
- **Invalid Due Date**: If the user enters an invalid due date (e.g., a date in the past), the system will display a message such as: "Please enter a valid future due date for the task."
- **Database Errors**: If there is an issue with the database during the task creation process (e.g., failure to insert the task), an error message will be shown: "An error occurred while saving your task. Please try again later."
- **Invalid Task Update**: If the user attempts to update a task with invalid input (e.g., a malformed due date), the system will notify them with a message like: "Please ensure all fields are correctly filled before updating your task."

Error messages will be displayed near the form where the user interacted, helping them understand and correct the issue.

3.2.3 Event Scheduling

Event scheduling is a key feature in **TempusFlow**, allowing users to organize personal events with specific dates and times. This section outlines the specific requirements for scheduling events, including inputs, processing, outputs, and error handling.

3.2.3.1 Introduction

The **Event Scheduling** functionality enables users to schedule personal events, such as meetings, appointments, or reminders, with specific dates and times. Users will be able to enter details for their events, store them in the system, and view them in a **calendar view** for easy management. The system will also allow for the **editing** and **deletion** of events as needed.

3.2.3.2 Inputs

To create or update an event, users will provide the following inputs:

- **Event Name**: A short title or description of the event (e.g., "Doctor's Appointment", "Team Meeting"). This field is required.
- **Event Date**: The date of the event, which must be in a valid **date format** (e.g., **YYYY-MM-DD**).
- **Event Time**: The time at which the event occurs, provided in a valid **time format** (e.g., **HH:MM**).

These inputs will be gathered through a form within the **Event Scheduling Interface**. The system will validate that the user has entered the correct date and time before allowing the event to be saved.

3.2.3.3 Processing

Once the user submits the event details, the following steps will take place:

- **Event Stored in Database**: The event details, including the name, date, and time, will be stored in the **MySQL database**. Each event will be associated with the user who created it, ensuring that only the relevant user can view and modify their events.
    - The database schema will include fields for the event name, date, and time, and will be linked to the user’s account using a **foreign key** reference.
    - Events will be stored in a **structured format** that allows for easy retrieval and display in a calendar view.
- **Event Retrieval for Calendar View**: The system will retrieve all events associated with the user’s account and display them in a calendar view. This calendar will allow users to view their events on specific dates, and it will update dynamically as events are added, updated, or deleted.
    - The events will be displayed in the appropriate **time slots** on the calendar based on their date and time.
    - The system will also support **event reminders** and **alerts** if required in the future.

3.2.3.4 Outputs

After processing the event data, the following outputs will be generated:

- **Calendar Update**: Once the event is successfully created, the calendar will update to reflect the new event. The event will be displayed in the calendar on the correct date and time.

Users will be able to view their upcoming events in a **day, week, or month view**, depending on the interface options provided.

- Each event in the calendar view will show the event name and may also display the time and additional details (such as location or description) when the user hovers or clicks on the event.
- Users will be able to click on an event to view or edit its details.

If an event is deleted or updated, the calendar will automatically reflect these changes.

3.2.3.5 Error Handling

The event scheduling system will include robust error handling to guide users and prevent data entry issues:

- **Invalid Date Format**: If the user enters an invalid date (e.g., **MM/DD/YYYY** instead of **YYYY-MM-DD**), the system will display an error message such as: "Please enter a valid date in the format YYYY-MM-DD."
- **Invalid Time Format**: If the time entered is in an incorrect format (e.g., **25:00** or **9:00 PM** instead of **HH:MM**), the system will display: "Please enter a valid time in the format HH:MM (24-hour format)."
- **Event in the Past**: If the user attempts to schedule an event in the past, the system will notify them: "Events cannot be scheduled in the past. Please choose a future date and time."
- **Empty Fields**: If any of the required fields (event name, date, time) are left empty, the system will display an error message: "All fields are required. Please provide a name, date, and time for the event."
- **Database Errors**: If there’s a failure in saving or retrieving events from the database, the system will display a message: "An error occurred while saving your event. Please try again later."

These error messages will be displayed clearly and will guide users in correcting their inputs.

3.2.4 Note Management

Note management allows users to create, store, and categorize personal notes within **TempusFlow**. This section outlines the specific requirements for the **Note Management** feature, including inputs, processing, outputs, and error handling.

3.2.4.1 Introduction

The **Note Management** functionality enables users to create personal notes, which can be used to jot down ideas, reminders, or any other type of information. Users can also categorize their notes by adding **tags**, making it easier to organize and retrieve them later. Notes can be **created**, **updated**, or **deleted** as needed.

3.2.4.2 Inputs

When creating or updating a note, users will provide the following inputs:

- **Note Content**: The main text of the note. This field is required and will be used to store the body of the note.
- **Tags (Optional)**: Users can add one or more tags to categorize their note (e.g., "work", "personal", "urgent"). This field is optional, but it helps in organizing notes for future searches.

These inputs will be gathered through a form in the **Note Management Interface**. The note content field will be a **text area** where users can freely input their text. The tags field will allow the user to enter tags, either by typing them manually or selecting from a predefined list of available tags.

3.2.4.3 Processing

Once the user submits the note details, the following processing steps will occur:

- **Save Notes in Database**: The note content and any tags will be stored in the **MySQL database**. If tags are provided, they will be stored in a related table that links each note with its tags, ensuring a proper **many-to-many** relationship between notes and tags.
    - The database will store each note's content, creation date, and the tags associated with it.
    - Notes will be associated with the user who created them using a **foreign key** reference.
- **Update Notes**: If a user decides to update an existing note, the system will overwrite the current content with the new information while preserving any tags (if applicable).
- **Delete Notes**: If a user decides to delete a note, it will be removed from the database entirely, including its associated tags.

These operations will be performed securely, with data integrity checks to ensure that notes are correctly stored, updated, or deleted.

3.2.4.4 Outputs

After processing the note data, the following outputs will be generated:

- **Display Notes in UI**: Once a note is created, updated, or deleted, it will be displayed in the **Notes Section** of the user's dashboard. The notes will be listed with their content, creation date, and any tags associated with them.
    - Notes will be displayed in a user-friendly format, with each note showing the content and an option to view or edit it.
    - If tags are associated with the note, they will be shown next to the note, and users can filter notes by tag.
- **Search and Filter Notes**: Users will have the ability to search for notes by keywords within the content and filter them by tags. The system will dynamically update the list of displayed notes based on the user's search or filter criteria.
    - If a tag is clicked, the system will display all notes associated with that tag.

