# Final Project Report

## Time Tracking and Invoicing Application for Freelancers and Small Businesses

APRIL 2023

**APPLICATION NAME**
Tempus

**PROJECT TEAM MEMBERS**
Karan Naik(21bcs051)
Nithish Chouti(21bcs074)
Shaikh Ammar(21bcs105)
Shriya Patare(21bcs115)

## Contents

# I. Project Description

## 1. Project Overview

The purpose of this project was to create a time tracking and invoice generation application for freelancers and small businesses. The application aimed to simplify the process of tracking hours worked, generating invoices, and managing payments.  It includes a timer function and the ability to manually enter time worked, customizable invoice templates, project management tools, reporting, and user management. The project was implemented using agile development methodology and consisted of multiple iterations. The final product was tested for functionality, user experience, and security.

## 2. The Purpose of the Project

### 2a. Background of the Project

This application is designed for small sized businesses and individuals that could benefit from digitizing their records and workflows to enable greater productivity and cost-saving. Currently, small businesses use manual tracking methods that are prone to error, inaccurate and easily destructible. Also these records can be manipulated if not digitalised. Also, small businessmen are not well aware of the invoice formats required and have a tough time generating invoices .Thus requiring a more automated system for keeping records and generating invoices. In case of freelancers who are paid on an hourly basis or on the basis of the feature being built, time tracking is extremely necessary and is a deciding factor of their salary.

Problems with the current system:

- Due in large part to the unconnected and remote nature of small business, not many applications are designed to ease their workflows. A lot of the book-keeping is done manually, on paper.

- This directly leads to lack of portability, data insecurity, lack of speed in insertion, updation and deletion and lack of built-in data analysis tools.
- Digitising said workflows would solve all of the above problems at a very feasible rate for the customers.
- Building this app with dynamic technologies would allow for adding updates to keep up with market trends and customer requests.

## 2b. Goals of the Project

- Efficient Time Tracking: The app should provide a convenient and accurate way to track time spent on various tasks or projects. It should allow users to easily log their time, categorize it, and provide relevant details.
- Streamlined Invoicing: The app should automate the process of generating professional invoices based on the tracked time. It should enable users to create and customize invoices, include billing rates, itemize services, and calculate totals accurately.
- Accuracy and Transparency: The app should ensure that time tracking and invoicing are accurate and transparent, reducing the chances of errors or disputes. It should provide clear records of time entries, allowing users to review and verify the logged hours.
- Productivity Analysis: The app may include features to analyze productivity based on time spent on different tasks or projects. It can provide insights into resource allocation, identify time-consuming activities, and help users optimize their work processes.
- Client Management: The app may incorporate features for managing client information, such as contact details, project history, and payment status. This can help users maintain a centralized database and improve client communication and relationship management.
- Integration and Collaboration: The app may integrate with other tools and platforms, such as project management software, accounting systems, or collaboration tools. Seamless integration enables smooth data exchange and improves overall workflow efficiency.
- Reporting and Analytics: The app should offer reporting capabilities to generate various time-related reports, such as timesheets, billing summaries, and

project-specific analytics. These reports can provide valuable insights into profitability, resource allocation, and project performance.

- Mobile Accessibility: The app may have a mobile version or companion app to facilitate time tracking and invoicing on the go. This allows users to log time and create invoices from anywhere, improving flexibility and convenience.
- Scalability and Customization: The app should be designed to accommodate the growing needs of businesses or individuals. It should be scalable to handle increased data volume and customizable to adapt to specific invoicing requirements or industry standards.
- User Experience and Usability: The app should prioritize a user-friendly interface and intuitive navigation to enhance usability. A well-designed user experience reduces the learning curve, increases adoption, and maximizes user satisfaction.

These goals collectively aim to simplify and streamline the time tracking and invoicing process, save time and effort for users, improve accuracy and transparency, and enhance productivity and financial management for individuals or organizations using the app.

## 3. Stakeholders

- **Small business owners**: Businesses with a small number of employees that need to track time and bill clients for their services. These are the stakeholders with the most influence as they are the intended main market.
- **Freelancers**: Independent contractors who need to track time and bill clients for their services. This group of individuals is tech-savvy and would freely embrace any technology that is intuitive, convenient and optimizes their workflow. They are the secondary stakeholders.
- **Agencies**: Businesses that provide services to multiple clients and need to track time and bill clients for their services.
- **Consultants**: Individuals who provide consulting services and need to track time and bill clients for their services.
- **Solo entrepreneurs**: Self-employed individuals who need to track time and bill clients for their services.

The end users will have different needs and use cases, but generally, they all need a tool to track time, create invoices, and manage clients and projects.

