Assignment 3: Smart City Management Platform

CS346: Software Engineering Laboratory

Assignment-3 Documentation

Group 2A Task:

Develop an integrated platform designed to improve urban living through technological innovation and interconnectivity.

Authors:

Group 2A

Instructor:

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

Contents

1 Problem

1.1 Problem Statement

The Smart City Management System is an integrated platform designed to improve urban living through technological innovation and interconnectivity. It centralizes essential services, providing residents with easy access to a wide range of professionals like doctors and electricians. Educational institutions, employment opportunities, and healthcare services are seamlessly integrated, enhancing the city's efficiency and the welfare of its inhabitants.

1.2 Goal and Motivation

The motivation here is to create a software where the specified features are integrated to provide a great experience to the user during the stay in the Smart City. We have broken down the various features that can be provided into multiple ministries and grouped similar features into modules. Students were divided into groups to take up the responsibility of one or several modules.

2 Installation & Setup

Download the zipped folder of the repository from this link . Unzip the folder and open the .sln file using Visual Studio 2022. Run the file and create an account. The software is all set up to use.

3 Hosting Database

Our database is hosted online in ASP.net. All the tables have been created and necessary entries have been made.

Thus, the software can be used by any person on the internet with ease.

4 Detailed Documentation of the Software

4.1 User registration & Management

4.1.1 Introduction

The User Registration and Management module is a crucial element of the Smart City Management System, providing users with a seamless experience to access and manage their accounts. This module ensures secure authentication, facilitates account recovery via the forget password option, and enables users to maintain their profile information. It acts as a gateway for residents to create profiles, manage personal details, access various city services, and view important notices from the home page. Additionally, it provides a secure login mechanism for module admins, with passwords being periodically updated for heightened security.

4.1.2 Features

1. **Login Page**: Users can access the Smart City Portal by logging in with their credentials. The system verifies the entered credentials against the Users database. Upon successful verification, users are directed to the home page.

2. **Forgot Password**: Users who have forgotten their password can initiate a password recovery process. They receive a One-Time Password (OTP) on their registered email address. After OTP verification, users can reset their password securely.

3. **User Registration (Sign Up)**: New users can sign up for an account. Upon registration, they receive an OTP on their email for verification. After OTP verification, users are directed to a details page where they enter basic information such as date of birth, phone number, profile picture etc. The entered details are saved in the Users database.

4. **Profile Update**: Users have the option to update their profile information. This feature allows users to edit their basic information and upload a new profile picture if desired. Any changes made are updated in the Users database.

5. **Admin Login**: Module admins can log in using the unique credentials reserved for them in the Users table. Passwords for admins are changed after elections for enhanced security, ensuring robust protection of sensitive module data.

6. **Impotant Notices**: The important notices option on the home page displays the various notices that are either broadcast to all the users or are specifically directed to the individual user by the various modules. The user can filter the notices based on the sending ministry and the date received.

7. **Database Integration**: User registration and profile information are stored and managed in the Users database and notifications are stored in the notifications table, ensuring data integrity and accessibility across the application.

4.1.3 Functions

1. **SendEmail**(*recipientEmail*, *randomNumber*): The `SendEmail` function is responsible for sending an One-Time Password (OTP) to the specified email address using the SMTP protocol. The OTP generated by the system is sent to the user's email, and upon user input, the entered OTP is compared with the OTP sent to verify the user's email address. An error message is displayed if the OTPs do not match.

2. **SaveDetails_Click and UpdateDetails_Click**: These functions validate user input to ensure that all mandatory fields are filled and that the entered values are valid. If any mandatory fields are left blank or if invalid values are entered, an error message is displayed. The SaveDetails_Click function adds a new row to the `users` table, while the UpdateDetails_Click function updates the existing row for the specified user UID.

3. **FilterNotifications**(*ministryId*): The `FilterNotifications` function displays notifications filtered by ministry ID. If the ministry ID argument is provided, notifications from the specified ministry are shown, otherwise, notifications from all ministries are displayed.

4. **CheckPasswordStrength**(*password*): The `CheckPasswordStrength` function helps in assessing the strength of a password and facilitates the creation of secure passwords.

4.1.4 Error Handling

1. **Validation at Login Stage**: Input provided by the user during the login stage undergoes validation against the users table. If the user UID or password is invalid or does not match, an error message is displayed.

2. **Account Creation and Profile Updation**: Details entered during account creation and profile updation are checked for validity. Mandatory fields are required to be filled, and if any are left empty, an error message is shown.

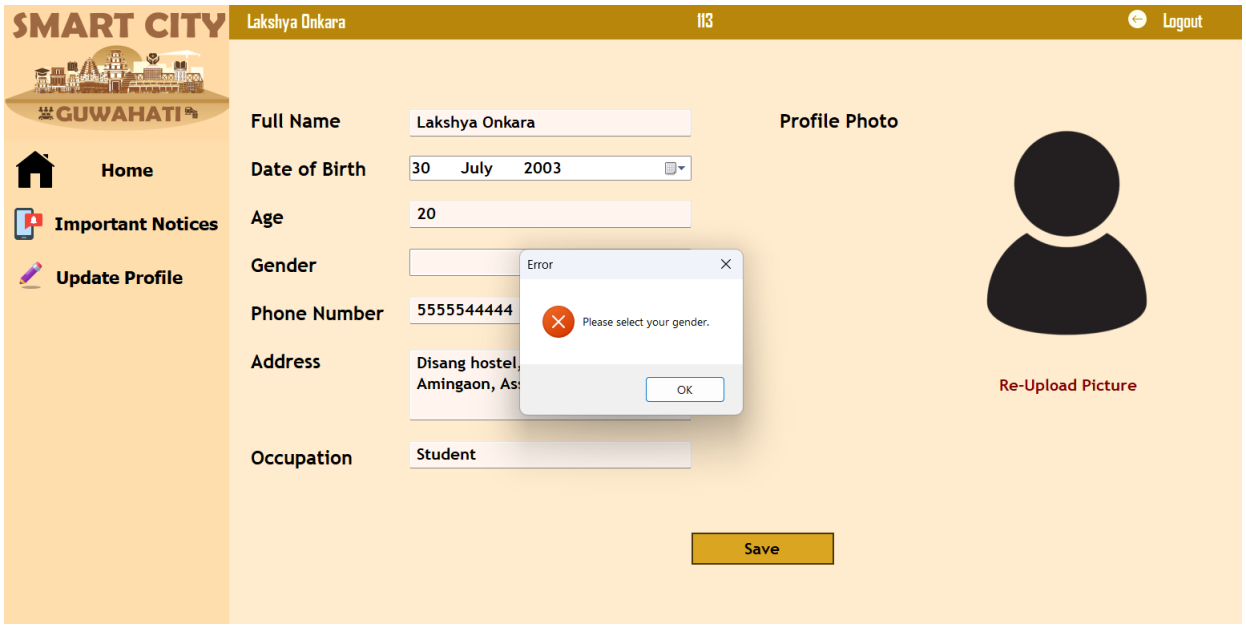

Figure 1: Error message displayed when mandatory fields are not filled in

3. **Contact Number** Contact number must be a 10 digit integer else a message prompt showing the same is displayed to the user/customer.

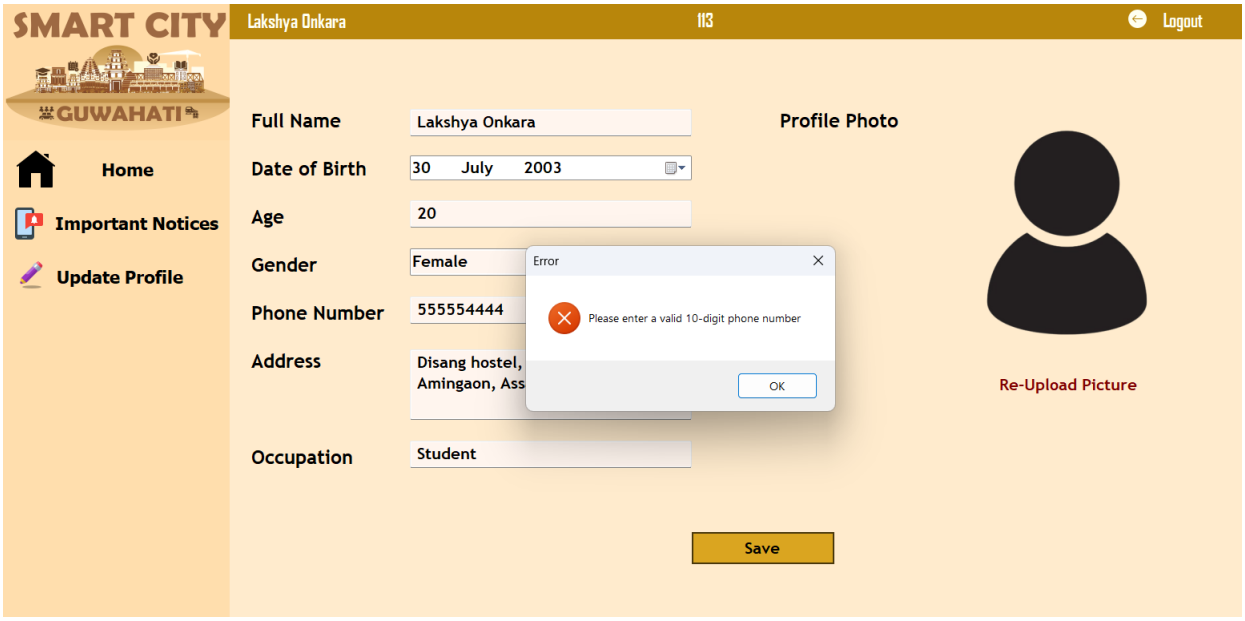

Figure 2: Error message displayed when invalid phone number is entered

4. **OTP Verification**: During the OTP verification stage, if the entered OTPs do not match, an error message is thrown.

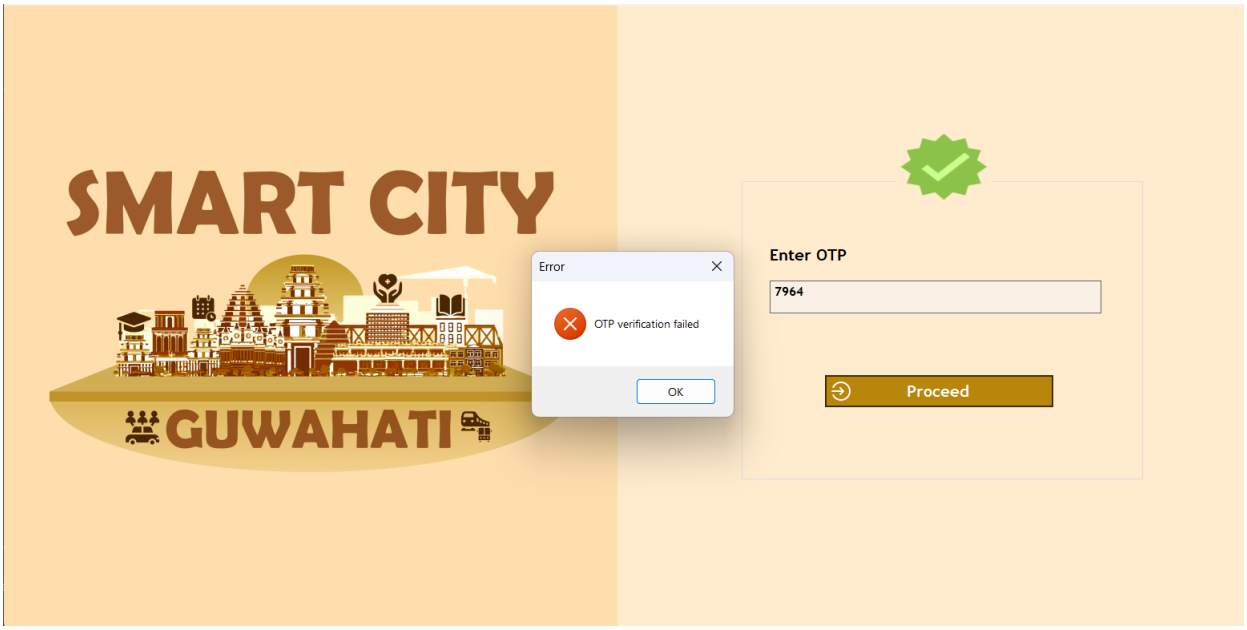

Figure 3: Error message displayed when incorrect OTP is entered

5. **Password Strength Checker** Strength of the password is displayed when the password is being entered ,if password is deemed weak then a message prompting the same is displayed urging the user to reinforce his password.

6. **Exception Handling for SQL Connections**: All SQL connections are established within a try-catch block to handle exceptions gracefully, ensuring that any errors encountered during database operations are handled appropriately without disrupting the application flow.

4.2 Professional Services Directory

4.2.1 Introduction

The professional service directory facilitates seamless access to services for all citizens of the smart city, aiming to optimize the service request and provision process. This platform offers various modes for both service providers and users, enabling easy booking, modification, and cancellation, along with online payment features. Service providers can register and manage their services effortlessly, accessing booking details and service histories conveniently.

4.2.2 Features

User side

1. **Search service** Users can utilize this feature to search for available services provided by different service providers. They simply need to input the service name or description along with the relevant department. The system then presents a tabulated view of active services, including service ID, service description, department, hourly charge, and operating hours.

2. **Request/Cancel Service** Once users find a suitable service, they can proceed to book it by providing the service ID, their user ID, preferred date, and hours. The booking is made for a one-hour slot. Users can also view their requested services and cancel them if needed. An online payment portal is available for convenient service booking.

3. **Booking History** Users can access their service booking history, displaying all successfully delivered services. Filter options are provided for organizing the booking history.

Service Provider side

1. **Post service** Service providers can register on the platform and post details about the services they offer, including descriptions, departments, hourly charges, and operating hours.

2. **Withdraw service** Providers have the option to permanently stop offering their services if desired.

3. **View offered service** Providers can review past services provided to users, serving as a historical record.