The display of notes will be optimized for usability, with pagination or infinite scrolling to ensure a smooth experience when the user has a large number of notes.

3.2.4.5 Error Handling

To ensure the integrity and usability of the note management system, the following error handling procedures will be implemented:

- **Empty Note Content**: If the user attempts to save a note without entering any content, the system will display an error message like: "Please enter some text for your note."
- **Invalid Tag Format**: If the user enters an invalid tag (e.g., special characters not allowed in tags), the system will display: "Please enter valid tags (letters, numbers, and spaces only)."
- **Database Errors**: If an error occurs when saving, updating, or deleting notes (e.g., a database connection issue), the system will display: "An error occurred while saving your note. Please try again later."
- **Note Update Conflict**: If a user attempts to update a note and there's a conflict (e.g., the note doesn't exist anymore), the system will display: "The note you are trying to update no longer exists. Please refresh your list."

Error messages will be displayed next to the relevant field or section to guide the user in correcting their input.

3.2.5 Daily Overview

The **Daily Overview** feature provides users with a consolidated view of their tasks, events, and notes for the day. This feature is aimed at helping users stay organized and focused by providing a summary of all the activities they need to complete or attend. It is automatically populated with the relevant data for the current day.

3.2.5.1 Introduction

The **Daily Overview** summarizes all the tasks, events, and notes that are scheduled for the current day. It will aggregate data from the **Task Management**, **Event Scheduling**, and **Note Management** sections, giving users a comprehensive snapshot of their daily activities. The goal is to provide users with a simple, at-a-glance view of everything they need to complete or attend on that particular day.

This feature eliminates the need for users to manually check individual sections for daily tasks, events, and notes. Instead, it aggregates all this information into a single view, making it easier for users to prioritize and manage their day.

3.2.5.2 Inputs

- **N/A**: The **Daily Overview** does not require any direct input from the user. All data displayed in this section is **automatically fetched** from the relevant parts of the application (tasks, events, and notes for the current day).

- **Tasks**: All tasks with today's due date.
- **Events**: All events scheduled for today.
- **Notes**: Any notes categorized with the day or tagged as reminders.

The system will automatically retrieve data from the **Task**, **Event**, and **Note** databases, filter it based on the current date, and aggregate it into the overview.

3.2.5.3 Processing

The system will perform the following steps to generate the daily overview:

- **Fetch Today's Data**: The system will fetch all tasks, events, and notes from the database that are scheduled for the current day.
    - **Tasks**: Tasks with the **due date** matching today's date will be included.
    - **Events**: Events that occur today, based on their **date and time**, will be included.
    - **Notes**: Notes that are specifically categorized for today or tagged as reminders will be included.
- **Aggregate Data**: The data from tasks, events, and notes will be aggregated into a single list or view. This will give the user a comprehensive snapshot of their day's activities.
    - **Tasks**: Will be listed with the title, due time (if applicable), and status (e.g., pending, completed).
    - **Events**: Will be listed with the event name, time, and any additional details (e.g., location).
    - **Notes**: Will be listed by their content, with any tags or reminders clearly displayed.
- **Sort Data**: The system will sort tasks, events, and notes by time, ensuring that the user sees their earliest events or tasks first. The sorting logic will ensure that the daily overview is presented in a chronological order to help users prioritize their day.

3.2.5.4 Outputs

Once the daily data is processed, the system will display the following outputs:

- **Dashboard Summary View**: The system will display the **Daily Overview** as a section on the user's dashboard. This view will include:
    - **Today's Tasks**: A list of tasks due today, sorted by priority or due time.
    - **Today's Events**: A list of events scheduled for today, sorted by time.
    - **Today's Notes**: A list of notes tagged for today or marked as reminders.

The dashboard will be updated dynamically to reflect any changes in tasks, events, or notes as the user interacts with the system. Any modifications to tasks, events, or notes will be reflected in the daily overview in real-time, ensuring that the user always has an accurate representation of their schedule.

3.2.5.5 Error Handling

To ensure smooth user experience, the system will handle any potential issues that may arise with the daily overview:

- **Empty Schedule**: If there are no tasks, events, or notes for the current day, the system will gracefully handle the absence of data by displaying a message like: "No tasks, events, or notes for today. Enjoy your day!"
    - The message will be clearly visible, ensuring that users know the system is functioning properly, even if they have no activities for the day.
- **Database Errors**: If an error occurs while fetching the data (e.g., database connection failure), the system will display: "Error retrieving your daily schedule. Please try again later."
    - This will inform the user that there was an issue in fetching the data and suggest they try again later, ensuring the user experience remains smooth even in the event of an issue.
- **Partial Data**: If one category (tasks, events, or notes) fails to load while the others are successfully displayed, the system will show a partial message like: "Some of your data couldn't be loaded. Please check your connection." This will alert the user to potential issues while ensuring the other categories are still usable.

3.3.1 Performance

Performance is a critical aspect of the **TempusFlow** system to ensure a smooth and efficient user experience. This section outlines the system's performance requirements, including the expected response times for key user interactions.

Introduction

The performance of **TempusFlow** is key to maintaining user engagement and satisfaction. A slow or unresponsive system can lead to user frustration, decreased productivity, and ultimately abandonment of the system. Therefore, optimizing for speed and responsiveness is a top priority.

Performance Criteria

- **Transaction Speed**: The system shall be optimized to ensure that **95% of all user transactions** (such as loading dashboards, submitting forms, updating notes, etc.) are completed within **2 seconds** under typical network conditions (e.g., stable internet connection with moderate latency).
- **Network Conditions**: The system should be capable of performing well under normal network conditions, including situations where network latency is moderate. Performance under slow or unstable network connections will be addressed through appropriate error handling and optimizations, such as caching or loading data in chunks.
- **Dashboard Loading**: Loading the **dashboard** (which aggregates tasks, events, notes, etc.) should take no longer than **2 seconds**. This includes fetching data from the database,

processing it, and rendering it on the user interface. If the data size is large, optimization strategies like **pagination**, **lazy loading**, or **caching** may be employed to keep load times minimal.