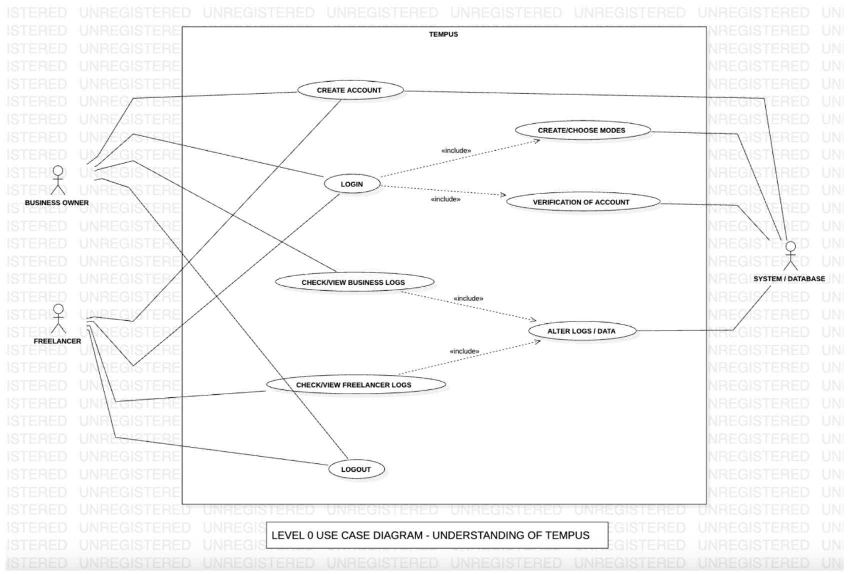
# II. Requirements

## 4. Product Use Cases

### 4a. Product Use Case Diagrams

We have created a Level 0 and Level 1 use case diagram.

Level 0 Use Case Diagram:



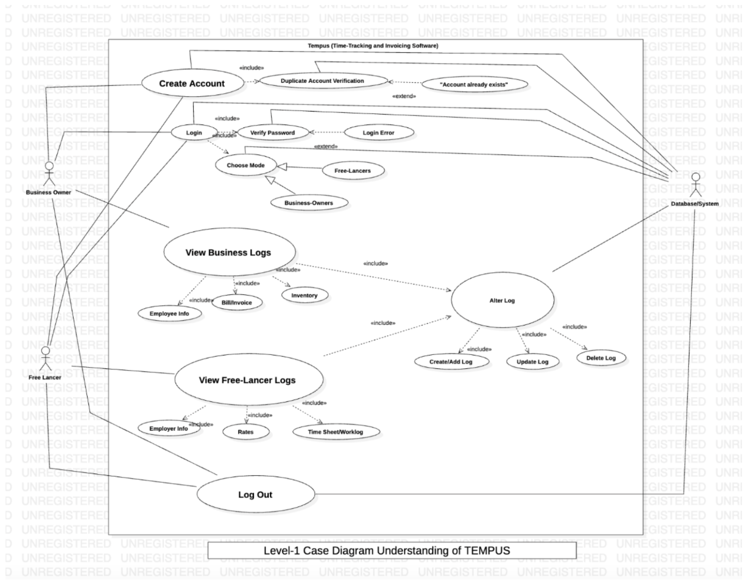
Here the use cases are as follows:

- Create Account: This allows the user to create a new account in the system.
- Login: This allows the user to log into the application once he/she has already created the account.
- Verification of Account: This is included in the 'Login' use case and is used to check if the user's account exists in the system and if the password matches the id.

- Create/Choose Modes: This is also included in the ‘Login’ use case and is used choose one of the models included in the application, i.e. either the Business Mode or the Freelancer Mode.
- Create/View Business Logs: This allows the user to create new logs or view pre-existing logs in the Business mode.
- Create/View Freelancer Logs: This allows the user to create new logs or view pre-existing logs in the Freelancer mode.
- Alter Logs/Data: This use case is included in the Create/View Business Logs and Create/View Freelancer Logs use cases and is used to help the user alter the logs that have been added to the respective modes.
- Logout: This allows the user to log out of the application.

  Actors: Actors are Businesspersons, Freelancers and Database. Businesspersons and freelancers are the users and the database stores all the data and returns data on request.

  Level 1 Use Case Diagram:



  The Level 1 use case diagram is shown above which is a detailed view of the Level 0 diagram.