4. **Scheduled services** Providers can view all requested services, as well as upcoming services they need to provide to specific customers.

5. **Leave scheduler** Providers can use this feature to schedule leave days, canceling their services for an entire day in the upcoming week if necessary.

4.2.3 Functions

1. **ComboBox2_SelectedIndexChnged** ()
Description: Sort data by date/price. In page searching, search happens on basis of description(any type of substring matching) or department.

2. **TransportationDashboard_Load** ()
Description: Service_Portal.vb - Loads all available services

3. **ScheduleButton_Click**()
Description: Request-Cancel_Service.vb - When service id, user id, date and time are provided through appropriate textboxes, a service is scheduled for requesting user on clicking 'Request Button'

4. **CancelButton_Click** ()
Description: Request-Cancel_Service.vb - Used for cancelling services whose checkboxes are selected.

5. **Dashboard_Load** ()
Description: Service_Provider.vb - Selects all the scheduled services of a particular provider and displays the current active service and next scheduled service.

6. **PostService** ()
Description: Service_Post.vb - Posts service taking data from provider(description, service offered, department, charge, start time, end time).

7. **Service_Scheduled_Load** ()
Description: Service_Scheduled.vb - Loads all the scheduled services.

8. **Service_View_Offered.vb**
Description: View offered services to different users utilising various filter options

9. **Service_Withdraw.vb**
Description: Loads active services of a service provider alongwith a checkbox to select services a provider wishes to withdraw.

10. **Service_Leave.vb**
Description: Provides an option to service provider to take leave in upcoming week.

4.2.4 Error Handling

1. **Empty fields in service provider leave page** The application throws error in case of confirm button is clicked without any selection.

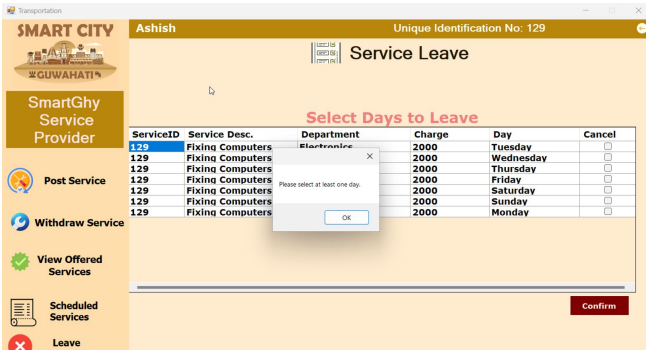

Figure 4: Error because of empty selection in leave page

2. **Empty fields in service provider withdraw page** The application throws error in case of confirm button is clicked without any selection.

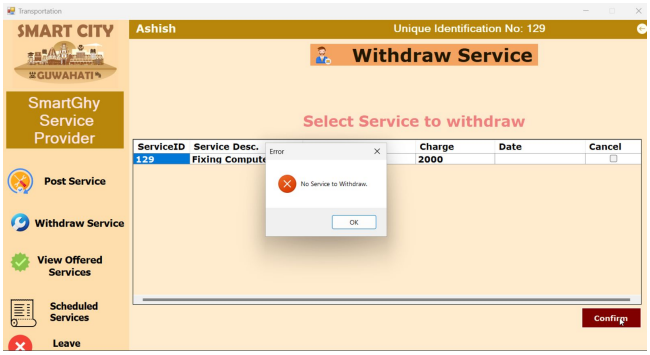

Figure 5: Error because of empty selection in withdraw page

3. **Empty fields in service provider post service page** The application throws error in case of post button is clicked without filling details.

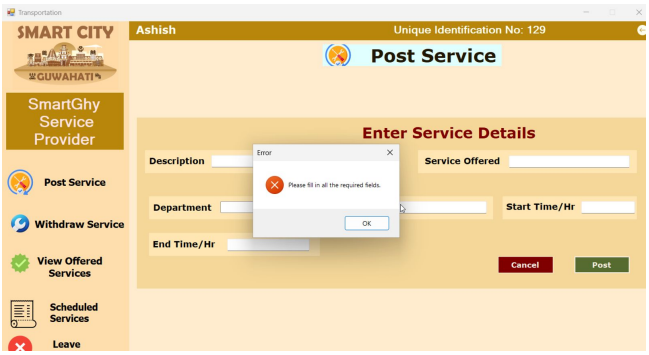

Figure 6: Error because of empty fields in post service

4. **Empty fields in service provider request/cancel page** The application throws error in case of request/confirm button is clicked without filling details.

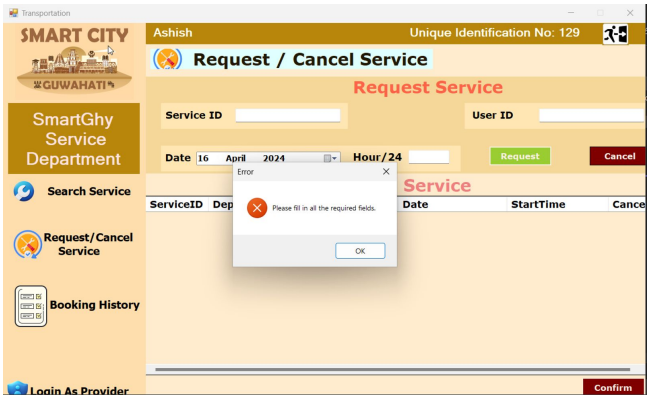

Figure 7: Error because of empty fields in request/withdraw service

4.3 Employment Portal

4.3.1 Introduction

The Employment Portal serves as a platform designed to facilitate efficient job search and application processes for both job seekers and recruiters. By offering user-friendly interfaces and intuitive functionalities, the portal aims to streamline the recruitment journey for all stakeholders involved.

4.3.2 Features

From job seekers' side

1. **Search for jobs** Here, job seekers have the ability to explore employment opportunities. Using a range of filter options, individuals can search jobs to find positions that align with their preferences. Features such as auto-predictive job descriptions and department fields makes the search process easier, enhancing user experience. Additionally, seekers can specify their salary lower bound to further filter their search criteria. Upon entering their search parameters, the platform presents available jobs in a clear tabular format, displaying essential details for each listing.

2. **Apply for jobs** Applying for a position is made simple for job seekers, who only need to input the job ID to initiate the process. The required details of candidates are already there in the smart city database which are then shared to the recruiter.

3. **View applied jobs** Job seekers can easily track their application status through the "View Applied Jobs" section. This feature provides filters to categorize applications based on acceptance, rejection, or pending status, enabling users to stay informed about the progress of their job applications.

From admin/recruiters' side

1. **Application Management** When a recruiter logins as a recruiter; they are shown all the applications; accepted, rejected and pending applications. Through a user-friendly interface, recruiters can quickly **review applicant credentials** and manage application statuses with simple clicks, helping with faster and efficient recruitment processes.

2. **Add new jobs** Posting new job opportunities is straightforward for recruiters, who can input essential details such as department, job description, salary, qualifications, and deadline. This streamlined process ensures that job listings are accurately and comprehensively presented to potential candidates.

3. **Withdraw jobs** Recruiters have the flexibility to remove job listings from the platform as needed. By selecting the checkboxes corresponding to the desired listings, recruiters can withdraw jobs, ensuring that outdated or no longer available positions are promptly removed from public view.

4.3.3 Functions

1. **ApplyToJob**()
   Description: Taking Job ID as input, it creates a job application sent to recruiter along with applicant ID.

2. **SearchJobs**()
   Description: It takes filter options from user and searches jobs that match the criteria. Also there is functionality to autofill entries in Job Description and Department field.

3. **ViewApplicationStatus**()
   Description: Shows the applicant their applications' status. They may filter applications based on applied, accepted and rejected applications.

4. **CheckCredentials** ()
   Description: Used by recruiter to check applicant details of a selected job application.

5. **AcceptApplicant**()
   Description: Accepts selected job application(change status to accepted) and sends a notification to the applicant.

6. **RejectApplicant**()
   Description: Rejects selected job application(change status to rejected) and sends a notification to the applicant .

7. **CreateJob**()
   Description: Creates a job with details as specified by recruiter in textboxes.

8. **WithdrawJobs**()
   Description: Deletes a job selected by recruiter.

4.3.4 Error Handling

1. **Exception Handling for SQL Connections**: All SQL connections are established within a try-catch block to handle exceptions gracefully, ensuring that any errors encountered during database operations are handled appropriately without disrupting the application flow.

2. **Recruiter Permission Feature Access**: The application checks the Profile of the user in the database. Only Those having admin access i.e registered themselves as a recruiter can access recruiter privileges.

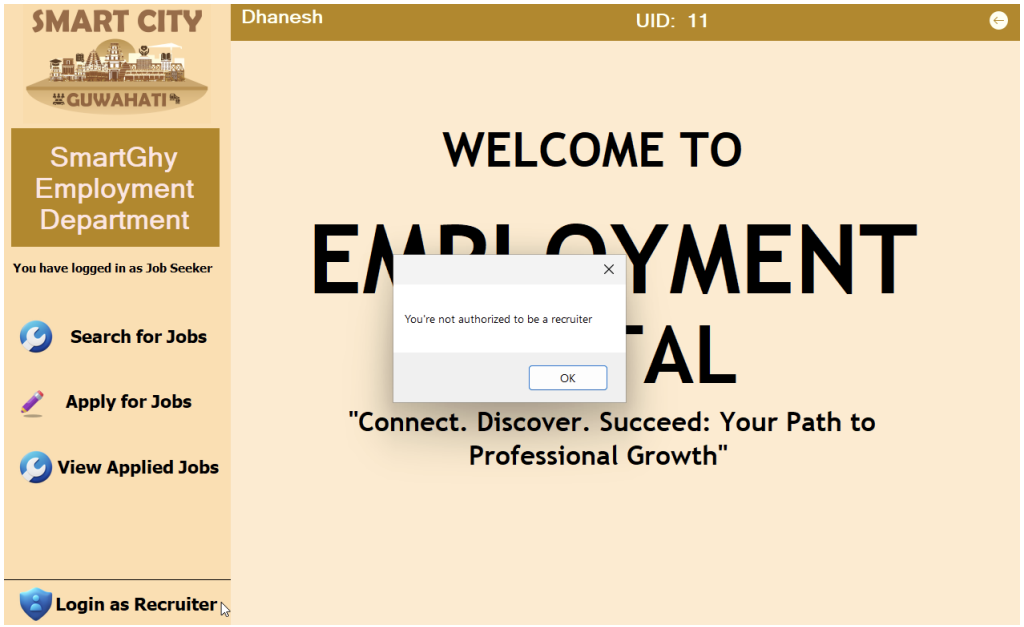

Figure 8: Only permitted users can access Recruiter facilities

3. **Invalid or closed job application detection**: The application queries the database if the job application to which the candidate is applying is valid or not(deadline has not passed and the job is existent.

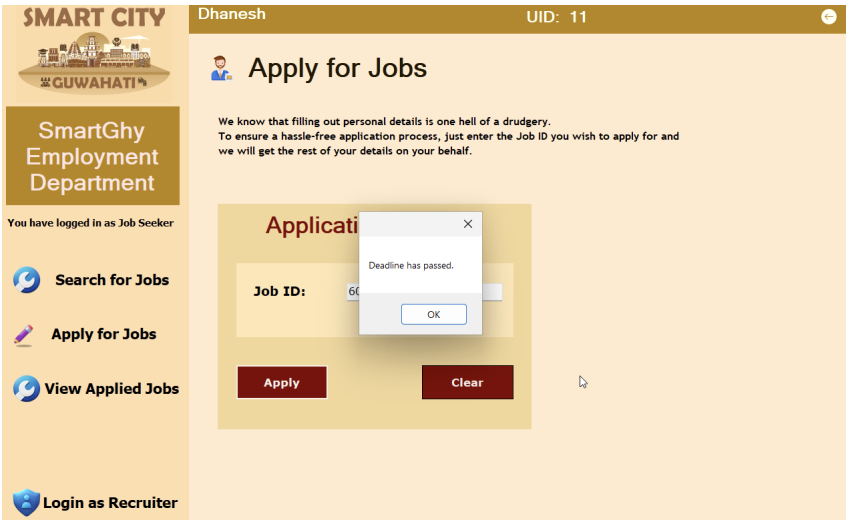

Figure 9: The deadline for the applied job application has already passed

4. **Invalid Job Search detection**: Our advanced Job search portal clearly handles all types of edge cases wherein extensive checks are made to display the job with the entered specifications.

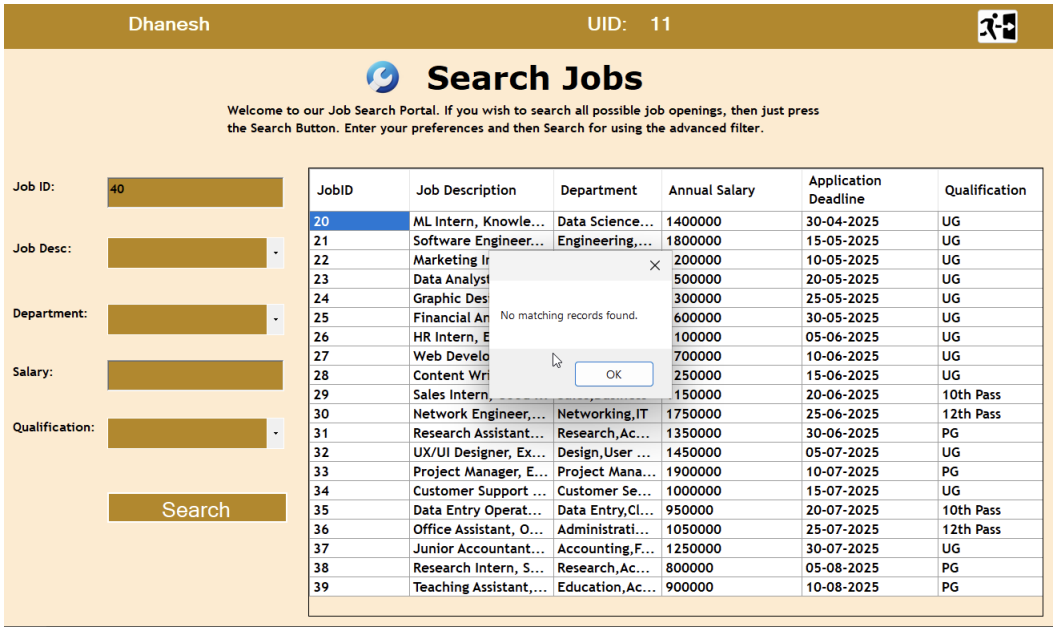

Figure 10: Shows that no record of a job exists with the required specs.