- **Form Submission**: Forms such as user registration, task creation, and event scheduling must be submitted and processed within **2 seconds**. This includes input validation, data storage, and the return of a success message or redirection.
- **Search and Filter Operations**: When users perform search or filter operations (e.g., searching for notes or filtering tasks by tags), the response time should also be within **2 seconds** for 95% of queries. This may involve implementing **indexing** and **optimized database queries** to reduce search times.

Scalability Considerations

- **Database Optimization**: As the number of users and data grows, the database queries must be optimized for large datasets. Proper indexing, query optimization, and database normalization techniques will be employed to ensure performance remains high even with increasing data volume.
- **Caching**: For frequently accessed data such as the **Daily Overview** or **task lists**, **caching** strategies like **in-memory caches (e.g., Redis)** may be used to reduce load times by serving pre-rendered content instead of fetching it from the database every time.
- **Load Testing**: Prior to deployment, load testing will be conducted to ensure the system can handle peak traffic, simulating the number of concurrent users and the volume of data expected. The system must maintain its performance standards under varying loads.

Error Handling and Timeouts

- **Timeouts**: If any operation takes longer than expected, the system should have a **timeout** mechanism in place. For example, if fetching user data from the database takes more than 10 seconds, the system should show an error message like: "The request timed out. Please try again."
- **Error Messages**: In case of performance-related issues, such as delays or timeouts, clear error messages should be shown to the user, and the system should gracefully handle the failure. For example, if an event retrieval fails, the user might see: "We are having trouble loading your events. Please try again later."

User Experience

- **Smooth Interactions**: Performance optimizations should not interfere with the quality of user interaction. The system should feel responsive even during data loading phases. Progress indicators (e.g., loading spinners or skeleton screens) will be used where necessary to give users feedback during loading times.
- **Minimal Delays**: All animations, transitions, and user interface updates should occur with minimal delays to create a smooth, responsive experience. This includes real-time updates when tasks or events are created or modified.

3.3.2 Reliability

Reliability is a crucial non-functional requirement for the **TempusFlow** system, ensuring that the application performs consistently and without errors. The system must be resilient to failures, minimize the impact of downtime, and maintain data integrity across all operations. This section outlines the reliability criteria for the system, focusing on system stability, data consistency, and fault tolerance.

Introduction

The **TempusFlow** system must be **reliable**, meaning that it should work consistently over time without unexpected crashes, data loss, or performance degradation. The application must also ensure that data remains intact, even in the event of system failures. To achieve this, robust recovery procedures, error handling mechanisms, and appropriate uptime monitoring must be in place.

Reliability Criteria

- **System Availability**: The system must maintain a high level of availability, with **downtime not exceeding 1 minute per day**. This ensures that users can rely on the application for their productivity needs with minimal interruptions. If the system is unavailable, the downtime must be brief and scheduled to minimize user impact (e.g., maintenance windows should be communicated in advance).
- **Data Integrity**: The system must ensure that no data is lost, corrupted, or compromised during any operation. In the event of a system crash or unexpected failure (e.g., server restart), **data consistency** must be maintained. The following measures will be implemented to ensure this:
    - **Transactional Integrity**: All critical database operations (such as task creation, event scheduling, or note storage) should be handled in transactions. This ensures that, in case of failure, incomplete data operations are rolled back to preserve data integrity.
    - **Backups**: Regular backups will be taken to safeguard against data loss. These backups should be stored securely, and the system should have a recovery procedure in place to restore data from the latest backup in the event of catastrophic failure.
    - **Database Transactions**: Use of **ACID** (Atomicity, Consistency, Isolation, Durability) properties in database transactions will be strictly followed. For example, if an event is being scheduled and a task is being added simultaneously, both operations must either complete successfully or not at all to prevent data inconsistency.
- **Error Handling and Fault Tolerance**:
    - **Graceful Failure**: In case of a system failure or crash, the application should not result in a blank screen or an unhandled error. Instead, the system will display a **friendly error message**, such as: "Something went wrong. Please try again later." This helps users understand that the system is aware of the issue and is being worked on.

    - **Automatic Recovery**: The application should have automated recovery mechanisms in place. In case of failure, such as server crashes or database disconnections, the system should automatically attempt to recover and restore operations. This could include retries of failed operations or switching to a backup server if the primary server fails.
    - **Logging and Monitoring**: All errors, exceptions, and system failures should be logged with detailed information (e.g., error codes, timestamps, and affected modules). These logs will help developers identify recurring issues and address them promptly. Monitoring tools will be used to alert administrators in case of critical failures or performance degradation.
- **Backup and Disaster Recovery**:
    - **Automated Backups**: The system will perform automated daily backups of all user data and system configurations. These backups should be stored securely and off-site (or in the cloud) to prevent loss in case of hardware failure.
    - **Backup Verification**: Backup processes must be verified periodically to ensure data integrity. If a backup cannot be restored correctly, immediate corrective actions will be taken.
    - **Disaster Recovery Plan**: In case of catastrophic system failure (e.g., server crash, hardware failure), a disaster recovery plan will be in place. The plan includes restoring the latest backup and ensuring minimal data loss. The system should also allow for recovery from the most recent consistent backup to minimize downtime.

Uptime and Maintenance

- **Scheduled Maintenance**: The system may require periodic maintenance, but downtime should be scheduled in advance, and users should be notified at least 24 hours before planned maintenance. Maintenance windows should be kept as short as possible, ideally not exceeding 15-30 minutes.
- **Monitoring and Alerts**: A **real-time monitoring system** will be in place to track system performance and uptime. In case of any downtime or performance degradation, **alerts** will be sent to the system administrators to take immediate action.

Testing and Validation

- **Reliability Testing**: Stress testing, fault injection testing, and load testing will be conducted regularly to identify potential weaknesses in the system. These tests will simulate failures and heavy traffic to ensure the system can recover gracefully without data loss or significant downtime.
- **System Health Checks**: Periodic health checks will be carried out to ensure that the system components (e.g., server, database, backup) are functioning properly. These checks will include verifying database integrity, ensuring that backups are taken successfully, and confirming that the recovery mechanisms are operational.

3.3.3 Availability

Availability is a critical non-functional requirement for **TempusFlow**, ensuring that users have uninterrupted access to the service with minimal downtime. The system must maintain **high availability** so that users can rely on it for managing their tasks, events, and notes at any time. This section outlines the criteria for maintaining system availability, including uptime expectations, maintenance procedures, and proactive monitoring.