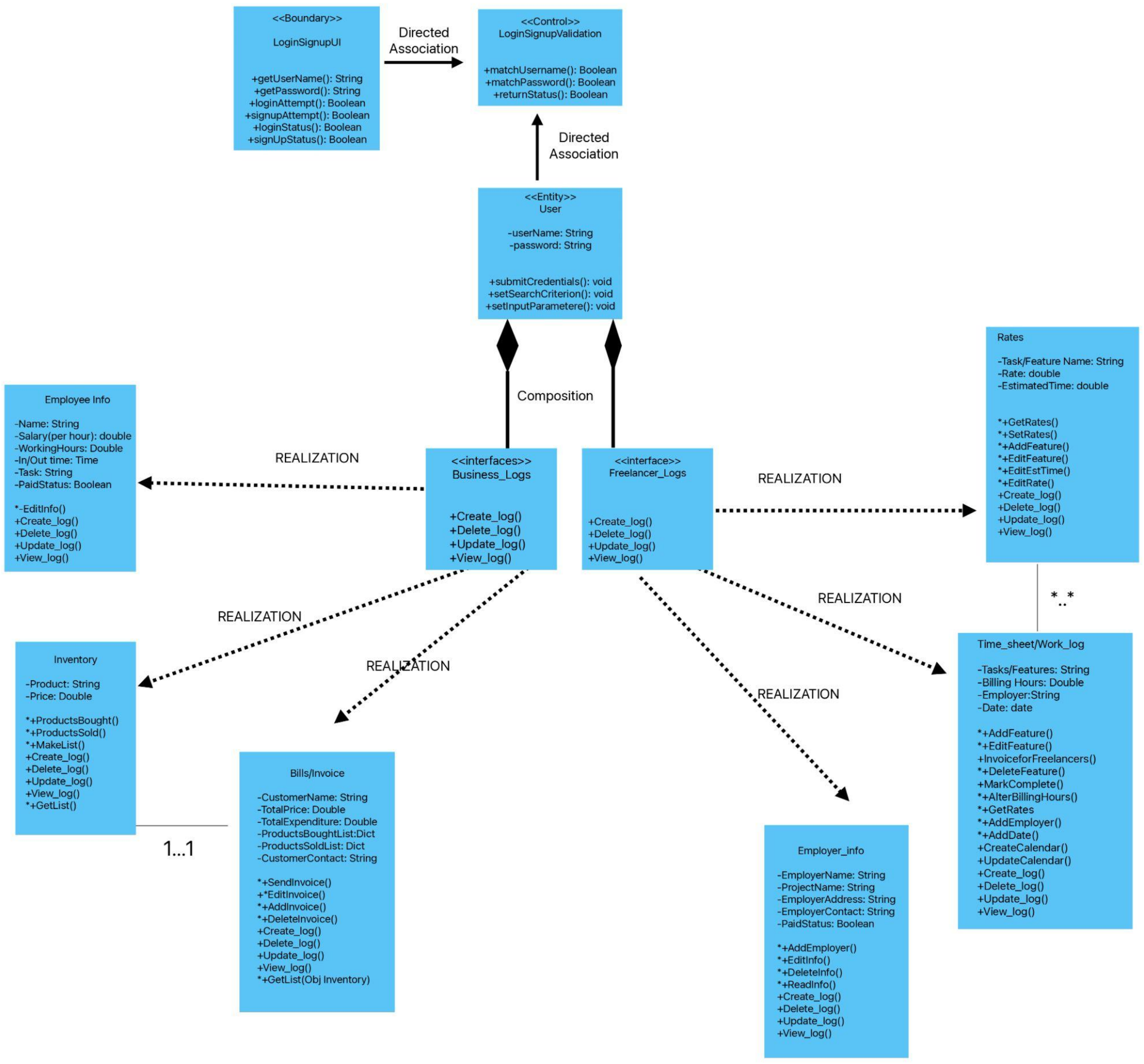
- Create Account use case includes the Duplicate Account Verification use case that verifies if a new account that is being created already exists in the system. The Duplicate Account Verification use case then extends to *Account Already

Exists*, this message is shown if a duplicate account similar to the account being created exists in the system.

- The Login use case includes the Verify Password and Chose Modes use cases. Verify Password use case extends to Login Error, where if the password entered for a certain account is incorrect, the system returns a Login Error. The Choose Mode extends to Freelancer and Business Modes. The Freelancer Mode is for freelancers and the Business Mode is for businesspersons.
- For Business Owners, we have the View Business Logs use case that includes Employee Info, Bills/Invoice and Inventory use cases. Employee Info stores the information of the employees hired by the business owners, this information can contain the name, age, date of birth, payment status, tasks performed, salary, working hours and in/out times of each employee. The Bills/Invoice use case stores information of the latest invoices, helps generate automated invoices based on given inputs and sends invoices to the clients. The Inventory use case stores information about the sold and bought goods, helping in tracking the expenses and income of the business.
- For Freelancers, we have the View Freelancer Logs use case that includes Employee/Client Info, Rates and Time Sheet/Worklog. The Employee/Client Info stores the name, contact number, email id, project and address of the client that hires the freelancer. Rates stores the rates that the freelancer demands to be paid for each feature or each hour. Time Sheet/Worklog stores the details about tasks/features that the freelancer has to complete/create. It helps measure the time taken by the freelancer to build that feature as freelancers can be paid on an hourly basis. This also helps the freelancer better understand how his/her time is divided and acts as a proof of hours spent for the client to pay accordingly.
- The View Business Logs and View Freelancer Logs include the Alter Log use case. Alter Log includes Create/Add Log, Update Log and Delete Log cases.
- The Logout case helps users log out of their accounts.
  Businesspersons, Freelancers and Database are the actors. Businesspersons use the Business Mode, Freelancers use the Freelancer Mode and the Database serves all requests made by the user.

# 5. Data Requirements

## 5a. Class Diagrams



The above is the class diagram of the proposed application that helps understand the data requirements for the project. The Tempus system has 11 classes. Starting with the LoginSignupUI class, this class takes the userID and password from the user. It uses the LoginSignupValidation class to verify if the password matches the ID. These two classes are related by directed association. The User class is also related by the directed association to the LoginSignupValidation class. This class uses the username and password entered by the user.