5. **Robust handling of other features**: The application must not crash in situations like a button being pressed which requires a candidate/jobid selected whenever nothing is selected.

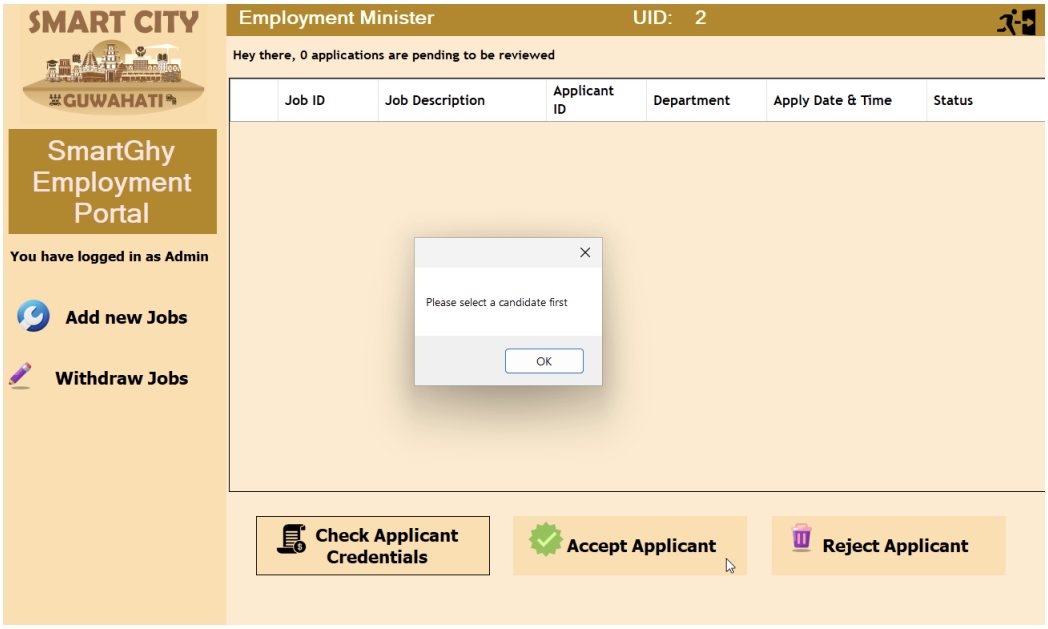

Figure 11: Shows that no candidate is selected

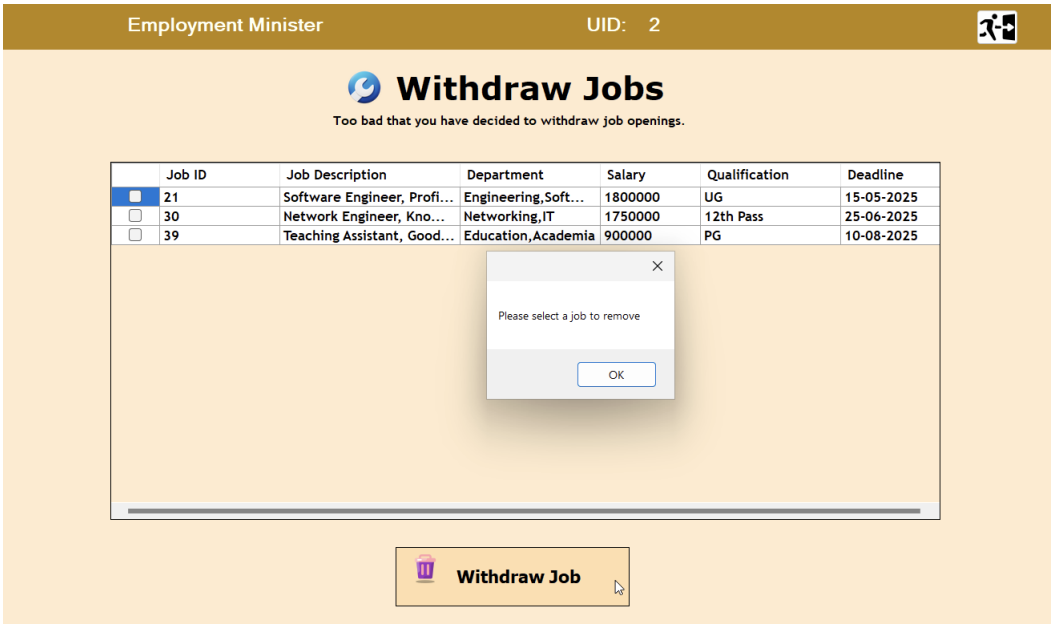

Figure 12: Shows that no job-id is selected to withdraw

6. **Incomplete Job openings not accepted**: We have made it mandatory that the job openings must have all the details so that it easy on the job seeker. Hence, only job openings with all details entered are accepted. Non numerical salary cannot be entered whatsoever

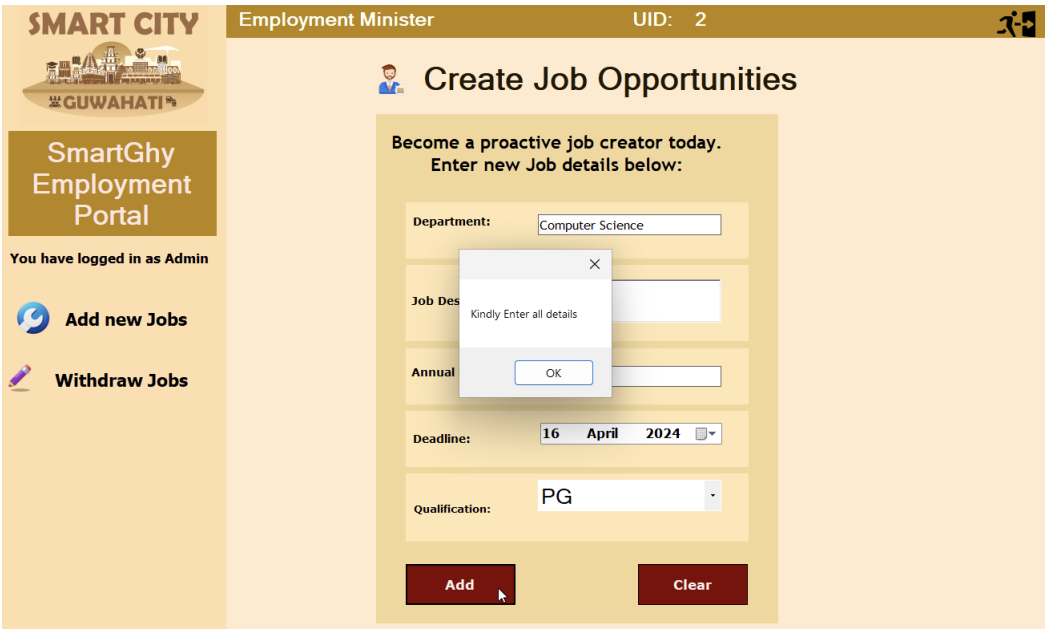

Figure 13: Error showing not all required details are entered

4.4 Education Management

4.4.1 Introduction

The Education management comes under the Ministry of Education in the smart city, which is elected through the Administrative Hierarchy module. This section of the Smart City Management provides various services aimed at enhancing the educational experience for all stakeholders. From student enrollment and course management to teacher facilitation and administrative oversight, our platform offers a comprehensive suite of tools and resources to ensure seamless operations and promote academic excellence.

4.4.2 Features

Main Users:

1. **Students:** Any resident, irrespective of profession, enrolled in educational programs.
2. **Teacher (Ministry):** Professionals employed by the Ministry of Education.
    a. Institute Teachers: Educators affiliated with specific institutes.
    b. E-Course Teachers: Instructors offering courses through the e-learning platform.
3. **Admins:**
    a. Institute Admins: Administrators responsible for managing operations within their respective institutes.
    b. E-Course Platform Admins: Administrators overseeing the e-learning platform's functionality and content.
4. **Education Minister:** Elected official holding a fixed account for administrative purposes within the education system.

User wise features:

- **Students**:
    - **E-Course**:
        1. Join courses by paying a designated course fee.
        2. Utilize video lessons to fulfill course requirements.
        3. Obtain certificates upon successful completion.
        4. Provide feedback by rating the courses they've undertaken.
    - **Institute**:
        1. Apply for admission to educational institutions.
        2. Pay tuition fees.
        3. Check and avail bus facilities provided by the institution.
        4. Receive pass certificates and extracurricular certificates upon completion of programs.
    - **Moodle**:

1. Access supplementary learning materials to institute classes
2. Access and submit assignments
3. View grades and assessments.

– **Entrance Exams**:

1. Enroll for city-wide entrance exams.
2. Receive admit cards containing exam date and venue information.
3. Obtain result certificates when results declared by the Education Minister.

- **Teachers**:
  - **E-Course**:
    1. Apply to add e-courses either on their personal account or on behalf of their institute.
    2. Develop lessons, and manage course content.
  - **Moodle**:
    1. Create institute classes within the Moodle platform.
    2. Add/modify/delete assignments and resources for an institute class
    3. Review student submissions and provide grades accordingly.
- **Admins**:
  - **E-Course Admin**:
    1. Review and approve/reject applied e-courses.
    2. Manage existing courses, including dropping them if necessary.
  - **Institute Admin**:
    1. Approve/reject student admission applications.
    2. Oversee student management and fee administration.
    3. Generate pass certificates by inputting marks and other necessary information.
- **Minister**:
  - **E-Course**:
    1. Administer the same features as e-course admin, including approving/rejecting e-courses and managing existing courses.
    2. Add e-course admins to assist in managing e-learning courses.
  - **Institute**:
    1. Add new institutions to the system.
    2. Edit institution details, such as contact information and addresses.
    3. Hire administrators, principals, and other staff members.
  - **Entrance Exams**:
    1. Set exam dates and venues.
    2. Define exam syllabus and topics.
    3. Release results and ranks to participants.

4.4.3 Functions

1. **Login Handler**

   (a) **GetEdProfileByUserID**($userID$) → Void
   Description: Retrieves user profile info based on provided ID. If profile exists in `ed_profile`, populates `Profile` object; otherwise, retrieves from `users` table and inserts into `ed_profile`.

2. **ECourse Handler**

   (a) **Course Class**
   Description: Describes the details of an E-Course.

   (b) **CourseContent Class**
   Description: Describes a resource within a course including its summary, video link, and other attributes.

   (c) **GetCourses**() → Course[]
   Description: Retrieves course information from the database.

   (d) **Add(or Update)Course**($affiliation$, $name$, $category$, $teacherName$, $teacherID$, $syllabus$, $introVideoLink$, $apprStatus$, $fees$) → Void
   Description: Adds(or Updates) a new course to the database.

   (e) **GetTeacherCourses**($teacherID$) → Course[]
   Description: Retrieves courses taught by a specific teacher.

   (f) **GetCourseDetails**($courseID$) → Course
   Description: Retrieves details of a course from the database.

   (g) **GetCourseContents**($courseId$) → CourseContent[]
   Description: Retrieves course contents from the database.

   (h) **Add(or Update or Delete)CourseContent**($courseId$, $contentName$, $contentType$, $videoLink$, $content$) → Void
   Description: Adds(or Updates or Deletes) new content to a course in the database.

   (i) **GetInProgress(or Completed)Courses**($studentId$) → Course[]
   Description: Retrieves courses in progress for a student.

   (j) **GetStudentCourseCompletionRecords**($studentID$, $courseID$) → Integer[]
   Description: Retrieves completion records for a student in a specific course.

   (k) **CompleteResource(Or Course)**($studentID$, $courseID$, $SeqNo$) → Void
   Description: Marks a specific course resource as completed for a student. If all resources are completed, then progress becomes 100 and course gets completed.

   (l) **RateCourse**($studentID$, $courseID$, $rating$) → Void
   Description: Records the rating given by a student to a course.

   (m) **Generate(Or Get)Certificate**($studentID$, $CertType$, $year$, $courseID$) → Void
   Description: Generates and saves (or Retrieves) a certificate for a student upon course completion.

3. **Moodle Handler**

   (a) **MoodleCourse Class**() → Course[ ]
   Description : This is a class to describe the course details of any MoodleCourse.

(b) **RoomContent Class** → Course[ ]
Description : This is a class to describe the resources and assignments in a course with its description, video link and other attributes.

(c) **GetCourses**() → Course[ ]
Description : This function retrieves course information from the database as an array of course class objects.

(d) **AddCourse**(*passKey*, *profId*, *instId*, *year*, *mClass*, *mSem*, *name*) → Integer
Description: Adds a new course to the database with the provided details, such as pass key, professor ID, institution ID, academic year, class or semester, and name. Returns the room ID of the newly added course.

(e) **GetRoomContents**(*roomID*) → RoomContent[]
Description: Retrieves the contents of the course room with the provided room ID.

(f) **LoadCourseDetails**(*roomID*) → MoodleCourse
Description: Retrieves details of the course with the provided room ID, including professor name and institution name.

(g) **GetEnrolledAssignments**(*studentID*) → RoomContent[]
Description: Retrieves a list of assignments for the courses in which the student with the provided student ID is enrolled.

(h) **AddCourseContent**(*roomID*, *contentName*, *contentType*, *videoLink*, *content*) → Void
Description: Adds content to the course with the provided room ID, including details such as content name, content type, video link, and content itself.

