Smart City Management System

CS346: Software Engineering Lab

Documentation

Authors:

Group 2B

Instructor:

Prof. Pradip K. Das, Dept. of CSE, IITG

Problem Statement

The objective is to develop a comprehensive smart city management system that revolutionizes traditional offline processes by digitalizing them, thereby enhancing the quality of life for residents. The envisioned app will seamlessly integrate various services, spanning from healthcare management, educational resources, and employment opportunities to travel arrangements, banking facilities, and community engagement activities.

The primary goal is to provide a user-centric platform that caters exhaustively to the daily requirements of the general populace. This entails digitizing essential services such as doctor appointments, course enrollments, job searches, and civic engagements like elections. Additionally, the app will streamline mundane tasks such as banking transactions, travel bookings, and access to newsletters and event updates, ensuring convenience and efficiency for users.

In essence, the app aims to be a one-stop solution for residents, offering a holistic digital ecosystem that simplifies and enhances their everyday lives.

SDLC Model followed

For creating this application we followed the **Incremental model** .It is a software development life cycle approach where the development process is divided into smaller, incremental portions or iterations. Each iteration passes through the phases of requirements gathering, design, implementation, testing, and deployment.

In this model, the software is developed and delivered incrementally, with each increment adding new functionality to the system. Each iteration builds upon the previous one, allowing for continuous feedback from stakeholders and users. This iterative approach enables flexibility and adaptability, as changes can be incorporated at each stage of development.

All the iterations in our model were done in parallel and there were 5 teams that were doing this.

Module 1 User Documentation

User Registration and Management Module

Introduction

The User Registration and Management Portal is the opening module of the Smart City Management Platform, designed to facilitate the registration and management of users accessing the platform's features and services. This report provides an overview of the functionalities, design and security features of the User Registration and Management Portal.

This report will include the design and functionality of the main features of the portal majorly including Login Page , Registration Page , Forgot Password Page, Profile Page and Edit Profile Page, all of which will collectively help in seamless User management.

Login Page

The login page serves as the entry point for users to access the Smart City Management Platform, providing a secure and user-friendly interface for authentication. This includes several key features, such as:

- Username and Password Fields: Users are required to enter their registered email address or SID and password to log in to the platform securely.
- Forgot Password Link: A "Forgot Password" link is provided for users to initiate the password reset process in case they forget their login credentials. This feature typically involves sending a password reset link to the user's registered email address.

- User Registration Link: A link is present on the page so that new users can register themselves on the portal. Clicking on the link will redirect the user onto a registration page.

Registration Page

The registration page serves as the gateway for new users to join the Smart City Management Platform, enabling them to create accounts and access platform features and services. The features of this page include :

- Input Fields: Users are prompted to provide essential information such as name, email address, contact details, designation, DOB , Gender , Profile Picture and desired password to create their accounts.
- Password Strength Indicator: An interactive password strength indicator assists users in creating secure passwords by providing real-time feedback on password complexity and strength based on established criteria such as length, complexity, and character diversity.
- Password Strength Check
  - The system performs a password strength check to ensure that the new password meets security requirements.
  - Criteria for a strong password include:
    - Minimum length of 8 characters.
    - At least one digit (0-9).
    - At least one uppercase letter.
    - At least one lowercase letter.
    - At least one special character.

Upon registering , Each user will be assigned a unique SID.

Profile Page

The Main Profile Page is a personalized dashboard that allows users to interact with various aspects of the User Management Portal, providing a comprehensive overview of their personal and account information. It serves as a central hub for user activities and settings.
Key Features:

- Profile Overview: Displays the user's profile picture and personal details such as name, DOB, gender, and designation, making the page more personalized and engaging.
- Navigation Bar: A dynamic navbar is included, offering quick access to different sections of the portal, including Events, Transport, Health and any additional features relevant to the user experience.
- Search Functionality: Incorporates a search bar that enables users to search for other users by name or mobile number, fostering connectivity and social interaction within the platform.
- Notifications Area: A dedicated section for notifications alerts users to new messages, approval requests, or any other personalized alerts, ensuring they stay informed about important activities or updates. These notifications delete themselves after the user has seen them.

Edit Profile Page

The Edit Profile Page allows users to update their personal and account information, ensuring their profile remains up-to-date and secure. This page is crucial for maintaining user engagement and personalization.
Key Features:

- Editable Personal Information: Users can update their personal details such as name, email address, contact details, DOB, gender, and profile picture, facilitating easy customization of their profile.
- Change Password Section: A dedicated section for changing the password includes the current password field, new password field, and new password confirmation field, along with a password strength indicator to encourage the creation of secure passwords.
- Save Changes Button: Allows users to save any changes made to their profile or account settings, with a confirmation prompt to ensure no accidental changes are made.
- Cancel Changes Option: Gives users the ability to cancel any modifications, returning them to their main profile page without saving any changes, preserving the original information.

Forgot Password Page

The Forgot Password Page offers a vital function for users who have forgotten their login credentials, providing a secure and straightforward method to reset their password and regain access to their accounts.
Key Features:

- User Identification: Prompts users to enter their registered email address to verify their identity and initiate the password reset process.
- Verification Process: The system sends a password OTP to the userâs registered email, ensuring secure verification.
- Reset Password Functionality: Once verified, users are directed to a secure page where they can set a new password.

Password Change Page

This page is for existing users to change their password. users are required to enter their old password, new password, and confirm the new password. The system verifies the old password before allowing the change. If the old password is correct and the new password meets the criteria, it is updated in the database. If the old password is correct and the new password does not meets the criteria , a warning is shown to the user that your password is weak , do you still want to change it , if clicked yes . it is updated in the database. Upon successful password change, users are notified, and they can continue using their accounts with the new password.

Festival and Event Planning Module

Introduction

The Festival and Event Planning Module serves as a digital platform designed to streamline the entire event life cycle, from initial conception to post-event analysis and and managing events efficiently. This report will give you an overview of its main parts: the main menu, where you can see all events; the add event page, where you create new events; the edit event page where you can edit events; and the event display, where you get more details about specific events.

This module is like a digital helper that guides you through the whole event process, from planning to analyzing what happened after the event. It's made to fit the needs of everyone involved in events: organizers, guests, vendors, and authorities. Our aim is to create a user-friendly platform where you can easily organize events, manage vendors, help guests register, and optimize Resource Allocation processes.

This report will delve into the key components of the Festival and Event Planning Module, providing insights into the design, development, and functionality of each submodule.

Main Menu Page

The main menu page serves as the central hub for accessing information about ongoing and upcoming events. It displays cards containing essential details of each event, such as event name, date, and venue. Users can click on a card to access more detailed information about the event or to initiate specific actions related to it, such as registering or offering services.

Add Event Page

The add event page enables users to create new events by filling out a comprehensive form. This form includes fields for essential event details, such as:

- Event Name: A name for the event.
- Event Description: A brief overview of the event.
- Cover Image: Visual representation of the event.
- Owner Name: Identification of the event organizer. (Automatically generated)
- Event Type: Categorization of the event for administrative purposes.
- Venue: Location where the event will take place.
- Date and Time: Schedule of the event.
- Restrictions: Restriction on who can participate in event like Age, Number of participants etc.

Moreover, the card page allows event owners to input additional resources, such as land and timing, which are later utilized in algorithms for optimization purposes.Upon submission, the event details are sent for approval to the respective minister, ensuring proper oversight and authorization before the event is finalized. Additionally, users have the option to add the event to a newsletter, enhancing its visibility and outreach.

Event Display Page

The card page provides detailed information about a specific event upon selection from the main menu. Users can view essential event details and take appropriate actions based on their role:

- For Vendors: Vendors can offer their services for the event, facilitating collaboration and partnership opportunities.Tags represent the services that the owners want to offer to vendors. A vendor can request the owner to assign them multiple tags.
- For Attendees: Options are available to register for the event, enabling seamless participation.
- For Event Owners: Event owners are equipped with buttons to approve vendor requests and manage attendee registrations, facilitating efficient event administration. These buttons trigger new forms that elucidate various functions, offering thorough insights into available actions and enhancing the event management process. The vendor approval page displays a list of vendors

seeking participation, with options to accept or reject. It also enables the addition of tags specifying required services for the event. In handling registrations, event owners have the flexibility to control event registration settings, including opening or closing registration and setting restrictions on the maximum number of attendees and age limits.

- For Minsters: The concerned minister has an approval button with which he can approve or disapprove certain event.

Edit Event Page

Here the owner can edit all the information about the event including Date , time , Restriction etc. and save the new information.

Technical Documentation

User Module

1. System Architecture

The backend code utilizes a simple three-tier architecture:

- Presentation Layer: Implemented using VB.NET and Windows Forms for the user interface.
- Business Logic Layer: Contains the backend code responsible for handling user interactions and business logic.
- Data Access Layer: Interacts with the MySQL database to perform CRUD (Create, Read, Update, Delete) operations.

2. Database Schema

The backend code interacts with a MySQL database named `smart_city_management`. The relevant table for this page is `User`, which contains fields such as `SID`, `EmailAddress`, and `PasswordHash`.

Database Dictionary

The following is the database schema of the module:

1 User

a User ID / SID:

- Type: Integer
- Constraints: Auto-increment, Not Null, Primary Key
- Description: Unique identifier for each user.

b Name:

- Type: String
- Constraints: Not Null
- Description: The full name of the user.

c Email Address:

- Type: String

- Constraints: Not Null, Unique
- Description: The user’s email address.

d Phone Number:

- Type: String
- Constraints: Not Null, Unique
- Description: The user’s phone number.

e Designation:

- Type: String
- Constraints: Not Null
- Description: The user’s job title or role.

f Gender:

- Type: String
- Constraints: Null
- Description: The user’s gender.

g DOB:

- Type: Date
- Constraints: Null
- Description: Date of Birth of the user.

h Profile Pic:

- Type: Blob
- Constraints: Null
- Description: User’s profile picture.

i Password Hash:

- Type: String
- Constraints: Not Null

- Description: A hashed version of the user’s password.

2 Login

a User ID/SID or Email:

- Type: Integer
- Constraints: Not Null, Foreign Key
- Description: Identifier used to log in.

b Password Hash:

- Type: String
- Constraints: Not Null
- Description: The hashed password for login verification.

3 Notifications

a Notification ID:

- Type: Integer
- Constraints: Auto-increment, Not Null, Primary Key
- Description: Unique identifier for each notification.

b Notification Recipient SID:

- Type: Integer
- Constraints: Not Null, Foreign Key
- Description: Identifier of the user who receives the notification.

c Type:

- Type: Integer
- Constraints: Not Null
- Description: Displays the type of notification and the corresponding page to be redirected to.

d Notification Content:

- Type: String

- Constraints: Not Null
- Description: A string displaying the content of notification.

User_Login Class

This class represents the user login functionality in the application.

Dependencies

- System.Text: Standard library for handling string operations.
- MySql.Data.MySqlClient: MySQL database connectivity library.
- System.Security.Cryptography: Library for cryptographic functions.

Attributes

- connString: String variable storing the connection string to the MySQL database.
- conn: MySqlConnection object for establishing a connection to the MySQL database.

Methods

HashPassword(password As String) As String

- Description: Hashes the provided password using the SHA256 hashing algorithm.
- Parameters:
- password (String): The password to be hashed.
- Returns:
- String: The hashed password.

Event Handlers

- LoginForm_Load: Handles the form load event.
- EmailTextBox_Enter, EmailTextBox_Leave, PasswordTextBox_Enter, PasswordTextBox_Leave: Handle events related to email and password textboxes, including focus and text manipulation.
- TogglePasswordButton_Click: Handles the event for toggling password visibility.
- LoginButton_Click: Handles the event when the login button is clicked. Performs user authentication and navigation to the main panel upon successful login.
- SignUpLabel_Click, Label2_Click: Handle events for navigating to the sign-up form and the forgot password form respectively.

User_SignUpForm Page

This class represents the user sign-up functionality in the application.

Dependencies

- `MySql.Data.MySqlClient`: MySQL database connectivity library.
- `System.IO`: Standard library for handling file input/output operations.
- `System.Text`: Standard library for handling string operations.
- `System.Security.Cryptography`: Library for cryptographic functions.

Attributes

- `connString`: String variable storing the connection string to the MySQL database.
- `conn`: MySqlConnection object for establishing a connection to the MySQL database.
- `selectedImagePath`: String variable storing the path of the selected profile picture.

Methods

`GenerateUniqueSID() As Integer`

- Description: Generates a unique 6-digit SID (System Identifier) for a new user.
- Returns:
  - `Integer`: Unique SID.

`HashPassword(password As String) As String`

- Description: Hashes the provided password using the SHA256 hashing algorithm.
- Parameters:
  - `password` (String): The password to be hashed.
- Returns:
  - `String`: The hashed password.

Password Strength Check Functions

- `ContainsUpperCase(inputString As String) As Boolean`: Checks if the input string contains at least one uppercase letter.
- `ContainsLowerCase(inputString As String) As Boolean`: Checks if the input string contains at least one lowercase letter.
- `ContainsDigit(inputString As String) As Boolean`: Checks if the input string contains at least one digit.
- `ContainsSpecialCharacter(inputString As String) As Boolean`: Checks if the input string contains at least one special character.
- `PasswordStrengthCheck(InputString As String) As Integer`: Checks the strength of the provided password based on length, presence of uppercase and lowercase letters, digits, and special characters.

Event Handlers

- `SignUpForm_Load`: Handles the form load event. Populates the gender ComboBox and sets initial visibility of StrengthCheckLabel.

- `PasswordTextBox_TextChanged`: Handles the event when the password text changes. Makes the strength label visible if the password is not empty.
- `CheckStrengthClick`: Handles the event when the strength label is clicked. Invokes the password strength check function.
- `SignUpButton_Click`: Handles the event when the sign-up button is clicked. Performs user registration and inserts the user data into the database.
- `LoginLabel_Click`: Handles the event when the login label is clicked. Navigates to the login form.
- `Button1_Click`: Handles the event when the "Upload Picture" button is clicked. Opens a file dialog to select a profile picture.

Note

- The `SignUpButton_Click` method performs user registration by validating the input data, checking for existing email and contact number in the database, generating a unique SID, hashing the password, and inserting the user data into the database.
- The password strength is checked during sign-up, and a message box is displayed to indicate the strength level.

Profile Page

Dependencies

- **MySql.Data.MySqlClient:** This library is used to interact with the MySQL database server.
- **System.Text:** This library is used for handling text-related operations.

Class Members

Properties

- **connString (String):** A string variable that holds the connection string used to connect to the MySQL database.
- **conn (MySqlConnection):** An instance of the `MySqlConnection` class used to establish a connection to the MySQL database.
- **userDetailsTable (DataTable):** A DataTable object used to store user details retrieved from the database.
- **notifications (List(Of (Integer, String))):** A list used to store notification data in the form of tuples containing notification IDs and messages.

Methods

- **ChildForm(Form childform):** This method accepts a child form and adds it to the `mypanel.panel1` control.
- **ChildForm2(Panel parentpanel, Form childform):** This method accepts a parent panel and a child form, clears the parent panel, and adds the child form to it.
- **User_Profile_Load(sender As Object, e As EventArgs):** This method is an event handler that loads user profile information when the form is loaded.
- **User_Profile_Click(sender As Object, e As EventArgs):** This method is an event handler that hides the search box results when the form is clicked.

- **PictureBox2_Paint(sender As Object, e As PaintEventArgs):** This method is an event handler that creates an elliptical region for the profile picture.
- **LoadNotifications():** This method fetches notifications for the current user from the database and displays them.
- **GetUserDetails(userID As Integer):** This method retrieves user details from the database based on the provided user ID.
- **GetUserDetailsByPartialName(partialName As String):** This method retrieves user details from the database based on a partial name match.
- **SearchTextBox_GotFocus(sender As Object, e As EventArgs):** This method is an event handler that displays the search results when the search box gains focus.
- **SearchTextBox_TextChanged(sender As Object, e As EventArgs):** This method is an event handler that updates the search results based on the text entered in the search box.
- **ListBox1_MouseClick(sender As Object, e As MouseEventArgs):** This method is an event handler that displays user details when a user is selected from the search results.
- **AddNotificationToPanel(ID As Integer, userID As Integer, type As Integer, message As String):** This method adds a notification to the notification panel.
- **NotificationLabel_Click(sender As Object, e As EventArgs):** This method is an event handler that handles the click event for notification labels.
- **DeleteNotification(ID As Integer, type As Integer, message As String):** This method deletes a notification from the database.
- **User_Profile_Shown(sender As Object, e As EventArgs):** This method is an event handler that loads notifications when the form is shown.
- **EditProfileButton_Click(sender As Object, e As EventArgs):** This method is an event handler that opens the user profile editing form.
- **ChangePasswordButton_Click(sender As Object, e As EventArgs):** This method is an event handler that opens the change password form.
- **LogoutButton_Click(sender As Object, e As EventArgs):** This method is an event handler that logs the user out and returns to the login form.
- **AdminButton_Click(sender As Object, e As EventArgs):** This method is an event handler that opens the user administration form.

Admin Page

Dependencies

- **MySql.Data.MySqlClient:** This library is used to interact with the MySQL database server.
- **System.Text:** This library is used for handling text-related operations.
- **System.Security.Cryptography:** This library is used for hashing passwords using the SHA256 algorithm.

Class Members

Properties

- **connString (String):** A string variable that holds the connection string used to connect to the MySQL database.

- **connection (MySqlConnection):** An instance of the `MySqlConnection` class used to establish a connection to the MySQL database.

Methods

- **User_Admin_Load(sender As Object, e As EventArgs):** This method is an event handler that loads user data when the form is loaded.
- **LoadUsersData():** This method executes a SQL query to select all users from the database and binds the results to a DataGridView for display.
- **EmailExistsInDatabase(EmailAddress As String) As Boolean:** This method checks if the specified email address exists in the database.
- **GetDetailsButton_Click(sender As Object, e As EventArgs):** This method retrieves user details from the selected row in the DataGridView and displays them in textboxes.
- **ClearButton_Click(sender As Object, e As EventArgs):** This method clears the input textboxes.
- **HashPassword(password As String) As String:** This method hashes the input password using the SHA256 algorithm.
- **AddUserButton_Click(sender As Object, e As EventArgs):** This method adds a new user to the database using the information provided in the input textboxes.
- **EditUserButton_Click(sender As Object, e As EventArgs):** This method updates an existing user's information in the database based on the email address provided in the input textboxes.
- **DeleteUserButton_Click(sender As Object, e As EventArgs):** This method deletes a user from the database based on the email address provided in the input textboxes.
- **Label_Click(sender As Object, e As EventArgs):** This method navigates to the user profile form when a label is clicked.

Change Password Page

1. Dependencies

The backend code relies on the following dependencies:

- `MySql.Data.MySqlClient`: MySQL database connectivity library.
- `System.Text`: Standard library for handling string operations.
- `System.Security.Cryptography`: Library for cryptographic functions.
- `System.Windows.Forms.VisualStyles.VisualStyleElement`: Library for UI elements in Windows Forms.

2. Backend Functionality

GetUserDetails Function

- Retrieves the email address associated with a given user ID (`SID`) from the database.

HashPassword Function

- Utilizes SHA256 hashing algorithm to securely hash passwords before storing them in the database.

Password Strength Check Functions

- Contains functions to verify password strength based on criteria such as length, presence of digits, uppercase and lowercase letters, and special characters.

Button1_Click Event Handler

- Handles the logic for changing the user's password.
- Verifies input fields, checks old password validity, and updates the password in the database if all conditions are met.

CheckBox1_CheckedChanged Event Handler

- Toggles password visibility in the UI based on the state of the checkbox.

Button3_Click Event Handler

- Handles navigation back to the user profile page upon clicking the “Back to Profile ”button.

Reset Password Page

1. Dependencies

The backend code relies on the following dependencies:

- `MySql.Data.MySqlClient`: MySQL database connectivity library.
- `System.IO`: Standard library for handling file input/output operations.
- `System.Text`: Standard library for handling string operations.
- `System.Security.Cryptography`: Library for cryptographic functions.
- `System.Windows.Forms.VisualStyles.VisualStyleElement`: Library for UI elements in Windows Forms.

2. Backend Functionality

HashPassword Function

- Utilizes SHA256 hashing algorithm to securely hash passwords before storing them in the database.

Password Strength Check Functions

- Contains functions to verify password strength based on criteria such as length, presence of digits, uppercase and lowercase letters, and special characters.

Button1_Click Event Handler

- Handles the logic for changing the user's password.
- Verifies input fields, checks password validity, and updates the password in the database if all conditions are met.

CheckBox1_CheckedChanged Event Handler

- Toggles password visibility in the UI based on the state of the checkbox.

Button3_Click Event Handler

- Handles navigation back to the login page upon clicking the “Back to Login” button.

Forgot Password Page

1. Dependencies

The backend code relies on the following dependencies:

- `MySql.Data.MySqlClient`: MySQL database connectivity library.
- `System.IO`: Standard library for handling file input/output operations.
- `System.Text`: Standard library for handling string operations.
- `System.Net`: Standard library for network-related operations.
- `System.Net.Mail`: Library for sending emails.
- `System.Windows.Forms`: Library for Windows Forms user interface components.

2. Backend Functionality

IsEmailPresent Function

- Checks if the provided email address is registered in the system's database.

Timer_Tick Event Handler

- Tracks the elapsed time for code verification and manages code expiration.

Button1_Click Event Handler

- Sends a verification code to the user's email address.
- Handles error conditions such as invalid email address or failed email sending.

Button2_Click Event Handler

- Validates the entered verification code.
- Redirects the user to the password reset page upon successful code verification.

ResendTimer_Tick Event Handler

- Manages the countdown timer for code resend functionality.

ResendLabel_Click Event Handler

- Initiates the process of resending the verification code.
- Implements logic to limit code resending to once per session.

Button3_Click Event Handler

- Redirects the user back to the login page upon clicking the “Back to Login” button.

Festival Module

Main menu page

Dependencies

- **CityManagement.FestivalEvents_Cards:** This namespace contains the `FestivalEvents_Cards` class, which is used to generate event cards for display.
- **Microsoft.VisualBasic.ApplicationServices:** This library is used for accessing application-level services.
- **MySql.Data.MySqlClient:** This library is used to interact with the MySQL database server.

Class Members

Properties

- **connString (String):** A string variable that holds the connection string used to connect to the MySQL database.
- **conn (MySqlConnection):** An instance of the `MySqlConnection` class used to establish a connection to the MySQL database.

Methods

- **Button1_Click(sender As Object, e As EventArgs):** This method is an event handler that clears the current panel controls and displays the form for adding a new festival event.
- **PictureBox1_Click(sender As Object, e As EventArgs):** This method is an event handler that currently does nothing. It might be intended for future functionality.
- **FestivalEvents_MainMenu_Load(sender As Object, e As EventArgs):** This method is an event handler that executes when the main menu form is loaded. It sets the form to full screen, hides the title bar, and fetches festival event data from the database to display on the form.

New Event Page

Dependencies

- **System.Diagnostics.Eventing.Reader:** This namespace provides classes for reading event logs.
- **System.IO:** This namespace provides classes for performing input and output operations, such as reading files.
- **MySql.Data.MySqlClient:** This library is used to interact with the MySQL database server.

Class Members

Properties

- **connString (String):** A string variable that holds the connection string used to connect to the MySQL database.
- **conn (MySqlConnection):** An instance of the `MySqlConnection` class used to establish a connection to the MySQL database.
- **selectedImagePath (String):** A string variable that stores the file path of the selected image for the event.

Methods

- **GenerateUniqueEventID():** This private method generates a unique event ID for the new event by checking if the randomly generated ID already exists in the database. If it exists, a new ID is generated recursively until a unique one is found.
- **Label1_Click(sender As Object, e As EventArgs):** This method is an event handler that clears the current panel controls and displays the main menu for festival events.
- **Label8_Click(sender As Object, e As EventArgs):** This method is an event handler that currently does nothing. It might be intended for future functionality.
- **Button2_Click(sender As Object, e As EventArgs):** This method is an event handler that executes when the user clicks the button to register a new event. It validates the input data, inserts the new event into the database, and displays a success message. If an error occurs, an appropriate error message is displayed.
- **Button1_Click(sender As Object, e As EventArgs):** This method is an event handler that executes when the user clicks the button to upload an image for the event. It opens a file dialog for the user to select an image file, loads the selected image into an Image control, and stores the file path for later use.

Event Details

Dependencies

- **Microsoft.VisualBasic.ApplicationServices:** This namespace provides classes and interfaces that support the Visual Basic Application Model.
- **MySql.Data.MySqlClient:** This library is used to interact with the MySQL database server.

Class Members

Properties

- **connString (String):** A string variable that holds the connection string used to connect to the MySQL database.
- **conn (MySqlConnection):** An instance of the `MySqlConnection` class used to establish a connection to the MySQL database.