Then we have the Business_Logs and Freelancer_Logs interfaces which each have the Create_log(), Delete_log, Update_log() and View_log() methods/functions. The User class is related to the  Business_Logs and Freelancer_Logs interfaces by Composition, where User is the container and the Business_Logs and Freelancer_Logs interfaces are contained in it. The Business_Logs interface extends to 3 other classes namely Employee Info, Inventory and Bills/Invoice classes. These 3 classes are related to the Business_Logs interface through Realization and implement the functionality of the Business_Logs interface by using the methods defined in Business_Logs. The  Freelancer_Logs interface extends to 3 other classes namely Employer_info, Rates and Time_sheet/Work_log classes. These 3 classes are related to the Freelancer_Logs interface through Realization and implement the functionality of the Freelancer_Logs interface by using the methods defined in Freelancer_Logs. The Employee Info class has attributes called Name, WorkingHours, Salary, In/Out time, Task and Paid Status. The new method it includes is EditInfo() that helps edit information stored in the Employee Info section. The Inventory class has Product and Price attributes to store the information/name of the product and its respective price. The new methods it includes are ProductsBought(), ProductsSold() and MakeList() where the ProductsBought() method shows the bought products, the ProductsSold() method shows the sold products and the MakeList() method shows the list of products in the inventory, it can asos be used to make a list of sold products and be used in the invoice generator. Bills/Invoice has CustomerName, TotalPrice, TotalExpenditure, ProductsBought, ProductsSold, CustomerContact attributes that store the client information and inventory information to include in the invoice. The new methods it includes are SendInvoice(), EditInvoice(), DeleteInvoice(), AddInvoice() and GetList() methods. SendInvoice() sends the invoice generated to the client on the given contact. EditInvoice() lets the user manually edit the invoice. DeleteInvoice() helps the user delete the invoice. AddInvoice() allows the user to delete the invoice. GetList() method gets list of the items to be included in the invoice. The MakeList() and GetList() methods have a one-to-one relationship as the items included in the MakeList() method are sent on calling the GetList() method to be included in the invoice. Employer_info class has EmployerName, ProjectName, EmployerAddress, EmployerContact and PaidStatus attributes to store the employer information. The new methods it includes are AddEmployer(), EditInfo(), DeleteInfo() and ReadInfo() to add, edit , delete and view the employer information. Rates classes has Task/Feature Name, Rate and EstimatedTime as attributes to store the task/feature name, rate for that task and the estimated time to be taken to complete the task. The new methods it includes are GetRates(), SetRates(), AddFeature(), EditFeature(), EditEstTime() and EditRate() methods to get rates from the user as input and set the. To get features from the user and be able to edit them. To be able to edit the estimated time and rate indeed by the user. Time_sheet/Work_log class has Tasks/Features, Billing Hours, Employer and Date as attributes to store the tasks, billing hours, employer name and the date the task is completed. The new methods it includes are AddFeature()[helps ads feature to the timesheet], EditFeature()[helps edit the added features], InvoiceforFreelancers()[generates invoice for the freelancer depending on the completed tasks and rates], DeleteFeature()[helps delete a feature], MarkComplete()[helps mark a task as complete after completion], AlterBillingHours()[allows user to alter the pre-added billing hours], GetRates()[helps get rates for each task from the Rates class], AddEmployer()[allows user to add the employer info for whom the specified feature is being built], AddDate()[helps add date of the task being worked upon], CreateCalendar()[helps create calendar to see the tasks and the dates they were completed on and the schedule new tasks] and UpdateCalendar()[helps update the calendar] methods. The Rates class and Time_sheet/Work_log class have a many-to-many relationship where on calling the GetRates() method from the Time_sheet/Work_log class, Rates for that feature are returned from the Rates class.

## 6. Technical feasibility

We will be developing a user-friendly web based application. The following aspects need to be considered:

- **Platform and Technology:** The application will be developed using Python(FLASK) for backend, JavaScript(ReactJS) along with HTML and CSS for front-end and MySQL as our databases(since ours would be a small database)

- **Integration:** Ensuring that the application can integrate with any existing systems that the small business or freelancer may already have in place, such as accounting software, to ensure a smooth and efficient workflow. Our app would have to have features allowing for this. This aspect would have to be considered during the design phase seriously.

- **Security:** The application should be secure to protect sensitive data and financial information of the clients. Our application would have authentication done while logging in and information and privileges would be granted or restricted based on the type of login.

## 7. Financial feasibility

Being a web application, it would have an associated hosting cost. In the beginning we would use heroku, a free hosting service and this would later be changed to a more secure server once it gets up and running. Since the system doesn't consist of multimedia transfer bandwidth required would be minimal.

The system will follow the trial model, meaning no cost will be charged for customers until a certain user base has been established then each customer will be given a free trial for a limited time period after which payment would be required.

Bug fixes and maintenance would have its associated costs and would need a dedicated team at the later stages of the project. Besides the associated costs, the benefits provided to the customer would be huge and would include portability, digitisation and security along with relevant data analysis tools.

So in the long run, this project would be financially feasible.