Introduction

The availability of **TempusFlow** is vital to user satisfaction and productivity. To meet the needs of its users, the system must be designed to ensure **reliable and continuous access** with minimal disruptions. This involves maintaining **high system uptime**, implementing a robust infrastructure, and planning for **scheduled maintenance** during low-traffic periods to reduce impact on users.

Availability Criteria

- **System Uptime**: **TempusFlow** must maintain a minimum of **99% uptime** over the course of any given month. This means that the system should be available to users for at least **99% of the time**, with acceptable downtime being no more than **approximately 7 hours 18 minutes per month** or **1 minute per day**. This availability level ensures that users can depend on the system for their daily productivity tasks without significant interruptions.
- **Unplanned Downtime**: The system must be designed to minimize unplanned downtime, including crashes, bugs, or infrastructure failures. In the case of unexpected failures, recovery procedures must be in place to restore service as quickly as possible, minimizing user disruption.
- **Planned Maintenance**: **Planned downtime** for system maintenance, such as updates, optimizations, and hardware upgrades, should be scheduled during **off-peak hours** (e.g., late night or early morning, depending on user base and traffic patterns). Maintenance downtime should be kept to a minimum and ideally should not exceed **30 minutes** per maintenance window.
    - **User Notifications**: Users should be notified in advance about any planned maintenance, ideally 24 to 48 hours before the scheduled downtime. Notifications should include the expected duration and specific times when the system will be unavailable.
    - **Maintenance Windows**: Maintenance windows should be established to ensure that the least number of users are impacted. Maintenance should be conducted during periods of low system usage (e.g., during the night or on weekends).
- **High Availability Design**: The system should employ a **high availability architecture** that minimizes single points of failure. This includes using **load balancers**, **redundant servers**, and **database replication** to ensure that the system remains operational even if one component fails.
    - **Load Balancing**: The use of **load balancers** ensures that traffic is distributed evenly across servers, preventing any one server from becoming overwhelmed and causing downtime.

- **Redundancy**: Critical components, such as database servers or application servers, should be set up in redundant configurations to ensure that a failure in one component does not lead to a system-wide outage.
- **Failover Mechanism**: The system should be capable of **automatic failover**, meaning that if one server or service fails, traffic will be automatically redirected to another available server or service without impacting users.

Monitoring and Alerts

- **Real-Time Monitoring**: The system will be continuously monitored to detect any availability issues, such as slow response times, server crashes, or database downtimes. Monitoring tools will track key performance indicators (KPIs) such as:
  - **Response time** for user requests
  - **Server uptime** and **health**
  - **Database connectivity**
  - **Application performance**

  This allows the operations team to quickly identify and address any potential problems before they escalate into more serious issues.
- **Automated Alerts**: In the event of an issue, **automated alerts** will notify the system administrators immediately. Alerts will include information about the nature of the problem (e.g., server down, database connection failure) and provide enough context to facilitate a quick resolution.

Fault Tolerance and Recovery

- **Failover and Recovery**: In the event of a failure, the system should have mechanisms in place to automatically recover and restore service. For example, if the primary web server becomes unavailable, the load balancer will redirect traffic to a backup server to ensure minimal disruption. If the primary database server fails, a replica database server will take over the role of the master to maintain data availability.
- **Data Redundancy**: Data redundancy is critical for maintaining system availability. Data should be regularly replicated to secondary storage or databases to ensure that there is no data loss in case of system failure. **Database replication** and **backup strategies** (as outlined in the **Reliability** section) should be implemented to ensure data can be restored quickly if a failure occurs.

Testing and Validation

- **Availability Testing**: Regular **stress testing** and **load testing** should be conducted to simulate high traffic scenarios and test the system's ability to handle large numbers of concurrent users without downtime. These tests will help identify potential weaknesses in the system’s infrastructure that could lead to reduced availability.

- **Failover Testing**: The failover system must be tested periodically to ensure that, in case of a failure, the system can switch to a backup server or service without significant downtime. This will verify that the high availability architecture works as intended.

3.3.4 Security

Security is a critical non-functional requirement for **TempusFlow**, given the sensitive nature of the user data it handles, including personal tasks, notes, events, and schedules. Protecting this data from unauthorized access, tampering, and exposure is essential to ensure the system remains trustworthy, secure, and compliant with industry standards. This section defines the security requirements that **TempusFlow** must meet to safeguard user information.

Introduction

The security of user data and the integrity of system operations are paramount for the success of **TempusFlow**. The system must implement a robust security framework to prevent unauthorized access, data breaches, and other security threats. Security measures should focus on protecting user data both at rest and in transit, ensuring that all interactions with the system remain confidential and protected from external threats.

Authentication and Authorization

- **User Authentication**: The system must ensure that only authorized users can access their accounts and personal data. User authentication will be based on a secure login process, which includes the following:
    - **Username/Email and Password**: Users will authenticate using their **username/email** and **password**.
    - **Password Hashing and Salting**: User passwords must be securely hashed and salted before storing in the database. This prevents storing plaintext passwords and protects user credentials in case of database breaches. Recommended hashing algorithms such as **bcrypt** or **argon2** should be used.
    - **Two-Factor Authentication (Optional)**: While not required by default, **two-factor authentication (2FA)** should be considered for an extra layer of security. This could involve using one-time passcodes (OTPs) sent via email or a mobile app for additional verification during login.
- **Session Management**: Once authenticated, the system will manage user sessions using secure **session tokens**. These tokens will be stored securely (e.g., as HttpOnly cookies) to prevent cross-site scripting (XSS) attacks. Tokens should expire after a set period of inactivity, and users should be logged out after a session timeout or when they manually log out.
- **Role-Based Access Control (RBAC)**: The system will enforce **role-based access control** to ensure that users can only access the data and features relevant to their role. For instance, regular users may have access to task management and event scheduling, while administrators may have access to user management and other system-level settings.