Methods

- **FestivalEvents_EventDetails_Load(sender As Object, e As EventArgs):** This method is an event handler that executes when the form loads. It retrieves event details from the database, adjusts the visibility of buttons based on user designation and ownership status, and displays the event information on the form.
- **Button2_Click(sender As Object, e As EventArgs):** This method is an event handler that executes when the user clicks the button to choose services for the event. It clears the current panel controls and displays the form for choosing services.
- **Label1_Click(sender As Object, e As EventArgs):** This method is an event handler that executes when the user clicks the label to return to the main menu for festival events. It clears the current panel controls and displays the main menu form.
- **Button1_Click_1(sender As Object, e As EventArgs):** This method is an event handler that executes when the user clicks the button to view registration restrictions for the event. It clears the current panel controls and displays the form for registration restrictions.
- **Button4_Click(sender As Object, e As EventArgs):** This method is an event handler that executes when the user clicks the button to approve the event (for municipal officers). It clears the current panel controls and displays the approval form.
- **Button3_Click(sender As Object, e As EventArgs):** This method is an event handler that executes when the user clicks the button to edit the event (for event owners). It clears the current panel controls and displays the event editing form.
- **Button5_Click(sender As Object, e As EventArgs):** This method is an event handler that executes when the user clicks the button to request services for the event (for merchants). It clears the current panel controls and displays the service request form.
- **Button6_Click(sender As Object, e As EventArgs):** This method is an event handler that executes when the user clicks the button to register for participation in the event. It clears the current panel controls and displays the participation registration form.

Edit Event

Dependencies

- **System.Drawing.Text:** Provides access to installed and private fonts on the system.
- **System.Windows.Forms.VisualStyles.VisualStyleElement:** Represents a visual style element for use with visual styles on Windows operating systems.
- **System.Xml:** Provides classes for processing XML data.
- **MySql.Data.MySqlClient:** Enables interaction with a MySQL database server.
- **Org.BouncyCastle.Asn1.IsisMtt.X509:** Contains classes for handling X.509 certificates and related functionality.

Class Members

Properties

- **connString (String):** A string variable that holds the connection string used to connect to the MySQL database.
- **conn (MySqlConnection):** An instance of the `MySqlConnection` class used to establish a connection to the MySQL database.
- **EventId (Integer):** The ID of the event being edited, retrieved from a global variable.
- **UserSID (Integer):** The ID of the current user, retrieved from a global variable.

Methods

- **Edit_Event_Load(sender As Object, e As EventArgs):** This method is an event handler that executes when the form loads. It fetches event details from the database using the `GetEventDetails` function and populates the form fields with the retrieved information.
- **GetEventDetails(EventID As Integer):** This private method retrieves festival details from the database using the provided `EventID`. It returns a dictionary containing event details.
- **Button1_Click(sender As Object, e As EventArgs):** This method is an event handler that executes when the user clicks the button to upload a picture. It opens a file dialog for selecting an image and displays the selected image in the `PictureBox` control.
- **Button2_Click(sender As Object, e As EventArgs):** This method is an event handler that executes when the user clicks the button to save changes. It updates the event details in the database based on the values entered in the form fields.

Choose Vendor Page

Dependencies

- **System.Windows.Forms:** Provides classes for creating Windows-based applications.
- **CityManagement.FestivalEvents_Offer_Service2:** Likely another class or module within the project.
- **MySql.Data.MySqlClient:** Enables interaction with a MySQL database server.

Class Members

Properties

- **selectedVendorButton (Button):** Tracks the currently selected vendor button.
- **vendorServiceTags (List(Of VendorServiceTag)):** A list of `VendorServiceTag` objects representing vendor service tags.
- **currentIndex (Integer):** The index indicating the current vendor being selected.
- **connString (String):** A string variable that holds the connection string used to connect to the MySQL database.
- **conn (MySqlConnection):** An instance of the `MySqlConnection` class used to establish a connection to the MySQL database.

Constructor

- **New(ByVal serviceTags As List(Of VendorServiceTag), ByVal index As Integer):** Initializes a new instance of the `FestivalEvents_ChooseVendor` class with the provided list of `VendorServiceTag` objects and the current index.

Methods

- **FestivalEvents_ChooseVendor_Load(sender As Object, e As EventArgs):** This method is an event handler that executes when the form loads. It fetches vendor details from the database and populates the form with vendor information, allowing the user to select a vendor.
- **GetVendorName(ByVal SID As Integer) As String:** Retrieves the name of a vendor based on their SID (System ID). It queries the database to fetch the vendor's name using the SID.
- **SelectButton_Click(sender As Object, e As EventArgs):** This method is an event handler that executes when the user clicks on a vendor selection button. It updates the selected vendor information and closes the form.

Choose Services page

Dependencies

- **CityManagement.FestivalEvents_Offer_Service2:** Likely another class or module within the project.
- **MySql.Data.MySqlClient:** Enables interaction with a MySQL database server.

Class Members

Properties

- **connString (String):** A string variable that holds the connection string used to connect to the MySQL database.

- **conn (MySqlConnection):** An instance of the `MySqlConnection` class used to establish a connection to the MySQL database.
- **vendorServiceTags (List(Of VendorServiceTag)):** A list of `VendorServiceTag` objects representing vendor service tags.
- **currentIndex (Integer):** The index indicating the currently selected service.
- **selectedVendorButton (Button):** Tracks the currently selected vendor button.

Methods

- **CheckBox_CheckedChanged(sender As Object, e As EventArgs):** This method is an event handler that executes when any of the checkboxes for service selection change. It toggles the visibility of associated UI elements based on checkbox state and updates the required service status.
- **ToggleVisibility(visible As Boolean, ParamArray controls As Control()):** Sets the visibility of multiple controls based on the provided boolean value.
- **ToggleIsRequired(isRequired As Boolean, index As Integer):** Updates the required status of a service based on the provided boolean value and index.
- **FestivalEvents_ChooseServices_Load(sender As Object, e As EventArgs):** This method is an event handler that executes when the form loads. It initializes the form's UI and fetches service-related information from the database.
- **updatePage(vendorServiceTags As List(Of VendorServiceTag)):** Updates the UI elements based on the provided list of `VendorServiceTag` objects.
- **GetVendorName(ByVal SID As Integer) As String:** Retrieves the name of a vendor based on their SID (System ID) from the database.
- **Button_Click(sender As Object, e As EventArgs):** This method is an event handler that executes when any of the service selection buttons are clicked. It updates the UI to display available vendors for the selected service.
- **SelectButton_Click(sender As Object, e As EventArgs, currentIndex As Integer):** This method is an event handler that executes when a vendor selection button is clicked. It updates the selected vendor information and updates the UI accordingly.
- **Button5_Click(sender As Object, e As EventArgs):** This method is an event handler that executes when the "Save Changes" button is clicked. It serializes the updated service tags and updates the database with the changes.
- **Button6_Click(sender As Object, e As EventArgs):** This method is an event handler that executes when the "Back" button is clicked. It navigates back to the event details form.

Event Approval Page

Dependencies

- **MySql.Data.MySqlClient:** Enables interaction with a MySQL database server.

Class Members

Properties

- **connString (String):** A string variable that holds the connection string used to connect to the MySQL database.
- **conn (MySqlConnection):** An instance of the `MySqlConnection` class used to establish a connection to the MySQL database.
- **isApproved (Boolean):** A variable to track the approval status of the event.

Methods

- **Public Sub New():** Constructor method that initializes the form.
- **FestivalEvents_Approval_Load(sender As Object, e As EventArgs):** This method is an event handler that executes when the form loads. It retrieves the current approval status of the event from the database and updates the button appearance accordingly.
- **ToggleButton_Click(sender As Object, e As EventArgs):** This method is an event handler that executes when the toggle button is clicked. It toggles the approval status, updates the button appearance, and updates the database with the new approval status.

Offer Service Page

Dependencies

- **System.Drawing.Text:** Provides functionality for managing fonts and text.
- **MySql.Data.MySqlClient:** Enables interaction with a MySQL database server.
- **Newtonsoft.Json:** Handles JSON serialization and deserialization.

Class Members

Properties

- **connString (String):** Holds the connection string used to connect to the MySQL database.
- **conn (MySqlConnection):** An instance of the `MySqlConnection` class used for database communication.
- **EventId (Integer):** Represents the ID of the current festival event.
- **UserSID (Integer):** Represents the ID of the current user.

Nested Class

- **VendorServiceTag:** Defines a data structure to represent vendor service tags, including service name, requirement status, list of vendor IDs, and approved vendor ID.

Methods

- **Event_OfferService_Load(sender As Object, e As EventArgs):** Handles the form load event. It fetches festival details from the database and populates the UI controls with the retrieved data.

- **PopulateVendorServices(jsonString As String, userId As Integer):** Parses and populates vendor services based on JSON data retrieved from the database.
- **VendorServiceButton_Click(sender As Object, e As EventArgs):** Handles the click event of vendor service buttons. It toggles the user's request status for a service and updates the database accordingly.
- **SerializeVendorServiceTags(vendorServiceTags As List(Of VendorServiceTag)) As String:** Serializes the list of vendor service tags to JSON format.
- **GetEventDetails(EventID As Integer) As Dictionary(Of String, Object):** Retrieves festival details from the database based on the provided event ID.
- **ApplyFontToControl(control As Control, font As Font):** Applies a specific font to a control and its child controls recursively.

Event Participation page

Dependencies

- **MySql.Data.MySqlClient:** Enables interaction with a MySQL database server.
- **Newtonsoft.Json:** Handles JSON serialization and deserialization.

Class Members

Properties

- **connString (String):** Holds the connection string used to connect to the MySQL database.
- **conn (MySqlConnection):** An instance of the `MySqlConnection` class used for database communication.
- **EventId (Integer):** Represents the ID of the current festival event.
- **UserSID (Integer):** Represents the ID of the current user.

Nested Class

- **Restrictions:** Defines a data structure to represent event restrictions, including minimum age, maximum age, and maximum number of participants.

Methods

- **Event_Participation_Load(sender As Object, e As EventArgs):** Handles the form load event. It fetches event details from the database and populates the UI controls with the retrieved data, including event name, venue, description, date, time, cover image, and restrictions. It also checks the user's registration status for the event.
- **Button1_Click(sender As Object, e As EventArgs):** Handles the click event of the registration button. It registers or unregisters the user for the event based on the current registration status, event openness, and user age restrictions.

- **GetEventDetails(EventID As Integer) As Dictionary(Of String, Object):** Retrieves festival details from the database based on the provided event ID, including name, description, venue, date, time, cover image, and restrictions.
- **ApplyFontToControl(control As Control, font As Font):** Applies a specific font to a control and its child controls recursively.

Registration restrictions page

Dependencies

- **MySql.Data.MySqlClient:** Enables interaction with a MySQL database server.
- **Newtonsoft.Json.Linq:** Provides classes to parse and manipulate JSON data.

Class Members

Properties

- **connString (String):** Holds the connection string used to connect to the MySQL database.
- **conn (MySqlConnection):** An instance of the `MySqlConnection` class used for database communication.
- **isAcceptingReg (Boolean):** Represents the current registration status of the event (open or closed).

Methods

- **FestivalEvents_RegRestrictions_Load(sender As Object, e As EventArgs):** Handles the form load event. It fetches event details, including restrictions, from the database and populates the UI controls with the retrieved data. It also updates the UI based on the current registration status of the event.
- **ToggleButton_Click(sender As Object, e As EventArgs):** Toggles the registration status of the event between open and closed.
- **TextBoxMaxParticipants_KeyPress(sender As Object, e As KeyPressEventArgs):** Handles the key press event for the maximum participants text box. It allows only numeric input.
- **ButtonSave_Click(sender As Object, e As EventArgs):** Saves the updated registration restrictions to the database. It validates the input data, checks for consistency between minimum and maximum age limits, and updates the event's registration status.
- **ButtonBack_Click(sender As Object, e As EventArgs):** Navigates back to the event details page.
- **CheckBox1_CheckedChanged(sender As Object, e As EventArgs):** Handles the checked changed event for the minimum age checkbox. It enables or disables the minimum age track bar based on the checkbox state.
- **CheckBox2_CheckedChanged(sender As Object, e As EventArgs):** Handles the checked changed event for the maximum age checkbox. It enables or disables the maximum age track bar based on the checkbox state.

- **CheckBoxApplyMaxParticipants_CheckedChanged(sender As Object, e As EventArgs):** Handles the checked changed event for the maximum participants checkbox. It enables or disables the maximum participants text box based on the checkbox state.
- **TrackBarMinAge_Scroll(sender As Object, e As EventArgs):** Handles the scroll event for the minimum age track bar. It updates the minimum age label with the selected value.
- **TrackBarMaxAge_Scroll(sender As Object, e As EventArgs):** Handles the scroll event for the maximum age track bar. It updates the maximum age label with the selected value.

User Documentation

Employment Portal

Table of Contents

1. Introduction

The Employment Portal page allows users to get employed in their desired field of interest, by fulfilling the corresponding criteria.

2. Navigation

- The first page displays the list of various organisations to which one can apply.
- On clicking any option, the user is taken to the application form.
- The user can click on new hospital proposal to apply for opening a new hospital in the city.
- Additionally, on clicking the Regular Services option, the user is taken to the list of Regular Services page, from which he can choose his desired organisation.

3. Features

Filling Details in Application Form

- The application form asks for various inputs from the user depending on his chosen option.
- The user can even upload a profile picture for his occupation.
- The user can, and in some cases, should upload a required document also, as a proof for his credibility for getting employed.
- Ensure all information is accurate and complete before submitting; incomplete or inaccurate applications may result in delays or rejection.
- The user receives a notification if his employment request gets approved, after which his designation no longer remains ‘Not Employed’.

Filling Proposal form for Hospital Listing

- The application form asks for various inputs from the user.
- The information that the user is required to fill includes Owner's Name , Contact Details,Bank Account Details, Location , Description of the Proposal.
- Review and "Send Proposal": Ensure all information is accurate and complete before submitting; incomplete or inaccurate proposals may result in delays or rejection.
- The user receives a notification if his proposal request gets approved, after which he becomes Hospital Owner.

4. Troubleshooting

Error Handling

- If any errors occur during the application filling, appropriate messages are displayed.

Common Issues

- Ensure all required fields are filled before applying.
- If encountering persistent issues, contact the system administrator for assistance.

Services Portal

Table of Contents

1. Introduction

Welcome to our Service Portal where customers and service providers come together to streamline service bookings and communication. Our platform offers a one click booking option for booking services along with cancellation and rescheduling, along with a built-in chat page to facilitate clear communication between customers and service providers along with the option for rating the worker for his service.

2. Navigation

At the top of the application interface, a navigation bar provides quick access to essential features:

- Service History: Allows users to view their service history.
- Chats (List of Chats): Directs users to the list of chat conversations with workers.
- Work Section: Provides access to his work section.

Home Page (List of Services Form):

- Upon launching the application, users land on the home page displaying a list of available services.
- Selecting a service redirects users to the list of workers page.

List of Workers Page:

- Users can view a list of workers offering the selected service.
- Clicking on the "Send Enquiry" button directs users to the Service History page.
- Clicking on the "Chat" option opens the Chats page corresponding to the selected worker.
- The navigation bar at the top remains accessible for easy page navigation.

Service History Page:

- Users can access their service history, which displays past interactions and transactions.
- A "Back" button allows users to return to the Home page.

Chats Page:

- The Chats page displays a list of chat conversations with different workers.
- Selecting a chat opens the chat conversation corresponding to that worker.
- A "Back" button is available to navigate back to the List of Chats page.

Page Interaction:

- Users can interact with each page to perform specific actions, such as sending enquiries, viewing chat conversations, and accessing service history.

Work Page

- If the user is employed as a regular service provider, he can see all the incoming service requests. He can also chat with the customer to fix details regarding the service to be provided.
- If the user is employed as the head of a regular service organisation, he can see all the incoming employment requests for that organisation, the details of each applying candidate, and either approve or reject the request.
- A "Back" button allows users to return to the Home page.

3. Features

Service Booking (Customer Side):

- Customer can request service from any worker and get the best service by searching and filtering workers based on rating and number of ratings.
- After selecting a worker they can send a booking request to him and their request is communicated to the worker.

- Along with this they can send details of the work and negotiate the price of service with the workers in the chat section of our App.
- Once the service is done the customer can do one click payment of the service without any issues
- Customer can also altruistically rate the service of the worker and ensure only good service providers are in good demand

Chats:

- Customers can see their chats in the chats window along with unread messages count shown to them along with the last message.
- User can select whom to chat and the chat occurs in real-time
- Chats are stored and they can see when it was received and whether they have seen it or not

Work Section (Regular Service Provider):

- The worker can see all the incoming service requests, their service time, bill amount, etc. He can also filter them according to their status.
- He can accept, reject the pending requests, see his completed requests, upcoming requests, and also requests whose payment is due.

Work Section (Head of Regular Service Organization):

- The button “Work Section” in the navigation bar routes differently for the Head of Regular Service Organization. It routes to a page where the head can see current employees and current employment requests present.
- The employees are listed in a table where the head can look at the rating corresponding to each employee.
- The head can also see incoming employment requests of currently unemployed people or of those who want to change their job. Each request displays only necessary information, additional information can be sought out by clicking on the ‘View’ button.
- The head can look at candidates’ resume and past experience, and then decide whether to ‘Approve’ or ‘Reject’ the particular candidate.

4. User Interface

User Flow

1. **List of Services:** User will select the service that he wants from Electrician, Plumber, Househelp, Mechant
2. **List of Workers:** Customer Can click on the Send Enquiry button to Request the booking and or click on the chat icon to start the chat
3. **Service History:** Customer is redirected to this page and whenever he Sends a new Enquiry and Can also Access it from the Navigation bar of service portal. Here the Customer can cancel sent enquiries, Cancel booked appointments, see upcoming Appointments, Do payment and Star Rate the Customer From 1 to 5

4. **Payment Gateway:** Customer is redirected to the payments page where he can pay the worker through various options like Netbanking, etc
5. **List of Chats:** User can see their all their previous chats in which they have sent one message and select whom to chat with
6. **Chats:** Customer can chat here with the Workers negotiate the the time of service and bill amount as well as ask for rescheduling the time of the service

List of workers

Overview:

- The "List of Workers" page provides users with a list of service workers available based on their selected job category. Users can view details such as worker name, organisation location, rating, number of ratings, and contact information and Send Enquiry Button.

Features:

1. Filtering Options: Users can filter the list of workers based on their preferences. Clicking on the "Filter" button (labelled as "Label1") allows users to access filtering options.
2. Search Functionality: Users can search for specific vendors using the search bar provided. Simply click on the search bar and enter the vendor's name to find them quickly.
3. Dynamic Updates: The list of workers updates dynamically based on user interactions and changes in the database. This ensures that users always have access to the most up-to-date information.
4. Sorting: Users can sort the list of workers based on different criteria such as worker rating and number of ratings. This allows users to find the most suitable service worker for their needs.
5. Worker Details: Each worker listed provides essential details including their name, organisation location, rating, number of ratings, and contact information.

How to Use:

1. Accessing the Page: Users can access the "List of Workers" page from the application's navigation menu or from other relevant pages.
2. Filtering: Click on the "Filter" button to access filtering options. Users can filter workers based on their preferences such as rating and number of ratings.
3. Searching: Use the search bar to quickly find specific vendors by entering their name.
4. Viewing Worker Details: Scroll through the list of workers to view their details. Each worker entry provides essential information to help users make informed decisions.
5. Interacting with Filters: Toggle checkboxes to apply sorting preferences for worker rating and number of ratings.
6. Navigating to Other Pages: Users can navigate to other relevant pages by clicking on the provided buttons, such as returning to the main services page.

Service History

Overview:

- The "Service History" page provides users with a comprehensive overview of their past service bookings. Users can view details such as booking ID, worker name, service time, bill amount, worker rating, Status and Action Buttons which change the status of the service based on what state they are in.

Features:

1. Filtering Options: Users can filter the service history based on various criteria such as service time and status. Clicking on the "Filter" button allows users to access filtering options.
2. Search Functionality: Users can search for specific service history entries using the search bar provided. Simply click on the search bar and enter relevant keywords to find specific entries.
3. Dynamic Updates: The service history updates dynamically based on user interactions and changes in the database. This ensures that users always have access to the most up-to-date information.
4. Sorting: Users can sort the service history entries based on different criteria such as service time. This allows users to easily find the desired entries.
5. Detailed Information: Each service history entry provides detailed information including booking ID, worker name, service time, bill amount, worker rating, and status. Users can use this information to track their past service bookings effectively.

How to Use:

1. Accessing the Page: Users can access the "Service History" page from the application's navigation menu or from other relevant pages.
2. Filtering: Click on the "Filter" button to access filtering options. Users can filter service history entries based on criteria such as service time and status and action to do.
3. Searching: Use the search bar to quickly find specific service history entries by entering relevant keywords.
4. Viewing Service History Details: Scroll through the list of service history entries to view detailed information. Each entry provides essential details to help users track their past service bookings effectively.
5. Interacting with Filters: Toggle checkboxes to apply filtering preferences for service time and status.
6. Navigating to Other Pages: Users can navigate to other relevant pages by clicking on the provided buttons, such as returning to the main dashboard.

List of Chats

Overview:

- The "List of Chats" page provides users with a list of their ongoing chats with service workers. Users can view details such as worker names, the last message exchanged, and the number of unread messages for each chat.

Features:

1. Real-time Updates: The page dynamically updates to reflect any changes in the user's chat history, such as new messages or unread messages.
2. Unread Message Indicator: Chats with unread messages are highlighted in a separate panel, making it easy for users to identify and prioritise chats that require attention.
3. Message Preview: Users can view a preview of the last message exchanged in each chat, allowing them to quickly catch up on recent conversations.
4. Navigation: Users can easily navigate between chats and access individual chat sessions by clicking on the desired chat item.
5. Return to Main Page: Users can return to the main page of the application by clicking on the provided button.

How to Use:

1. Accessing the Page: Users can access the "List of Chats" page from the application's navigation menu or from other relevant pages.
2. Viewing Chats: Scroll through the list of chats to view ongoing conversations with service workers. Each chat item displays the worker's name, last message, and unread message count.
3. Interacting with Chats: Click on a chat item to view the full conversation and send new messages. Unread messages are highlighted in a separate panel for easy identification.
4. Real-time Updates: The page updates in real-time to reflect changes in the user's chat history, ensuring users have access to the latest information.
5. Returning to Main Page: Click on the provided button to return to the main page of the application at any time.

Chats

Overview:

- The "Chats" page allows users to engage in conversations with service workers. Users can send and receive messages, view message history, and see unseen messages for new messages.

Features:

1. Real-time Messaging: Users can send and receive messages in real-time, enabling seamless communication with service workers.
2. Message History: The page displays the history of messages exchanged between the user and the service worker, allowing users to review past conversations.
3. Unseen Message: Users can see unseen messages and their count along with the last message given
4. Return to Main Page: Users can easily return to the main page of the application by clicking on the provided button.

How to Use:

1. Accessing the Page: Users can access the "Chats" page from the application's navigation menu by clicking the chat button on any workers card in the list of workers section.
2. Viewing Messages: Scroll through the chat history to view messages exchanged between the user and the service worker. Each message displays the sender's name, message content, and timestamp.
3. Sending Messages: Type a message in the text input area at the bottom of the page and press Enter or click the send button to send the message to the service worker.
4. Receiving Messages: New messages from the service worker are automatically displayed in the chat history area. Users receive notifications for new messages, even if the page is not currently active.
5. Returning to Main Page: Click on the provided button to return to the main page of the application at any time.

5. Troubleshooting

Error Handling

- The worker cannot accept a pending request without providing the service time for it.
- The worker can also not click the done button for an upcoming request without providing the bill amount for it.

Common Issues

- Ensure that the list of workers and list of Appointments is up to date along with the state of the appointments
- Ensure that the Payment occurs without any inconsistencies in the front end
- Ensure that chats are placed in the right order.

Health Portal

Table of Contents

1. Introduction

Welcome to our Health Portal, your gateway to seamless healthcare management. With our platform, users can effortlessly book appointments with their preferred doctors and access a comprehensive directory of healthcare professionals in the city. Meanwhile, doctors can efficiently manage their appointment schedules, ensuring smooth operations and enhanced patient care. Also, the hospital owners can manage the recruitment of their doctors by creating the new vacancies and managing existing ones. Apart from that, the users are empowered to take control of their health by providing easy access to their medical history through the health record tracker.

2. Navigation

- Users can access the different pages using the top navbar in the Health portal.
- The user can see the doctors in different specialisation fields under the specialisation tab.
- The user can see the list of hospitals under the hospitals tab, and then select a doctor among the doctors of that hospital.
- The user can even see his health record track from the health record tab.
- Also, if the user is employed as a doctor or hospital owner, he can navigate to the respective work section. If he is a doctor, he can see all the appointment bookings, and if he is a hospital owner, he has options to create a new vacancy in the hospital, view previous vacancies, or accept or reject incoming doctor employment requests.