(i) **LoadCourseContent**(*roomID*, *seqNo*) → RoomContent
Description: Retrieves the details of the content with the provided sequence number in the course with the given room ID.

4. **Entrance Exam Handler**

(a) **get_ee_details**(*examID* As Integer) → *EEDetails*
Description: Retrieves entrance exam details based on the provided exam ID from the database.

(b) **get_venue_by_id**(*venueId* As Integer) → *EEVenue?*
Description: Retrieves venue details by the provided venue ID from the database.

(c) **add_new_venue**(*name* As String, *address* As String, *capacity* As Integer) → *Void*
Description: Adds a new venue to the database with the provided name, address, and capacity.

(d) **approve_pending_request**(*examID* As Integer, *studentID* As Integer, *year* As Integer) → *Void*
Description: Approves a pending request for a student admit for a specific exam and year in the database.

(e) **reject_pending_request**(*examID* As Integer, *studentID* As Integer, *year* As Integer) → *Void*
Description: Rejects a pending request for a student admit for a specific exam and year in the database.

(f) **get_ee_studentadmit**(*studentID* As Integer, *examID* As Integer, *year* As Integer) → *EEStudentAdmit*
Description: Retrieves student admit details for a specific student, exam, and year from the database.

(g) **AddStudentAdmit**(*studentID* As Integer, *examID* As Integer, *year* As Integer) → *Void*
Description: Adds a new student admit for a specific student and exam to the database.

(h) **UpdateFeeStatus**(*studentID* As Integer, *examID* As Integer, *year* As Integer) → *Void*
Description: Updates the fee status for a specific student, exam, and year in the database.

5. **Institute Handler**

(a) **GetAllInstitutions**() → *EdInstitution[]*
Description: Retrieves details of all institutions from the database and returns an array of EdInstitution objects.

(b) **GetAllRequests**(*principalID* As Integer) → *Admissions[]*
Description: Retrieves all pending admission requests for a specific principal from the database and returns an array of Admissions objects.

(c) **GetLastStudentRequest**(*studentID* As Integer) → *(Status As String, InstituteID As Integer)*
Description: Retrieves the status and the associated institute ID of the last admission request made by a student from the database.

(d) **InsertCertificate**(*certData As CertificateData*) → *Void*
Description: Inserts certificate data into the database based on the provided CertificateData object.

(e) **GetCertificatesByType**(*studentID* As Integer, *TypeOfCert* As String) → *List(Of CertificateData)*
Description: Retrieves certificates of a specific type for a given student from the database and returns a list of CertificateData objects.

4.4.4 Error Handling

1. **Exception Handling for SQL Connections**: All SQL connections are established within a try-catch block to handle exceptions gracefully, ensuring that any errors encountered during database operations are handled appropriately without disrupting the application flow.

2. **Memory Management (BC2004)**: The application monitors memory usage and handles the BC2004 error, indicating an out-of-memory situation. It accomplishes this by tracking the forms that are opened and closing them when they are no longer needed, effectively managing memory usage and preventing memory-related errors.

3. **Permission Feature Access**: The application checks the Profile of the user in the database. Only Those having Teacher as a profession can access teacher privileges. People

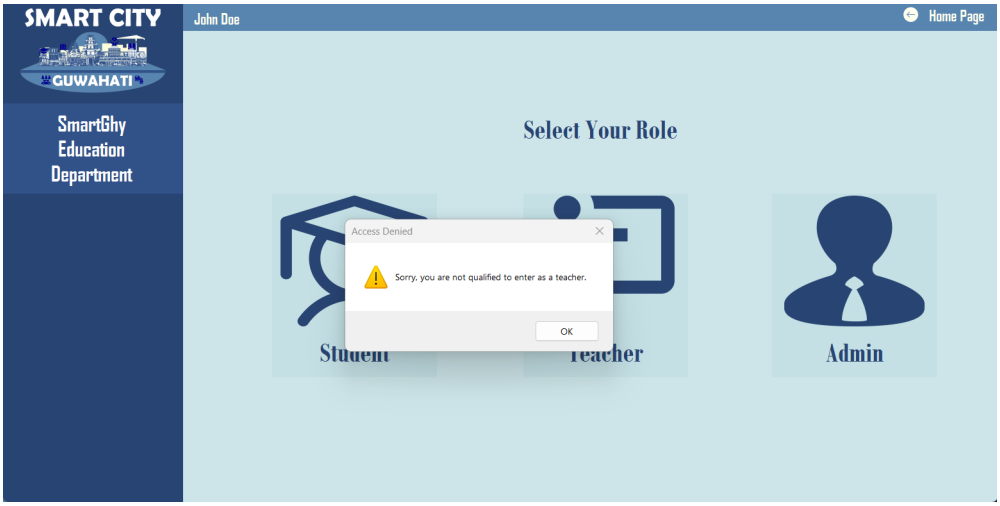

Figure 14: Only permitted users can access Teacher and Admin facilities

with admin profession and particular roles can access admin privileges for institute or ECourse.

4. **Student Only Access**: The application allows only Students to enter the Institute and Moodle sections as O=only they are qualified to get admitted to institutions. The Ecourse and Entrance Exam features are available for everyone. Appropriate error messages displayed.

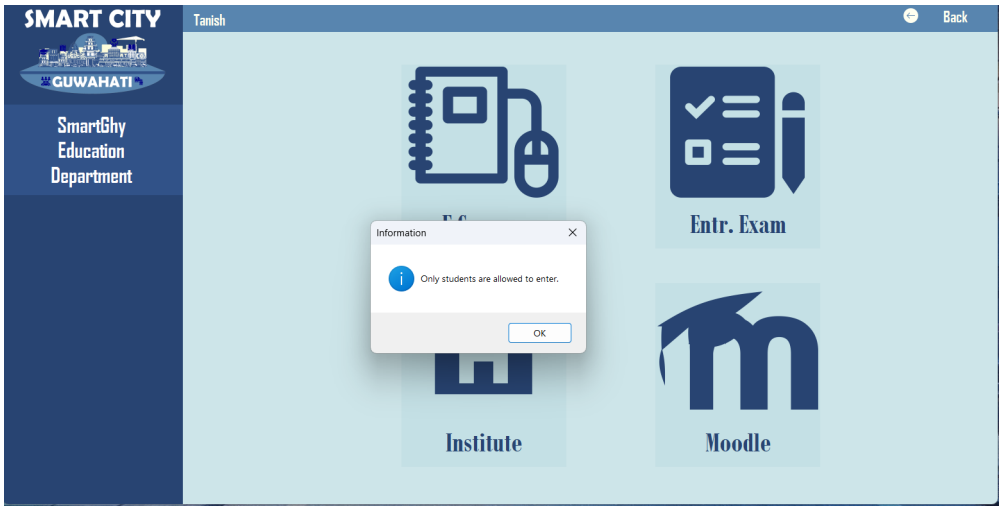

Figure 15: Only Students can enter the Institutions and Moodle Sections

5. **Invalid Video Link**: The Resouce pages for the courses in Moodle and ECourse sections check if the link given for Intro video or the Resource video is valid or not and displays appropriate error message.

6. **Student Institute Admission**: A Student can only be part of one Institution at a time. If not in any institution, he/she can apply to an Institution but only one at a time. After the admin approves/rejects the student he can apply elsewhere. In case student is part of some institution then he/she has the option to apply for transfer to some other institution.

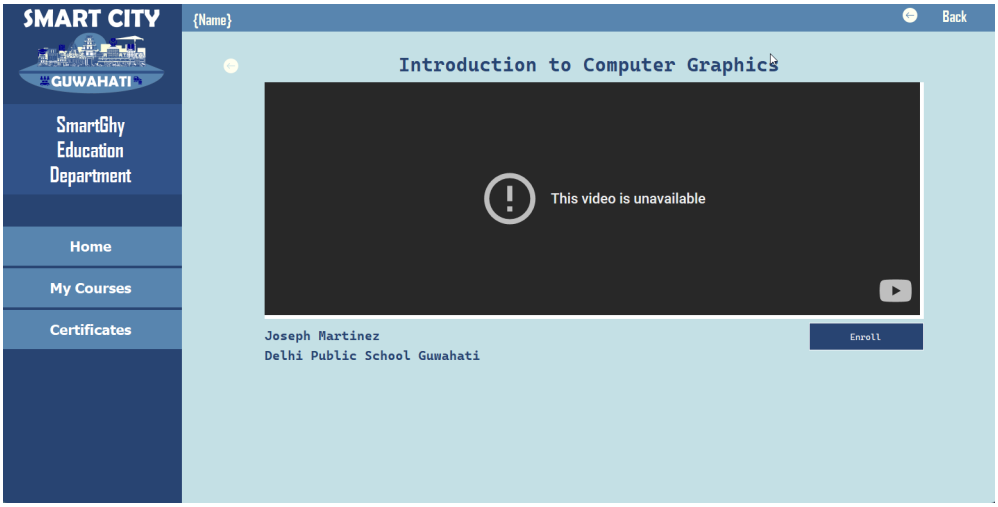

Figure 16: Invaid Youtube links will give error

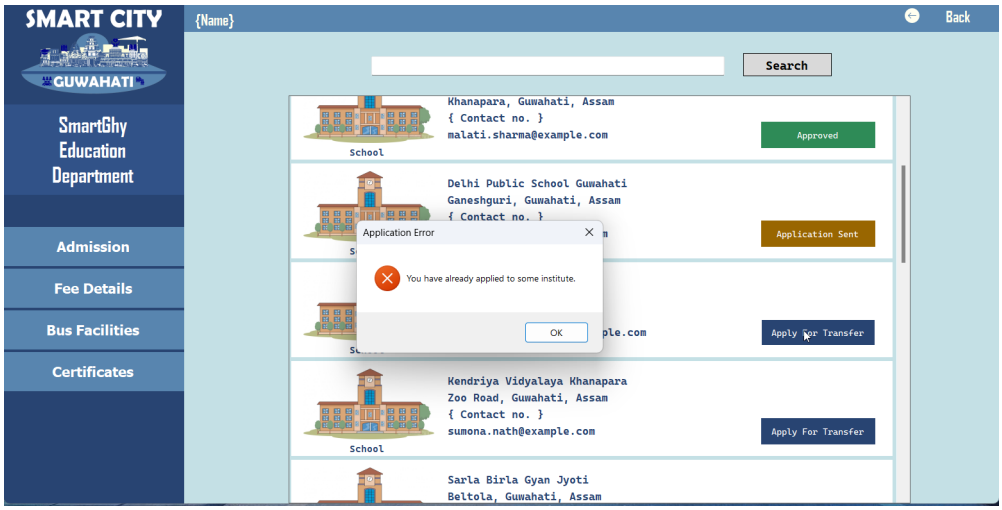

Figure 17: Students can apply for admissions only one institute as a time

4.5 Healthcare Management

4.5.1 Introduction

The Healthcare Management module serves as a comprehensive solution to address the diverse needs of healthcare facilities, encompassing various sectors crucial for efficient operations and enhanced patient care.Within this module, distinct sectors from Doctor appointment and Pharmacy to Blood Donation and Patient history are meticulously designed to streamline processes, optimize resource allocation, and prioritize patient well-being.It provides a comprehensive solution to efficiently manage patient data, appointments, and medical records.

4.5.2 Features

From User side

1. **Book an Appointment** Users can choose their preferred hospital and select the relevant department. Following selection, users can input their symptoms into a designated

textbox. Upon pressing the enter button, the message containing the user's symptoms is promptly dispatched to the hospital for appointment scheduling

2. **Buy Medicines** Users can initiate a search for a specific medicine within the module. After hitting the search button, a list of pharmacies stocking the requested medicine is displayed. Users can then choose their preferred pharmacy from the provided options to proceed with their purchase.

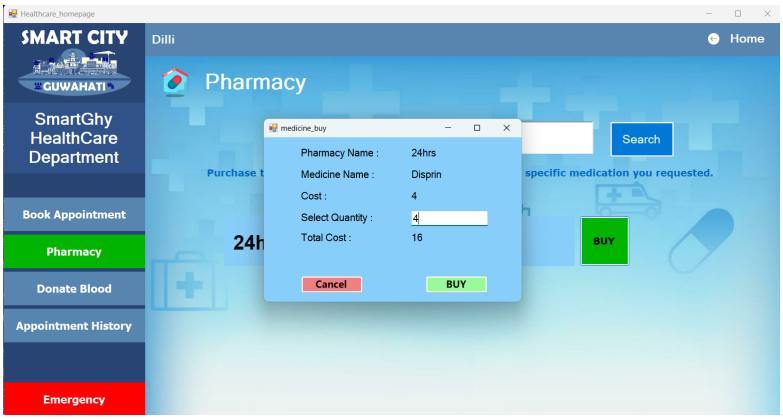

Figure 18: Popup shows when user try to buy medicine

3. **Blood donation** Users can select their preferred hospital for blood donation and provide necessary details including donation date, time, age, and blood group. Upon pressing the enter button, the system promptly schedules a blood donation appointment for the user on the specified date, facilitating their altruistic contribution to the community's healthcare needs.

4. **View User History** Users can access a comprehensive overview of their healthcare journey, encompassing doctor appointment bills, pharmacy transactions, and blood donation history. Through this feature, users can review past appointments, track medication purchases, and view records of their altruistic contributions through blood donation.

5. **Request an Ambulance** Users simply input their current address into the designated field. The system automatically dispatches an ambulance from the nearest hospital to the user's location, ensuring prompt emergency response and timely access to medical assistance.

From Admin side

1. **Appointment corner** Administrators oversee doctor assignments and approve or cancel appointments based on doctor availability and user-requested appointment dates, ensuring efficient scheduling and optimal utilization of medical resources.

2. **Doctor and Staff corner** Administrators have the capability to oversee and update doctor information, as well as manage hospital staff assignments and roles, ensuring smooth operation and coordination within the healthcare facility.

3. **Medicine corner** Administrators maintain a comprehensive record of medicines purchased by users, tracking the inventory levels across pharmacies and monitoring the quantity of each medicine available. This ensures efficient management of medicine stocks and facilitates timely replenishment to meet patient needs.