Data Protection

- **Data Encryption**:
    - **In-Transit Encryption**: All data transmitted between the client (user's web browser) and the server must be encrypted using **HTTPS (TLS/SSL)** to prevent man-in-the-middle (MITM) attacks. This ensures that sensitive information such as login credentials, task details, and notes are encrypted during transmission.
    - **At-Rest Encryption**: Sensitive data stored in the database should be encrypted at rest to prevent unauthorized access in the event of a data breach. This includes encrypting data such as passwords, notes, and personal events. **AES-256** encryption is recommended for encrypting sensitive data stored in the database.
- **Data Minimization**: Only the minimum necessary amount of user data should be collected and stored. For example, users' passwords should not be stored directly; only their hashed versions should be stored. Similarly, other sensitive personal data should be collected and stored only when absolutely necessary, and for the shortest period required.

Threat Protection

- **SQL Injection Prevention**: The system must be designed to prevent **SQL injection** attacks. This can be achieved by using **prepared statements** and **parameterized queries** for all database interactions. User input should never be directly concatenated into SQL queries.
- **Cross-Site Scripting (XSS) Prevention**: The system must guard against **XSS attacks**, where malicious scripts are injected into web pages. This can be done by **escaping** all user-generated content before displaying it on the webpage and using **Content Security Policy (CSP)** headers to restrict script execution.
- **Cross-Site Request Forgery (CSRF) Prevention**: **CSRF** attacks occur when a user unknowingly performs an action on the system while authenticated, which can lead to unintended operations. The system should use **anti-CSRF tokens** in all forms to verify that a request is coming from an authenticated user and not a malicious third-party site.
- **Brute-Force Protection**: The system should prevent **brute-force attacks** (where attackers attempt to guess user passwords). This can be done by implementing the following:
    - **Account Lockout**: After a set number of failed login attempts, the system should temporarily lock the account or impose a delay between further login attempts.
    - **Captcha Verification**: Adding CAPTCHA verification (e.g., reCAPTCHA) after multiple failed login attempts can help prevent automated brute-force attacks.

Logging and Monitoring

- **Security Logging**: The system must maintain **security logs** to track significant security events, such as failed login attempts, access to sensitive data, and any changes to user accounts or permissions. These logs should be regularly reviewed for suspicious activity.
- **Audit Trails**: All sensitive actions, such as user login/logout, password changes, and data updates, should be recorded in an **audit trail** to ensure accountability and enable forensic investigation in the event of a security breach.

- **Real-Time Alerts**: The system should be configured to generate real-time security alerts for suspicious activities, such as a large number of failed login attempts, unauthorized access to sensitive data, or abnormal user behavior patterns. These alerts should be sent to the system administrators or security team for prompt investigation.

Backup and Recovery

- **Data Backup**: Regular backups of user data and system configurations should be performed to ensure that, in the event of a security incident, data can be quickly restored. Backup data should be securely encrypted to prevent unauthorized access.
- **Disaster Recovery Plan**: A comprehensive **disaster recovery plan** must be in place to handle security breaches, such as data corruption or loss, and to ensure the system can quickly return to a functional state with minimal data loss.

3.3.5 Maintainability

Maintainability is a crucial non-functional requirement for **TempusFlow**. As the application evolves and new features are added, it is essential that the codebase remains easy to maintain, update, and extend. This section outlines the guidelines and best practices that will be followed to ensure the system is maintainable over time.

Introduction

Maintainability refers to how easily the system can be updated to correct defects, improve performance, or add new features. A maintainable system minimizes the time and effort required for developers to implement changes, fix bugs, and enhance functionality. To achieve maintainability, **TempusFlow** must adhere to principles of clean, modular, and well-documented code, and use tools that facilitate efficient development and collaboration.

Modular Architecture

- **Separation of Concerns**: The application must follow the principle of **separation of concerns** (SoC) to ensure that different parts of the system are decoupled and easier to maintain. This will be achieved by organizing the system into distinct layers, such as the **presentation layer** (UI), the **logic layer** (business logic), and the **data layer** (database).
    - **Presentation Layer**: Responsible for displaying data and interacting with the user. This layer will be implemented using HTML, CSS, and JavaScript.
    - **Business Logic Layer**: Contains the core functionality of the system, such as task management, event scheduling, and user authentication. This layer will be implemented in **PHP** and should follow the **MVC (Model-View-Controller)** pattern to enhance maintainability.
    - **Data Layer**: Responsible for interacting with the database. This layer should utilize **MySQL** and employ **prepared statements** and **ORM** (Object-Relational Mapping) techniques where appropriate to abstract database operations.

- **Modular Code**: The application should be developed in **modular components** that can be independently modified, replaced, or extended without affecting other parts of the system. For example, each feature (task management, event scheduling, note storage) should be implemented as a separate module or class, making it easy to debug or extend.

Code Standards and Best Practices

- **Consistent Coding Style**: The codebase should follow a consistent coding style to improve readability and ease of understanding. This includes naming conventions, indentation, commenting, and code structure.
    - **PHP**: The code should adhere to **PSR-12** (PHP Standards Recommendation) guidelines for formatting. This includes consistent use of indentation, spacing, and bracket placement, as well as following appropriate naming conventions for classes, methods, and variables.
    - **HTML/CSS/JavaScript**: For frontend code, **HTML5** and **CSS3** best practices should be followed, along with modern JavaScript standards (ES6+). JavaScript functions should be small, well-named, and modular.
- **Comments and Documentation**: Proper **inline comments** and **documentation** are crucial for maintainability. Every function, class, or module should have a clear description of its purpose, inputs, outputs, and any side effects. This documentation should follow standards like **PHPDoc** for PHP code and **JSDoc** for JavaScript. Additionally, each module and feature should have high-level documentation explaining how the system works as a whole and how individual components interact.
- **Code Review Process**: To ensure that the code remains of high quality, a **code review process** should be established. Code reviews help identify bugs, security vulnerabilities, and potential areas for improvement before new code is merged into the main codebase. All new code should undergo peer review by another developer before being deployed.

Version Control

- **Git**: **Git** will be used as the version control system to manage the development of **TempusFlow**. Git allows for efficient collaboration, tracking changes, and rolling back to previous versions if necessary.
    - **Branching Strategy**: A **Git branching strategy** (such as **GitFlow**) will be used to manage feature development, bug fixes, and releases. This will help ensure that the codebase remains stable and that new features are introduced in an organized manner.
    - **Commit Messages**: All commits must have clear and descriptive messages that explain what changes have been made and why. This helps future developers understand the context of each change.
    - **Tagging and Releases**: **Tags** should be used to mark significant milestones, such as stable releases, beta versions, or feature completions. This will make it easier to track changes across versions.