3. Features

Specialisation Section

1. Users can conveniently navigate to the Specialization Section from the health module's navigation bar. However, if they are not registered as hospital owners, they will receive a message indicating their non-employment status.
2. Directory of City Doctors: Within this section, users gain access to a comprehensive directory of all doctors within the city, complete with their affiliated hospitals, ratings, gender, hospital locations, and contact numbers.
3. Filtering Capabilities: The page offers robust filtering features, allowing users to refine their search based on the specialisation of the doctor, gender preferences, and ratings. This functionality ensures that users can easily locate and connect with healthcare professionals tailored to their specific needs.
4. Empowering Users: By providing such a feature-rich directory, this section empowers users to make informed decisions regarding their healthcare needs. Whether seeking a specialist or a preferred gender of healthcare provider, users can effortlessly navigate through the directory to find the right doctor for their requirements

Hospital Section

1. Accessing the Hospitals Page: Users can access the Hospitals Page directly from the health module's navigation bar. However, if they are not registered as hospital owners, they will receive a message indicating their non-employment status.
2. Comprehensive Hospital Directory: The Hospitals Page provides users with a comprehensive list of all hospitals within the city, along with their locations and contact numbers. This directory serves as a convenient resource for individuals seeking healthcare services or information about nearby hospitals.
3. Search Functionality: The page includes powerful search features that enable users to find hospitals based on specific criteria, such as location or hospital name. This functionality enhances user experience by facilitating quick and targeted access to relevant information.
4. Appointment Booking: Users can seamlessly transition from the Hospitals Page to the Appointment Page by selecting a hospital from the directory. This streamlined process allows users to schedule appointments with doctors affiliated with their chosen hospital, ensuring a hassle-free experience for healthcare seekers.

Appointment Page

1. Accessing the Appointment Page: Once users select a hospital from the directory, they can seamlessly navigate to the Appointment Page, where they can schedule appointments with doctors affiliated with that hospital.
2. Doctor Selection and Date Filtering: On the Appointment Page, users will find a comprehensive list of all doctors associated with the selected hospital. They can filter the list based on the desired specialisation and available appointment dates. The filtering system ensures that only doctors with available appointment slots for the selected date and specialisation are displayed, streamlining the appointment booking process.
3. Available Time Slots: For each doctor listed, users can view the remaining available time slots for appointments. If a particular time slot is already booked, it will not appear in the results, ensuring that users only see options that are open for scheduling. This feature allows users to easily identify and book available appointments with their preferred doctors.

Hospital Owner’s Work Section

1. Accessing the Work Section: Users can easily access the work section if they are employed as a Hospital Owner from the navigation bar of the health module, Else they will get a message showing Not Employed.

2. Managing Doctor Employment Requests: Hospital owners can review incoming doctor employment requests. Details such as resumes, experience, and qualifications of applicants are provided for evaluation. Hospital owners have the authority to accept or reject applications based on their assessment.
3. Managing Vacancies: Hospital owners can view vacancies within their hospital, including details such as department, required experience, etc. They can delete the vacancies which have been satisfied.
4. Creating New Vacancies: Hospital owners have the capability to create new job vacancies within their hospital. When creating a new vacancy, owners can specify details such as department, required experience, qualifications, etc.
5. This feature enables hospital owners to effectively manage staffing needs and recruit suitable candidates to fulfil roles within the hospital.

Doctor's Work Section

1. Accessing the Work Section: Users can easily access the work section if they are employed as a Doctor from the navigation bar of the health module, Else they will get a message showing Not Employed.
2. This allows doctors to view all their scheduled appointments in one convenient location. This includes both upcoming and completed appointments.
3. Doctors can easily access essential information about each appointment, such as the patient's name, gender, appointment date and time, and the current status of the appointment.
4. Doctors have the ability to update the status of appointments with a simple double-click. For instance, they can change the status from "booked" to "completed" once an appointment has been successfully conducted.
5. This feature empowers doctors to maintain organised records of their appointments, facilitating better patient care and communication.

Health Record Tracker

1. Accessing the Health Record Tracker: Users can easily access the Health Record Tracker feature from the navigation bar of the health module.
2. The Health Record Tracker provides users with access to their medical history, including basic medical details like blood group, height, weight, etc., and a comprehensive record of all past appointments across various hospitals.
3. Users can view details such as the date of the appointment, department, hospital, etc., for each appointment and see the bill, prescription of the appointments.
4. The Health Record Tracker empowers users to take control of their health by providing easy access to their medical history.

Technical Documentation

Employment Portal

Table of Contents

1. Introduction

This technical documentation provides insights into the architecture, functionality, and implementation details of the backend code.

2. System Architecture

The backend code utilises a simple three-tier architecture:

- Presentation Layer: Implemented using VB.NET and Windows Forms for the user interface.
- Business Logic Layer: Contains the backend code responsible for handling user interactions and business logic.
- Data Access Layer: Interacts with the MySQL database to perform CRUD (Create, Read, Update, Delete) operations.

3. Dependencies

The backend code relies on the following dependencies:

- `MySql.Data.MySqlClient`: MySQL database connectivity library.
- `System.Text`: Standard library for handling string operations.

4. Database Schema

The backend code interacts with a MySQL database named `smart_city_management`. The relevant tables for this page are `workerEmployReq`, which contains fields such as `userID, orgID, resume, detailsPrevWork, profile_picture, date, bank_account_no, name, contactDetails, email, address` and `employmentReq`, which contains fields such as hos_id, user_id, dept, resume.

5. Backend Functionality

Employ_application page

applyBtn_Click Event Handler

- Checks whether all required details are filled before applying.

SaveDetailsToDatabase Function

- Parses and inserts the details filled by the user into the `workerEmployReq` table.

Document_Click Event Handler

- Allows user to select a document from their local machine, to upload while applying.

Document_DragDrop Event Handler

- Allows user to drag and drop a document from their local machine, while applying.

upload_Click Event Handler

- Allows user to select a profile picture from their local machine, to upload while applying.

resume_Click Event Handler

- Allows the user to select a PDF file from their local machine, to upload a resume while applying for work.
- Functionality:
    1. Open a file dialog to choose a PDF file.
    2. Reads the selected PDF file as bytes.
    3. Displays an error message if there's any issue loading the file.

Health_DoctorsEmployment page

Label3_Click Event Handler

- Allows the user to select a PDF file from their local machine, to upload a resume while applying for work.
- Functionality:
    4. Open a file dialog to choose a PDF file.
    5. Reads the selected PDF file as bytes.
    6. Displays an error message if there's any issue loading the file.

ComboBox1_SelectedIndexChanged Event Handler

- Handles the selection change event of the ComboBox, filtering hospitals based on the selected department.

6. Error Handling

- Implements error handling mechanisms using try-catch blocks to gracefully handle exceptions.
- Displays informative error messages to users in case of any failures during applying or database operations.

7. Conclusion

The backend code provides a robust and secure mechanism for users to apply for employment opportunities in the city. By following industry best practices in error handling, and modularity, the code ensures a reliable and user-friendly experience.

Services Portal

Table of Contents

1. Introduction

This technical documentation provides insights into the architecture, functionality, and implementation details of the backend code.

2. System Architecture

Modular Component Architecture :

This Ensures that each logic Is handled in its own Class Rather than All of them Being Dumped into the same class and Reducing the Code Readability and Maintainability. This pushes the Complexity of the code more towards the edges Leaving a more simpler Core in the code Tree of Classes.

Optimised backend :

Showing the most up to date UI to the user is the utmost priority and must be handled with heavy backend so we needed to optimise the Queries as much as possible and fetch new data only when it is necessary.

A) ListofWorkers_Citizen

1. **ListofWorkers_Citizen Class**
   1.1. **Description:**
   - This class manages the list of workers displayed to citizens, including filtering and updating the UI.
   1.2. **Properties:**
   - WorkerCount: Integer to store the count of workers.
   - OriginalWorkersList: List of Worker objects to store the original list of workers fetched from the database.
   - Filterform: Instance of Workersfilter class used for filtering workers.
   1.3. **Constructor:**
   - Initialises the component and subscribes to events.
   1.4. **Methods:**
   - Timer1_Tick: Handles the periodic refresh of the worker list based on job selection.
   - ListofWorkers_Citizen_VisibleChanged: Handles the visibility change of the ListofWorkers_Citizen form.
   - Job_SelectedChanged_Handler: Fetches and updates the list of workers based on the selected job.
   - UpdateUIWithFilteredWorkers: Updates the UI with filtered and sorted list of workers.
   1.5. **Event Handlers:**
   - Timer1_Tick: Triggered on timer tick for periodic update.
   - ListofWorkers_Citizen_VisibleChanged: Triggered when visibility of the form changes.
   - Job_SelectedChanged_Handler: Triggered when job selection changes.
2. **Worker Class**
   2.1. **Description:**
   - Represents a worker entity with properties such as ID, name, location, rating, etc.
   2.2. **Properties:**
   - WorkerId: String representing worker's ID.
   - WorkerName: String representing worker's name.
   - OrgLocation: String representing worker's organisation location.
   - WorkerRating: Double representing worker's rating.
   - WorkerNumberOfRatings: Integer representing the number of ratings received by the worker.
   - WorkerPhoneNumber: String representing worker's phone number.
   2.3. **Constructor:**
   - Initializes Worker objects with provided properties.

B) UrbanClapNav

1 **UrbanClapNav Class**
   1.1. **Description:**

a. The UrbanClapNav class manages the navigation within the UrbanClap application, displaying different forms based on user actions and permissions.

1.2. **Properties:**

a. userID: Integer representing the user ID of the current user.
b. connString: String representing the connection string for the MySQL database.
c. conn: MySqlConnection object for database connectivity.
d. designation: String representing the designation of the user.

1.3. **Methods:**

UrbanClapNav_Load: Handles the loading of the UrbanClapNav form and initialises the UI.
Button1_Click: Handles the click event of Button1 (Service Request) and displays appropriate forms based on the user's designation.
Button2_Click: Handles the click event of Button2 (Chats) and displays the ChatsForm.
Button3_Click: Handles the click event of Button3 (Work Section) and toggles between displaying service request or work section forms based on user's designation.
ShowFormInPanel1: Displays a specified form inside Panel1.
HideCurvedLabels: Hides curved labels CurvedLabel1 and CurvedLabel2.
ShowCurvedLabels_chats: Shows CurvedLabel2 for chat-related sections.
ShowCurvedLabels_history: Shows CurvedLabel1 for history-related sections.

C) Service History

1. **ServiceHistory Class**

1.1. **Description:**

- This class manages the service history of a user, displaying booking details and providing filtering options.

1.2. **Properties:**

- OriginalBookingsList: List of BookingItem objects to store the original list of service bookings fetched from the database.
- HistoryCount: Integer to store the count of service history items.
- OriginalStatusCount: Dictionary to store the count of each appointment status.
- semaphore: Semaphore for thread synchronisation to access shared resources.

1.3. **Constructor:**

- Initialises the component, sets up initial status count, and starts the timer.

1.4. **Methods:**

- InitializeStatusCount: Initialises the OriginalStatusCount dictionary with status counts set to zero.
- ServiceHistory_Load: Loads the service history from the database and updates the UI.
- Timer1_Tick: Handles the periodic refresh of service history based on status changes.
- MapStatusToAppointmentState: Maps status strings to AppointmentState enum values.
- UpdateUIWithFilteredBookings: Updates the UI with filtered and sorted list of service history items.

- FilterAndSortBookings: Filters and sorts the service history items based on user-selected filters.

1.5. **Event Handlers:**

- Timer1_Tick: Triggered on timer tick for periodic update.
- CurvedLabel9_Click: Triggered on clicking the button to navigate back to the home page.
- TextBox_GotFocus: Triggered when textbox gains focus to clear placeholder text.
- TextBox_LostFocus: Triggered when textbox loses focus to restore placeholder text.
- Label1_Click: Triggered on clicking the filter option to show or hide the filter form.

D) BookingItem

1. **BookingItem Class**

1.1. **Description:**

- Represents a single booking item in the service history.

1.2. **Properties:**

- BookingId: String representing the booking ID.
- WorkerName: String representing the worker's name associated with the booking.
- ServiceTime: String representing the service time of the booking.
- BillAmount: String representing the bill amount of the booking.
- WorkerRating: Double representing the worker's rating associated with the booking.
- Status: AppointmentState enum representing the status of the booking.

2.3. **Constructor:**

- Initializes BookingItem object with provided booking details.

E) History Filter

1. **Historyfilter Class**

1.1. **Description:**

- Manages the filtering options for the service history.

1.2. **Methods:**

- FilterAndSortBookings: Filters and sorts the service history items based on user-selected filters.

3.3. **Event Handlers:**

- CheckBoxes_CheckStateChanged: Triggered on checkbox state change to filter service history.

F) ListOfChats

1. **ListOfChats Class**

1.1. **Description:**

- This class manages the list of chats between a user and service workers, displaying unread message counts and last messages.

1.2. **Constructor:**

- Initialises the component, sets up a timer for periodic updates, and loads initial worker message counts.

1.3. **Methods:**

- LoadChats: Loads the list of chats from the database and populates the UI with ChatListItems.
- UpdateReceivingTime: Updates the receiving time for relevant messages in the database.
- LoadWorkerMessageCounts: Fetches worker IDs and their respective total message counts from the database.

1.4. **Event Handlers:**

- refreshTimer_Tick: Triggered on timer tick to refresh the list of chats by updating worker message counts.
- CurvedLabel9_Click: Triggered on clicking the button to navigate back to the home page.

1.5. **Dependencies:**

- MySql.Data.MySqlClient: Library for MySQL database connectivity.

G) Chats

1. **Chats Class**

1.1. **Description:**

- This class manages the chat interface between a user and a service worker, allowing the user to send and receive messages.

1.2. **Constructor:**

- Initialises the component, sets up event handlers, and starts a timer for periodic updates.

1.3. **Methods:**

- LoadChatMessages: Retrieves and displays chat messages from the database.
- UpdateReceiveTime: Updates the receiving time for messages and marks them as seen.
- CheckForNewMessages: Checks for new messages from the database and updates the UI accordingly.
- GetLastMessageIDFromDatabase: Retrieves the last message ID from the database.
- GetNewMessagesFromDatabase: Retrieves new messages from the database based on the last message ID.
- InsertMessageIntoChatsTable: Inserts a new message into the Chats table in the database.
- MakePictureBoxRound: Sets the PictureBox's region to a circle shape for rounded appearance.

1.4. **Event Handlers:**

- Button1_Click: Navigates back to the list of chats page when clicked.
- Timer1_Tick: Triggers the periodic update of chat messages and checks for new messages.
- Label1_Click: Sends the typed message when clicked.
- RichTextBox1_GotFocus: Handles the got focus event of the message input textbox.
- RichTextBox1_LostFocus: Handles the lost focus event of the message input textbox.
- Label2_Paint: Handles the paint event of a label to create a gradient background.

1.5. **Dependencies:**
- System.Drawing.Drawing2D: Required for graphics-related operations.
- MySql.Data.MySqlClient: Library for MySQL database connectivity.
- Windows.ApplicationModel.Chat: Namespace for chat-related functionality (not used in the provided code).

H) Chatlistitem

1. **Chatlistitem Class**
1.1. **Description:**
- This class represents an item in the list of chats, displaying information about a specific worker including their name, last message, and unread message count.
1.2**. Constructor:**
- Initialises the control with optional parameters for worker ID, name, last message, and unread message count. Sets the values to the respective controls and adjusts the display of the last message if it exceeds 30 characters.
1.3. **Methods:**
- MakePictureBoxRound: Sets the PictureBox's region to a circle shape for rounded appearance.
1.4. **Event Handlers:**
- CurvedLabel1_Click, Label1_Click, Label2_Click, PictureBox1_Click, Label3_Click: Handles click events of various controls within the Chatlistitem. Updates the global variable for the selected chat ID, marks the messages as seen in the database, and navigates to the chat page.
1.5. **Dependencies:**
- System.Drawing.Drawing2D: Required for graphics-related operations.
- MySql.Data.MySqlClient: Library for MySQL database connectivity.

I) ChatElement

1. **ChatElement Class**
1.1. **Description:**
- This class represents a chat element in the chat interface, consisting of a curved label to display the message text and a label to display the date and time of the message.
1.2. **Constructor:**
- Initialises the control with optional parameters for the message text and specific date and time. Sets the text for the CurvedLabel and the DateTimeLabel, adjusting their sizes based on the content.
1.3. **Methods:**
- MakePictureBoxRound: Sets the PictureBox's region to a circle shape for rounded appearance.
- AdjustLabelSize: Adjusts the size of the label based on the text content to ensure proper display.
1.4. **Event Handlers:**
- ChatElement_Load: Handles the load event of the ChatElement control. Makes the PictureBox round.

- Label1_Paint: Handles the paint event of the CurvedLabel control. Draws a radial gradient background for the CurvedLabel.

1.5. **Dependencies:**

- System.Drawing.Drawing2D: Required for graphics-related operations.
- System.Windows.Media.Effects: Library for applying effects to UI elements.

J) HistoryItem Class

1. **HistoryItem Class**

1.1. **Description:**

- This class represents an item in the service history, displaying information about past appointments. It allows users to perform actions such as withdrawing an enquiry, cancelling an appointment, paying for a service, and rating the service provider.

1.2. **Fields:**

- BookingId: Stores the ID of the booking associated with the history item.
- stateColors: Dictionary mapping appointment states to colors for display.
- actionColors: Dictionary mapping customer actions to colors for display.
- currentAppointmentState: Tracks the current state of the appointment.

1.3. **Enums:**

- AppointmentState: Defines different states of the appointment.
- CustomerAction: Defines different actions that can be performed by the customer.

1.4. **Constructor:**

- Initialises the control with optional parameters for booking ID, customer name, service time, bill amount, rating value, and current appointment state. Sets the values of the components based on the parameters and initialises the appointment state.

1.5. **Methods:**

- MakePictureBoxRound: Sets the PictureBox's region to a circle shape for rounded appearance.
- UpdateAppointmentState: Updates the appointment state and corresponding actions based on the new state.
- HandleAction: Handles customer actions such as withdrawing an enquiry, cancelling an appointment, paying for a service, and rating the service provider.
- UpdateStatusInDatabase: Updates the status of the appointment in the database.
- ChildForm, ChildForm2: Utility functions to display child forms within a panel.
- GetUserIdFromBookingsFor, GetUsername, GetWorkerIdFromBookings: Helper functions to retrieve user information from the database.

1.6. **Event Handlers:**

- CurvedLabel4_Click: Handles the click event of the action label. Performs actions based on the current state of the appointment.
- HistoryItem_Load, HistoryItem_Load_1: Handles the load event of the HistoryItem control.

K) Workersfilter

3. **Workersfilter Class**

3.1. **Description:**

- Manages the filtering options for the list of workers.

3.2. **Methods:**
- TrackBar1_Scroll: Handles the scrolling of TrackBar for filtering by rating.
- CheckBoxes_CheckStateChanged: Handles the change in state of checkboxes for sorting.
- Button2_Click: Resets all filter options and updates the worker list.
- Button1_Click: Updates the worker list based on applied filters.

3.3. **Event Handlers:**
- TrackBar1_Scroll: Triggered on scrolling of TrackBar.
- CheckBoxes_CheckStateChanged: Triggered on checkbox state change.
- Button2_Click: Triggered on clicking the reset button.
- Button1_Click: Triggered on clicking the filter button.

4. **ListItem Class**

4.1. **Description:**
- Represents a single item in the list of workers displayed to citizens.

4.2. **Properties:**
- Worker_ID: String representing the worker's ID.

4.3. **Constructor:**
- Initializes ListItem object with provided worker details.

4.4. **Methods:**
- MakePictureBoxRound: Makes the PictureBox round for displaying the worker's image.

4.5. **Event Handlers:**
- Label5_Click: Triggered on clicking the chat option to open the chat page for the worker.
- Label6_Click: Triggered on clicking the send enquiry option to send an enquiry to the worker.

L) HistoryFilter Class

1. **HistoryFilter Class**

1.1. **Description:**
- This class represents a filter control for the service history. It provides options to filter history items based on various criteria such as status, date, etc.

1.2. **Event Handlers:**

1.2.1. CheckBoxes_CheckStateChanged:
- Handles the CheckStateChanged event of the checkboxes. Changes the appearance of the checkboxes based on their state (checked, unchecked, indeterminate).
- Updates the text and colour of the checkbox to indicate sorting order (newest first or oldest first).

1.2.2. CheckBox_CheckedChanged:
- Handles the CheckedChanged event of the checkboxes. Sets the background image and adjusts the layout of the checkbox when it is checked.
- Clears the background image when the checkbox is unchecked.

1.2.3. Button2_Click:
- Handles the Click event of the "Reset" button. Resets all filter options to their default state and updates the UI with filtered bookings.

1.2.4. Button1_Click:

- Handles the Click event of the "Filter" button. Updates the UI with filtered bookings based on the selected filter options.

M) Rate Class

1. **Rate Class:**
   1.1. **Description:**
   - This class represents a form for rating service workers. It allows users to select the number of stars for rating and updates the worker's rating in the database accordingly.
   1.2. **Properties:**
   - SelectedStars: Gets or sets the number of stars selected by the user.
   1.3. **Fields:**
   - workerID: Stores the ID of the service worker being rated.
   1.4. **Constructor:**
   - New(ByVal workerID As Integer): Initialises a new instance of the Rate class with the specified worker ID.
   1.5. **Methods:**
   - CurvedLabel1_Click(sender As Object, e As EventArgs): Event handler for the click event of the rating label. Sets the SelectedStars property and updates the rating in the database.
   - UpdateRatingInDatabase(): Updates the rating of the service worker in the database based on the selected stars.

N) Services_WorkSect (Regular Service Provider)

1. **Services_WorkSect Class**
   1.1. **Description:**
   - This class manages the list of service requests, including filtering and updating the UI.
   1.2. **Methods:**
   - GetWorkerID:Fetches the worker id corresponding to the current user who is logged in
   - LoadServiceRequests:Fetches all the service requests corresponding to the current worker.
   1.3. **Event Handlers:**
   - Guna2ComboBox1_SelectedIndexChanged: Triggered on selecting any option in the filter component. It filters the requests based on their status.

O) SerReq_worker_accepted

1. **SerReq_worker_accepted Class:**
   1.1. **Description:**
   - This class represents a component for a service request which has been accepted by the worker.
   1.2. **Methods:**
   - UpdateStatus: Updates the status of the request in the serviceBooking table.

1.3. **Event Handlers:**

- Button2_Click: Triggered on clicking the done button. It implies that the worker has completed the work, and now the customer can pay him.
- Button1_Click: Triggered on clicking the cancel button. It implies that the worker has cancelled the previously accepted request.

P) SerReq_worker_pending

1. **SerReq_worker_pending Class:**
   1.1. **Description:**
   - This class represents a component for a service request which has not been accepted by the worker.
   1.2. **Methods:**
   - UpdateStatusServiceTimeAndBillAmount: Updates the status of the request in the serviceBooking table.
1.3. **Event Handlers:**
   - Button2_Click: Triggered on clicking the done button. It implies that the worker has accepted the request.
   - Button1_Click: Triggered on clicking the cancel button. It implies that the worker has rejected the request.

Q) SerReq_worker_completed

1. **SerReq_worker_completed Class:**
   1.1. **Description:**
   - This class represents a component for a service request which has been completed and the payment has also been done.

R) SerReq_worker_paymentDue

1. **SerReq_worker_paymentDue Class:**
   1.1. **Description:**
   - This class represents a component for a service request which has been completed but its payment is pending.

S) IncomingEmploymentReq

1.1 Description: The IncomingEmploymentReq class is a form in a Visual Basic .NET application designed to manage incoming employment requests. It interacts with a MySQL database to retrieve and display employment requests based on a specified organisation ID.

1.2 Properties and Private Fields

- OrgHeadWorkSectionForm: Stores a reference to the parent form.
- _orgID: Stores the organisation ID used in database queries.

1.3 Methods

- Label1_Click and Label1_Click_1: Placeholder methods for handling clicks on Label1, used for user interaction or error messages.

T) Incoming_Emp_Req

1.1 Description: It is designed for managing employment requests within an organisation. It deals with different operations such as approving, rejecting, and displaying details of employment requests.

1.2 Properties and Private Fields

- OrgHeadWorkSectionForm: Stores a reference to the parent form.
- _orgID: Stores the organisation ID used in database queries.

1.3 Methods

- **SetEmpDesignation:** Uses a Select Case statement to assign a job title based on the organisation ID (orgID).
- **Button2_Click:** Handles the approval process of an employment request. It:
    - Opens a MySQL connection and starts a transaction.
    - Executes several SQL statements to update user status and insert new records into the database.
    - Commits the transaction and displays success messages, or rolls back in case of an error.
    - Finally, it removes the user control from the parent form, indicating the request has been processed.
- **Button1_Click:** Handles the rejection of an employment request by updating the database to reflect the rejection status and removes the control from the display.
- **Button3_Click:** Displays a new form with detailed information about the employment request.

1.4 Namespace Imports:

- System.Runtime.InteropServices and System.Transactions are imported but not directly used in the provided code, which suggests they might be used elsewhere in the application or are unnecessary imports.
- DocumentFormat.OpenXml.Bibliography and MySql.Data.MySqlClient are crucial as they handle database operations.

1.5 Constructor (Public Sub New):

- Initialises the class with parameters for each piece of user information and employment request detail.
- Calls SetEmpDesignation to set the employment designation based on orgID.
- Handles the displaying of an optional image if provided.