4. **Health record corner** Administrators are responsible for updating the health records of users following doctor appointments, medicine purchases, and blood donations. Additionally, they generate birth and death certificates for users as needed, ensuring accurate and comprehensive documentation of healthcare events and vital statistics.

5. **Emergency corner** Administrators oversee the allocation and dispatch of ambulances in urgent situations.

6. **Blood donation corner** Administrators oversee blood donation appointments and manage the blood bank inventory. They ensure efficient scheduling of donation appointments and maintain adequate blood supply levels in the blood bank to meet the healthcare facility's needs and support patient care.

4.5.3 Functions

Book appointment inner form

1. **LoadandBindDataGridView**()
   Description: Displays a list of hospitals available in the city.

2. **LoadandBindDataGridView2**()
   Description: Displays a list of departments available in the selected hospital.

3. **Button1_Click**()
   Description: Books the appointment for the user by collecting information about symptoms.

   Pharmacy inner form

4. **LoadandBindDataGridView**()
   Description: Shows a list of pharmacies where the requested medication is available.

5. **Buy_Click**()
   Description: Displays a messagebox allowing users to select the quantity of medication they wish to purchase.

   History inner form

6. **d1_Click**()
   Description: Displays the user's history of doctor appointments including date, hospital ID, hospital name, doctor ID, doctor name, status, and bill.

7. **d2_Click**()
   Description: Shows the user's history of medication bills including date, medication name, quantity, and bill.

8. **d3_Click**()
   Description: Displays the user's blood donation history including date, time, blood group, and status.

   Blood Donation inner form

9. **LoadandBindDataGridView**()
   Description: Displays a list of hospitals where users can donate blood.

10. **d1_Click**()
    Description: Checks if user inputs are valid and stores the blood donation appointment in the database if valid and books appointment for blood donation.

    Emergency inner from

11. **d1_Click**()
    Description: Collects the user's current address location and dispatches an ambulance from the nearest available hospital to the user's location.

4.5.4 Error Handling

It involves implementing mechanisms to gracefully manage unexpected situations, such as incorrect user input,system failures, or unexpected behavior, to ensure the stability and reliability of the software.

we will face this problem in Book appointment and Donate blood inner forms.Figure 15 and Figure 16 depict the error handling mechanisms implemented in the Book Appointment inner form. Similarly Figure 17 and Figure 18 depict the error handling mechanisms implemented in the Donate Blood inner form.

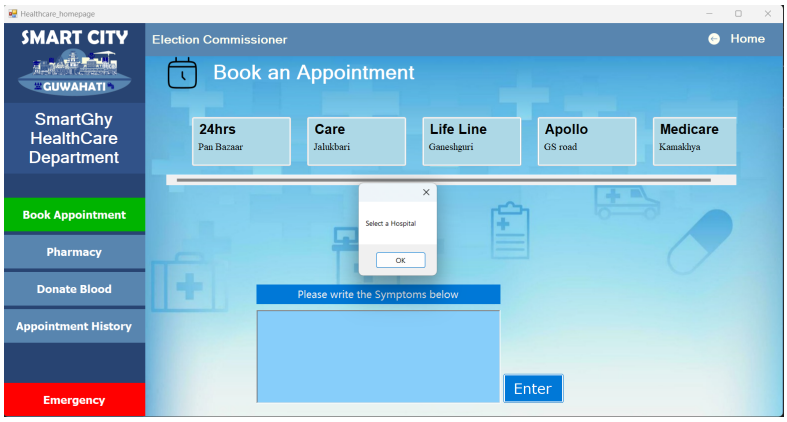

Figure 19: Popup shows when user hit enter button without selecting hospital.

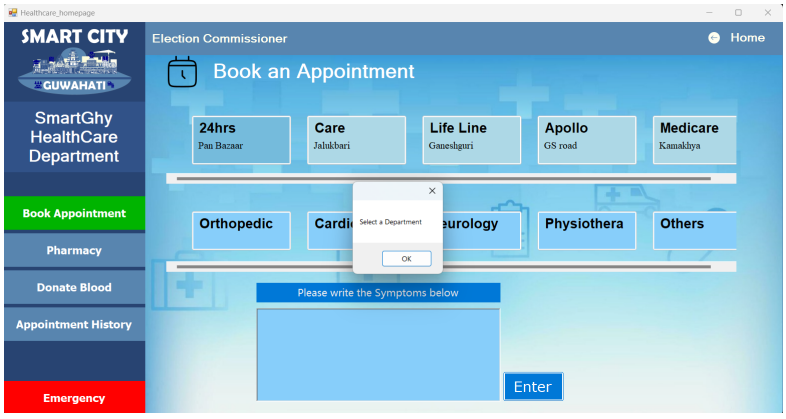

Figure 20: Popup shows when user hit enter button without selecting the department

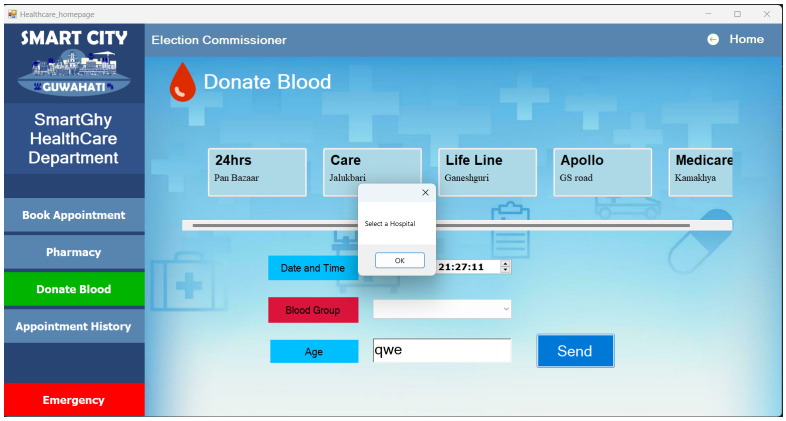

Figure 21: Popup shows when user hit enter button without selecting hospital.

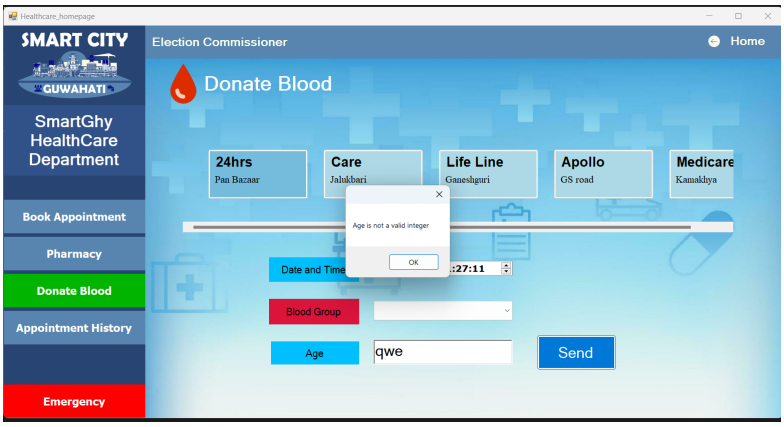

Figure 22: Popup shows when user age type is other than Interger

4.6 Transportation Management

4.6.1 Introduction

The Transportation management comes under Ministry of Transportation in the smart city, who will be elected in the election conducted by Administrative Hierarchy module. This Section of the Smart City Management provides various services like Driving License Acquisition, Toll Booth Management, Ride Sharing, Public Transport Management, Vehicle Registration and also administrative features for the minister of transportation.

4.6.2 Features

Stakeholders:

1. **Citizens:** Any resident, irrespective of profession, registered in Smart City Portal.
2. **Transportation Minister:** Duly elected Minister and also the bank account holder to which payment paid in transportation department goes to.
3. **Department Officers:**
    a. **R.T.O Officer:** Administrator for Vehicle Registration and Driving License Management.
    b. **Public Transport Officer:** Administrator for Toll Plaza, Fastag Divisions.
    c. **Ride Sharing regulators:** Administrator for regulating Ride Sharing Posts.
4. **Drivers and Riders:** *Drivers* post Ride Sharing Notifications for *Riders* to choose them.

User wise features:

1. **Citizens:**
    (a) **Driving License Registration**
        a. **User Data Display:** Displays user information and existing driver's license entries.
        b. **Payment Gateway Integration:** Allows users to make payments for new or renewed driving licenses.
        c. **Dynamic UI:** Updates UI elements based on user actions like payment initiation or cancellation.
    (b) **Vehicle Registration Request**
        a. **Display Registered Vehicles:** Displays registered vehicles for users.
        b. **View PDF and Image:** Users can view PDF invoices and vehicle images associated with each registration.
        c. **Payment Integration:** Users can make payments for vehicle registration with validation checks.
        d. **Image and PDF Upload:** Users can upload vehicle images and PDF invoices for registration.

(c) **Fastag Acquisition and Renewal:**

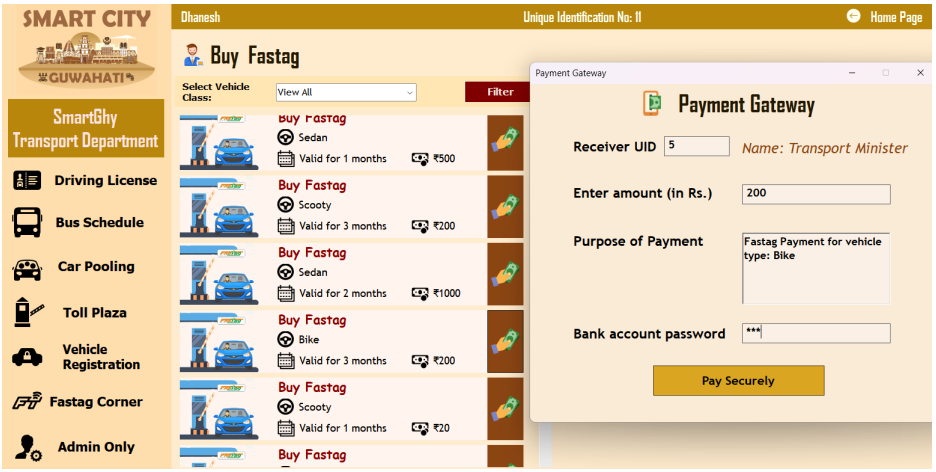

Figure 23: User can pay and buy Fastag for the vehicles to which they have Driving License

a. **Fastag Buying:** Users can buy Fastag with their money for the vehicle that they own.

Figure 24: Users can topup/renew the fastag that they have bought by paying the appropriate fee.

b. **Fastag Renewal:** Users can renew or Topup the Fastag in the same portal after its expiry.

(d) **Toll Plaza Simulation**

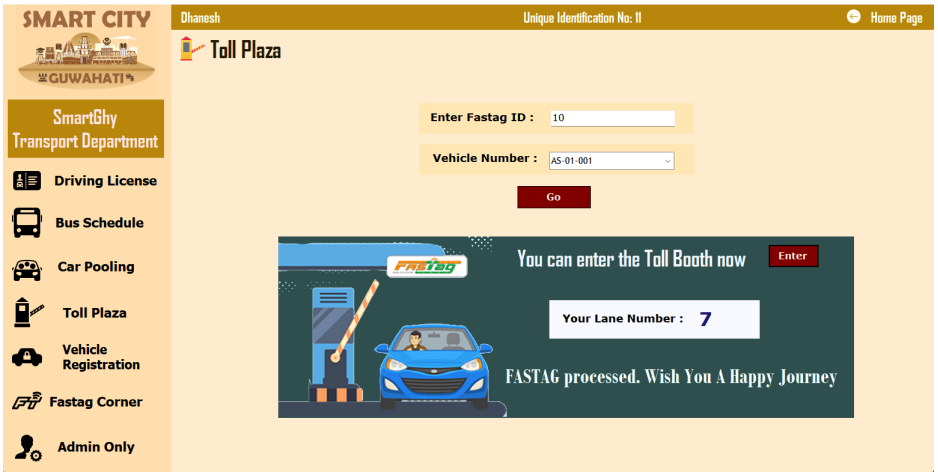

Figure 25: Successful toll plaza entry after fastag validation

a. **Toll Gate Enter:** Toll Gate is simulated by having the user enter the vehicle number after which a suitable lane would be automatically chosen.

b. **Fastag Validation:** The balance of Fastag is checked after which the processing is completed.

(e) **Ride Sharing and Ride Sharing Chats**

a. **Post Ride Feature:** Drivers can post a ride sharing request for ride sharing regulator's approval.

b. **View Ride Sharing Posts:** Any valid user can view all the approved ride sharing requests along with the fare and capacity left.

c. **View Ride Sharing Chats:** Riders and drivers can engage in chat feature and the drivers can update the fare based on rider's responses.

d. **Payment Integration:** Payment Gateway is integrated for payments upon which drivers can accept riders.

(f) **TransportBusSchedule**

The user can buy bus tickets from the available buses. He has to select source and destination, then different buses available will be shown to him, he has to then click on buy button. A payment gateway will then open and he has to enter his password to complete the transaction.

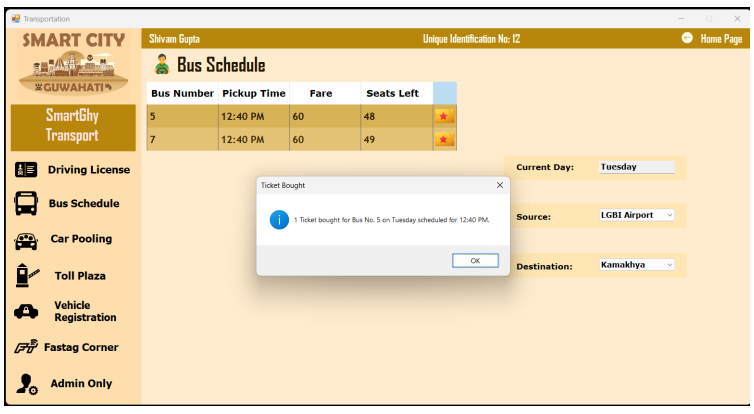

Figure 26: Successfully bought a bus ticket

2. **Transportation Minister:**

   (a) **Admin HomePage**

      a. **Viewing Various Corners:** Administrators can view view various corners like R.T.O corner, Public Transport Corner and also regulate the department sub-admins.

      b. **Authentication:** Transport minister can access all pages while department sub-admins can access only particular parts that are relevant to their position. Eg: Public Transport officer can't access Vehicle Request Regulation page.