Scalability and Extensibility

- **Scalability**: The system should be designed to handle future growth. As more users are added to the system or new features are required, the code should be flexible enough to scale without significant rework. For example:
    - **Database Scalability**: The database schema should be designed to handle growing amounts of data, such as tasks, notes, and events, without performance degradation. Indexes should be created on frequently queried fields, and database queries should be optimized for performance.
    - **Code Scalability**: The system should be able to accommodate new features without requiring significant changes to the existing codebase. For example, new types of tasks or notes should be added as separate modules or extensions that can integrate with the existing system.
- **Extensibility**: As the needs of users evolve or new requirements emerge, the system should be easily extendable. This can be achieved by designing the system with clear interfaces and abstracted components that can be updated or replaced independently of the rest of the application.

Automated Testing

- **Unit Tests**: Automated **unit tests** should be written for key components of the system to verify that each part functions correctly in isolation. This will allow for quick identification of issues and will help maintain the integrity of the system as new changes are made.
- **Integration Tests**: In addition to unit tests, **integration tests** should be written to verify that different components of the system interact correctly. These tests can ensure that tasks are properly stored in the database, events appear correctly in the calendar, and that user authentication functions as expected.
- **Test-Driven Development (TDD)**: Developers should follow **Test-Driven Development (TDD)** practices, where tests are written before the actual code. This ensures that code is tested thoroughly and that new features are added with proper test coverage from the beginning.

Change Management

- **Changelog**: A **changelog** should be maintained to track significant changes to the system, including bug fixes, new features, and performance improvements. This will help developers understand what has changed between versions and provide a history of modifications to the system.
- **Documentation Updates**: Every change made to the system should be reflected in the system's documentation. This includes both high-level documentation (e.g., user guides, system architecture) and low-level documentation (e.g., API references, database schema changes).

3.3.6 Portability

Portability is a key non-functional requirement for **TempusFlow**, ensuring that the application can be accessed and used across various platforms and environments without issues. This section outlines the requirements and guidelines to achieve platform independence and cross-browser compatibility for the application.

Introduction

Portability refers to the ability of the system to run on different operating systems and devices without requiring modifications or additional setup. **TempusFlow** should be able to function smoothly on major operating systems (Windows, Linux, and macOS) and be accessible through standard web browsers, ensuring ease of access for a wide range of users.

Platform Independence

- **Cross-Platform Compatibility**: TempusFlow should be platform-independent, meaning that it should work across different operating systems, including:
    - **Windows**: The application should run without issues on all modern versions of Windows (e.g., Windows 10, Windows 11).
    - **Linux**: The application should be compatible with popular Linux distributions such as **Ubuntu**, **Debian**, **Fedora**, and **Arch Linux**. The backend should be deployable on a **LAMP stack** (Linux, Apache, MySQL, PHP).
    - **macOS**: The application should work seamlessly on macOS, and it should be tested to ensure compatibility with the latest versions of the macOS operating system.
- **No Operating System-Specific Dependencies**: TempusFlow should not rely on any OS-specific libraries or features. It should be built with technologies that are supported on all major platforms, ensuring that no platform-specific configuration or code is required for deployment.

Web Browser Compatibility

- **Supported Browsers**: The application should be compatible with the following modern web browsers:
    - **Google Chrome** (latest stable release)
    - **Mozilla Firefox** (latest stable release)
    - **Microsoft Edge** (latest stable release)
    - **Safari** (latest stable release for macOS)
- **No Special Plugins Required**: The application should be accessible and fully functional without requiring any additional plugins, extensions, or external software. This means relying on standard web technologies such as **HTML5**, **CSS3**, **JavaScript**, and **PHP**, all of which are supported natively by modern web browsers.
- **Responsive Design**: The UI of the application should be designed using **responsive web design** principles to ensure that it is usable and visually appealing across various devices,

including desktops, tablets, and smartphones. The layout should adapt to different screen sizes and resolutions to provide a consistent experience for all users.

Deployment Environment

- **LAMP Stack Compatibility**: TempusFlow should be designed to run on a **LAMP stack** environment, ensuring that it can be deployed on any system supporting **Linux, Apache, MySQL, and PHP**. This ensures the application can be hosted on a variety of server environments across different platforms.
- **Web Server Compatibility**: While the system is optimized for **Apache**, it should also be compatible with other popular web servers (e.g., **Nginx**), which may be chosen by the hosting environment.

No Special Installation Requirements

- **Web-Based Access**: TempusFlow is a **web application** and should be accessible via a URL through any modern browser. Users should not need to install any additional software, plugins, or configuration steps to access the application. The system should function out of the box once the web server is set up and the application is hosted.
- **No Client-Side Dependencies**: The client-side code (HTML, CSS, and JavaScript) should be compatible with the standard set of web technologies supported by modern browsers. There should be no need for users to install browser extensions or any custom software to interact with the application.

Testing for Portability

- **Cross-Platform Testing**: The application should undergo **extensive testing** on all supported platforms (Windows, Linux, and macOS) to ensure that it functions as expected. This includes testing for:
    - User authentication and task management functionality.
    - Event scheduling and notes handling.
    - UI responsiveness and layout on different screen sizes.
    - Browser compatibility (ensuring consistent functionality and appearance across all supported browsers).
- **Continuous Integration (CI)**: A **CI pipeline** should be established to automatically run tests across different platforms and browsers. This will help identify any platform-specific issues early in the development process.

3.4 Design Constraints

Design constraints are critical guidelines that limit or direct how the system is developed and deployed. For **TempusFlow**, the following design constraints will ensure the system adheres to

specified standards, technologies, and methodologies while maintaining a high level of functionality, security, and scalability.