U) Incoming_Emp_Req

1.1 Description: The OrgHeadWorkSection class handles specific functionalities related to organisational department heads. It connects to a MySQL database to fetch user-related data and adjusts the UI based on the user's role.

1.2 Private Fields and Constructor

- **connString**: This is the connection string used to connect to a MySQL database. It's fetched from a module named Module1, ensuring that the connection string is centralised for easier management.
- **conn**: A MySQL connection object initialised with the connString.
- **_orgID**: An integer that stores an organisation ID corresponding to the department.

1.3 Methods

- **SetPvtOrgId(dept As String):** A private method that sets _orgID based on the department name. This ID helps in loading relevant data/forms specific to a department.

1.4 Event Handlers

- **OrgHeadWorkSection_Load(sender As Object, e As EventArgs):** This method executes when the user control is loaded. It tries to connect to the database to fetch the designation of the currently logged-in user. Depending on the user's designation, it adjusts the UI and potentially other functionalities.
    - If the designation starts with "Head", it parses the designation to determine the specific department and sets the private _orgID accordingly. The designation is also displayed on the UI.
    - If there is no user found or an error occurs, appropriate messages are shown.
- **ShowFormInPanel(formToShow As Form):** This method accepts a form and displays it within a Panel control on the user control. It is a versatile method for displaying different forms within the same panel.
- **Guna2Button2_Click(sender As Object, e As EventArgs):** Handles clicks on a specific button (Guna2Button2), which manages visibility of certain UI elements and loads forms related to employment requests.
- **Guna2Button1_Click(sender As Object, e As EventArgs):** Similar to the above, but it is used for displaying a different form, related to managing employee tables.

3. Dependencies

The backend code relies on the following dependencies:

- `MySql.Data.MySqlClient`: MySQL database connectivity library.

- `System.IO`: Standard library for handling file input/output operations.
- `System.Text`: Standard library for handling string operations.
- `System.Windows.Forms.VisualStyles.VisualStyleElement`: Library for UI elements in Windows Forms.

4. Backend Optimization

- Implemented Gradient which changes according to rating(from green to yellow) also done the same for chats page.
- In service history the state is defined by the status and its respective count. So at every tick we are checking whether any status count has changed. If it has then we fetch the updated history again.
- In List of workers we just check the count of workers and get the updated list if it changes. Also if any update is done from the user side we are not refreshing it again as we are changing the state on the user side itself so as to remove inconsistency.
- In List of chats we get the WorkerId and Corresponding total count of messages, if that number changes we fetch the new messages Also if a user is sending any messages we are again updating the state then and there and reducing backend load and removing inconsistency.
- In order to get the latest state of chats we are pinging the database according to exponential backoff as whenever we get a new data we are pinging the data more frequently and if our fetching results in no new data we do the next query after 2*times the current interval.

5. Error Handling

- Implements error handling mechanisms using try-catch blocks to gracefully handle exceptions.
- Displays informative error messages to users in case of any failures during any database error.

6. Future Updates

1. The backend Updates can be optimised using caches for reading new messages in chats.
2. Separating the “state” from the database as in order to get the state, Also querying the database can be made more efficient by making a denormalized table just for storing the state and all the queries for the state will go to this database.

Health Portal

Table of Contents

1. Introduction

This technical documentation provides insights into the architecture, functionality, and implementation details of the backend code.

2. System Architecture

The backend code follows a three-tier architecture:

- Presentation Layer: Implemented using VB.NET and Windows Forms for the user interface.
- Business Logic Layer: Contains the backend code responsible for handling user interactions and business logic.
- Data Access Layer: Interacts with the MySQL database to perform CRUD (Create, Read, Update, Delete) operations.

3. Dependencies

The backend code relies on the following dependencies:

- `MySql.Data.MySqlClient`: MySQL database connectivity library.
- `System.IO`: Standard library for handling file input/output operations.
- `System.Text`: Standard library for handling string operations.
- `System.Windows.Forms`: Library for Windows Forms user interface components.

4. Backend Functionality

GetDoctorID Function

- Purpose: Retrieves the doctor ID associated with the current user from the database.
- Functionality: Executes a SQL query to fetch the doctor ID based on the user ID.

cMakeAppointment (Component)_load Event handler

Purpose: cMakeAppointment serves the purpose of presenting individual doctors to facilitate the appointment-making process. It retrieves doctor details from the "Doctors" table and fetches additional information such as

doctor name, gender, and affiliated hospital from the "User" table and the hospital database.

Functionality:

- Fetches Doctor Details: cMakeAppointment component calls upon database queries to fetch doctor details from the "Doctors" table, including unique identifiers and other relevant information.
- Joins Tables: Utilises inner joins with the "User" table to acquire supplementary data such as doctor name and gender, enhancing the comprehensiveness of the displayed information.
- Hospital Affiliation: Further enhances doctor profiles by conducting inner joins with the hospital database, providing users with insights into the hospitals where each doctor practices.

Apply Button_OnClick Event Handler (makeAppointment)

- Purpose: Filters doctors once date of appointment and specialisation is selected

Functionality:

- Sends query to select rows of doctors table where date and specialisation matches
- Assuming average time of Appointment to be 20 min, Using FCFS it counts the number of appointments of doctors and it decreases the available time and ignores if already full.

Make Appointment Button_Onclick Event Handler

- Purpose: Adds Appointment of doctor to Hospital_appointment table
- Functionality:Checks if any doctor is selected or not if this occurs it pops a message box showing Appointment is made

cSpecialisation (Component)

- Purpose:Consists of individual doctors with their respective information
- Functionality:This is dynamically created depending on number of doctors

ComboBox (Specialisation, Rating ,Gender)_OnSelected Event Handler

Purpose:Dynamically Applies Filter to Doctors directory
Functionality:

- Each of Specialisation , Rating, Gender Combobox when its item is selected checks where or not the items of other two are selected and dynamically filters doctor's table on basis of those which are selected

cListHospital (Component)_onClick Event Handler

Purpose : Consists of individual hospitals of the city fetched as per rows
Functionality:

- Created for each row of hospital table in database and acts as gateway to hospital appointment page. Has on click functionality , when clicked stores the hospital id selected and sends this to the makeAppointment page to open the hospital's information for appointment.

Combobox_onSelect Event, Search_onclick Event handler(ListHospital)

Purpose :

- SearchBox: Dynamically filters Hospitals based on matched subpart
- ComboBox: To set Criteria for Search Box to search

Functionality:

- SearchBox: Regularly sends requests for filtering hospital tables based on text entered. If the substring of the string in the database matches to text it filters.
- ComboBox: It consists of two options of Hospital and Location. User selecting any one will result in directing search bar to search on that selected field

Health_ViewAppointment_Load Event Handler

- Purpose: Loads the form and populates a ListView with appointment details.
- Functionality:
  - Call `GetDoctorID` to retrieve the doctor ID.
  - Constructs and configures a ListView to display appointment information.
  - Retrieves appointment data from the database based on the doctor ID.
  - Populates the ListView with appointment details fetched from the database.

GetUserDetails Function

- Purpose: Retrieves user details (name and gender) based on the provided user ID.

- Functionality: Executes a SQL query to fetch user details from the database and returns them as a comma-separated string.

ListView1_ItemActivate Event Handler

- Purpose: Handles activation of ListView items (appointments).
- Functionality:
    - Checks if an appointment is selected in the ListView.
    - Retrieves the current status of the selected appointment.
    - Updates the appointment status in the ListView and the database based on certain conditions.

UpdateStatusInDatabase Subroutine:

- Purpose: Updates the status of an appointment in the database.
- Functionality: Executes a SQL update query to modify the status of the specified appointment ID.

Button2_Click Event Handler

- Validates the entered verification code.
- Redirects the user to the password reset page upon successful code verification.

ResendTimer_Tick Event Handler

- Manages the countdown timer for code resend functionality.

ResendLabel_Click Event Handler

- Initiates the process of resending the verification code.
- Implements logic to limit code resending to once per session.

Button3_Click Event Handler

- Redirects the user back to the login page upon clicking the “Back to Login” button.

Health Record Tracker Page:-

Health_Record_Tracker_Load Event Handler

- Initialises the Health Record Tracker form and populates a ListView with appointment details.
- Retrieves the user's name using GetUserDetails. Configures a ListView to display appointment information. Fetches appointment data from the database based on the user ID.

GetUserDetails Function

- Purpose: Retrieves user details (name and gender) based on the provided user ID.
- Functionality: Executes a SQL query to fetch user details from the database and returns them as a string.

GetUserDesignation Function

- Purpose: Retrieves the user's designation based on the provided user ID.
- Functionality:
    - Executes a SQL query to fetch the user's designation from the database.
    - Returns the user's designation as a string.

Label1_Click Event Handler

- Purpose: Navigates to the appropriate form based on the user's designation.
- Functionality:
    - Retrieves the user's designation using GetUserDesignation.
    - Navigates to either the Health View Appointment form or the Health Doctor Employment Requests form based on the designation.

Doctor Employment Request Page :-

Health_Doctor_Employment_Requests_Load Event Handler

- Purpose: Initialises the Health Doctor Employment Requests form and populates a ListView with doctor employment requests.
- Functionality:
    - Configures the form's appearance.
    - Creates a new Panel control to host the ListView.
    - Configures and adds a ListView control to the Panel.
    - Retrieves data from the DoctorEmploymentRequest table and populates the ListView with doctor employment requests.

Doctor_Emp_Req (Component)

Doctor_Emp_Req Constructor

- Purpose: Initialises a new instance of the Doctor_Emp_Req form.
- Functionality:
    - Sets the parent form and initialises component properties.
    - Sets the date, name, and resume data based on the provided parameters.

Button2_Click Event Handler

- Purpose: Handles the click event of the "Accept" button.
- Functionality:
    - Retrieves the user ID associated with the button clicked.
    - Removes the employment request entry from the database using the user ID.
    - Adds the doctor entry to the Doctors table if the employment request is successfully removed.

Button1_Click Event Handler

- Purpose: Handles the click event of the "Reject" button.
- Functionality:
    - Retrieves the user ID associated with the button clicked.
    - Removes the employment request entry from the database using the user ID.

RemoveEntryFromDatabase Function

- Purpose: Removes an entry from the DoctorEmploymentRequest table in the database.
- Functionality:
    - Executes a MySQL delete query to remove the entry based on the provided user ID.
    - Returns true if the entry is successfully deleted; otherwise, returns false.

AddDoctorToTable Function

- Purpose: Inserts a doctor entry into the Doctor's table in the database.
- Functionality:
    - Executes a SQL query to insert the doctor entry with default values into the Doctor's table.
    - Returns true if the entry is successfully inserted; otherwise, returns false.

Button4_Click Event Handler

- Purpose: Handles the click event of the "View Resume" button.
- Functionality:
    - Saves the resume data to a temporary PDF file.
    - Open the PDF file using Microsoft Edge for viewing.

View Previous Vacancy Page :-

Health_View_Previous_Vacancies_Load Event Handler

- Purpose: Loads the form and populates a ListView with previous vacancies.
- Functionality:
    - Creates a Panel control and a ListView control within it.
    - Retrieves data from the hospitalVacancy table in the database.
    - Creates Hospital_Vac_Comp instances for each vacancy and adds them to the ListView.

Hospital_Vac_Comp Class

- Purpose: Represents a hospital vacancy component.
- Constructor:
    - Initialises the component with the parent form, hospital ID, department, vacancy count, and description.
    - Sets the text values of labels based on the parameters.

Button2_Click Event Handler:

- Purpose: Handles the click event of the "More Info" button.
- Functionality: Opens a new form (Hos_Desc_Form) to display detailed description.

Button1_Click Event Handler:

- Purpose: Handles the click event of the "Delete" button.
- Functionality: Removes the vacancy entry from the database.

RemoveEntryFromDatabase Function:

- Purpose: Removes the vacancy entry from the database.
- Functionality:

- Executes a MySQL delete query to remove the entry from the database based on hospital ID.
- Returns true if the entry is successfully deleted; otherwise, returns false.

Create New Vacancy Page:
Health_Create_New_Vacancy Class

- Purpose: Handles the creation of a new vacancy entry in the hospitalVacancy table.
- Button1_Click Event Handler:
  - Purpose: Handles the click event of the "Create" button.
  - Functionality:
    - Retrieves data from the textboxes (hospital ID, department, vacancy count, description).
    - Constructs an SQL INSERT query to add a new vacancy entry to the database.
    - Executes the INSERT query to add the new vacancy entry to the database.
    - Displays a message box indicating successful insertion or an error message box if an exception occurs.

5. Error Handling

- Implements error handling mechanisms using try-catch blocks to gracefully handle exceptions.
- Displays informative error messages to users in case of any failures during database operations or other operations.

User Documentation- Transport Module

Transport Landing Page

Table of Contents

1. Introduction

This page gives the user four options either to go to bus booking, cab booking, map or transport ministry.

2. Getting Started

Accessing the Page

1. **Login:** Users must be logged into the smart city platform
2. **Navigation:** Once logged in, navigate to the transport section.

3. Features

Allows user to navigate to different pages

- Allow the user to go to bus booking, cab booking, map or transport ministry.

4. User Interface

Buttons and Controls

- **Map:** Directs user to Map.
- **Bus Booking Button:** Directs user to bus booking page.
- **Cab Booking Button:** directs user to cab booking page.
- **Transport Ministry Page:** Directs a user to Transport Ministry page

5. Troubleshooting

Common Issues

- If encountering persistent issues, contact the system administrator for assistance.

Cab Booking Page

Table of Contents

1. Introduction

This page enables a user to fill the parameters to be able to swiftly book a cab.

2. Getting Started

Accessing the Page

1. **Login:** Users must be logged into the smart city platform
2. **Navigation:** Once logged in, navigate to the transport section.
3. **Cab Feature:** Within the transport module, locate and click on the "Cab Booking" option.

3. Features

Allows user to fill details of cab required

- Upon accessing the page, the user can fill the source location, destination location and the date-time for which you want a cab.
- This ensures that users are offered flexibility to book a cab for any.

Driver can access info page

- If the user is driver, he or she can access their activity page using ‘Are you a driver button?’, so using the SID of the user we fetch the designation of the person to check

whether or not he is a driver, if he/she is then on clicking on that button they are directed to their activity page otherwise and error message is displayed.

Users can access their activity page

- User can click on the 'User Activity' button control so that they are directed to their activity page wherein they can view current and previous rides.

4. User Interface

Input Fields

1. **Source Location:** Combo-box from where user can select the source location from the specified locations
2. **Destination Location:** Combo-box from where user can select the destination location from the specified locations
3. **Date-time:** Date time picker to select the time for which you want to schedule the cab.

Buttons and Controls

- **Book Button:** Initiates the cab booking process by directing the user to the available cab page.
- **'Are you a driver' Button:** Directs a user to his/her info page in case he/she is a driver.
- **Back to Main Page Button:** directs user back to the transport landing page.
- **User Activity Page:** Directs a user to his/her activity page

5. Troubleshooting

Common Issues

- Ensure all required fields are filled before attempting to book the cab.
- If encountering persistent issues, contact the system administrator for assistance.

Bus Booking Page

Table of Contents

1. Introduction

This page enables a user to fill the parameters to get the bus that take them to their desired location.

2. Getting Started

Accessing the Page

1. **Login:** Users must be logged into the smart city platform
2. **Navigation:** Once logged in, navigate to the transport section.
3. **Bus Feature:** Within the transport module, locate and click on the "Bus Booking" option.

3. Features

Allows user to fill details of Bus required

- Upon accessing the page, the user can fill the source location, destination location and the date-time for which they want a bus.
- This ensures that users are offered flexibility to book a bus for any source to destination.

4. User Interface

Input Fields

1. **Source Location:** Combo-box from where user can select the source location from the specified locations
2. **Destination Location:** Combo-box from where user can select the destination location from the specified locations
3. **Date-time:** Date time picker to select the time for which you want to schedule the cab.

Buttons and Controls

- **Book Button:** Initiates the bus booking process by directing the user to the available bus page.
- **Back to Main Page Button:** directs user back to the transport landing page.

5. Troubleshooting

Common Issues

- Ensure all required fields are filled before attempting to book the cab.
- If encountering persistent issues, contact the system administrator for assistance.

Cab Available Page

Table of Contents

1. Introduction

This page enables a user to select parameters swiftly to be able to book a cab.

2. Getting Started

Accessing the Page

1. **Login:** Users must be logged into the smart city platform
2. **Navigation:** Once logged in, navigate to the transport section.
3. **Cab Feature:** Within the transport module, locate and click on the "Cab Booking" option. Then enter the details of the cab they want to book, after this the user will be directed to this page.

3. Features

Display the details of the most suitable cab for user

- Using the details filled by the user, we run an algorithm in the backend that fetches a suitable cab for the user efficiently.
- To promote sharing, our algo will first search among all the currently running cabs if we can extend the path to satisfy the needs of the user. If there is no such cab, then he/she is allotted the first free available cab.
- After finding the most appropriate cab, its details- source, destination, time of arrival at the pickup location, cost to be incurred by the user, Driver name are all displayed in a card to the user.

Click on ‘Book’ button to begin booking process

- The user will then be directed to the payment process to complete payment and finally confirm the booking.

4. User Interface

Display Fields

1. **Source Location:** Label which shows the pickup location for the cab.
2. **Destination Location:** Label which shows the destination location for the cab
3. **Date-time:** Shows the time of arrival of the cab at the specified pickup location.
4. **Cost:** Shows the cost that will be incurred by the user for booking this cab.

Buttons and Controls

- **Book Button:** Initiates the cab booking process by directing the user to the payment portal.
- **Back to Booking Page Button:** directs user back to the cab booking page.

5. Troubleshooting

Common Issues

- Ensure all required fields are filled before attempting to book the cab.
- If encountering persistent issues, contact the system administrator for assistance.

Bus Available Page

Table of Contents

1. Introduction

This page enables a user to select parameters swiftly to be able to book a cab.

2. Getting Started

Accessing the Page

1. **Login:** Users must be logged into the smart city platform
2. **Navigation:** Once logged in, navigate to the transport section.
3. **Bus Feature:** Within the transport module, locate and click on the "Bus Booking" option. Then enter the details of the bus they want to book, after this the user will be directed to this page.

3. Features

- **Display the details of the all the bus running on the required route on that day**
- Using the details filled by the user, we make a query on the ddataabasse that fetches allll suitable buses for the user efficiently.

Click on ‘Book’ button to begin booking process

- The user will then be directed to the bus info page which shows all the information about bus and also gives you option to do final payment and booking.

4. User Interface

Display Fields

1. **Source Location:** Label which shows the pickup location for the cab.
2. **Destination Location:** Label which shows the destination location for the cab
3. **Date-time:** Shows the time of arrival of the bus at the specified pickup location and of the drop location.
4. **Cost:** Shows the cost that will be incurred by the user for booking this bus, this cost is per unit distance travelled by the bus.

Buttons and Controls

- **Book Button:** Initiates the bus booking process by directing the user to the bus info page.
- **Back to Booking Page Button:** directs user back to the bus booking page.

5. Troubleshooting

Common Issues

- Ensure all required fields are filled before attempting to book the bus.
- If encountering persistent issues, contact the system administrator for assistance.

Cab User Activity Page

Table of Contents

1. Introduction

This page fetches the details of the activity of the particular user that is logged in displaying his/her ongoing ride if any as well as his/her ride history if there are any previous rides.

2. Getting Started

Accessing the Page

1. **Login:** Users must be logged into the smart city platform
2. **Navigation:** Once logged in, navigate to the transport section.
3. **Cab Feature:** Within the transport module, locate and click on the "Cab Booking" option. On that page, the user should click on the ‘User Activity’ Page.

3. Features

Display the Ongoing Ride of the user

- Using the details filled by the user, we run an algorithm in the backend that fetches the ongoing ride of the user if there is any. In case there isn’t any, then a message box is displayed saying that the user doesn’t have any ongoing rides.
- On the card displaying the ongoing ride, the following details are displayed- source, destination, time that the ride had started, the cost of the ride, Driver name.

Option to end the Ongoing Ride

- The user will have to click on the button labelled ‘End Ride’ to end the ride, then accordingly the database will be updated and the page will then be refreshed, and the ride which has ended will then be displayed now in the previous rides card.

Display all the previous rides of the user

- On the card displaying each previous ride, the following details are displayed- source, destination, time that the ride had started, the cost of the ride.

4. User Interface

Display Fields

1. **Card displaying Ongoing Ride:** Card which shows the ongoing ride with relevant parameters.
2. **Card displaying Previous Rides:** Multiple cards which show the previous ride with relevant parameters.

3. **Destination Location:** Label which shows the destination location for the cab

Buttons and Controls

- **End ride Button:** Ends ongoing ride of the user.
- **Back to Booking Page Button:** directs user back to the cab booking page.

5. Troubleshooting

Common Issues

- Ensure all required fields are filled before attempting to book the cab.
- If encountering persistent issues, contact the system administrator for assistance.

Cab Driver Info Page

Table of Contents

1. Introduction

This page fetches the details of the activity of the particular user that is logged in displaying his/her ongoing ride if any as well as his/her ride history if there are any previous rides.

2. Getting Started

Accessing the Page

1. **Login:** Users must be logged into the smart city platform
2. **Navigation:** Once logged in, navigate to the transport section.
3. **Cab Feature:** Within the transport module, locate and click on the "Cab Booking" option. On that page, the user who is a driver should click on the ‘Are you a driver button’.

3. Features

Display the name and email of the Driver

- Using the SID of the user, we fetch the name and email of the user and display it on different textboxes.

Display the Ongoing Ride of the driver

- Using the details of the driver fetched from the database, we run an algorithm in the backend that fetches the ongoing ride of the driver, if there is any. In case there isn't any, then a message box is displayed saying that the driver doesn't have any ongoing rides.
- On the card displaying the ongoing ride, the following details are displayed- source, destination, time that the ride had started, the cost per unit distance of the ride.

4. User Interface

Display Fields

1. **Card displaying Ongoing Ride:** Card which shows the ongoing ride with relevant parameters.
2. **Textbox displaying email:** shows the email of the driver logged in.
3. **Textbox displaying name:** shows the name of the driver logged in.

Buttons and Controls

- **Back to Booking Page Button:** directs user back to the cab booking page.

5. Troubleshooting

Common Issues

- Page can only be accessed if the user is a driver.
- If encountering persistent issues, contact the system administrator for assistance.

Bus Info Page

Table of Contents

1. Introduction

This page fetches all the details of the bus from the database calculating the cost for your distance travelled and also gives you information about bus driver. You can book the bus from this page and can get your ticket printed.

2. Getting Started

Accessing the Page

1. **Login:** Users must be logged into the smart city platform
2. **Navigation:** Once logged in, navigate to the transport section.
3. **Bus Feature:** Within the transport module, locate and click on the "Bus Booking" option. On that page, search for the buses in the bus booking page and then you will be directed to the bus results page and then click on the book button to go to this page.

3. Features

Display the information about the bus and bus driver

- Using the ID of the bus and the pickup and drop location it finds the total time of travel and driver name and number and other information like total capacity of the bus and price, we fetch these values and display it on different textboxes.

Calculates total cost according to your requirement

- Using the number of girls seat booked and boys seat booked it calculates the total cost of travel and display on the textbox.

Get your ticket printed as a pdf

- User can get his tickets printed by clicking on the button get tickets after the payment is made.

4. User Interface

Display Fields

1. **Textboxes displaying different information about the bus and bus drivers :** Card which shows the ongoing ride with relevant parameters.

Buttons and Controls

- **Back to Booking Page Button:** directs user back to the cab booking page.
- **Button to calculate total amount**.
- **Button to do payment of the bus ticket**
- **Button to get the pdf of your ticket downloaded to the my documents**

5. Troubleshooting

Common Issues

- Tickets can’t be printed if you haven’t paid for it.
- If encountering persistent issues, contact the system administrator for assistance.

Add Bus Page

Table of Contents

1. Introduction

This page allows a specific govt official, namely the transport minister to add a new bus route with the parameters specified.

2. Getting Started

Accessing the Page

1. **Login:** Users must be logged into the smart city platform
2. **Navigation:** Once logged in, navigate to the transport section.
3. **Bus Feature:** Within the transport module, locate and click on the "Transport Ministry" option.

3. Features

Add a New Bus with specified parameters

4. User Interface

Display Fields

1. Labels specifying what all parameters need to be filled for the bas to be added

Buttons and Controls

- **Back to main Page Button:** directs user back to the transport landing page.
- **Button to confirm the addition of new bus**

5. Troubleshooting

Common Issues

- All parameters need to be filled to be able to add the bus with correct parameters.
- If encountering persistent issues, contact the system administrator for assistance.

Technical Documentation- Transport Module

Cab Booking Page

Table of Contents

1. Introduction

The Cab Booking page is a component of the Smart City Management system designed to allow registered users to cab a cab from the given set of source locations and destination locations. This technical documentation provides insights into the architecture, functionality, and implementation details of the backend code.

2. System Architecture

The backend code utilizes a simple three-tier architecture:

- Presentation Layer: Implemented using VB.NET and Windows Forms for the user interface.
- Business Logic Layer: Contains the backend code responsible for handling user interactions and business logic.
- Data Access Layer: Interacts with the MySQL database to perform CRUD (Create, Read, Update, Delete) operations.