   (b) **Admin Ride Sharing**

      a. **Viewing Ride Sharing Requests:** Citizens can view ride sharing requests submitted by others.

      b. **Filtering Requests:** Requests can be filtered based on their status for easier management.

      c. **Dynamic Post Generation:** Requests are dynamically generated as visual components within the interface.

      d. **User Profile Display:** User profiles, including names and profile photos, are displayed alongside each request.

   (c) **Admin Driving License Request**

      a. **Viewing Driving License Requests:** Transport administrators can view driving license requests.

      b. **Accepting or Rejecting Requests:** Administrators can manage requests directly from the interface.

      c. **Dynamic Data Update:** The interface dynamically updates upon accepting or rejecting requests.

      d. **User Profile Display:** Information such as name, age, and requested vehicle type is displayed.

   (d) **Admin Vehicle Registration Request**

      a. **Viewing Vehicle Registration Requests:** Administrators can view vehicle registration requests.

      b. **Accepting or Rejecting Requests:** Requests can be accepted or rejected from the interface.

      c. **Displaying Vehicle Images:** Vehicle images provide visual context alongside registration requests.

      d. **Accessing PDF Documents:** PDF documents associated with registration requests are accessible.

      e. **Dynamic Data Update:** The interface dynamically updates upon accepting or rejecting requests.

3. **Department Officers:**

   (a) **R.T.O Officer:**

   - **Admin Driving License Request**

      a. **Viewing Driving License Requests:** Transport administrators can view driving license requests.

b. **Accepting or Rejecting Requests:** Administrators can manage requests directly from the interface.

c. **Dynamic Data Update:** The interface dynamically updates upon accepting or rejecting requests.

d. **User Profile Display:** Information such as name, age, and requested vehicle type is displayed.

- **Admin Vehicle Registration Request**

a. **Viewing Vehicle Registration Requests:** Administrators can view vehicle registration requests.

b. **Accepting or Rejecting Requests:** Requests can be accepted or rejected from the interface.

c. **Displaying Vehicle Images:** Vehicle images provide visual context alongside registration requests.

d. **Accessing PDF Documents:** PDF documents associated with registration requests are accessible.

e. **Dynamic Data Update:** The interface dynamically updates upon accepting or rejecting requests.

(b) **Ride Sharing regulators:**

- **Admin Ride Sharing**

a. **Viewing Ride Sharing Requests:** Citizens can view ride sharing requests submitted by others.

b. **Filtering Requests:** Requests can be filtered based on their status for easier management.

c. **Dynamic Post Generation:** Requests are dynamically generated as visual components within the interface.

d. **User Profile Display:** User profiles, including names and profile photos, are displayed alongside each request.

(c) **Public Transport Officer:**

- **TransportManageTollGatesAdmin**
  The Public Transport Officer can use this page to add new toll gates, update an existing toll gate or delete an existing toll gate. He does by specifying the Lane ID, description, type of vehicles allowed and fare for that toll gate service pressing on Add button to add a new toll gate or press on Update button to update an existing toll gate.

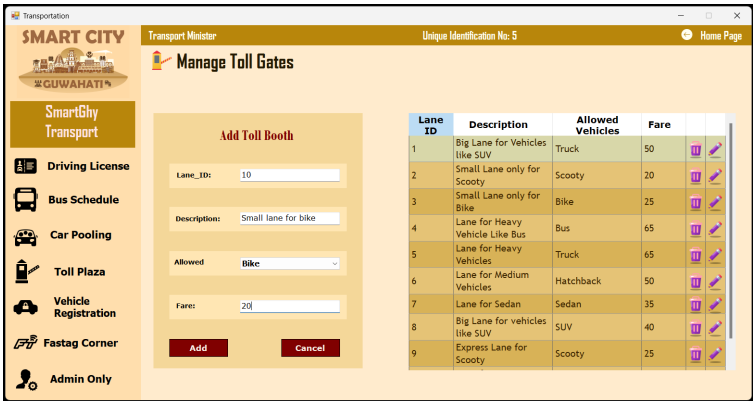

Figure 27: Adding a New Toll Booth

- **TransportationBusSchedulesAdmin**
  The Public Transport Admin can use this page to add new bus schedules, update existing bus schedules or delete an existing bus schedule. He has to enter Bus Number, days when bus will be operating, pickup time, Source, Destination, Fare and Number of seats. He should click on Add to add new bus schedule and on Update to update existing bus schedule.
- **TransportationManageFastagAdmin**
  The Toll Gate Officer can use this page for adding new fastag plan, update existing fastag plans or delete existing fastag plan. He has to specify plan number, vehicle class allowed, fee amount and validity.
- **TransportManageBusStopAdmin**
  Public Transport Officer can use this page to add new bus stop, edit bus stop name or delete an existing bus stop. He has to enter the bus stop name and click on add/update button.

4.6.3 Functions

1. **Admin Home**
   (a) **FormClosed**(*sender*, *e*) → Void
   Description: Disposes all child forms when the parent form is closed.
   (b) **TransportAdminDLReq_Click**(*sender*, *e*) → Void
   Description: Opens the Driving License request page if the user is authorized.
   (c) **TransportAdminVRReq_Click**(*sender*, *e*) → Void
   Description: Opens the Vehicle Registration request page if the user is authorized.
   (d) **TransportMangeBusStopAdmin_Click**(*sender*, *e*) → Void
   Description: Opens the Bus Stops management page if the user is authorized.
   (e) **TransportManageTollGatesAdmin_Click**(*sender*, *e*) → Void
   Description: Opens the Manage Toll Gates page if the user is authorized.
2. **Transport TG Enter**
   (a) **CompareTypes**(*Integer id*) → Boolean
   Description: Compares the vehicle types obtained from different sources and returns True if they match, False otherwise.

(b) **GetVehicleNumberComboBox**() → Void
Description: Retrieves the vehicle numbers for the user and populates them into the ComboBox.

(c) **Button3_Click**(*sender*, *e*) → Void
Description: Checks for valid Fastag ID and selected vehicle number, then compares vehicle types and fetches a random toll ID.

(d) **Button1_Click**(*sender*, *e*) → Void
Description: Checks for sufficient balance in Fastag, decrements balance, and inserts tollgate entry if conditions are met.

(e) **TransportTGEnter_Load**(*sender*, *e*) → Void
Description: Loads the form and invokes the GetVehicleNumberComboBox subroutine to populate vehicle numbers.

3. **RideSharingMain**

(a) **GetPlacesFromDatabase**() → *List(Of Place)*
Description: Retrieves place data from the database and returns a list of places.

(b) **LoadandBindDataGridView**() → Void
Description: Loads and binds data to the DataGridView for tracking ride-sharing posts.

(c) **AddPost**(*name: String*, *poster_uid: Integer*, *vehicleNum: String*, *note: String*, *datetime: String*, *fromPlace: String*, *toPlace: String*, *fare: Integer*, *capacity: Integer*, *postNum: Integer*, *image: Image*) → Void
Description: Adds a ride-sharing post to the UI with specified details.

(d) **GetVehicleNumberComboBox**() → Void
Description: Retrieves the vehicle numbers for the user and populates them into the ComboBox.

(e) **TransportationInnerScreen_Load**(*sender: Object*, *e: EventArgs*) → Void
Description: Loads the form and invokes the GetVehicleNumberComboBox subroutine to populate vehicle numbers.

(f) **ClearAddRidePost**() → Void
Description: Clears the input fields for adding a ride-sharing post.

(g) **Button3_Click**(*sender: Object*, *e: EventArgs*) → Void
Description: Handles the click event of the "Add" button to add a ride-sharing post.

(h) **LoadPosts**() → Void
Description: Loads ride-sharing posts from the database and adds them to the UI.

(i) **ComboBox1_SelectedIndexChanged**(*sender: Object*, *e: EventArgs*) → Void
Description: Handles the event when a selection is changed in ComboBox1 to prevent selecting the same place in both ComboBoxes.

(j) **ComboBox2_SelectedIndexChanged**(*sender: Object*, *e: EventArgs*) → Void
Description: Handles the event when a selection is changed in ComboBox2 to prevent selecting the same place in both ComboBoxes.

(k) **Label4_Click**(*sender: Object*, *e: EventArgs*) → Void
Description: Reloads the DataGridView and ride-sharing posts when clicked.

(l) **DataGridView1_CellContentClick**(*sender: Object*, *e: DataGridViewCellEventArgs*) → Void
Description: Handles the click event on DataGridView cells, specifically for deleting ride-sharing posts.

(m) **Button2_Click**(*sender: Object*, *e: EventArgs*) → Void
Description: Clears the input fields for adding a ride-sharing post when clicked.

4. **RideSharingChats**

(a) **DataGridView1_CellContentClick**(*sender: Object*, *e: DataGridViewCellEventArgs*) → Void
Description: Handles the click event on DataGridView cells, specifically for approving or rejecting ride-sharing requests.

(b) **AddChat**(*name: String*, *postNum: Integer*, *datetime: String*, *comment: String*, *image: Image*, *isRider: Boolean*) → Void
Description: Adds a chat bubble to the UI with specified details.

(c) **LoadChats**() → Void
Description: Clears all existing chats and loads new ride-sharing chats from the database into the UI.

(d) **DataGridView1_CellFormatting**(*sender: Object*, *e: DataGridViewCellFormattingEventArgs*) → Void
Description: Formats the cells in the DataGridView, specifically the fee_paid column, to display human-readable values.

(e) **Button1_Click**(*sender: Object*, *e: EventArgs*) → Void
Description: Handles the click event of the "Pay" or "Withdraw" button for ride-sharing fees.

(f) **Button3_Click**(*sender: Object*, *e: EventArgs*) → Void
Description: Handles the click event of the "Set Fare" button to update fare information for ride-sharing.

(g) **Button5_Click**(*sender: Object*, *e: EventArgs*) → Void
Description: Handles the click event of the "Send" button to send a message in the ride-sharing chat.

5. **TransportTollPlaza**

(a) **GetVehicles**() → *List(Of Vehicle)*
Description: Retrieves a list of vehicles from the vehicleTypeMap and returns it.

(b) **LoadPosts**(*Optional filter_type As Integer = -1*) → Void
Description: Loads posts related to fastag plans from the database and adds them to the UI. If a filter type is specified, only posts matching that type are loaded.

(c) **LoadPurchases**() → Void
Description: Loads purchases related to fastag plans from the database and adds them to the UI.

(d) **AddPostPlan**(*postNum: Integer*, *vehtype: String*, *Optional validity: String = "Valid for 3 months"*, *Optional fare: Integer = 0*) → Void
Description: Adds a fastag plan post to the UI with specified details.

(e) **AddPostPurchase**(*postNum: Integer*, *ft_id: Integer*, *validity: Integer*, *Optional drvlicensenum: String = "12345"*, *Optional dt: String = "21st March 2024"*, *Optional fare: Integer = 0*) → Void
Description: Adds a fastag purchase post to the UI with specified details.

(f) **Label4_Click**(*sender: Object*, *e: EventArgs*) → Void
Description: Handles the click event of Label4 to filter and reload posts based on the selected vehicle type.

6. **Admin Ride Sharing Request**

(a) **LoadPosts**() → Void
Description: Retrieves ride-sharing requests from the database and populates the interface with them, optionally filtered by status.

(b) **AddPost**(*reqid: Integer*, *dname: String*, *vehid: String*, *datetime: String*, *fromPlace: String*, *toPlace: String*, , *dlid: Stringfare_per_person: Integer*, *capacity: Integer*, *postNum: Integer*, *Status: String*, *image: Image*) → Void
Description: Adds a ride-sharing post to the UI with specified details.

(c) **Button1_Click**(*sender*,*e*) → Void
Description: Handles the filter button click event, triggering the reloading of requests based on the selected filter.

7. **Admin Driving License Request**

(a) **DataGridView1_CellContentClick**(*sender As Object*, *e As DataGridViewCellEventArgs*) → Void
Description: Handles cell click events in the DataGridView, enabling administrators to accept or reject driving license requests.

(b) **LoadandBindDataGridView**() → Void
Description: Retrieves and binds driving license requests from the database to the DataGridView, updating the interface accordingly.

8. **Admin Vehicle Registration Request**

(a) **DataGridView1_CellContentClick**(*sender As Object*, *e As DataGridViewCellEventArgs*) → Void
Description: Handles cell click events in the DataGridView, allowing administrators to view vehicle images and associated PDF documents, as well as accept or reject registration requests.

(b) **ByteArrayToImage***Byte()* → *Image*
Description: Converts byte array data retrieved from the database into an Image object.

(c) **ResizeImage**(*Image*,*Integer*,*Integer*) → *Image*
Description: Resizes the original image to the specified width and height.

(d) **LoadandBindDataGridView**() → Void
Description: Retrieves and binds vehicle registration requests from the database to the DataGridView, updating the interface accordingly.

9. **Driving License Registration**

(a) **LoadAndBindData**() → Void
Description: This function loads and binds user data and driver's license entries from the database to the UI components. It handles the display of user information, including name, age, address, gender, and profile photo, as well as existing driver's license entries such as license ID, vehicle type, issue date, validity period, and test status.

(b) **Paytb_Click():**(*sender*,*e*) → Void
Description: This function handles the payment button click event, setting a flag to initiate payment processing. It triggers the payment gateway integration and initiates the payment process for new or renewed driving licenses.

(c) **Canceltb_Click():**(*sender*,*e*) → Void
Description: This function handles the cancel button click event, resetting the vehicle type selection in the ComboBox. It allows users to cancel their selection and revert any changes made during the license registration or renewal process.