1. Must Use LAMP Stack

- **LAMP Stack Requirement**: TempusFlow must be developed and deployed using the **LAMP stack** (Linux, Apache, MySQL, PHP). This constraint ensures compatibility and reliability across different server environments, particularly on Linux-based systems. The LAMP stack is widely supported and offers a cost-effective and stable platform for web applications.
    - **Linux**: The backend of TempusFlow should be deployed on a Linux server, which is the standard for hosting web applications due to its stability, security, and performance. The system should be compatible with major Linux distributions, such as Ubuntu, Debian, Fedora, and CentOS.
    - **Apache**: The system must run on the **Apache** web server. Apache is a reliable, widely-used HTTP server that is compatible with PHP and offers strong support for various web technologies.
    - **MySQL**: The database should be MySQL 5.7 or higher. This will be used to store user data, tasks, events, notes, and any other system information. The system should rely on standard SQL queries and ensure efficient database operations.
    - **PHP**: The backend logic of TempusFlow should be written in **PHP 8+**. PHP is ideal for developing dynamic web applications and is supported by the LAMP stack. The application should follow best practices for writing secure, scalable, and maintainable PHP code.

2. Adherence to HTML5/CSS3 Standards

- **HTML5 Compliance**: TempusFlow's front-end should strictly adhere to **HTML5** standards. HTML5 offers improved semantics, better multimedia handling, and is supported by all modern web browsers. The use of outdated or non-standard HTML elements should be avoided.
    - The system should leverage HTML5's form controls, video, audio, and other rich media capabilities to provide an engaging and modern user interface.
    - The code should be semantic, using appropriate tags for sections, navigation, headers, footers, and articles. This improves accessibility, search engine optimization (SEO), and readability of the code.
- **CSS3 Standards**: The application's styling should adhere to **CSS3** standards. CSS3 enables the use of modern design techniques such as animations, transitions, and responsive layouts, which will enhance the user experience.
    - **Responsive Design**: The front-end should be designed with **responsive design** principles, ensuring that the layout adapts to different screen sizes and devices.
    - **Grid and Flexbox**: CSS3 Grid and Flexbox should be utilized for the layout, providing flexibility and consistency across various devices and screen resolutions.

- **Minimalist and Consistent Styling**: The styling should be clean, modern, and consistent across all pages. It should be simple and intuitive, providing an easy-to-navigate interface.

3. No Use of Third-Party Authentication Libraries

- **Custom Authentication Implementation**: TempusFlow must implement its authentication mechanisms **without relying on third-party libraries**. While many libraries exist for handling user authentication, relying on custom code allows for greater control over security, functionality, and performance.
    - **Password Handling**: Passwords must be stored securely using **hashing and salting** techniques. PHP's **password_hash()** and **password_verify()** functions should be used for this purpose.
    - **Session Management**: Custom session management should be implemented, using PHP's built-in session handling mechanisms. The system should create and manage user sessions securely after a successful login, with proper session expiration and renewal features.
    - **No OAuth or Social Logins**: Social media login methods (such as Facebook, Google, or Twitter) or third-party authentication services (like Firebase Auth) should not be integrated. Instead, TempusFlow should focus on implementing a straightforward user authentication flow based on email and password.

4. Security Constraints

- **Data Encryption**: All sensitive data, such as passwords and user data, must be encrypted both at rest and in transit. The application should always use **HTTPS** for secure communication between the client and server.
- **SQL Injection Protection**: The system should implement **prepared statements** and **parameterized queries** to prevent SQL injection attacks. All user inputs should be validated and sanitized before being processed.
- **Cross-Site Scripting (XSS) Prevention**: Input data should be properly sanitized to prevent malicious code from being executed in the user's browser.

5. Performance Constraints

- **Efficient Database Queries**: The application should be optimized to minimize the load on the database by using **indexed columns** and **optimized SQL queries**. Complex queries should be broken down or cached where necessary to reduce response times.
- **No Blocking Operations**: The system should ensure that requests to the backend are non-blocking and responsive. If any long-running processes are required, they should be handled asynchronously or in the background.

6. Scalability Constraints

- **Modular Code Design**: The codebase should be modular, following the **MVC** (Model-View-Controller) design pattern to ensure that new features can be added with minimal disruption to the existing system.

- **Database Normalization**: The database schema should follow proper normalization principles to avoid redundancy and ensure scalability as the number of users and tasks grows.

3.5 Other Requirements

In addition to the functional and non-functional requirements outlined above, there are several important system requirements that address the system's operational, security, and monitoring aspects. These include regular database backups and the implementation of an audit log for user activities. These requirements are critical for ensuring data integrity, system reliability, and effective system management.

1. Scheduled Database Backups Every 24 Hours

- **Backup Frequency**: The system must perform a **database backup** every 24 hours to ensure that no data is lost in case of a system failure, disaster, or unexpected downtime. This regular backup schedule will safeguard user data and ensure business continuity.
    - **Automated Backup Process**: Backups should be automated and run without manual intervention to eliminate human error and ensure that the system remains up-to-date. A cron job or scheduled task should be configured on the server to run the backup process at a fixed time every 24 hours.
    - **Backup Storage Location**: The backup files should be stored securely, either on the same server or in an off-site backup location (cloud storage, external drives, etc.). The backup storage must have restricted access to avoid unauthorized data manipulation.
    - **Backup Integrity**: Periodically, the integrity of the backups should be verified to ensure they can be successfully restored in the event of a system failure. The backup process should include both the data (tables) and the structure (schema) of the MySQL database.

2. Audit Log of User Activity for Admin Review

- **Purpose**: The system must maintain an **audit log** to track significant user activities within the application. This log should be accessible by system administrators for security, compliance, and operational monitoring purposes. The log will help administrators identify suspicious activities, troubleshoot issues, and ensure compliance with organizational policies.
    - **Logged Activities**: The audit log should capture key actions performed by users, such as:
        - User logins and logouts
        - Task creation, updates, and deletions
        - Event scheduling and deletions
        - Note creation, edits, and deletions
        - Any system errors or failed login attempts