3. Dependencies

The backend code relies on the following dependencies:

- `MySql.Data.MySqlClient`: MySQL database connectivity library.

- `System.Text`: Standard library for handling string operations.
- `System.Windows.Forms.VisualStyles.VisualStyleElement`: Library for UI elements in Windows Forms.

4. Database Schema

The backend code interacts with a MySQL database named `smart_city_management`. The relevant table for this page is `User`, which contains fields such as `SID`, `EmailAddress`, and `Designation`.

5. Backend Functionality

Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click:

- This is an event handler for the click event of Button1.
- It retrieves the selected values from ComboBox1 (FromLocation) and ComboBox2 (ToLocation), as well as the selected date and time from DateTimePicker1.
- Converts the selected date and time to a SQL-formatted string.
- Clears the controls of **panel1** in the main form.
- Instantiates a new **transport_cabavailable** form, sets it as a child of **panel1**, and displays it.

Private Sub transport_cabbooking_Load(sender As Object, e As EventArgs) Handles MyBase.Load:

- This is an event handler for the form load event.
- Sets the initial state of the form to Maximized, sets the form border style to Sizable, hides the control box (minimize, maximize, close buttons), and sets the form title to an empty string.

Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click:

- This is an event handler for the click event of Button2.
- Executes a SQL query to fetch the designation of the user based on their SID (Staff ID).
- If the designation is "Driver", clears the controls of **panel1**, instantiates a new **transport_cabdriverinfo** form, sets it as a child of **panel1**, and displays it. Otherwise, displays a message indicating that the control is only accessible to drivers.

Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click:

- This is an event handler for the click event of Button3.
- Clears the controls of **panel1** in the main form.
- Instantiates a new **transport_cabuserhistory** form, sets it as a child of **panel1**, and displays it.

Cab Available Page

Table of Contents

1. Introduction

The Cab Available page is a component of the Smart City Management system designed to allow registered users to view a cab to book from the available set of cabs from the given set of source locations and destination locations. This technical documentation provides insights into the architecture, functionality, and implementation details of the backend code.

2. System Architecture

The backend code utilizes a simple three-tier architecture:

- Presentation Layer: Implemented using VB.NET and Windows Forms for the user interface.
- Business Logic Layer: Contains the backend code responsible for handling user interactions and business logic.
- Data Access Layer: Interacts with the MySQL database to perform CRUD (Create, Read, Update, Delete) operations.

3. Dependencies

The backend code relies on the following dependencies:

- `MySql.Data.MySqlClient`: MySQL database connectivity library.
- `System.Text`: Standard library for handling string operations.
- `System.Windows.Forms.VisualStyles.VisualStyleElement`: Library for UI elements in Windows Forms.

4. Database Schema

The backend code interacts with a MySQL database named `smart_city_management`. The relevant table for this page is `User`, `running_cabs`, `all_cabs`, `cab_paths`, `cab_nodes`, `cab_user`.

5. Backend Functionality

Private Sub CardButton_Click(sender As Object, e As EventArgs):

- This function handles the click event of a card button.
- It retrieves user data (name, email, contact number) based on the user's Staff ID (SID) from the User table in the database.
- It then checks if the user has already booked a cab. If not, it inserts a new record into the **cab_user** table with booking details. If the user has already booked a cab, it updates the existing record.
- It also updates the **running_cabs** table with the new booking details if a new cab is booked or extends the current path if the user extends their ride.
- Finally, it displays a message confirming the cab booking.

Private Function GetNewPathId(fromLocation As String, toLocation As String) As Integer:

- This function retrieves the path ID from the database based on the provided 'fromLocation' and 'toLocation'.
- It takes the 'fromLocation' and 'toLocation' as input parameters and returns the corresponding path ID.

Private Function extend_path(cabId As Integer) As Boolean:

- This function checks whether the current ride can be extended by comparing the existing path of the cab with the new path.
- It returns a Boolean value indicating whether the new path is a prefix of the existing path.

Private Function GetNodeId(location As String) As Integer:

- This function retrieves the node ID corresponding to a location from the database.
- It takes a location name as input and returns the corresponding node ID.

Private Sub RetrieveAndStoreNodesList():

- This function retrieves the nodes list for each cab from the database and stores it in a dictionary.
- It iterates over the retrieved data, processes it, and stores it in the **pathNodesMap** dictionary.

Function CalculateArrivalTimes():

- This function calculates the arrival times for each cab based on its path and departure time.
- It iterates over each cab in **relpathNodesMap**, calculates the total time for the journey, and determines the arrival time for each cab.

Function GetEdgeWeight(sourceNodeId As Integer, targetNodeId As Integer) As Integer:

- This function retrieves the edge weight (travel time) between two nodes (sourceNodeId and targetNodeId) from the database.
- It takes source and target node IDs as input parameters and returns the corresponding edge weight.

Function GetDepartureTime(cabId As Integer) As DateTime:

- This function retrieves the departure time of a cab from the database based on its ID (cabId).
- It takes the cab ID as input and returns the corresponding departure time.

Transport_cabavailable_load()

Clearing Data Structures: It clears several data structures (**pathNodesMap**, **relpathNodesMap**, and **cab_arrival_times**) to prepare them for fresh data.

Window Settings: It sets up the window state, border style, and control box properties of the form.

Database Operations:

- It opens a database connection (**conn2**).
- Executes a SELECT query to retrieve data from the **running_cabs** table.
- Retrieves the node IDs for the from and to locations using the **GetNodeId** function.
- Calls the **RetrieveAndStoreNodesList** function to retrieve and store node lists.
- Calculates arrival times for cabs using the **CalculateArrivalTimes** function.

Processing Cabs:

- It iterates through the cab arrival times.
- Retrieves additional data (driver name) for each cab from the database.
- Compares arrival times with the specified booking time.
- If a cab is available and matches the required route, it:
  - Retrieves additional data (price, node list, vacancies) from the database.
  - Calculates the total price based on the route and time.
  - Checks if the cab contains the required route as a subpath of the current cab route.
  - Displays the cab details on a card.
  - Attaches event handlers to the card buttons for payment and confirmation.
  - Updates database records related to cab booking and vacancies.

Database Connection Handling: It ensures proper opening and closing of the database connection (**conn2**).

Vacancy Check:

- We query the database to retrieve the number of vacancies for the current cab (**vacancyquery**).
- If there are vacancies (**vacancies > 0**), it means the cab is available for booking.

Creating UI Cards:

- If a cab is available, it creates a new UI card (**transport_cabCards**) to display information about the cab, such as the pickup and drop-off locations, arrival time, driver name, and price.
- Two event handlers are added to the UI card buttons (**Button1** and **Button2**) for different actions.

Button2 Click Event Handler:

- This event handler (**card.Button2.Click**) is triggered when the user clicks on a button (presumably to book the cab).
- It sets some global variables related to banking, such as the receiver username and the payment amount.
- Then, it clears some panels (**Banking_Main.Panel1** and **Newsletter_Main.Panel1**) and loads new forms (**Banking_Homepage** and another form) onto those panels.
- This section also fetches user information (name, email, contact number) from the database based on the user's session ID (**SID**).
- Depending on whether the user has made a payment (**banking_payment_done**), it either shows a message to make the payment or proceeds to book the cab.
- If the user has made the payment, it inserts or updates booking details into the **cab_user** table in the database.

Button1 Click Event Handler:

- This event handler (**card.Button1.Click**) is triggered when the user clicks on another button (possibly to view details or take another action).
- Similar to the Button2 event handler, it first checks if the user has made the payment.
- If the payment is done, it sets some variables related to the selected cab, such as the cost and the new path ID.
- It then updates the **running_cabs** table in the database with the new path ID, decrements the vacancies, and updates the destination location.
- Finally, it shows a confirmation message for the cab booking.

Add Card to Panel:

- After setting up the UI card and attaching event handlers, the card is added to a FlowLayoutPanel (**FlowLayoutPanel1**) for display to the user.

Finding an Available Cab:

- It begins by querying the database to find an available cab from the **all_cabs** table where **is_available = 1**. It limits the query to return only one cab (**LIMIT 1**).
- If an available cab is found, it proceeds to retrieve its price, driver name, and path nodes list.

Calculating Price and Total Minutes:

- Once the available cab is identified, it calculates the total price for the trip based on the path nodes list and the price per unit time (presumably per minute).

- It iterates over the path nodes list, calculating the total time required for the trip by summing up the edge weights (in minutes) between consecutive nodes.

Fetching Driver Name:

- It fetches the driver's name associated with the available cab from the database.

Creating UI Card and Event Handlers:

- Similar to the previous code, it creates a UI card to display information about the available cab and attaches event handlers to its buttons (**Button1** and **Button2**).

Button2 Click Event Handler:

- This event handler is responsible for initiating the payment process and redirecting the user to the banking homepage for payment.

Button1 Click Event Handler:

- This event handler is triggered when the user clicks on a button to book the cab.
- It first checks if the user has made the payment. If not, it prompts the user to make the payment first.
- If the payment is done, it proceeds to book the cab by inserting or updating booking details into the **cab_user** table in the database.
- It also updates the availability of the cab in the **all_cabs** table and inserts a new entry into the **running_cabs** table to track the booked cab's status.

Displaying UI Card:

- Finally, the UI card containing information about the available cab is added to the **FlowLayoutPanel** for display to the user.

Error Handling:

- The code is wrapped in a try-catch block to handle any exceptions that may occur during the execution of database queries or other operations. If an error occurs, it displays an error message to the user.

User Activity Page

Table of Contents

1. Introduction

The User Activity page is a component of the Smart City Management system designed to allow registered users view their ongoing as well as previous rides. This technical documentation provides insights into the architecture, functionality, and implementation details of the backend code.

2. System Architecture

The backend code utilizes a simple three-tier architecture:

- Presentation Layer: Implemented using VB.NET and Windows Forms for the user interface.
- Business Logic Layer: Contains the backend code responsible for handling user interactions and business logic.
- Data Access Layer: Interacts with the MySQL database to perform CRUD (Create, Read, Update, Delete) operations.

3. Dependencies

The backend code relies on the following dependencies:

- `MySql.Data.MySqlClient`: MySQL database connectivity library.
- `System.Text`: Standard library for handling string operations.
- `System.Windows.Forms.VisualStyles.VisualStyleElement`: Library for UI elements in Windows Forms.

4. Database Schema

The backend code interacts with a MySQL database named `smart_city_management`. The relevant table for this page is `User`, `running_cabs`, `all_cabs`, `cab_paths`, `cab_nodes`, `cab_user`.

5. Backend Functionality

transport_cabuserhistory_Load: This is an event handler for the form's Load event. It executes when the form is loaded. Within this function:

- The form's window state, border style, control box, and text are customized to maximize the window, make it sizable, remove the control box (minimize, maximize, close buttons), and set the title text to an empty string.
- It retrieves the user's email address based on their SID (Session ID) from the "User" table in the database.
- It checks if the user has any ongoing rides. If not, it shows a message box indicating that there are no ongoing rides.

- If the user has an ongoing ride, it retrieves details of the current ride such as driver name, pickup, and drop-off locations, departure time, and cost from the database.
- It populates a UI card with the details of the ongoing ride and adds a click event handler to end the ride.
- It retrieves the user's ride history from the "cab_history" table and populates UI cards with the details of previous rides.
- It handles any exceptions that may occur during the process and displays an error message box if an exception occurs.

Button1_Click: This event handler executes when Button1 (presumably a back or home button) is clicked. It clears the controls in a panel named "mypanel.panel1" and replaces them with a new instance of the "transport_cabbooking" form.

Button4 Click Handler

- It opens a new connection to the database (**conn2**).
- It inserts the details of the current ride into the "cab_history" table.
- It updates the "cab_user" table to indicate that the ride has ended for the current user.
- It retrieves the maximum limit of vacancies for the current ride from the "all_cabs" table.
- It updates the "running_cabs" table to increment the vacancies count for the current ride.
- If the vacancies count equals the maximum limit, it deletes the entry from the "running_cabs" table and updates the "all_cabs" table to mark the cab as available.
- It displays a message box indicating that the ride has ended successfully.
- It closes the current form, clears controls in a panel named "mypanel.panel1", creates a new instance of the "transport_cabuserhistory" form, adds it to the panel, and displays it.
- It handles any exceptions that may occur during the process and displays an error message box if an exception occurs.

Cab Driver Info Page

Table of Contents

1. Introduction

The Driver Info page is a component of the Smart City Management system designed to allow registered users who are drivers to information as well as their ongoing ride if any. This technical documentation provides insights into the architecture, functionality, and implementation details of the backend code.

2. System Architecture

The backend code utilizes a simple three-tier architecture:

- Presentation Layer: Implemented using VB.NET and Windows Forms for the user interface.
- Business Logic Layer: Contains the backend code responsible for handling user interactions and business logic.
- Data Access Layer: Interacts with the MySQL database to perform CRUD (Create, Read, Update, Delete) operations.

3. Dependencies

The backend code relies on the following dependencies:

- `MySql.Data.MySqlClient`: MySQL database connectivity library.
- `System.Text`: Standard library for handling string operations.
- `System.Windows.Forms.VisualStyles.VisualStyleElement`: Library for UI elements in Windows Forms.

4. Database Schema

The backend code interacts with a MySQL database named `smart_city_management`. The relevant table for this page is `User`, `running_cabs, all_cabs, cab_paths, cab_nodes, cab_user.`

5. Backend Functionality

transport_cabdriverinfo_Load subroutine:

- This subroutine is an event handler for the **Load** event of the **transport_cabdriverinfo** form.
- It is triggered when the form is loaded.
- It sets various properties of the form such as **WindowState**, **FormBorderStyle**, **ControlBox**, and **Text** to customize its appearance.
- Inside a **Try** block, it performs database operations to retrieve information about a cab driver and their previous rides.
- It first retrieves the **SID** (System Identifier) of the cab driver from the **transport_landingPage** form.
- Then, it executes SQL queries to fetch the cab driver's name, email address, contact number, and cab ID from the database.
- Next, it retrieves information about the cab's previous rides such as departure time, from location, to location, and price.
- It dynamically creates UI elements (**previousRideCard**) to display the details of each previous ride and adds them to a **FlowLayoutPanel**.
- If any error occurs during the process, it displays a message box with the error message.
- Finally, it closes the database connection in the **Finally** block.

Button4_Click subroutine:

- This subroutine is an event handler for the **Click** event of **Button4**.
- It is triggered when **Button4** is clicked.
- It clears the controls of **panel1** inside **mypanel**.
- It creates a new instance of the **transport_cabbooking** form.
- It sets the **TopLevel** property of the form to **False**.
- It adds the **transport_cabbooking** form to **panel1** inside **mypanel**.
- It shows the **transport_cabbooking** form.

Bus Booking Page

Table of Contents

1. Introduction

The Bus Booking page is a component of the Smart City Management system designed to allow registered users to book a bus if any is available for the route that he/she has mentioned. This technical documentation provides insights into the architecture, functionality, and implementation details of the backend code.

2. System Architecture

The backend code utilizes a simple three-tier architecture:

- Presentation Layer: Implemented using VB.NET and Windows Forms for the user interface.
- Business Logic Layer: Contains the backend code responsible for handling user interactions and business logic.
- Data Access Layer: Interacts with the MySQL database to perform CRUD (Create, Read, Update, Delete) operations.

3. Dependencies

The backend code relies on the following dependencies:

- `MySql.Data.MySqlClient`: MySQL database connectivity library.
- `System.Text`: Standard library for handling string operations.
- `System.Windows.Forms.VisualStyles.VisualStyleElement`: Library for UI elements in Windows Forms.

4. Database Schema

The backend code interacts with a MySQL database named `smart_city_management`. The relevant table for this page is `User, bus, path etc`

5. Backend Functionality

transport_busbooking_Load(sender As Object, e As EventArgs):

- This function is an event handler triggered when the form **transport_busbooking** loads.
- It attempts to establish a connection to the database and calls **RetrieveBusInfo** to fetch information about a specific bus.

RetrieveBusInfo(busId As Integer):

- Queries the database to retrieve information about a bus using its ID.
- If the bus is found, it populates various labels on the form with details like driver name, seat availability, and prices.

RetrieveFromAndEndPt(pathId As Integer) As Tuple(Of String, String):

- Queries the database to retrieve the starting and ending points of a route based on the given path ID.
- Returns a tuple containing the starting and ending points.
    - **Button3_Click(sender As Object, e As EventArgs):**
- Event handler for a button click that calculates the total price based on the number of seats entered in text boxes and the prices per seat fetched from the database.
- Updates labels to display the calculated total price.

Button1_Click(sender As Object, e As EventArgs):

- Event handler for a button click that loads another form (**transport_busSearch**) into a panel (**mypanel.panel1**) on the current form.

Button2_Click(sender As Object, e As EventArgs):

- Event handler for a button click that generates a PDF ticket based on the information entered by the user.
- Uses GemBox.Pdf library to create a PDF document containing details such as route, date, time, driver info, booked seats, and total price. Then saves the PDF file.

Button4_Click(sender As Object, e As EventArgs):

- Event handler for a button click that prepares for a payment process.
- Sets some global variables related to payment, clears panels, and navigates to another form (**Banking_Homepage**).

Bus Available Page

Table of Contents

1. Introduction

The Bus Available page is a component of the Smart City Management system designed to allow registered users to view the available buses if any on the route that he/she has mentioned. This technical documentation provides insights into the architecture, functionality, and implementation details of the backend code.

2. System Architecture

The backend code utilizes a simple three-tier architecture:

- Presentation Layer: Implemented using VB.NET and Windows Forms for the user interface.
- Business Logic Layer: Contains the backend code responsible for handling user interactions and business logic.
- Data Access Layer: Interacts with the MySQL database to perform CRUD (Create, Read, Update, Delete) operations.

3. Dependencies

The backend code relies on the following dependencies:

- `MySql.Data.MySqlClient`: MySQL database connectivity library.
- `System.Text`: Standard library for handling string operations.
- `System.Windows.Forms.VisualStyles.VisualStyleElement`: Library for UI elements in Windows Forms.

4. Database Schema

The backend code interacts with a MySQL database named `smart_city_management`. The relevant table for this page is `User, bus, path etc`

5. Backend Functionality

transport_busResults_Load(sender As Object, e As EventArgs):

- This function is an event handler triggered when the form **transport_busResults** loads.

- It retrieves data from the database regarding available buses and their routes.
- It then dynamically creates and adds instances of a user control (**searchComponent**) to a flow layout panel based on the retrieved data, displaying bus information like driver name, prices, and departure/arrival times.

CheckString(pathString As String, startPoint As String, endPoint As String) As Boolean:

- This function checks whether both the start and end points are present in a given path string and if the start point appears before the end point.
- It's used in **transport_busResults_Load** to filter out buses whose routes do not match the selected start and end points.

FindTime(pathString As String, point As String, departTime As DateTime) As DateTime:

- This function calculates the arrival time at a specified point along a route based on the departure time and the time taken to travel between points.
- It's used in **transport_busResults_Load** to determine the departure and arrival times for each bus displayed.

Button1_Click(sender As Object, e As EventArgs):

- Event handler for a button click that loads another form (**transport_busSearch**) into a panel (**mypanel.panel1**) on the current form.
- It allows users to navigate back to the bus search form from the results page.

Add Bus Page

Table of Contents

1. Introduction

The Add Bus page is a component of the Smart City Management system designed to allow a specific govt official to add a bus on a specific route specifying various parameters. This technical documentation provides insights into the architecture, functionality, and implementation details of the backend code.

2. System Architecture

The backend code utilizes a simple three-tier architecture:

- Presentation Layer: Implemented using VB.NET and Windows Forms for the user interface.
- Business Logic Layer: Contains the backend code responsible for handling user interactions and business logic.
- Data Access Layer: Interacts with the MySQL database to perform CRUD (Create, Read, Update, Delete) operations.

3. Dependencies

The backend code relies on the following dependencies:

- `MySql.Data.MySqlClient`: MySQL database connectivity library.
- `System.Text`: Standard library for handling string operations.
- `System.Windows.Forms.VisualStyles.VisualStyleElement`: Library for UI elements in Windows Forms.

4. Database Schema

The backend code interacts with a MySQL database named `smart_city_management`. The relevant table for this page is `User, bus, path etc`

5. Backend Functionality

InsertIntoBusTable:

- This function inserts a new record into the **bus** table in the database.
- It takes various parameters representing bus details such as ID, driver information, seat availability, prices, route ID, and departure time.
- The function constructs and executes an SQL query using parameterized commands to safely insert the data into the database. After execution, it displays a success or failure message to the user.

Button1_Click(sender As Object, e As EventArgs):

- Event handler for a button click that triggers the insertion of bus data into the database.
- It retrieves input values from text boxes and date time picker, validates them, and then calls the **InsertIntoBusTable** function to perform the insertion.
- If the input values are invalid, it displays an error message informing the user to enter valid numeric values.

Module 4 Technical Documentation

Administration Hierarchy

Features and Functionalities

1. This portal can be accessed only after user login and authentication.
2. This portal aims to visualize the hierarchy structure of our smart city.
3. Ministers would be able to view the requests sent to them using Ministers Request's Page.
4. Institute owners would be able to view the requests sent to them using Ministers Request's Page.
5. List of all people in the hierarchy can be seen by interacting with elements in this page.

Code Structure

Forms The following forms are available in administration portal:

1. Election portal
2. job listing in hierarchy
3. Work section* for ministers and owners

Flow Administration portal can accessible after the user has logged in through left panel. Admin home page is first visible through which all other forms can be accessible like election portal, work section*, etc.

This module displays the structure of the entire city.It also reflects the communication flow between various department.We decided to implement 5 ministries, which are Education, Health, Transport, finance and Municipality. Education department includes Education minister, Educational institutions owners, Teachers in each of the institution.Health department includes Health minister, Hospitals owners, Doctors.Transport department includes Transport minister, Transport officers, drivers.Commerce department includes Commerce minister, Municipal Officers, Vendors.Home ministry department includes Home minister,Police. Hence this structure shows communication flow, from top to bottom for commands and bottom to top for requests.

Features and functions of each page

1. Admin home
   This page can be accessed by pressing administration in side navbar.This page leads to all other pages related to administration.Each Button click is associated with changing the inner panel of the main form.This is achieved by global visual basic file.Similar approach is used for implementing back functionality also.
2. Job listing
   On interacting with the buttons of designations in home page, we get a page which lists the people having those designations.
3. Election Portal
   This page can be accessed from administration portal.This page leads to all other pages available in election portal.Each Button click is associated with changing the inner panel of the main form.This is achieved by global visual basic file.Similar approach is used for implementing back functionality also.
4. Work section*
   This button is only visible for people whose designation is administration like ministers and owners.

Election Portal

Features and Functionalities

1. Any person eligible i.e above 18 years can apply to become voter.After becoming a voter, his/her name will be available in voters list.
2. All rules for voting and nomination are visible for everyone.
3. Any person eligible can nominate if the election period is on.He can submit his agenda and manifesto(in a pdf format) and pay deposit, which link banking application to complete the payment.
4. For all people the people nominating and their details like agenda and manifesto are visible.
5. Election Results and people who won are visible.For people who won their votes secured and victory margin are also calculated.Newly elected people's Designations are changed.

Code Structure

Forms The following forms are available in election portal:

1. Election Dashboard
2. Voter application page
3. Voting Portal
4. Nomination Application Page
5. Voting Rules Page
6. Nomination Rules Page
7. Current Ministers Page
8. All Election Results Page
9. View Nominees Page

Flow Election portal can accessible after the user has logged in and entered administration page through left panel.Election Dashboard page is first visible through which all other forms can be accessible like voting portal,Nomination Application Page etc.

Any Person eligible to vote can enter into Voter Application page and apply.During elections any valid voter can Vote (if he has not already voted) through Voting Portal.Eligible candidates will be allowed to nominate through Nomination Application Page.Results,Current Ministers, Voting Rules,Nomination Rules, Nominees Pages are available for general information of all people.Nomination Application Page will lead to Payment Portal, through which deposit payment of any person occurs.

Features and functions of each page

1. Election DashBoard
   This page can be accessed from administration portal.This page leads to all other pages available in election portal.Each Button click is associated with changing the inner panel of the main form.This is achieved by global visual basic file.Similar approach is used for implementing back functionality also.

2. Voter Application Page
   This page can be accessed from Election dashboard.This page is only accessible if the person is not a voter and is eligible to apply for voter.
   Methods-To know whether the user is already a voter or not and to find out if he is eligible to become voter.MySQL queries to our database have been written.The results have been retrieved and analysed. If the person applies, his details would be added into voters table using INSERT MYSQL query.The votes will not exist until he votes(default -2 will be stored).The person can exit after applying.
   Edge case handling- If the person tries to reapply , an error message would be shown and his details wont be entered again into the voters table.

3. Voting Rules
   This page can be accessed from Election dashboard.This displays the general rules everyone must satisfy during voting.Back functionality is added in this page.

