Technical Software Proposal

1. Project Overview:

Project Title: “**UrbanTide Fare Matrix”**

Project Summary: Complete with a point-to-point bus fare with digital or printed receipts for users to examine how much the bus fare would cost, a user-friendly bus ticketing matrix will enable commuters obtain a bus ticket with simplicity.

Project Scope:

* Easy way to buy a bus ticket fare.
* Receipt will only be online or printed (if there is a printer).
* Commuters will see the pick-up and drop-off stations.
* Generates tickets using html or pdf file.
* Audit trail.
* User management.
* Computes fare rate based on pick-up and drop-off.
* Dashboard.

1. Problem Statement:

The need for effective public transit networks is increased by urban populations, particularly in crowded places like Muntinlupa. Inaccurate computations, delays, and misunderstandings result from manual ticketing and fare calculations. Inefficiencies arise in the absence of an organized fare matrix system, which can lead to operator discontent and possible revenue loss. The passenger experience, operational effectiveness, and fare administration can all be enhanced by a user-friendly fare matrix.

1. Solution:

The proposed Bus Ticketing Fare Matrix named “UrbanTide Fare Matrix” will be guide for the commuters in purchasing bus fare tickets, help them to locate the exact pick-up and drop-off station of the bus point-to-point terminals. UrbanTide offers the best way to get a bus ticket, and it shows the population of each bus terminal around Muntinlupa.

1. Methodology & Approach:

Outline your development approach based on the principles of 12-Factor Apps:

1. Codebase

The project will be organized in a single repository, using a structured folder hierarchy:

* **Models/**: Contains the data models such as User, Document, and AuditTrail.
* **Services/**: Holds the business logic and service classes (e.g., UserService, DocumentService, EmailService).
* **Views/**: Includes the form classes and UI components (e.g., Form1, UserManagementForm).
* **Properties/**: Contains application settings and resources.
* **bin/ and obj/**: Standard output folders for compiled binaries.
* **Program.cs**: The main entry point of the application.

2. Dependencies

Dependencies will be managed using NuGet, which simplifies package management in C#. This will ensure that libraries like Entity Framework or any third-party libraries for email sending (e.g., SendGrid) are easily added, updated, or removed. The relevant dependencies will be listed in the packages.config file or specified in the .csproj file, depending on the project setup.

3. Config

Configuration settings, such as database connection strings and email service credentials, will be managed in the Database.cs and EmailService file. This allows sensitive information to be stored separately from the codebase.

4. Backing Services

External services like the database and email services will be connected through the service layer (e.g., DocumentService, UserService). The services will use the connection strings from the configuration file and appropriate APIs for email delivery.

5. Build, Release, Run

To ensure consistency in building and running the app, a build script (e.g., using MSBuild) will be included. Continuous Integration/Continuous Deployment (CI/CD) tools like Azure DevOps or GitHub Actions can automate building and releasing the application. The application will be run as a Windows Forms application after building.

6. Processes

The application’s processes will include a main UI thread for user interactions and asynchronous background tasks for operations like document uploads and email sending. For instance, the button click events (like btnUpload\_Click) will call asynchronous service methods to avoid blocking the UI.

7. Port Binding

While this is a Windows Forms application and not typically exposed over HTTP ports like web applications, any web APIs (if added later) could use ASP.NET Core for RESTful services, allowing binding to specific ports via configuration.

8. Concurrency

Concurrency will be handled using asynchronous programming (async/await) for I/O-bound operations. This will allow the application to scale well with multiple requests, especially during tasks like document upload/download, ensuring the UI remains responsive.

9. Disposability

The application will be designed to start quickly and shut down gracefully. On startup, the application will initialize necessary resources (like services and data context) quickly. On shutdown, event handlers will be used to release resources, like closing database connections or saving user session states.

10. Dev/Prod Parity

To maintain similarity between development and production environments, Docker can be utilized for containerization. Configuration files can be adjusted for different environments, ensuring the same version of dependencies and application code is used in both.

11. Logs

Logs will be stored in a database to provide an audit trail for user actions and system events, allowing easier debugging and monitoring.

12. Admin Processes

Administrative tasks like database migrations, user management, and email notifications will be handled through scripts or administrative forms within the application. Additional administrative functions could include periodic cleanup of old documents or user sessions.

1. Project Requirements:

\*\*Technical Requirements:\*\* List any tools, programming languages, frameworks, and software dependencies.

\*\*Hardware Requirements:\*\* Specify if any specific hardware is needed for development or testing.

1. Milestones & Timeline:

\*\*Milestones:\*\* List the key phases of the project (e.g., Planning, Development, Testing, Deployment).

\*\*Timeline:\*\* Provide a timeline with expected completion dates for each milestone.

1. Deliverables:

- Working software application

- User documentation

- Source code and version control repository

1. Evaluation & Metrics:

The application’s success will be measured based on whether it is able to satisfy the scope of the proposal. The application must be able to: Perform basic uploading of pdf, doc, ppt, csv, and xlsx documents per user, Documents can be seen by their own users, Users can upload and download their own documents, must feature Audit trail, Manage the users, Users could change their password, Provide the user a solution if they forgot their password, Admin can lock users, and Users can register, the default role being ‘standard’.

Numerous trials and quality assurance will be performed to ensure the reliability of the application’s features and functions.

1. Team & Roles:

Team Members:

* Cabalo, Francis Xavier: Documentation
* Coja, Dennis Mark: Technical leader
* Cuartero, Adrian Benedict: User Interface
* Gonio, Nicolette Sairylle: Backend Developer

Advisors:

* Professor Nino Francisco Alamo

1. Conclusion:

The project’s purpose is to provide a bus ticketing matrix that includes a point-to-point bus fee and digital or printed receipts that allow customers to review the fare amount, commuters will be able to purchase a bus ticket with ease. This is the project's goal in addressing the problem of bus fare and ticketing issues. On the dashboard of the system, every bus station is shown in real time.