## 8. Resource feasibility

Resources required are as follows:

- **Hardware**(Computers capable of programming i.e laptops and desktops along with required peripherals such as data drives, keyboards etc)
- **Hosting Space**(Heroku, freely available, will be used in the beginning but a secure server will be required at the later stages)

- **Programming tools**(Text editors, DBMS, compilers and interpreters. We shall use the open source versions of these softwares)
- **Human Resource**s(People capable of programming the relevant tech and all the associated costs such as healthcare, vacation, salary etc that comes with it. Currently, we are a team of four)

Granted the resources in question($2 million) it is safe to assume that this project is feasible at least in terms of resources.

# III. Design

## 9. System Design

### 9a. Design Goals

The design goals considered while designing Tempus to meet the specific needs and requirements are as follows:

- **Simplicity and Ease of Use**: The app should have an intuitive and user-friendly interface, making it easy for freelancers and small business owners to navigate and use the app efficiently. Minimizing the learning curve and providing clear instructions can help users quickly adopt the app.
- **Time Efficiency**: The app should optimize the time tracking and invoicing processes, allowing users to quickly log their time, generate invoices, and manage client information. Automation features, such as pre-populated fields or templates, can save time and reduce manual data entry.
- **Flexibility and Customization**: Freelancers and small businesses often have unique invoicing requirements. The app should allow for customization of invoices, such as adding branding elements, modifying layouts, or including specific payment terms. Providing flexibility ensures that the app caters to individual business needs.
- **Mobile Accessibility**: Many freelancers and small business owners work remotely or on the go. Offering a mobile version or companion app that is

optimized for smartphones and tablets allows users to track time and create invoices anytime, anywhere, enhancing flexibility and convenience.

- **Integration with Accounting Systems**: Small businesses often use accounting software for financial management. Integrating the time tracking and invoicing app with popular accounting systems (e.g., QuickBooks, Xero) enables seamless data transfer, eliminating the need for manual entry and improving accuracy.
- **Real-Time Reporting and Insights**: The app should provide real-time reporting and analytics features that give freelancers and small business owners visibility into their time usage, project profitability, and financial performance. Clear and concise reports help users make informed decisions and identify areas for improvement.
- **Client Management**: Effective client management is crucial for freelancers and small businesses. The app should include features to store and manage client information, track project history, and monitor payment statuses. Additionally, communication tools (e.g., email notifications, reminders) can facilitate timely interactions with clients.
- **Security and Data Privacy**: Protecting sensitive client information and maintaining data privacy is essential. Implementing robust security measures, such as encryption, secure authentication, and regular backups, instills confidence in users regarding the safety and privacy of their data.
- **Cost-Effectiveness**: As freelancers and small businesses often have limited budgets, the app should provide value for money. Offering affordable pricing plans, free tiers, or flexible subscription options can help ensure that the app remains accessible and affordable for its target users.
- **Customer Support and Documentation**: Providing comprehensive documentation, tutorials, and responsive customer support is crucial to assist users with any questions or issues they may encounter while using the app. Clear communication channels and timely assistance contribute to a positive user experience.

By focusing on these design goals,Tempus can cater specifically to the needs of freelancers and small businesses, helping them streamline their administrative tasks, improve productivity, and enhance financial management.

## 10. Proposed Software Architecture

1. Presentation Layer:
   - User Interface (UI): This layer focuses on the app's visual representation and user interactions, providing screens, forms, and controls for users to interact with the app.
   - User Experience (UX): Design principles and considerations are applied to create an intuitive and user-friendly interface that enhances the overall user experience.

2. Client-Side Application Layer:
   - Frontend Framework: A frontend framework can use React, HTML and CSS to build the client-side application. It handles the presentation logic, UI components, and user interactions.

3. Server-Side Application Layer:
   - Backend Framework: A backend framework can use Node.js with Express.js to handle server-side logic, API endpoints, and database interactions.
   - Business Logic: This layer encapsulates the core functionalities of the app, including time tracking, invoicing, reporting, and analytics. It handles the processing and manipulation of data.
   - API Controllers: These controllers expose API endpoints to handle requests from the client-side application, facilitating communication between the frontend and backend.

4. Data Layer:
   - Database: A relational database management system of MySQL can be used to store and manage structured data. The database handles user profiles, client details, time entries, invoices, and other relevant information.

5. Integration Layer:

- External Integrations: This layer enables integration with external systems, such as accounting software (e.g., QuickBooks, Xero) or payment gateways (e.g., PayPal, Stripe), allowing seamless data exchange and processing in the future.
- API Design and Documentation: Well-defined APIs, following RESTful principles, enable integration with third-party services and provide documentation to guide developers in utilizing the APIs.

6. Security Layer:

- Authentication and Authorization: This layer handles user authentication, ensuring secure access to the app and its features.
- Input Validation and Sanitization: Proper validation and sanitization techniques should be implemented to protect against common security vulnerabilities, such as SQL injection or cross-site scripting (XSS) attacks.
- Security Best Practices: Employing secure coding practices, encryption algorithms, and secure communication protocols (e.g., HTTPS) helps protect sensitive data and prevent unauthorized access.