4. Nomination Rules
   This page can be accessed from Election dashboard.This displays the rules and details of the nomination application procedure.It also explains the eligibility criterion to contest in the elections.Back functionality is added in this page.

5. Nomination Application Page
   This page can be accessed from Election dashboard.Through this page any eligible person can contest in elections.Agenda, Manifesto as pdf are taken as input.
   Methods-All the edge cases are handled.The Nominees table is searched with SELECT MySQL query and checked to stop duplicate entries.After successful entry of details, candidates details are entered into Nominees table using INSERT MySQL query.Gembox extension is used to upload the pdf.Paydeposit is used to pay an amount of 20,000 required for nomination.

6. Voting portal
   This page can be accessed from Election dashboard.This page is only accessed during voting period which is decided by mayor.In rest of the times, an appropriate error message will be shown.
   Methods-All Nominee details are fetched from Nominees table using SELECT MySQL queries.A usercontrol is added which holds all the details regarding a nominee.His name, agenda,manifesto are fetched using MySQL queries.A checkbox is added to store the vote of the person i.e id of the chosen candidate is thus stored.NOTA option is also provided which is also appropriately stored.When Vote button is clicked, the stored votes are added to voter table.
   Edge cases- Cases like is the person a voter, has he checked all the options are appropriately handled and tested exhaustively.Manifesto is downloaded and stored local path is shown in messagebox.

7. Current Ministers Page
   This page can be accessed from Election dashboard.A user control is created which holds all the details of ministers in each of the departments i.e Health ministry,Education Ministry,Transport Ministry,Home ministry,Finance Ministry.The Victory margin by which they have won and vote count are retrieved using JOIN and SELECT MySQL queries.If votes are tied, then age is considered to break the tie.

8. All Election Results Page
   This page can be accessed from Election dashboard.This is similar to Current Ministers Page, all candidates vote counts are also added in this page.

9. View Nominees Page
   This page can be accessed from Election dashboard.This page is for general information of people, to help them understand their candidates well before voting.

Complaints and Redressal Portal

Features and Functionalities

1. Users who encounter any dissatisfaction with the service can file a complaint directed towards either an individual or a group of people, depending on the nature of the issue.
2. Users have the ability to monitor their complaints and observe real-time progress within the history section*.
3. Individuals who receive complaints can access them through the redressal portal and take appropriate actions to address them, either partially or fully, depending on the nature of the issue.
4. If they are employed at a bank or police station, they can specifically view complaints filed within that institution.

Code Structure

Forms The following forms are available in election portal:

1. Complaints Dashboard
2. Institutes List Page
3. Complaints submission Page
4. Complaints History Page
5. Redressal Home
6. Complaints Resolving Page

Flow The Complaints portal can be accessed once the user logs in and navigates to the Complaints page via the left panel. The main page of the portal is the Complaints Dashboard, from which all other section*s can be accessed, such as the Institutes List Page, Complaint Submission Form and Complaints History Page.

The Redressal portal can be accessed once the user logs in and navigates to the Redressal page via the left panel. When accessing the redressal portal, users can easily view complaints directed to them or their organization. They have the ability to access and process each complaint efficiently, taking necessary actions for resolution. This streamlined process ensures prompt attention and effective handling of grievances.

Features and functions of each page

1. Complaints Dashboard
   This page can be accessed from Complaint portal.This page leads to all other pages available in Complaint portal.Each Button click is associated with changing the inner panel of the main form.This is achieved by global visual basic file.Similar approach is used for implementing back functionality also.
2. Institutes List Page
   This page can be accessed through Complaints Dashboard. This page shows the list of Institutes in cases related to Education , Health, Law.
3. Complaints submission Page
   This page will be showed after selecting institutes in case of Education, Health, Law. Users can file complaint to one or more members working in Institution from the list shown. In case of Law the complaint filed goes directly to the selected Institute. In case of Transport, this page can be accessed directly from complaints dashboard. In case of commerce, this page can be directly accessed from complaints dashboard and the complaint goes directly to the Bank.

4. Complaints History Page
   This page can be accessed from complaints dashboard. Users can view their past resolved complaints and also track the status of unresolved complaints.

5. Redressal Home
   This page can be accessed from Redressal portal.Through this page user can view all the complaints addressed to his organization if he is a Bank employee or a Police officer and complaints addressed to him if he works in any other Institute.

6. Complaints Resolving Page
   This page can be accessed from Redressal Home once the user clicks on the Resolve button on a particular complaint. Here the user can see more information about the complaint and decide to either resolve it immediately or save it for resolving latter.

Module 4 User Documentation

Election Portal

Election Dashboard which can be accessed through Administration Portal.Election portal can accessible after the user has logged in and entered administration page through left panel.Election Dashboard page is first visible through which all other forms can be acces- sible like voting portal,Nomination Application Page etc. Any Person eligible to vote can enter into Voter Application page and apply.During elections any valid voter can Vote (if he has not already voted) through Voting Portal.Eligible candidates will be allowed to nominate through Nomination Application Page.Results,Current Ministers, Voting Rules,Nomination Rules, Nominees Pages are available for general information of all people.Nomination Application Page will lead to Payment Portal, through which deposit payment of any person occurs.

Complaints Portal

This platform acts as a central point for citizens to report service issues or theft incidents. It allows users to track their complaints and view responses. Whether itâs service quality, administrative matters, or theft concerns, individuals can voice their grievances and stay updated on resolutions, promoting transparency.

Upon accessing the complaints portal, users are greeted with two primary options: either to review the status of previous complaints or initiate the process of registering a new one, ensuring flexibility and efficiency in addressing their concerns. For those opting to check the status of prior complaints, the system seamlessly retrieves relevant details from the complaint database by utilizing the userâs unique identification, offering transparency and easy access to historical information.

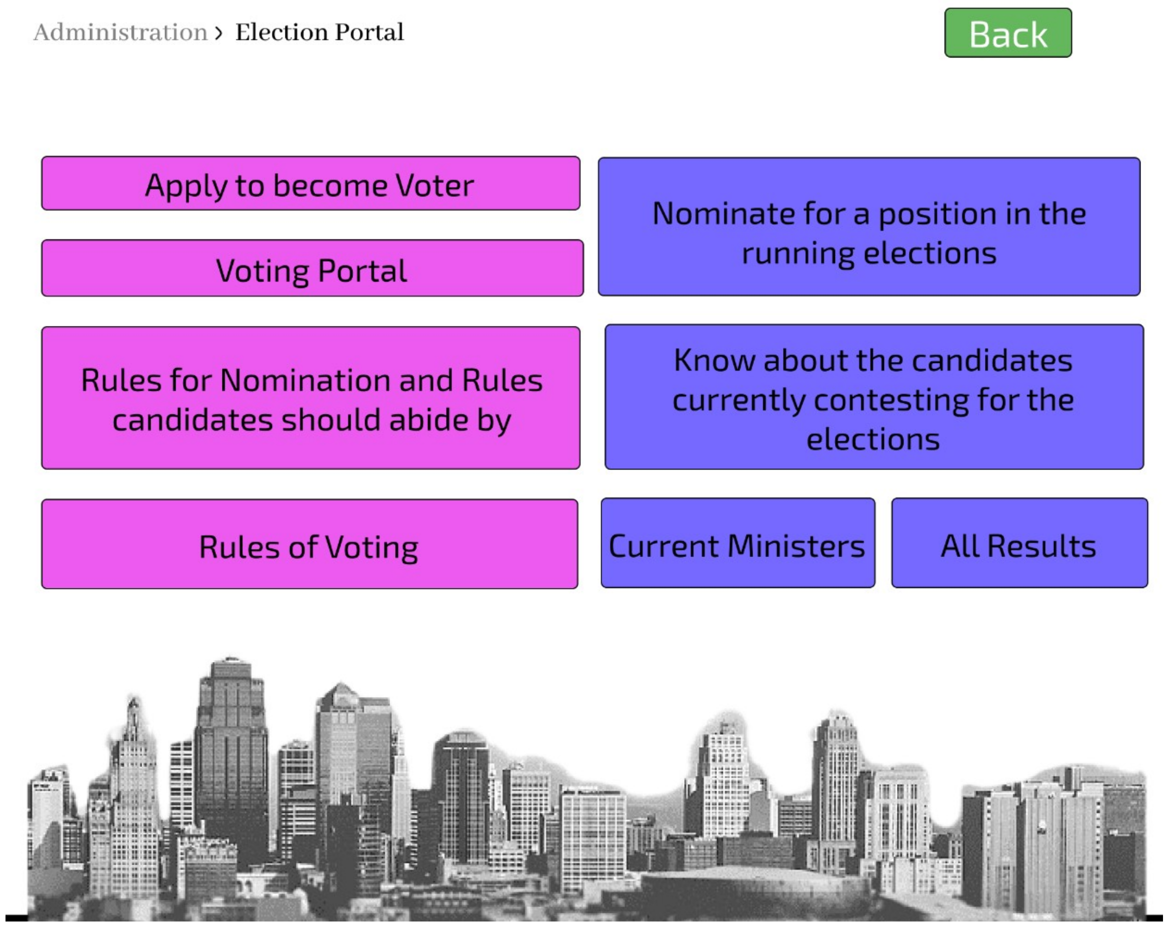

Figure 1: Election Dashboard with links to all other pages.The user can access these pages through this dashboard.

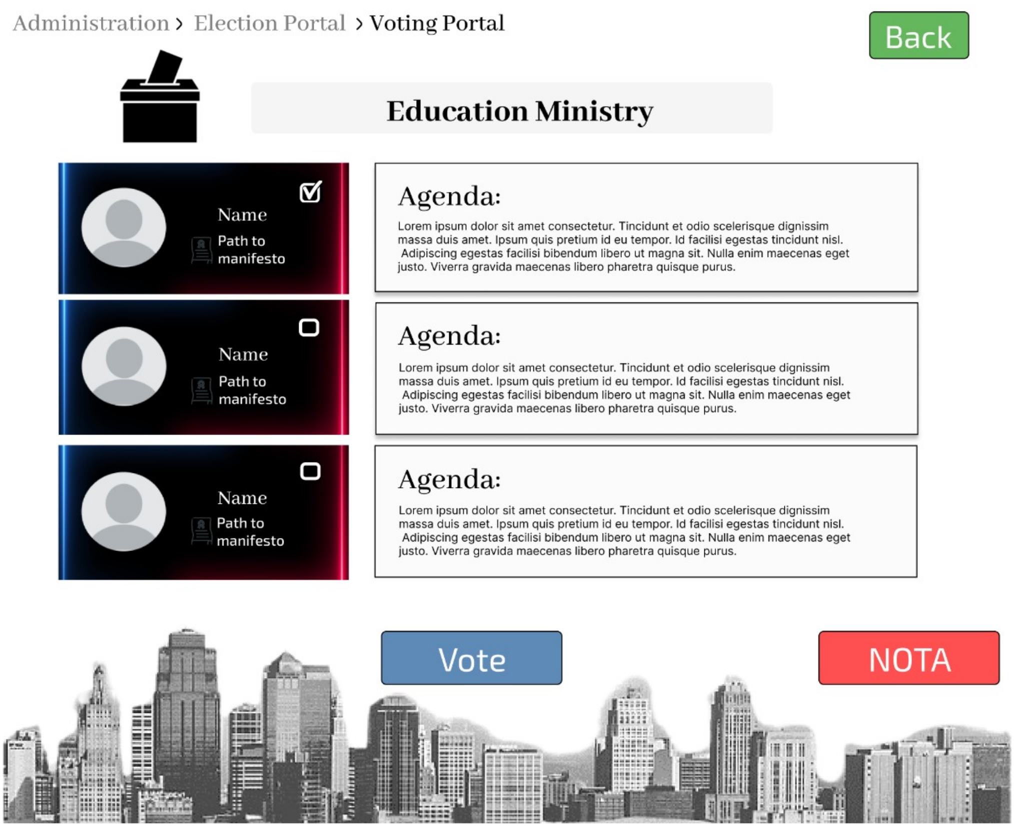

Figure 2: Through this page eligible candidates can apply to become voters.

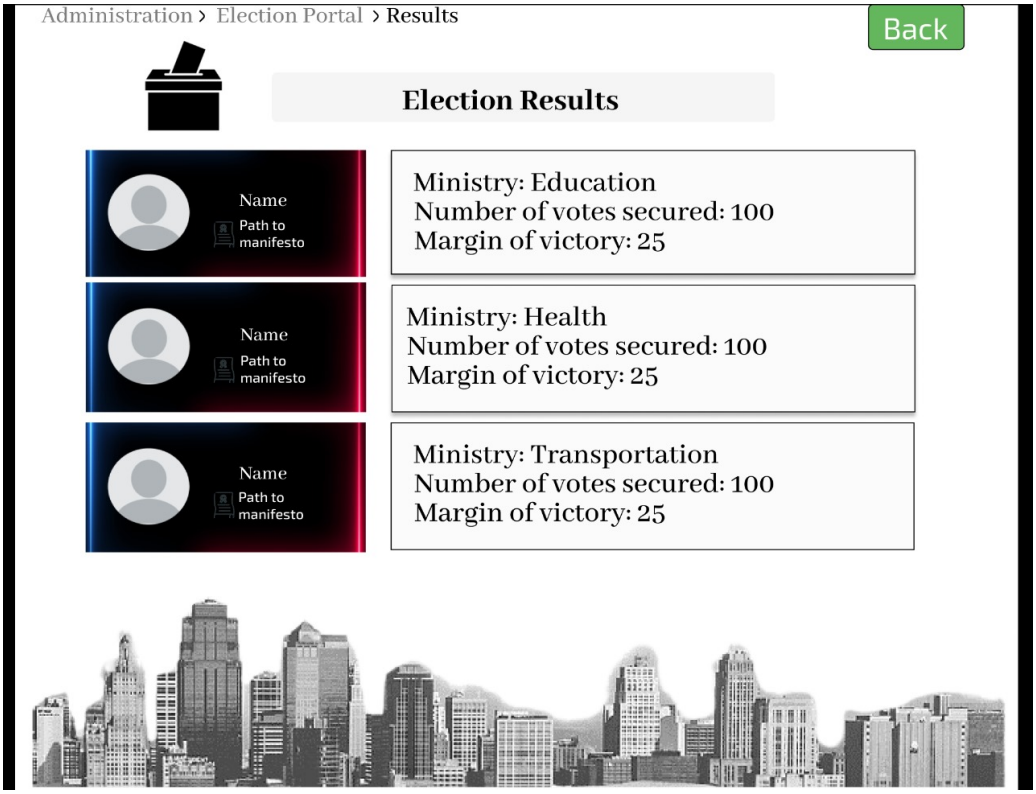

Figure 3: This shows the current ministers along with their votecount, victory margin, link to manifesto and name.

Figure 4: Through this page candidates can nominate to contest in the elections.

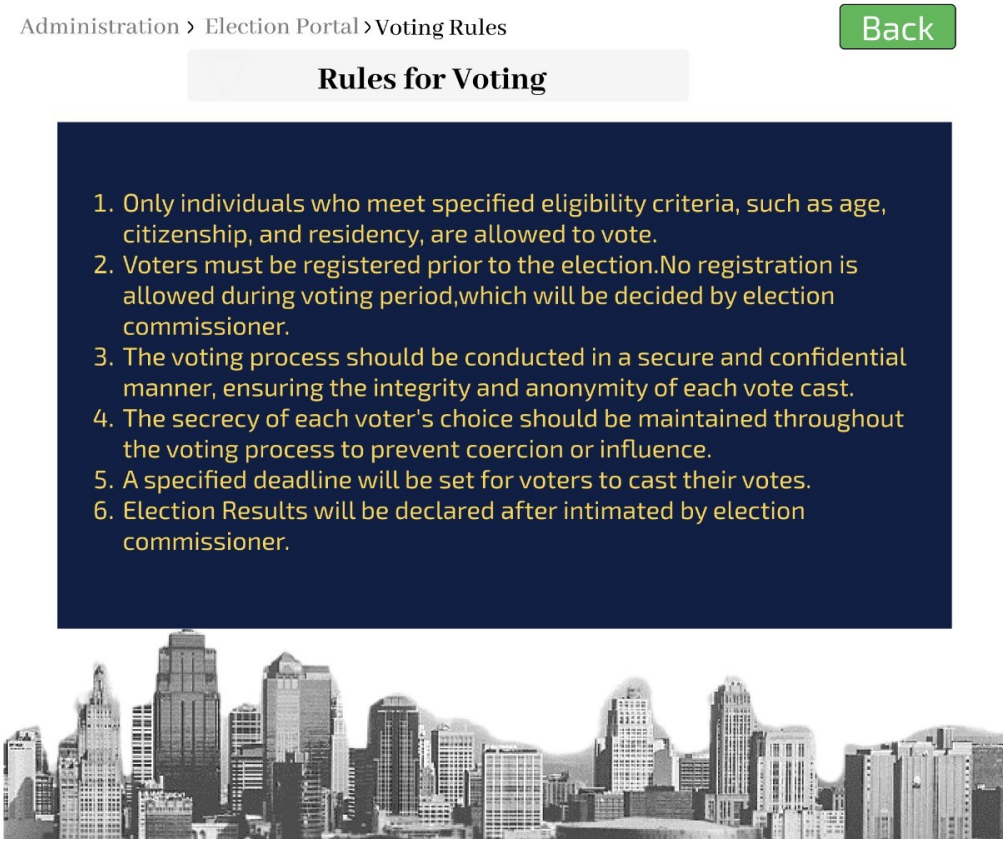

Figure 5: Rules to be followed during voting are given in this page.

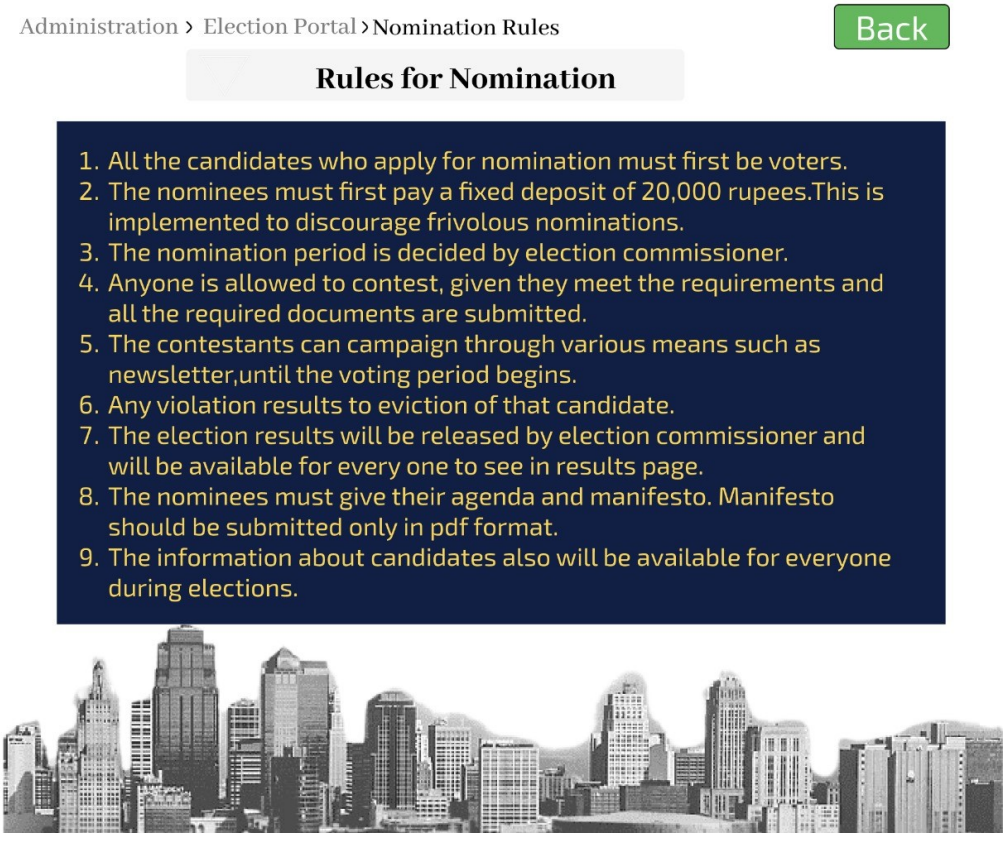

Figure 6: Rules to be followed by Nominating condidates are mentioned here.

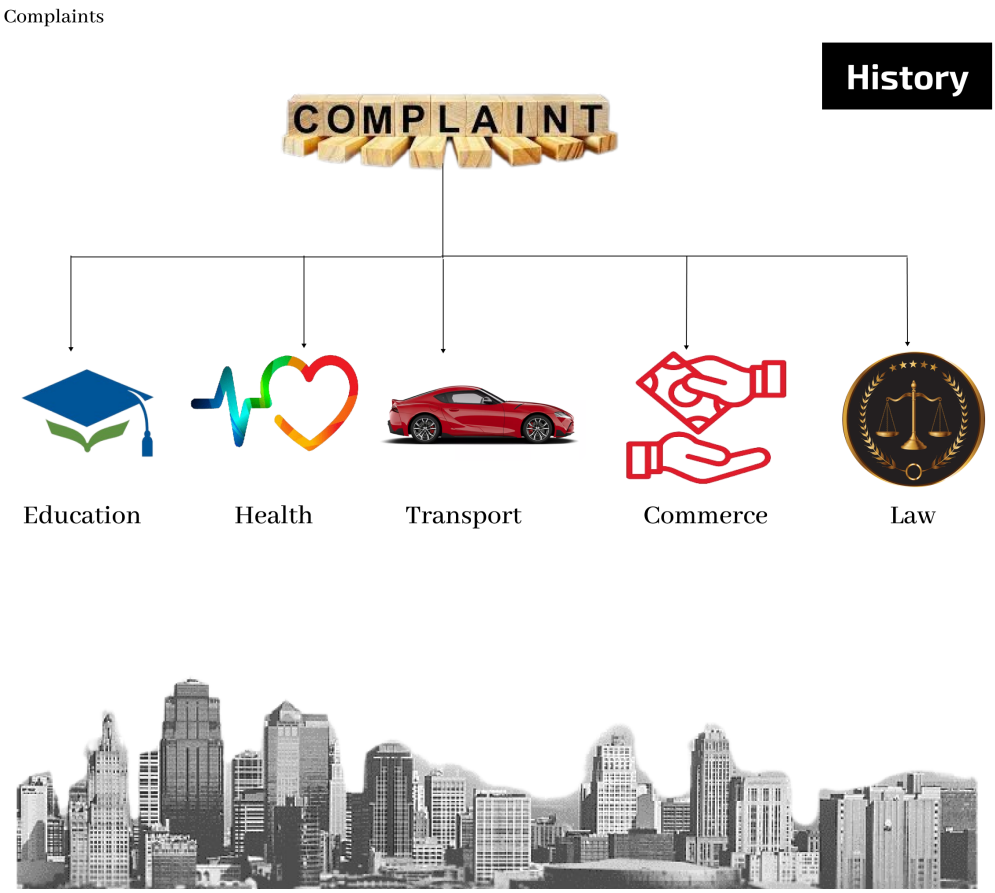

Figure 7: Complaint Page: The first page on accessing the complaints portal.

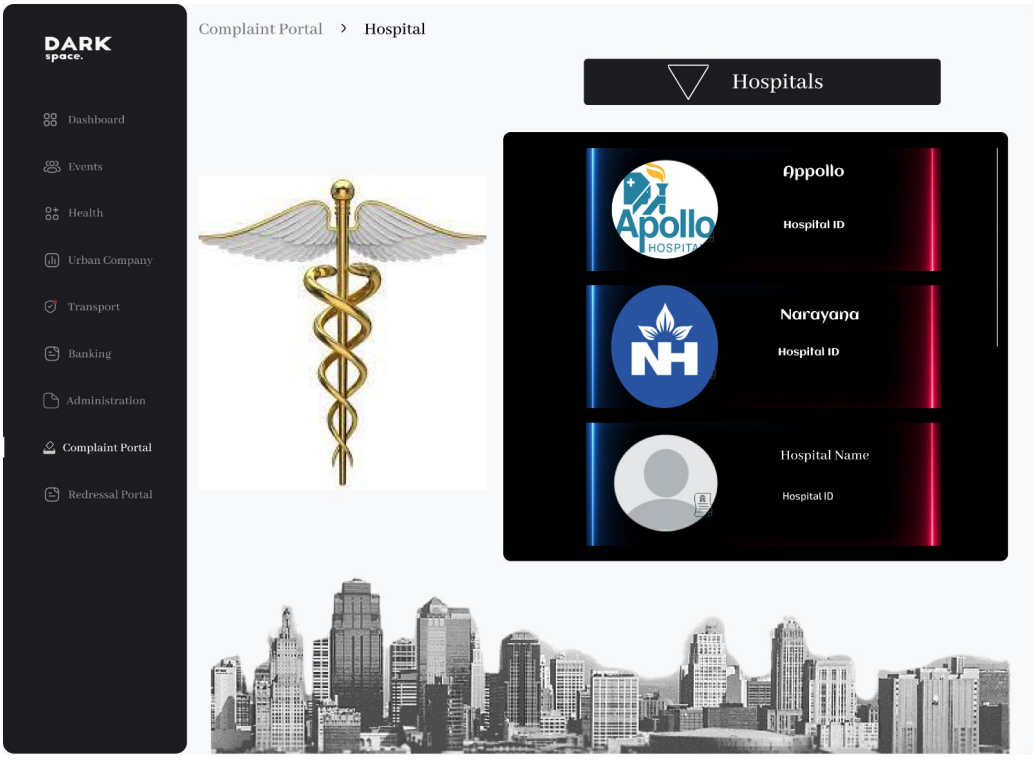

Figure 8: Choosing the hospital for which complaint to beÂ forwarder.