10. **Vehicle Registration Request**

(a) **DataGridView Cell Click**(*sender As Object*, *e As DataGridViewCellEventArgs*) → Void
Description: This function handles cell clicks in the DataGridView, enabling users to view PDF invoices or vehicle images associated with each registered vehicle. It allows users to access additional details and documents related to their registered vehicles.

(b) **ImageToByteArray***Image* → *Byte()*
Description: Converts an Image object given by the user into byte array data to store in the database.

(c) **ResizeImage**(*Image*,*Integer*,*Integer*) → *Image*
Description: This function resizes vehicle images to fit within the DataGridView cell for better presentation. It ensures that vehicle images are appropriately sized and displayed within the interface, optimizing space utilization and improving visual clarity.

(d) **LoadandBindDataGridView** () → Void
Description: This function fetches and binds vehicle registration data to the DataGridView, including user details and registered vehicles. It populates the interface with relevant data, ensuring that users have access to up-to-date information regarding their registered vehicles.

(e) **GenerateRandomId**() → *String*
Description: This function generates a random unique ID for new vehicle registrations to avoid duplicates. It ensures that each vehicle registration request is assigned a unique identifier, preventing conflicts and ensuring data integrity.

(f) **CheckIdExists**(*String*) → *Boolean*
Description: This function checks if a generated ID already exists in the database to ensure uniqueness. It verifies the uniqueness of generated IDs and prevents duplicate entries in the database.

11. **TransportManageTollGatesAdmin**

(a) **ShowEditOption**(*String*, *String*,*String*,*String*) → Void
Description: Displays edit options for updating a toll gate entry. Takes four input parameters representing the lane ID, description, allowed vehicle types, and fare per vehicle. No output.

(b) **GetVehicles** () → *List(Of Vehicle)*
Description: Retrieves a list of vehicle types. Returns a list of Vehicle objects.

(c) **LoadandBindDataGridView** () → Void
Description: Loads toll gate data from the database and binds it to the DataGridView control. No input or output parameters.

(d) **DataGridView1_CellContentClick** () → Void
Description: Handles the cell content click event for the DataGridView. Performs actions such as editing and deleting toll gate entries based on user interaction. Takes DataGridViewCellEventArgs as input parameter. No output.

(e) **Button2_Click**() → Void
Description: Handles the click event for the "Clear" button. Clears input fields on the form. Takes Object and EventArgs as input parameters. No output.

(f) **Button3_Click**() → Void
Description: Handles the click event for the "Add/Update" button. Adds a new toll gate entry or updates an existing one based on user input. Takes Object and EventArgs as input parameters. No output.

12. **TransportationBusSchedulesAdmin**

(a) **ShowEditOption**(*String*) → Void
Description: Populates the form fields with data from the database for editing a bus schedule.

(b) **GetPlacesFromDatabase** () → *List(Of Place)*
Description: Retrieves place data from the database.

(c) **LoadandBindDataGridView** () → Void
Description: Loads bus schedule data from the database and binds it to the DataGridView.

(d) **DataGridView1_CellContentClick** () → Void
Description: Handles cell click events in the DataGridView, allowing editing or deletion of bus schedules.

(e) **Button2_Click**() → Void
Description: Handles the click event of Button2, clearing form fields for adding a new bus schedule.

(f) **Button3_Click**() → Void
Description: Handles the click event of Button3, adding or updating a bus schedule based on form input.

13. **TransportationManageFastagAdmin**

(a) **ShowEditOption**(*String*, *String*,*String*,*String*) → Void
Description: Updates UI elements to show options for editing a Fastag plan.

(b) **GetVehicles** () → *List(Of Vehicle)*
Description: Retrieves a list of vehicle types. Returns a list of Vehicle objects.

(c) **LoadandBindDataGridView ()** → Void
Description: Loads Fastag plans from the database and binds them to the DataGridView.

(d) **DataGridView1_CellContentClick ()** → Void
Description: Handles cell click events in the DataGridView, allowing editing or deleting of Fastag plans.

(e) **Button2_Click**() → Void
Description: Handles the click event of the "Clear" button, clearing input fields.

(f) **Button3_Click**() → Void
Description: Handles the click event of the "Add" or "Update" button, adding or updating Fastag plans in the database.

14. **TransportGlobals**

(a) **GetVehicleType**(*Integer*) → *String*
Description: Retrieves the vehicle type based on the provided vehicle ID.

(b) **GetVehicleTypeID** (*String*) → *Integer*
Description: Retrieves the vehicle ID based on the provided vehicle type.

15. **TransportManageBusStopAdmin**

(a) **ShowEditOption**(*String*) → Void
Description: Displays the edit option by populating the UI elements with the details of the selected bus stop.

(b) **LoadandBindDataGridView ()** → Void
Description: Loads bus stop data from the database and binds it to the DataGridView for display.

(c) **DataGridView1_CellContentClick ()** → Void
Description: Handles cell clicks in the DataGridView. It performs edit and delete operations based on the user's actions.

(d) **Button1_Click**() → Void
Description: Handles the click event of the Add/Update button. It adds a new bus stop or updates an existing one based on user input.

(e) **Button3_Click**() → Void
Description: Resets the UI elements to their default state for adding a new bus stop.

16. **TransportBusSchedule**

(a) **ShowBuyOption**(*String*) → Void
Description: Displays the option to buy a ticket for the selected bus schedule.

(b) **GetPlacesFromDatabase** () → *List(Of Place)*
Description: Retrieves place data from the database to populate the source and destination ComboBoxes.

(c) **LoadandBindDataGridView ()** → Void
Description: Loads bus schedule data from the database and binds it to the DataGridView for display.

(d) **DataGridView1_CellContentClick ()** → Void
Description: Handles cell clicks in the DataGridView, particularly for buying tickets.

17. **TransportFRAdmin**

(a) **TransportFRAdmin_Load** () → Void
Description: This method is triggered when the TransportFRAdmin form loads. It establishes a connection with the database, executes a SQL query to fetch administrative records, and updates the UI labels with the retrieved data.

18. **TransportSendNotification**

(a) **TransportationInnerScreen_Load** () → Void
Description: This method is triggered when the TransportSendNotification form loads. It serves as an initialization function for the form but currently does not contain any specific functionality.

(b) **Button1_Click**() → Void
Description: This method is invoked when the user clicks the "Send Notification" button. It validates the user inputs for Ministry ID, Receiver UID, notification title, and message. If all inputs are valid, it triggers the SendNotifications function to send the notification and displays a success message. If any input is invalid, appropriate error messages are shown.

(c) **Button2_Click**() → Void
Description: This method is triggered when the user clicks the "Clear" button. It clears all the text boxes on the form, allowing the user to enter new notification details..

4.6.4 Error Handling

Note: We have handled many possible errors but are writing only few here to avoid verbosity

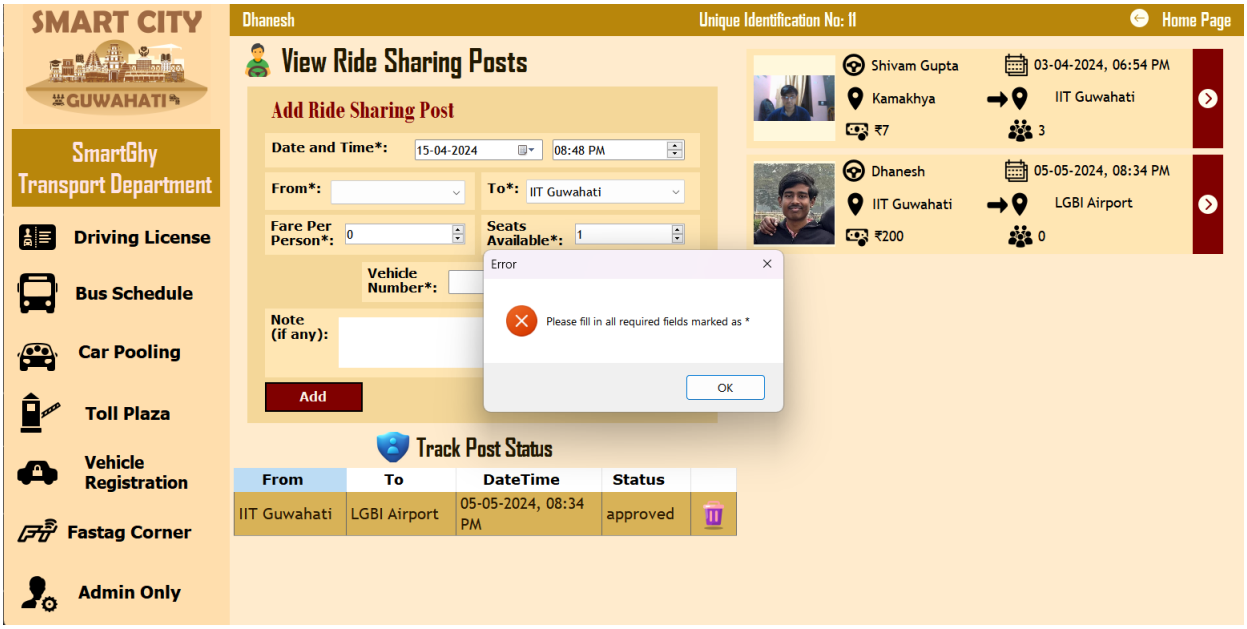

Figure 28: Error displayed when mandatory fields are not filled

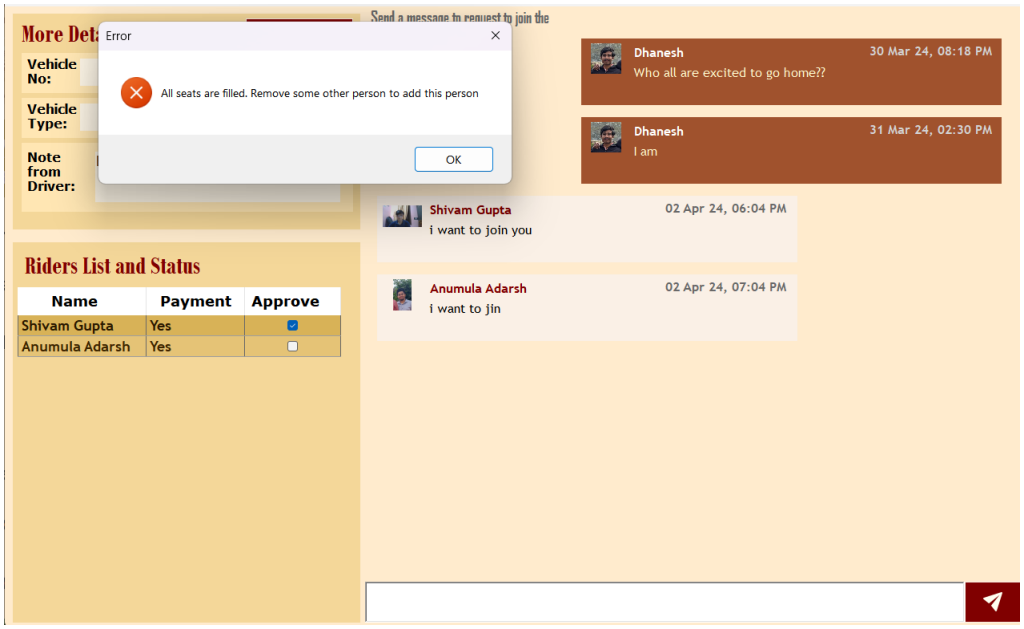

Figure 29: Error displayed when the driver selects more people for ride sharing than the capacity mentioned

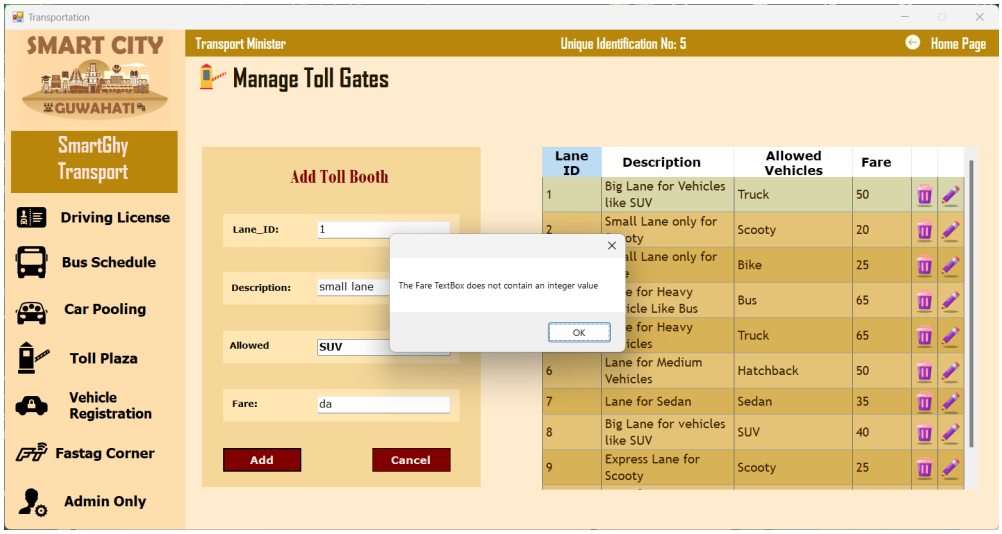

Figure 30: In Fare textbox if we enter some other data apart from integer then an error message box is shown to the user.