7. Infrastructure Layer:

- Deployment: The app can be deployed on servers or cloud platforms (e.g., AWS, Azure, Heroku) to make it accessible to users. Deployment tools and techniques (e.g., Docker, containerization) can be utilized for efficient deployment and scalability.
- Monitoring and Scalability: Monitoring tools can be employed to track app performance, handle errors, and scale resources based on user demands, ensuring a reliable and scalable infrastructure.

The proposed software architecture provides a modular and scalable structure that separates concerns, promotes code reusability, and facilitates maintenance and future enhancements. It leverages well-established frameworks, libraries, and technologies to build a robust and efficient time tracking and invoicing app for freelancers and small businesses.

## 11. User Interface

We aim to design an intuitive, user-friendly, and efficient user interface (UI) for Tempus Here are some key elements and components that were  included in the UI:

1. Dashboard:
   - A centralized dashboard provides a choice between the Business and the Freelancer Mode.
   - It may include widgets or cards displaying key metrics, notifications, and quick access to frequently used features.

2. Time Tracking:
   - Timer: A prominent timer component allows users to start, pause, and stop tracking their time for specific tasks or projects as part of the Time sheet.
   - Task/Project Addition: Users can add the task or project they are working on.
   - Task Details: Users can provide additional details, such as task descriptions or tags, to provide context for their time entries.

3. Invoicing:
   - Invoice Creation: A dedicated section where users can create and customize professional invoices, including the client's name, billing address, payment terms, and invoice line items.
   - Invoice Templates: Pre-designed invoice templates or the ability to create custom templates can help users quickly generate consistent and professional-looking invoices.
   - Billing Rates: Users can specify different billing rates for various tasks, projects, or clients to ensure accurate invoicing.

4. Client Management:
   - Client List: Users can maintain a list of their clients, including contact information, project history, and payment status.

- Client Details: Users can view and edit detailed information about each client, such as billing address, email, phone number, and notes.
- Communication: Integration with email or messaging platforms can enable direct communication with clients from within the app.

5. Reports and Analytics:
   - Timesheets: Users can view detailed timesheets that summarize their tracked hours and tasks, allowing them to review and verify their time entries.
   - Financial Reports: Reports that provide insights into income, expenses, and project profitability, helping users analyze their financial performance.
   - Visualizations: Charts and graphs can present data visually, making it easier for users to understand trends and patterns in their time usage and invoicing.

6. Settings and Preferences:
   - User Profile: Users can view and update their personal information, such as name, profile picture, and contact details.
   - Preferences: Customization options for app settings, such as time zone, currency, language, and notification preferences.
   - Integrations: Settings for integrating the app with external platforms, such as accounting software or project management tools.

7. Responsive Design:
   - The UI should be responsive and optimized for different devices and screen sizes, ensuring a consistent and seamless experience across desktops, tablets, and mobile devices.

The UI will be kept  clean, uncluttered, and visually appealing. Proper organization of components, logical grouping of related features, and consistent use of color schemes and typography will be implemented to create  a positive user experience. Additionally, clear labels, tooltips, and contextual help will be used to guide users and reduce any learning curve associated with the app.

## 12. Object Design

The Tempus system has 11 classes. Detailed explanation can be found at 5a.

1. LoginSignupUI:
    - This class takes the userID and password from the user. It uses the LoginSignupValidation class to verify if the password matches the ID. These two classes are related by directed association.
    - Attributes: None
    - Methods: getUserName(), getPassword(), loginAttempt(), signupAttempt(), loginStatus(), signUpStatus()
2. LoginSignupValidation:
    - Attributes: None
    - Methods: matchUsername(), matchPassword(), returnStatus()
3. User:
    - It is also related by the directed association to the LoginSignupValidation class.
    - Attributes: userName and password.
    - Methods: submitCredentials(), setSearchCriterion(), setInputParameter()
    - The User class is related to the Business_Logs and Freelancer_Logs interfaces by Composition, where User is the container and the Business_Logs and Freelancer_Logs interfaces are contained in it.
4. Business_Logs (interface):
    - Methods: Create_log(), Delete_log, Update_log() and View_log()
    - The Business_Logs interface extends to 3 other classes namely Employee Info, Inventory and Bills/Invoice classes. These 3 classes are related to the Business_Logs interface through Realization and implement the functionality of the Business_Logs interface by using the methods defined in Business_Logs.
5. Freelancer_Logs (interface):
    - Methods: Create_log(), Delete_log, Update_log() and View_log()
    - The Freelancer_Logs interface extends to 3 other classes namely Employer_info, Rates and Time_sheet/Work_log classes. These 3 classes are related to the Freelancer_Logs interface through Realization and implement the functionality

of the Freelancer_Logs interface by using the methods defined in Freelancer_Logs.