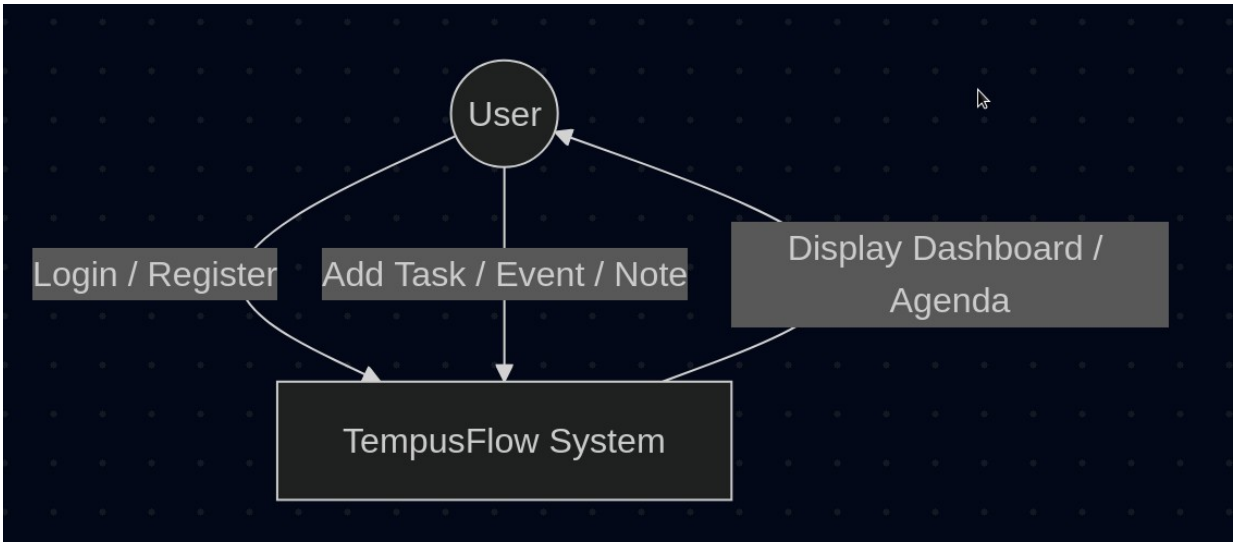
- **Log Details**: For each activity, the log should record the following details:
    - **Timestamp**: The exact time of the activity.
    - **User ID**: The unique identifier of the user performing the activity.
    - **Activity Type**: The type of activity (e.g., login, task creation, event scheduling).
    - **Details/Description**: Additional context or details, such as the task name, event time, or any error messages associated with the activity.
- **Log Access and Retention**: The audit log should be protected from unauthorized access, ensuring that only system administrators can view or modify the logs. The system should implement appropriate user roles and permissions for accessing this log. Logs should be retained for a specified period (e.g., 30 days), after which they can be automatically archived or deleted based on the retention policy.
- **Audit Log Storage**: The audit log should be stored in a separate, secure table in the database or in a separate logging system, with restricted access. It should be protected from tampering, ensuring the accuracy and integrity of the logs.

By implementing regular database backups and maintaining an audit log of user activities, TempusFlow ensures that the system remains reliable, secure, and compliant with best practices for data protection and operational monitoring. These requirements will help safeguard against data loss, provide valuable insights for administrators, and enhance overall system integrity.

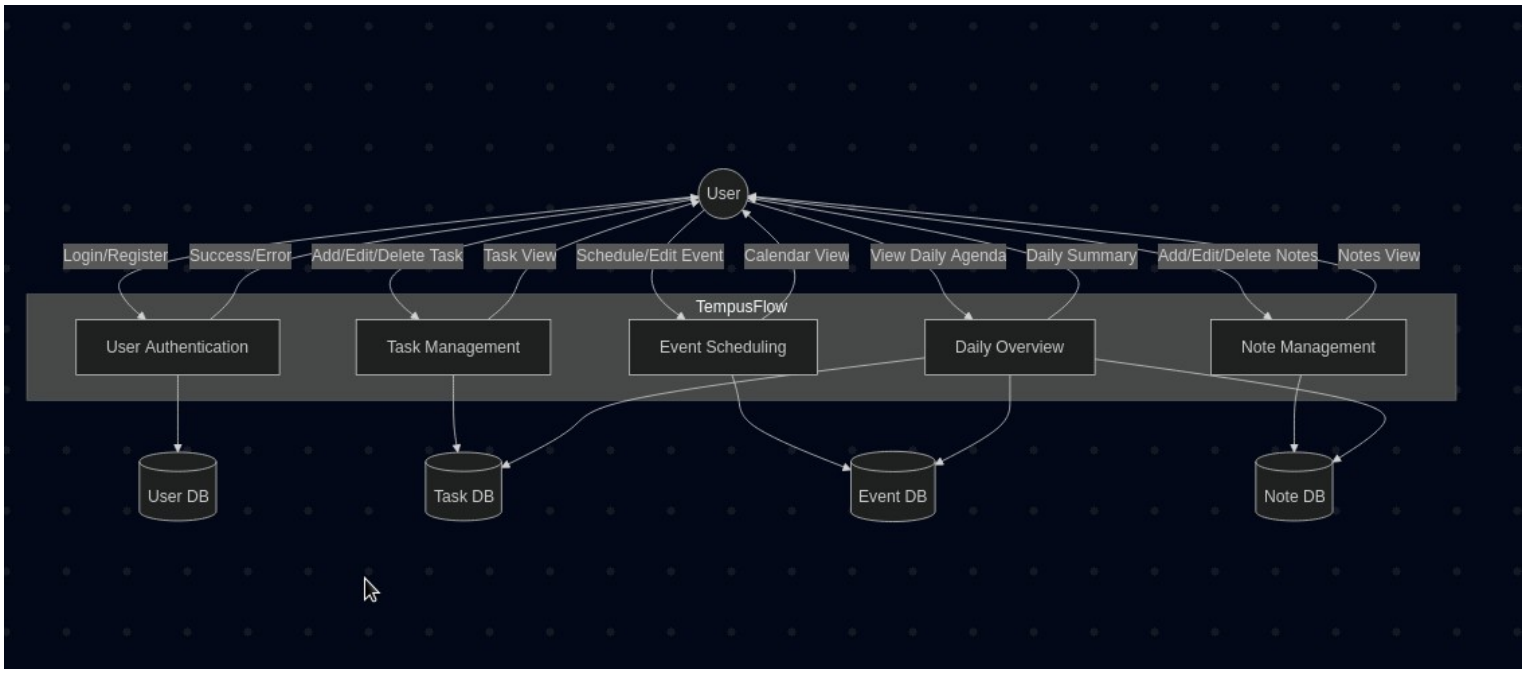
4. Analysis Models

4.1 Data Flow Diagrams (DFD)

- **Level 0 DFD:** Shows user interacting with modules for tasks, events, notes



- **Level 1 DFD:** Expands CRUD operations on task/event/note management



---

5. Github Link

https://github.com/theSolution12/TempusFlow