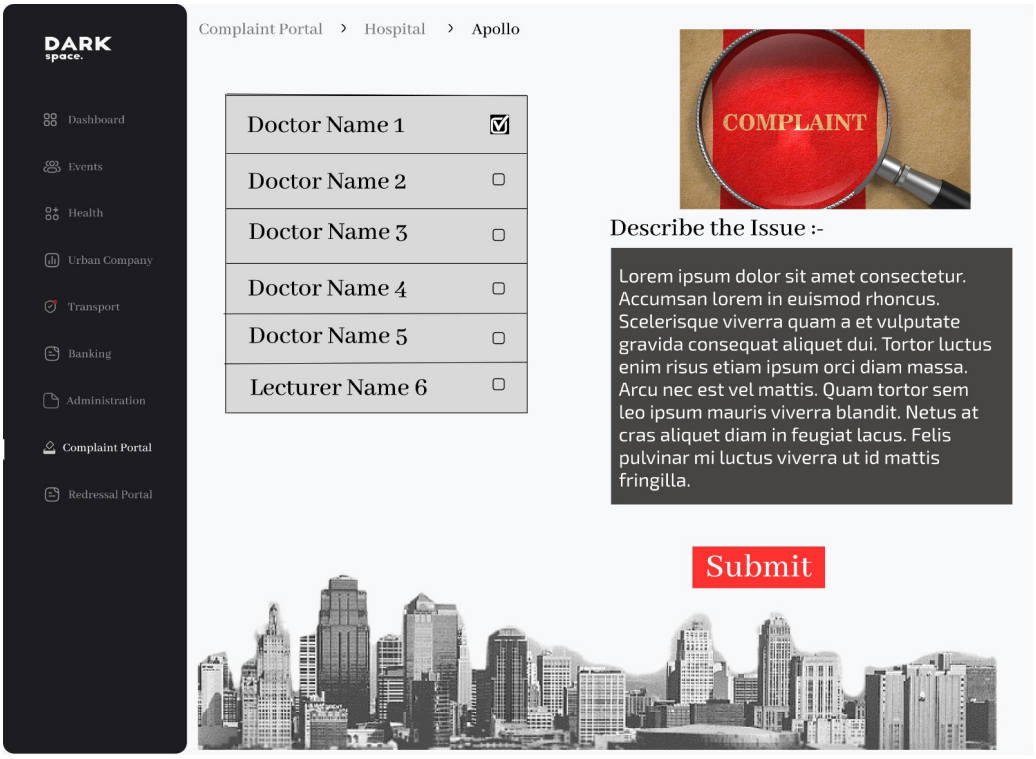

Figure 9: selecting the people to which complaints to beÂ forwarded.

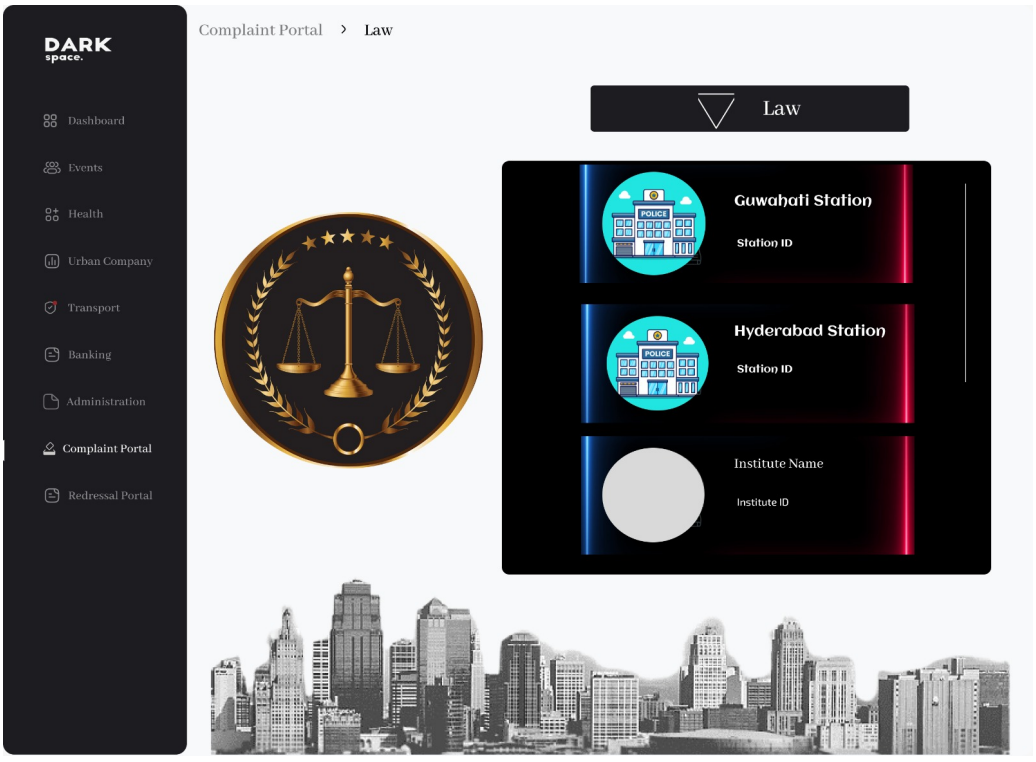

Figure 10: selecting the station for which complaint to beÂ forwarded.

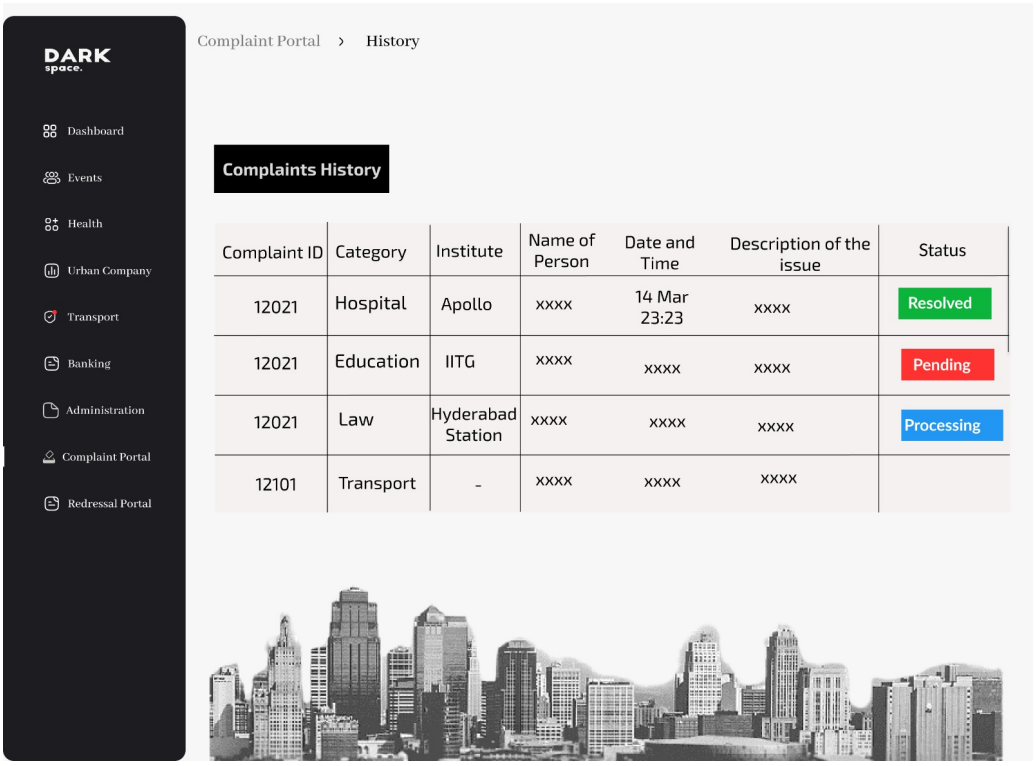

Figure 11: History page: for viewing the status of the complaints.

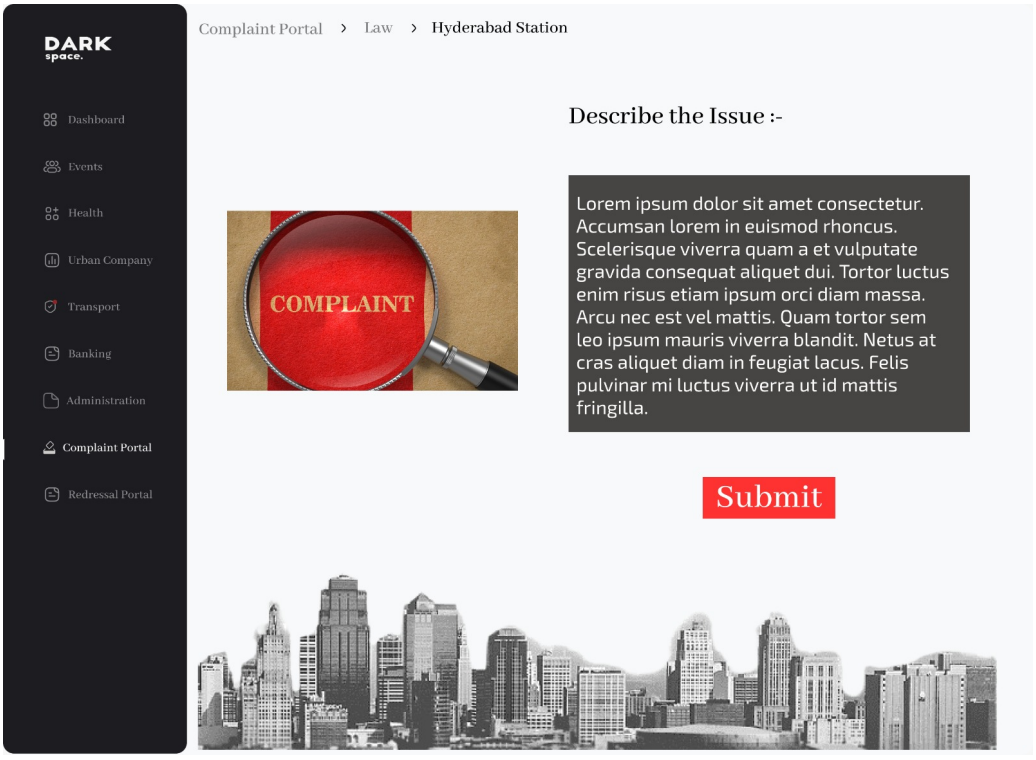

Figure 12: describing the complaint when it is related to Law.

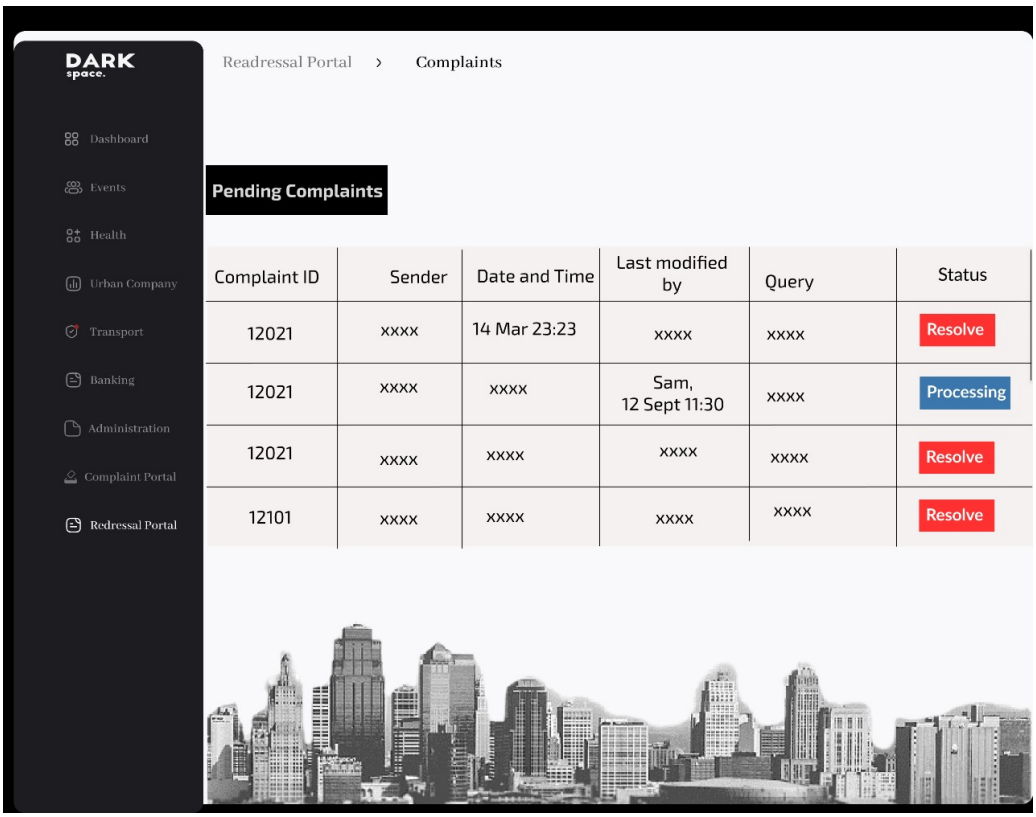

Figure 13: Redressal Home: The first page on accessing the Reddresal portal. Users can view pending complaints

Redressal Portal

In the redressal portal, individuals with access to forwarded complaints can view them and exercise autonomy in selecting which ones to address based on personal interest or priority. However, in cases involving law or banking complaints, all personnel within the relevant institution, like the bank or station, have visibility of these issues. This inclusive system allows employees to choose complaints that match their expertise or departmental responsibilities for prompt resolution. After selecting acomplaint, they can either reply with a solution or exit the process.

Module 5 User and Technical Documentation

Banking Service

Introduction

The Banking Service module stands as a foundational pillar within the smart city management platform, offering a robust platform for managing financial transactions and services. Developed using Visual Basic within Visual Studio, this module provides users with a seamless banking experience while ensuring security, reliability, and efficiency. The backend code relies on the following dependencies:

- Microsoft.SqlServer: This imports namespaces related to Microsoft SQL Server, providing access to classes and functionality for interacting with SQL Server databases.
- MySql.Data.MySqlClient: This imports namespaces for MySQL database connectivity, allowing interaction with MySQL databases using the MySQL .NET Connector.
- System.IO: This imports the System.IO namespace, providing classes and functionality for input and output operations, such as reading from and writing to files and streams.

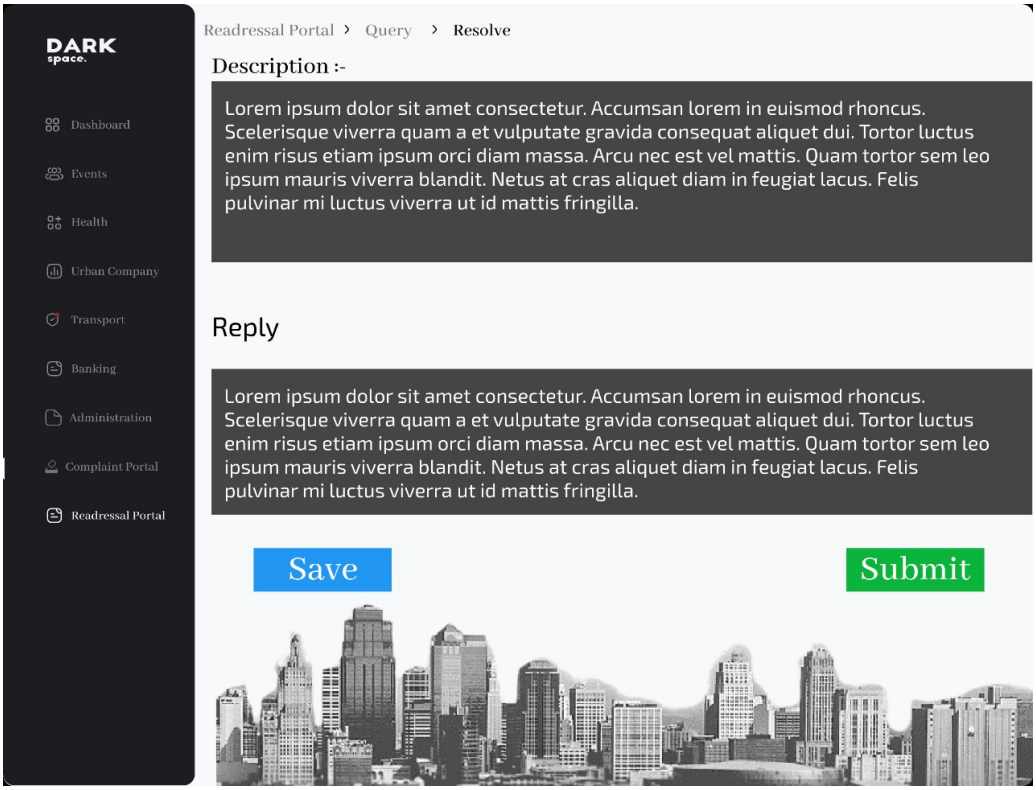

Figure 14: Complaint resolving page: Users can resolve the pending complaints.

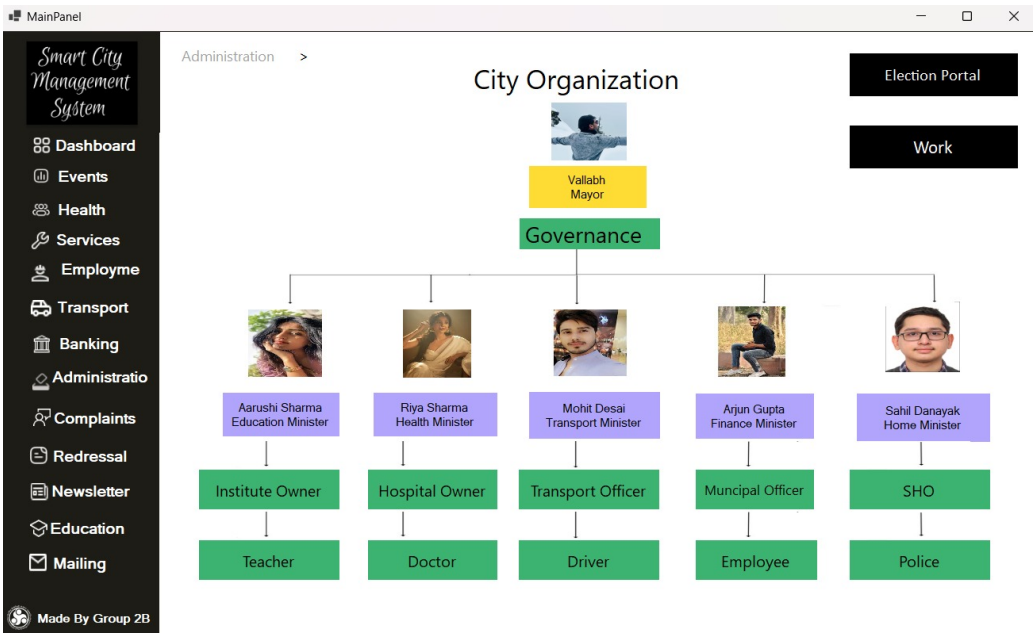

Figure 15: City hierarchy is shown here.All ministers can access the job requests through this page.

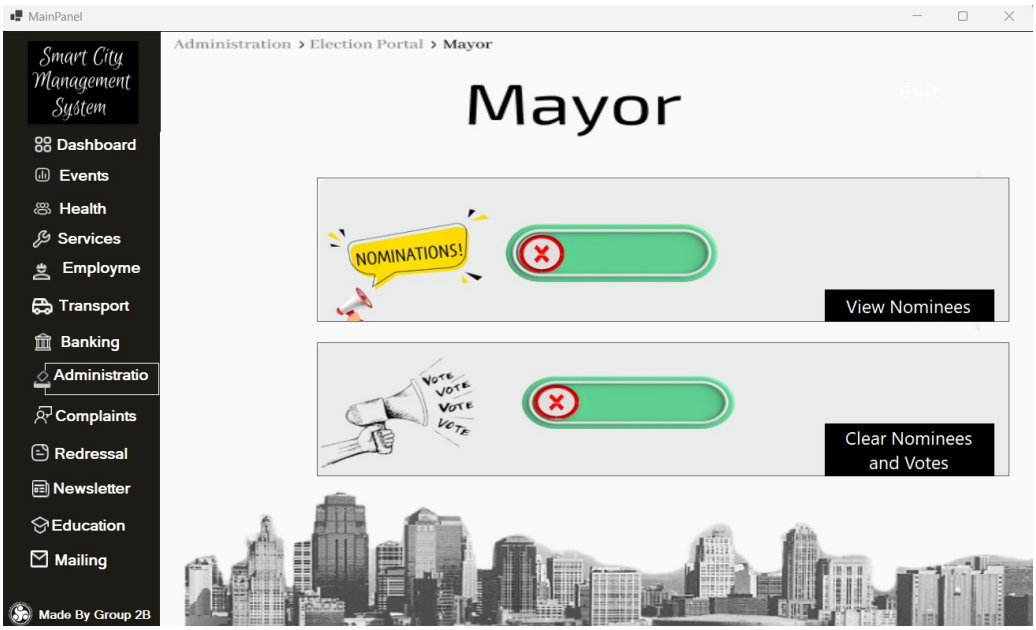

Figure 16: Mayor can start new elections and check nominees list.He can clear the table for nominees and votes once elections are over.

Figure 17: Ministers can check for their requests through this page.Similar pages are made for other minister also.

- Chart = System.Windows.Forms.DataVisualization.Charting.Chart: This imports the Chart class from the System.Windows.Forms.DataVisualization.Charting namespace. It also creates an alias Chart for the System.Windows.Forms.DataVisualization.Charting.Chart class, allowing you to refer to it using the alias Chart instead of the fully qualified name.
- System.Drawing: This imports the System.Drawing namespace, providing classes and functionality for working with graphics, images, and drawing operations in .NET applications.

Figure 18: Banking Homepage Interface

Query & Passbook Management Module

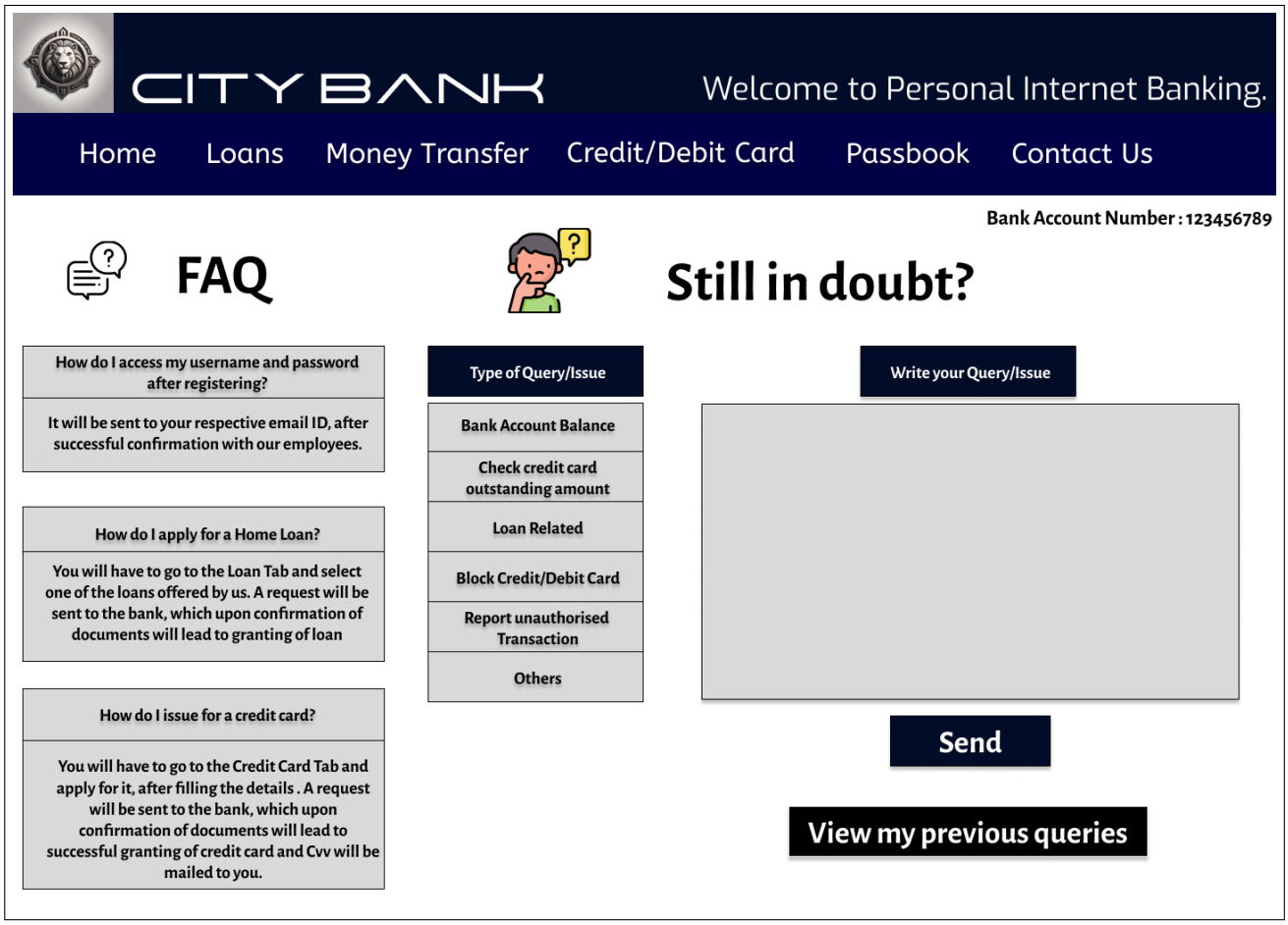

Figure 19: Contact Us Interface

User Features (Contact Us) :

- Navigation: Use the navigation menu at the top to access different sections such as Home, Loans, Money Transfer, Credit/Debit Card, Passbook, and Contact Us.
- FAQ:

  Click on the FAQ section to find answers to common questions regarding account access, applying for loans, issuing credit cards, and more.
- Still in Doubt?