6. Employee Info:
   - Attributes: Name, WorkingHours, Salary, In/Out time, Task and Paid Status.
   - Methods(excluding those extended by the interface): EditInfo() that helps edit information stored in the Employee Info section.
7. Inventory:
   - Attributes: Product, Price to store the information/name of the product and its respective price.
   - Methods(excluding those extended by the interface): ProductsBought(), ProductsSold() and MakeList() where the ProductsBought() method shows the bought products, the ProductsSold() method shows the sold products and the MakeList() method shows the list of products in the inventory, it can asos be used to make a list of sold products and be used in the invoice generator.
8. Bills/Invoice:
   - Attributes: CustomerName, TotalPrice, TotalExpenditure, ProductsBought, ProductsSold, CustomerContact to store the client information and inventory information to include in the invoice.
   - Methods(excluding those extended by the interface): SendInvoice(), EditInvoice(), DeleteInvoice(), AddInvoice() and GetList() methods. SendInvoice() sends the invoice generated to the client on the given contact. EditInvoice() lets the user manually edit the invoice. DeleteInvoice() helps the user delete the invoice. AddInvoice() allows the user to delete the invoice. GetList() method gets list of the items to be included in the invoice. The MakeList() and GetList() methods have a one-to-one relationship as the items included in the MakeList() method are sent on calling the GetList() method to be included in the invoice.
9. Employer_info
   - Attributes: EmployerName, ProjectName, EmployerAddress, EmployerContact and PaidStatus to store the employer information.
   - Methods(excluding those extended by the interface): AddEmployer(), EditInfo(), DeleteInfo() and ReadInfo() to add, edit , delete and view the employer information.

10. Rates classes:
    - Attributes: Task/Feature Name, Rate and EstimatedTime to store the task/feature name, rate for that task and the estimated time to be taken to complete the task.
    - Methods(excluding those extended by the interface): GetRates(), SetRates(), AddFeature(), EditFeature(), EditEstTime() and EditRate() methods to get rates from the user as input and set the. To get features from the user and be able to edit them. To be able to edit the estimated time and rate indeed by the user.
11. Time_sheet/Work_log:
    - Attributes: Tasks/Features, Billing Hours, Employer and Date to store the tasks, billing hours, employer name and the date the task is completed.
    - Methods(excluding those extended by the interface): AddFeature()[helps ads feature to the timesheet], EditFeature()[helps edit the added features], InvoiceforFreelancers()[generates invoice for the freelancer depending on the completed tasks and rates], DeleteFeature()[helps delete a feature], MarkComplete()[helps mark a task as complete after completion], AlterBillingHours()[allows user to alter the pre-added billing hours], GetRates()[helps get rates for each task from the Rates class], AddEmployer()[allows user to add the employer info for whom the specified feature is being built], AddDate()[helps add date of the task being worked upon], CreateCalendar()[helps create calendar to see the tasks and the dates they were completed on and the schedule new tasks] and UpdateCalendar()[helps update the calendar] methods.
    - The Rates class and Time_sheet/Work_log class have a many-to-many relationship where on calling the GetRates() method from the Time_sheet/Work_log class, Rates for that feature are returned from the Rates class.

# IV. Test Plans

## 13. Features to be Tested

When testing our app, it is important to ensure that all key features are thoroughly tested to verify their functionality, accuracy, and usability. Here are some of the features that should be tested:

1. User Registration and Authentication:
    - Verify that users can successfully register for an account.
    - Test the login process to ensure users can access their accounts securely.

2. Time Tracking:
    - Test the ability to start, pause, and stop timers accurately.
    - Verify that time entries are recorded correctly for different tasks and projects.
    - Test manual time entry functionality, allowing users to input time manually.

3. Task and Project Management:
    - Ensure users can create, update, and delete tasks and projects.
    - Verify that tasks can be assigned to specific projects and have the correct attributes (name, description, billable status).

4. Client Management:
    - Test the ability to add, update, and delete client information.
    - Verify that clients can be associated with relevant projects and invoices.

5. Invoicing:
    - Test the invoice creation process, ensuring accurate calculation of billable hours and amounts.

- Verify that invoices can be customized with relevant client and business information.
- Test the ability to preview, download, and send invoices to clients.
- Verify that invoice numbering and tracking are accurate.

6. Reporting and Analytics:
    - Test the generation of timesheets and financial reports based on tracked time and invoicing data.
    - Verify that reports provide accurate summaries, charts, and visualizations.
    - Test report customization options, such as date range selection and filtering.

7. Integration with External Systems:
    - When the app integrates with accounting software or payment gateways, test the data synchronization and communication with these external systems.
    - Verify that invoices and financial data can be seamlessly exported or imported.

8. Notifications and Reminders:
    - Test the functionality of notifications and reminders for tasks, deadlines, and overdue invoices.
    - Verify that users receive timely notifications via email or in-app notifications.

9. Security and Privacy:
    - Test user authentication and authorization mechanisms to ensure secure access to the app.
    - Verify that user data is stored securely and protected against unauthorized access.
    - Test data encryption during transmission and storage.

10. Usability and User Experience:
    - Verify that the app's user interface is intuitive, responsive, and easy to navigate.
    - Test the app's compatibility with different devices, browsers, and screen sizes.
    - Conduct user acceptance testing (UAT) to gather feedback on usability and overall user experience.

# V. Project Issues

## 14. Tasks

### 14a. Project Planning

Project planning for Tempus involves defining the project scope, setting goals, determining tasks and milestones, allocating resources, and establishing a timeline. Here are the key steps in the project planning process:

1. Define Project Scope:
   - Clearly define the objectives, features, and functionalities of the app.
   - Identify the target audience (freelancers and small businesses) and their specific needs and requirements.
   - Determine the key deliverables and project constraints, such as budget and timeline.

2. Set Project Goals:
   - Establish measurable goals that align with the app's purpose, such as improving time tracking accuracy, streamlining invoicing processes, and enhancing user experience.
   - Define success criteria, such as increased productivity, reduced invoicing errors, or positive user feedback.

3. Identify Project Tasks:
   - Break down the project into manageable tasks and activities.
   - Identify the core functionalities, such as user registration, time tracking, invoicing, reporting, and integration with external systems.
   - Prioritize tasks based on their dependencies and importance to the overall project.

4. Create Work Breakdown Structure (WBS):

- Develop a hierarchical breakdown of project tasks, subtasks, and deliverables.
- Organize the WBS into logical phases or modules, such as UI design, backend development, testing, and deployment.

5. Determine Milestones and Deliverables:

- Define key milestones and intermediate deliverables to track progress.
- Milestones could include completing UI design, finishing backend development, conducting user acceptance testing, and launching the app.

6. Estimate Resources:

- Identify the resources required for each project task, including personnel, technology, tools, and infrastructure.
- Determine the roles and responsibilities of team members, such as developers, designers, testers, and project managers.
- Estimate the effort, duration, and dependencies for each task to allocate resources effectively.

7. Develop Project Schedule:

- Create a timeline or Gantt chart to visualize task dependencies, durations, and milestones.
- Assign start and end dates to each task, considering resource availability and dependencies.
- Identify critical paths and potential bottlenecks to optimize the schedule.

8. Risk Management:

- Identify potential risks and uncertainties that may impact the project's success.
- Develop strategies to mitigate risks, such as contingency plans, alternative approaches, or resource allocation adjustments.
- Continuously monitor and assess risks throughout the project lifecycle.

9. Communication and Collaboration:

- Establish clear communication channels and protocols to ensure effective collaboration among team members.
- Schedule regular meetings and status updates to review progress, address issues, and make informed decisions.
- Use project management tools or collaboration platforms to facilitate communication and document sharing.

10. Project Documentation:
- Create project documentation, including requirements specifications, design documents, test plans, and user manuals.
- Maintain a repository for project artifacts, ensuring version control and easy access for team members.

11. Quality Assurance and Testing:
- Define a testing strategy, including functional testing, usability testing, performance testing, and security testing.
- Develop a comprehensive test plan, test cases, and test data to validate the app's functionalities and ensure quality.
- Conduct regular reviews and inspections to identify and resolve any defects or issues.

12. Iterative Development and Agile Practices (Optional):
- Consider adopting an iterative development approach, such as Agile methodologies (e.g., Scrum, Kanban).
- Break the project into iterations or sprints, allowing for incremental development, feedback, and continuous improvement.

## 14a. Planning of the Development Phases

When using the Agile model for developing a time tracking and invoicing app for freelancers and small businesses, the development process typically consists of several iterative phases.

1. Project Initiation:

- Gather requirements and understand the needs of freelancers and small businesses.
- Identify the core features and functionalities of the app.
- Create a product backlog that prioritizes the development tasks.

2. Sprint 1: Minimum Viable Product (MVP)
    - Define the initial set of features that constitute the minimum viable product.
    - Break down the features into user stories with clear acceptance criteria.
    - Estimate the effort for each user story and select the stories to be developed in the first sprint.
    - Conduct sprint planning, where the development team commits to delivering the selected user stories within the sprint duration (usually 1-4 weeks).
    - Implement the selected user stories, focusing on delivering a functional and usable MVP.

3. Sprint Review and Feedback:
    - Conduct a sprint review meeting to demonstrate the completed user stories to stakeholders (e.g., product owner, users).
    - Gather feedback and incorporate any necessary changes or enhancements into the product backlog.
    - Use the feedback to prioritize and refine the backlog for future sprints.

4. Sprint 2 and Beyond:
    - Select user stories from the product backlog based on their priority and value.
    - Conduct sprint planning for each sprint, focusing on delivering incrementally improved versions of the app.
    - Implement, test, and integrate the selected user stories during each sprint.
    - Conduct regular meetings, such as daily stand-ups, to track progress, discuss challenges, and make adjustments.

5. Continuous Integration and Testing:
    - Emphasize continuous integration, ensuring that the developed features are regularly integrated and tested.

- Conduct unit testing, integration testing, and functional testing throughout the development process.
- Automate tests where possible to streamline the testing process and ensure the stability of the app.

6. Sprint Review and Retrospective:
   - Conduct a sprint review at the end of each sprint to review the completed user stories and gather feedback.
   - Reflect on the sprint process during the retrospective meeting and identify areas for improvement.
   - Adjust the development approach and backlog priorities based on the feedback and lessons learned.

7. Iterative Enhancements and Releases:
   - Continuously refine and enhance the app based on user feedback and changing requirements.
   - Prioritize user stories based on their value and impact, and develop them iteratively in subsequent sprints.
   - Plan and schedule regular releases to deploy new features and updates to users.