  If you still have questions or issues, you can select the type of query or issue from the dropdown menu. Write your query or issue in the text box provided.
- Click "Send" to submit your query.
- View Previous Queries:

  Click on "View my previous queries" to see a history of your previously submitted queries.
- Bank Account Information: Your bank account number is displayed prominently at the top right corner of the page for reference.

Technical Features :

- Navigation Menu:

  A set of buttons at the top of the form for navigating to different sections of the application.

- FAQ Section:

  Displays frequently asked questions and their corresponding answers.

- Query Submission Section: Text box for entering the query or issue details. "Send" button to submit the query.

- View Previous Queries Button:

  Button to view a history of previously submitted queries.

- Bank Account Number Display:

  Displays the bank account number of the user at the top right corner of the form.

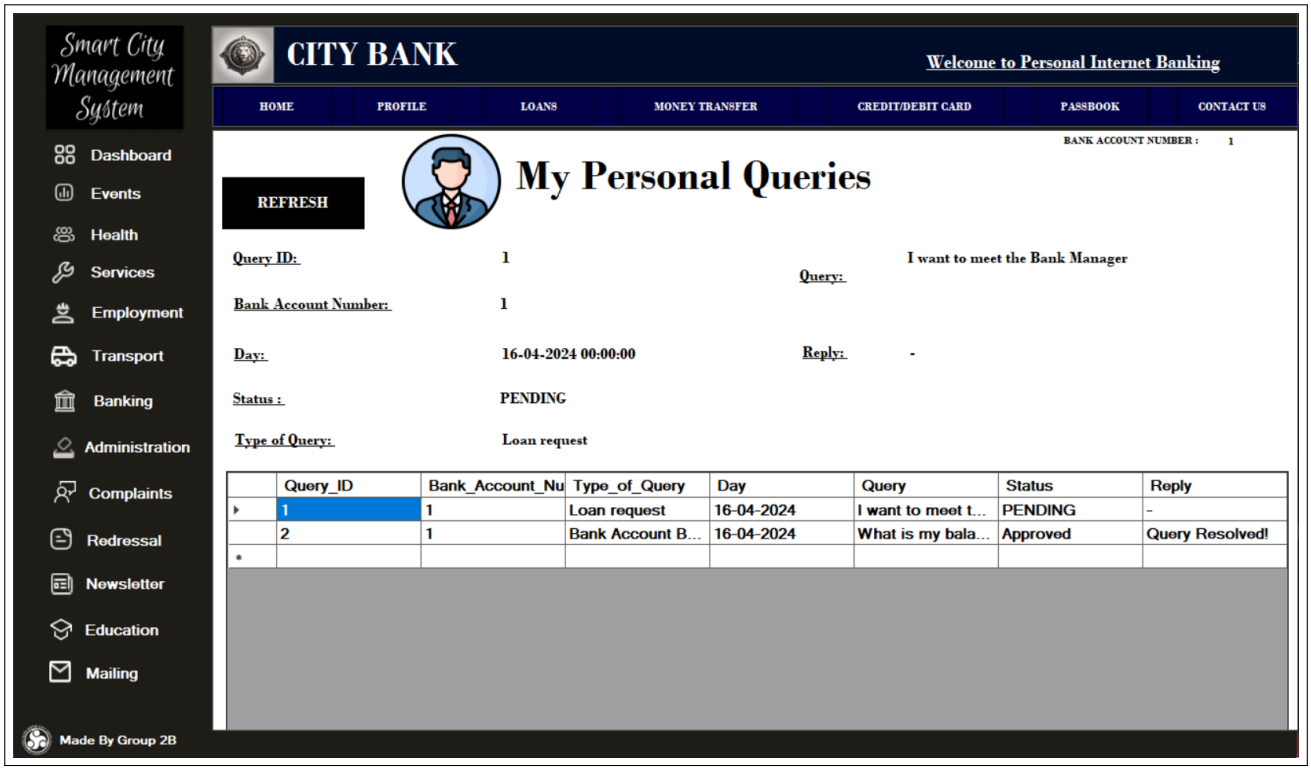

Figure 20: Banking Query Interface

User Features (Query Us) :

- Navigation:

  Utilize the sidebar menu on the left to navigate through different sections including Dashboard, Events, Health, Services, Employment, Transport, Banking, Administration, Complaints, Redressal, Newsletter, Education, and Mailing.

- My Personal Queries: Displays a list of personal queries submitted by the user. Each query includes details such as Query ID, Bank Account Number, Day of submission, Status, Type of Query, and Reply (if any).

- Refresh Button:

  Clicking on the "Refresh" button updates the list of queries to display the latest information.

- Bank Account Number Display:

  The bank account number of the user is shown at the top right corner of the page for reference.

Technical Features :

- Sidebar Menu:

  A set of icons representing different sections of the application, allowing users to navigate between them

- My Personal Queries Section:

  Displays a list of personal queries submitted by the user, including relevant details such as Query ID, Bank Account Number, Day of submission, Status, Type of Query, and Reply.

- Refresh Button:

  Button to update the list of queries with the latest information.

- Bank Account Number Display:

  Displays the bank account number of the user at the top right corner of the form.

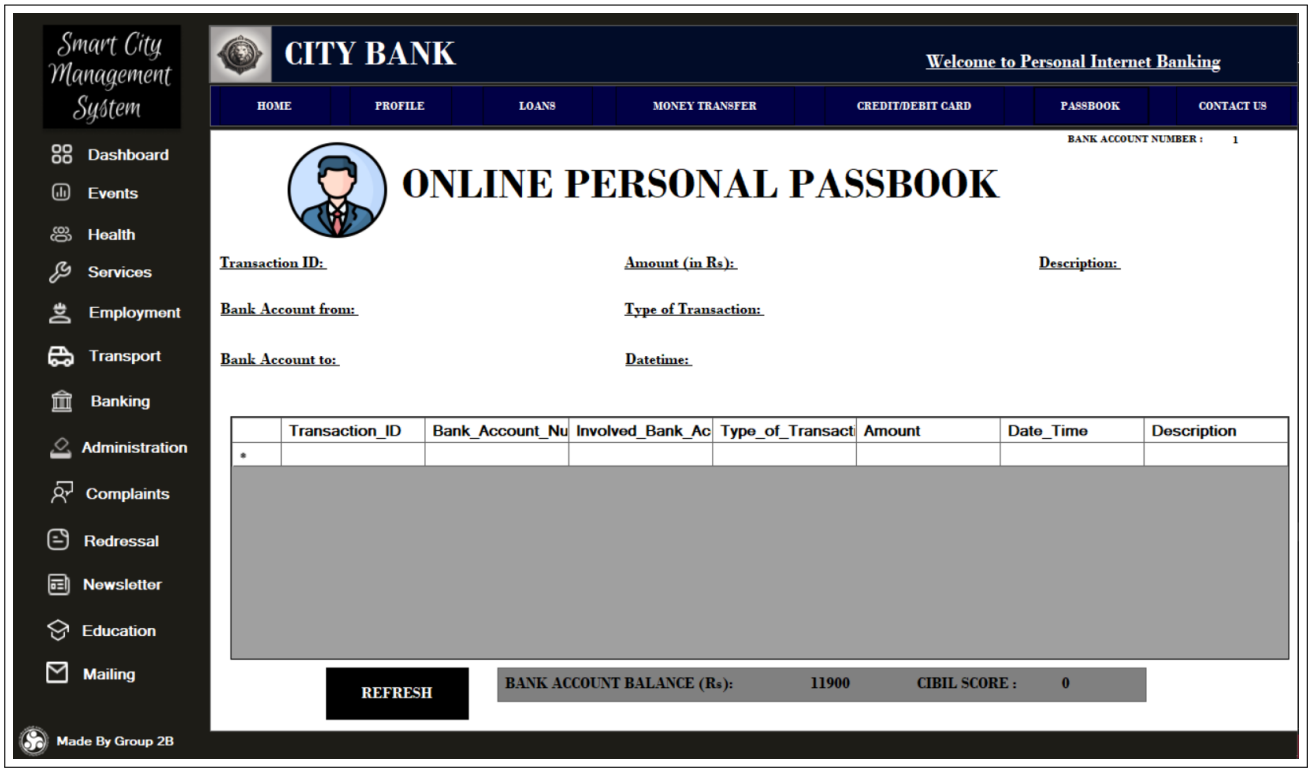

Figure 21: Banking Passbook Interface

User Features (Passbook):

- Navigation:

  Use the sidebar menu on the left to navigate through different sections including Dashboard, Events, Health, Services, Employment, Transport, Banking, Administration, Complaints, Redressal, Newsletter, Education, and Mailing.

- Online Personal Passbook:

  Provides access to the user's online personal passbook, displaying transaction details and account balance. Users can view transactions based on various parameters such as Transaction ID, Bank Account from/to, Type of Transaction, Amount, Date/Time, and Description.

- Refresh Button:

  Clicking on the "Refresh" button updates the passbook display with the latest transaction information.

- Bank Account Balance and CIBIL Score:

  Displays the current bank account balance in rupees and the CIBIL score of the user.

Technical Features :

- Sidebar Menu:

  A set of icons representing different sections of the application, allowing users to navigate between them.

- Online Personal Passbook Section:

  Displays transaction details including Transaction ID, Bank Account from/to, Type of Transaction, Amount, Date/Time, and Description.

- Refresh Button:

  Button to update the passbook display with the latest transaction information.

- Bank Account Balance and CIBIL Score Display:

  Displays the current bank account balance and CIBIL score of the user.

Administration Module

User Features

- **HomePage:**
  - The Home page serves as the main dashboard for administrators, providing access to the primary database of bank users.
  - Administrators can view and modify user information, including fields such as CIBIL Score, Phone Number, Bank Account Number, and Account Balance
  - Modifications to user data can be made directly from the Home page, ensuring flexibility and efficiency in managing user accounts.
- **Loan Approvals:**
  - In the Loan Approvals section, administrators can review and approve pending loan applications submitted by bank customers.
  - With a simple click of a button, administrators can approve loans based on predefined criteria, streamlining the loan approval process and ensuring timely access to funds for eligible applicants.
- **Queries and Complaints:**
  - The Queries and Complaints section provides a platform for administrators to address customer queries and complaints.
  - Administrators can view a log of queries submitted by customers, categorize them by type, and take appropriate actions to resolve issues in a timely manner.
- **New User Registration**
  - In the New User Registration section, administrators can approve new user registrations within the banking system.
  - Administrators review new user applications, verify applicant details, and approve or reject registrations based on predefined criteria, facilitating the onboarding process for new customers.

Technical Features : The Administrator Side of the Banking Module is implemented with robust functionality and user-friendly interfaces to streamline administrative tasks. Below are key technical details regarding the module's features and implementation:

- **Database Access:**
  - Administrators have direct access to the main database of bank users from the Home page.
  - Database queries and modifications are executed securely within the application, ensuring data integrity and confidentiality.
- **User Information Modification:**
  - Database transactions are managed efficiently to ensure that modifications are reflected accurately in the user database.
- **Loan Approval Process**
  - Loan approval decisions are made based on predefined criteria, with administrators able to approve loans with a single click of a button.
  - A query is sent to the Loan Table which flips the approved bit, indicating that the loan is now approved.
- **New User Registration Approval**
  - The New User Registration section enables administrators to review and approve new user registrations within the banking system.
  - A bank account number is randomly generated paying heed to the fact that a clash does not occur by recursively calling the random generator till a new value is obtained.

Newsletter Service

Introduction

The Newsletter Service module serves as a vital component within the smart city management platform, offering a centralized platform for disseminating news and updates to residents and stakeholders. Developed using advanced technologies, this module facilitates the submission, approval, and publication of news articles from various city services and provides users with compre- hensive statistics on city-wide metrics. The backend code relies on the following dependencies:

- Microsoft.SqlServer: This imports namespaces related to Microsoft SQL Server, providing access to classes and functionality for interacting with SQL Server databases.
- MySql.Data.MySqlClient: This imports namespaces for MySQL database connectivity, allowing interaction with MySQL databases using the MySQL .NET Connector.
- System.IO: This imports the System.IO namespace, providing classes and functionality for input and output operations, such as reading from and writing to files and streams.
- Chart = System.Windows.Forms.DataVisualization.Charting.Chart: This imports the Chart class from the System.Windows.Forms.DataVisualization.Charting namespace. It also creates an alias Chart for the System.Windows.Forms.DataVisualization.Charting.Chart class, allowing you to refer to it using the alias Chart instead of the fully qualified name.
- System.Drawing: This imports the System.Drawing namespace, providing classes and functionality for working with graphics, images, and drawing operations in .NET applications.

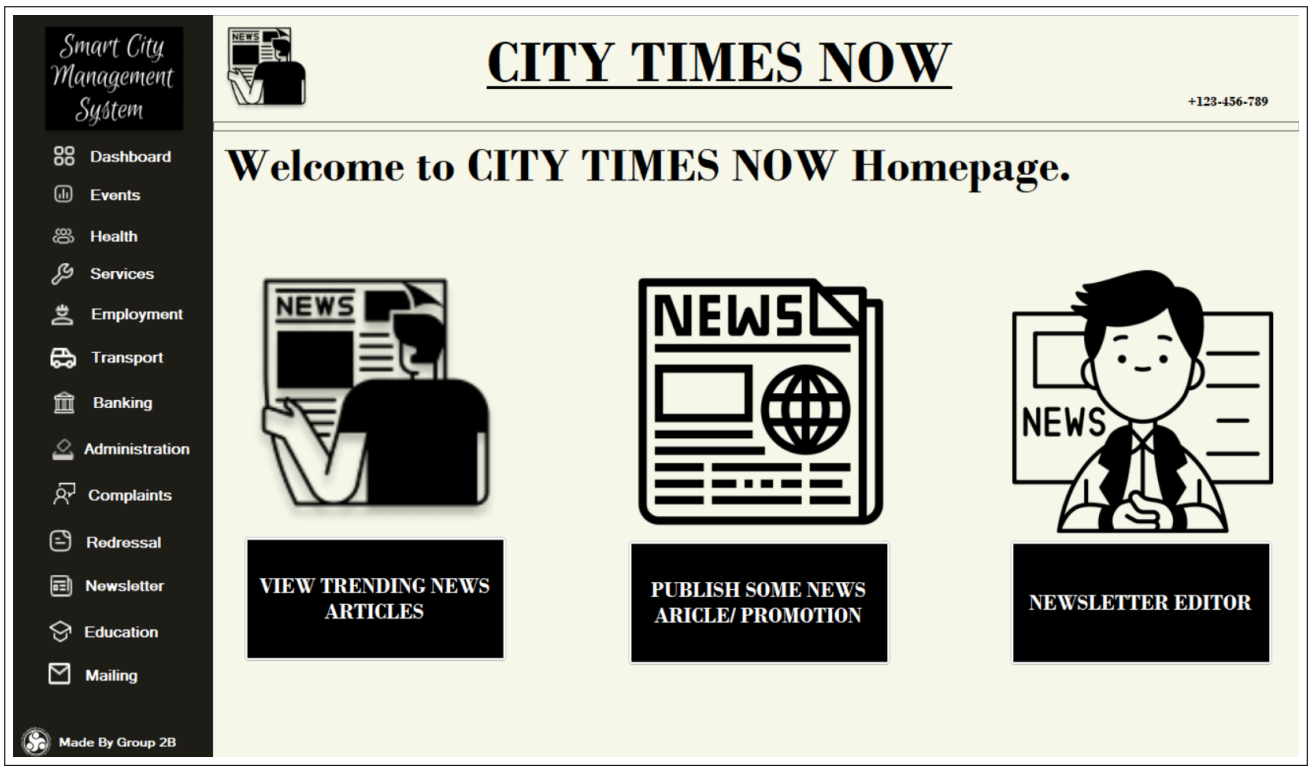

Figure 22: Newsletter Homepage Interface

Editor and Publishing Module

Figure 23: Newsletter Addnews Interface

User Features (Publish News) :

- Navigation:

Utilize the sidebar menu on the left to navigate through different sections including Dashboard, Events, Health, Services, Employment, Transport, Banking, Administration, Complaints, Redressal, Newsletter, Education, and Mailing.

- Publish News/Advertisement:

  Fill in the headline, publisher, select the type, and provide content for the news or advertisement. Optionally, upload an image related to the news or advertisement.

- Send to Publish Button:

  Clicking on the "Send to Publish" button submits the news or advertisement for publishing.

- Pay Now Button:

  Clicking on the "Pay Now" button initiates the payment process for publishing the news or advertisement.

- Contact Information:

  City Times Now contact number is displayed at the top right corner of the page for assistance.

Technical Features

- Sidebar Menu:

  A set of icons representing different sections of the application, allowing users to navigate between them.

- Publish News/Advertisement Section:

  Allows users to input headline, publisher, select type, and provide content for the news or advertisement. Option to upload an image related to the news or advertisement.

- Send to Publish Button:

  Button to submit the news or advertisement for publishing.

- Pay Now Button:

  Button to initiate the payment process for publishing the news or advertisement.

- Contact Information Display:

  City Times Now contact number is displayed at the top right corner of the form for user assistance.

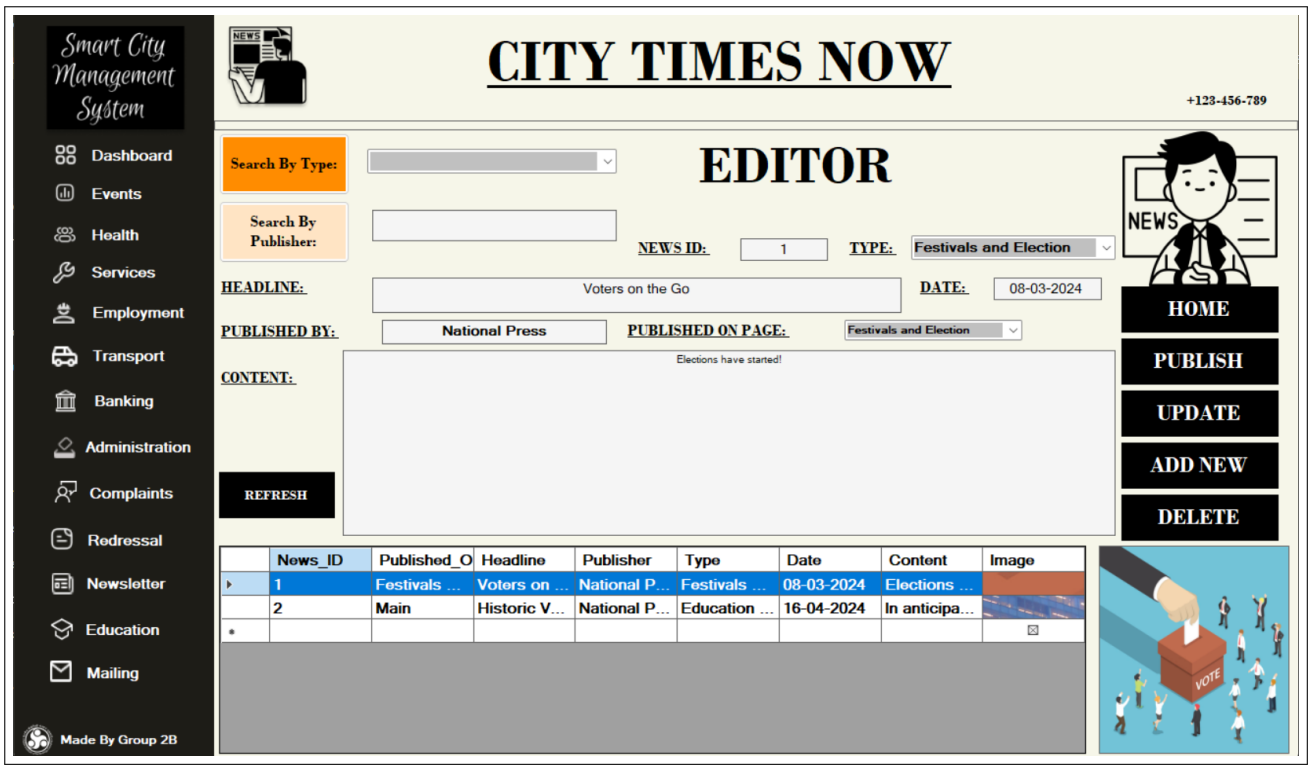

Figure 24: Newsletter Editor Interface

User Features (Editor)

- Navigation:

  Utilize the sidebar menu on the left to navigate through different sections including Dashboard, Events, Health, Services, Employment, Transport, Banking, Administration, Complaints, Redressal, Newsletter, Education, and Mailing.

- Editor Section:

  Search for news by type or publisher using the respective dropdown menus and search boxes. View news details including News ID, Headline, Publisher, Type, Date, Content, and Image. Edit or update existing news articles. Add new news articles. Delete news articles.

- Refresh Button:

  Clicking on the "Refresh" button updates the displayed news articles with the latest information.

- Home, Publish, Update, Add New, Delete Buttons:

  Use these buttons to navigate between different editor functionalities such as returning to the home page, publishing news, updating existing news, adding new news articles, and deleting news articles.

- Contact Information:

  City Times Now contact number is displayed at the top right corner of the page for assistance..

Technical Features:

- Sidebar Menu:

  A set of icons representing different sections of the application, allowing users to navigate between them.

- Editor Section:

  Provides functionalities to search, view, edit, add, and delete news articles. Includes fields for searching news by type or publisher, displaying news details, and managing news articles.

- Refresh Button:

  Button to update the displayed news articles with the latest information.

- Home, Publish, Update, Add New, Delete Buttons:

  Buttons to navigate between different editor functionalities such as home page, publishing news, updating existing news, adding new news articles, and deleting news articles.

- Contact Information Display:

  City Times Now contact number is displayed at the top right corner of the form for user assistance.

Email Service

Introduction

This module serves as a crucial component in facilitating efficient communication within our smart city ecosystem. Developed using Visual Basic, the Email Module offers a range of features designed to streamline email correspondence for city officials, administrators, and residents alike. The backend code relies on the following dependencies:

- Microsoft.SqlServer: This imports namespaces related to Microsoft SQL Server, providing access to classes and functionality for interacting with SQL Server databases.
- MySql.Data.MySqlClient: This imports namespaces for MySQL database connectivity, allowing interaction with MySQL databases using the MySQL .NET Connector.
- System.IO: This imports the System.IO namespace, providing classes and functionality for input and output operations, such as reading from and writing to files and streams.

User Features:

- **Composing Emails:** Emails can be saved as drafts for future editing or sent immediately.
- **Inbox Management:** The module allows users to access their inbox, where they can view received emails sorted by sender or other criteria.
- **Sent Emails:** Users can review a history of emails they have sent through the portal.

Technical Features:

- **Visual Basic Implementation:**
  - The Email Module is developed using Visual Basic, ensuring compatibility with the existing framework of our smart city management portal.
  - Visual Basic provides a user-friendly development environment, allowing for rapid iteration and customization of features.
- **Database Integration**
  - The module seamlessly integrates with the portal's database system to store and retrieve email data.
  - Structured Query Language (SQL) queries are used to interact with the database, ensuring efficient data retrieval and manipulation.
  - Database transactions are managed using Visual Basic's built-in database access libraries, ensuring data integrity and consistency.
- **Draft and Sent Email Management**

  - Draft emails are stored in a dedicated table within the database, along with metadata such as recipient information. Sent emails are archived in a separate table, allowing users to review their sent communications.

- **Integration with Smart City Portal:**
  - The email address of the sender is retrieved from the current instance of the smart city portal, ensuring accurate identification of the sender in outgoing emails.
  - Clicking on the Compose button initiates a new email composition session, while clicking on the Inbox or Sent mails button redirects users to their respective email folders within the module.