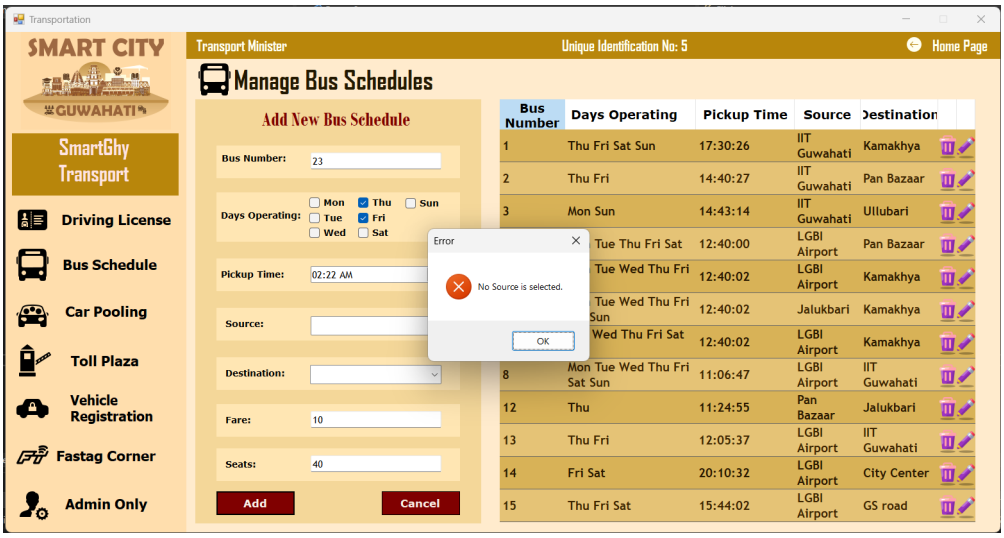

Figure 31: If we click on add button without selecting the source location then an error is prompted to the screen.

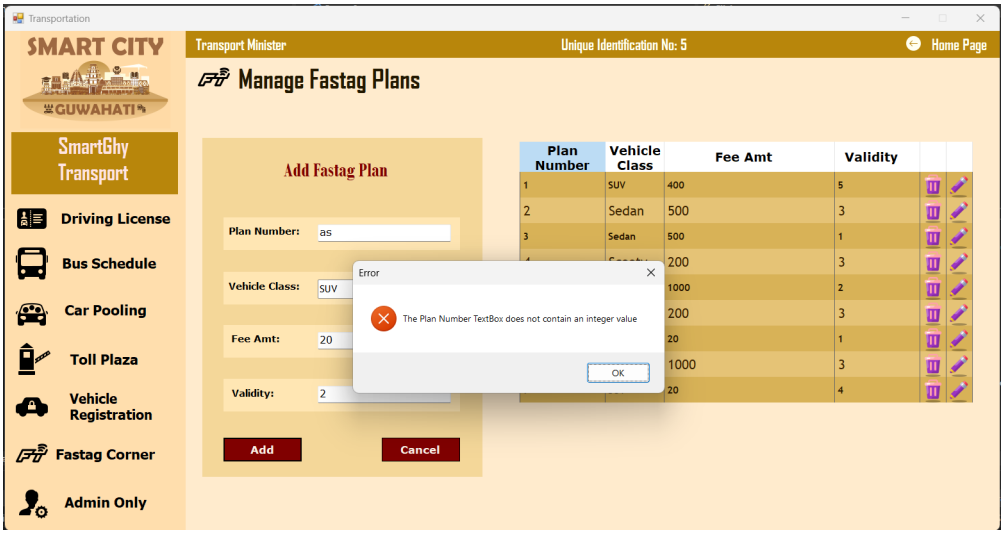

Figure 32: When an non integer value is entered in the Plan Number Textbox an error is shown to the user.

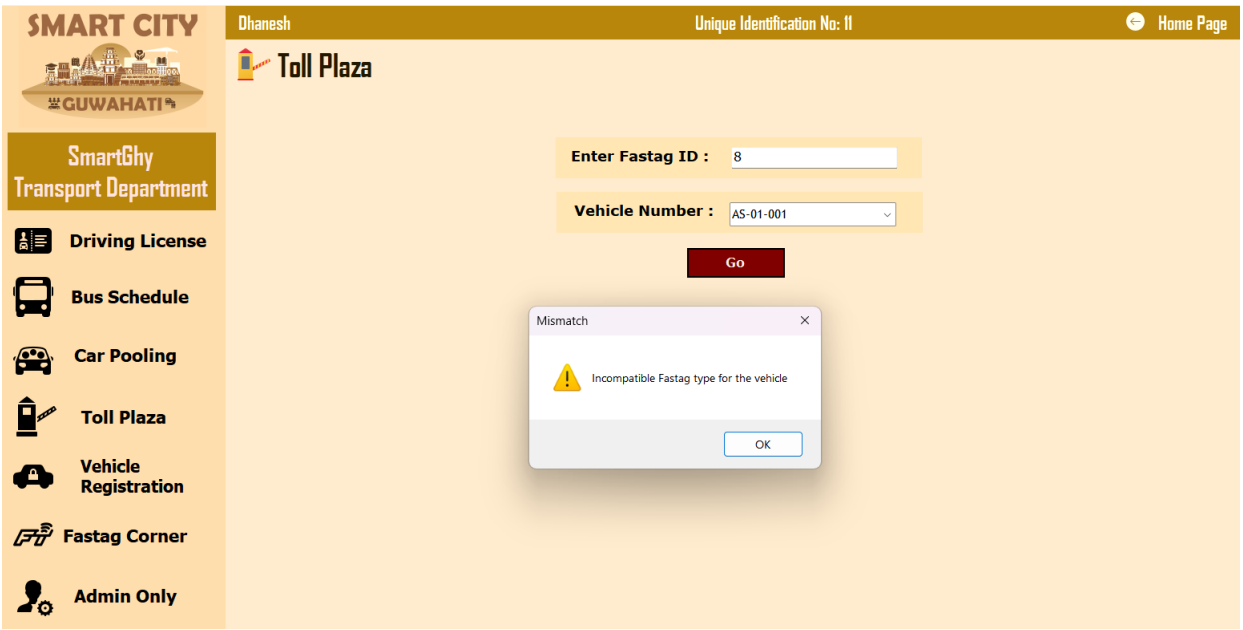

Figure 33: Error displayed when the Fastag ID chosen is incompatible to the vehicle type of the vehicle number selected.

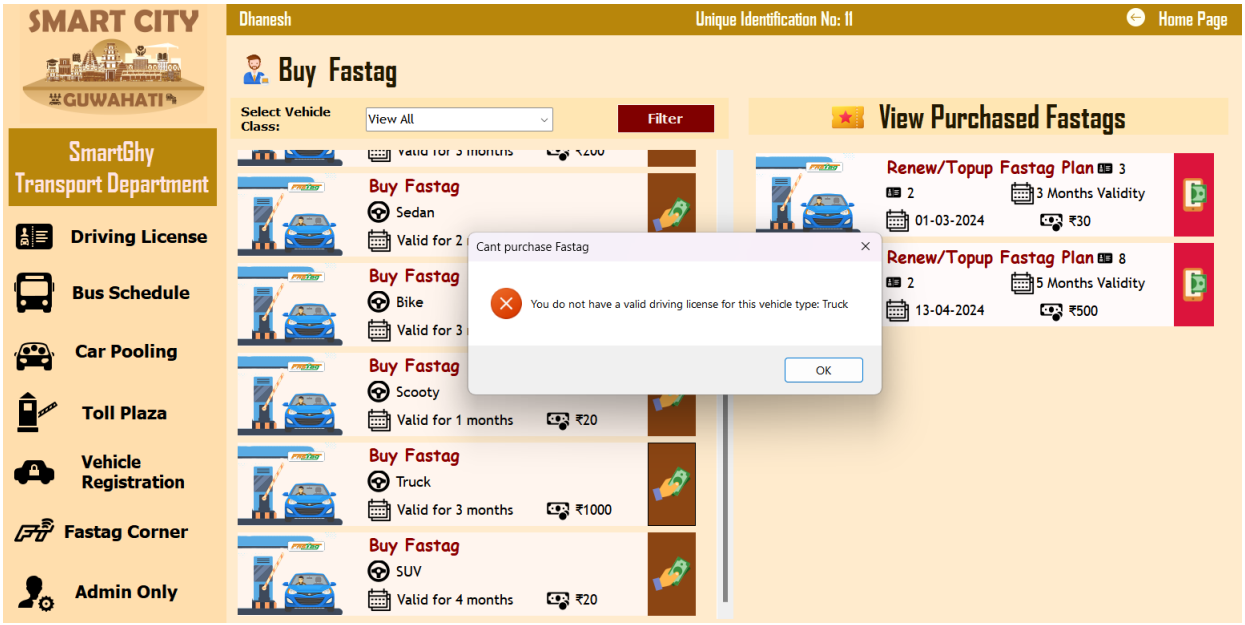

Figure 34: Error displayed when a user tries to buy a fastag that he doesn't have driving license to

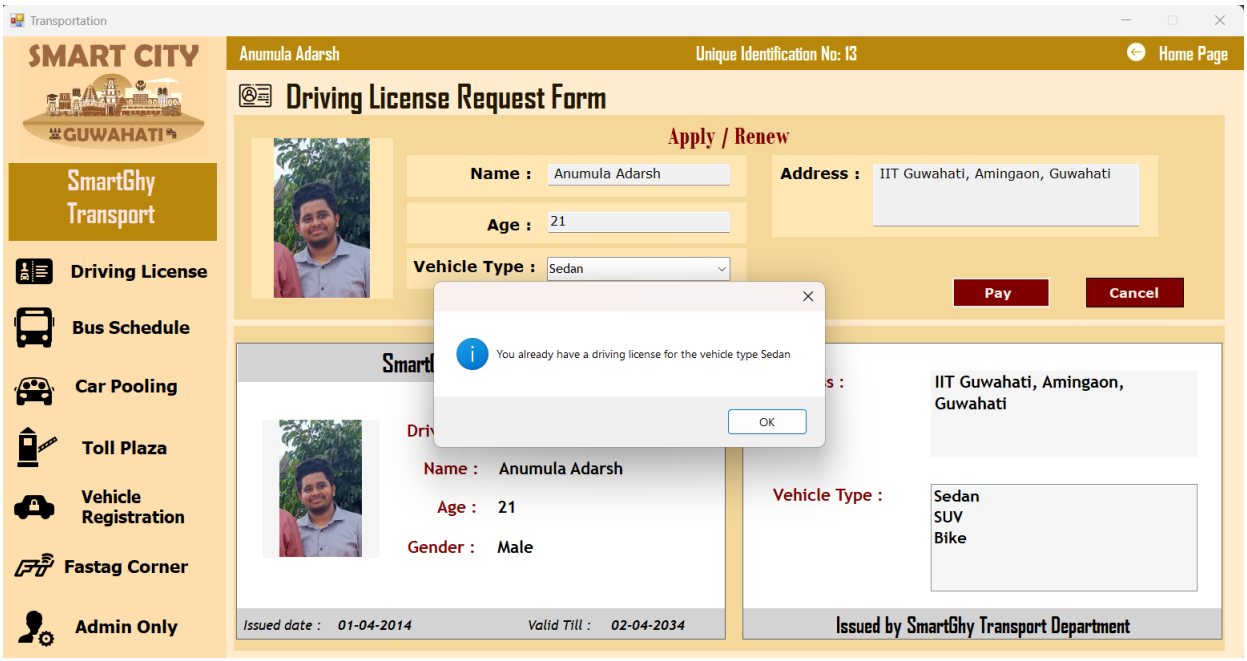

Figure 35: when a user tries to apply for a driving license for a vehicle type but he already has a license for it

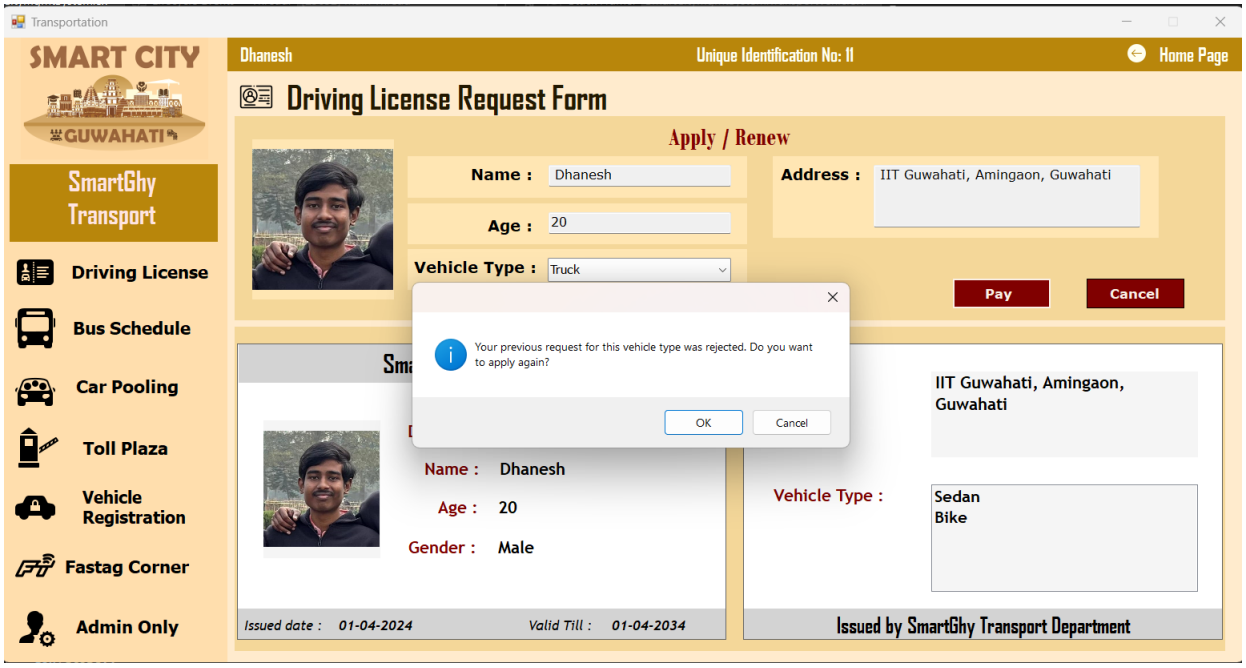

Figure 36: when a user tries to apply for driving license for a vehicle type and his previous request was rejected a message box is displayed and based on the user's response the request is processed

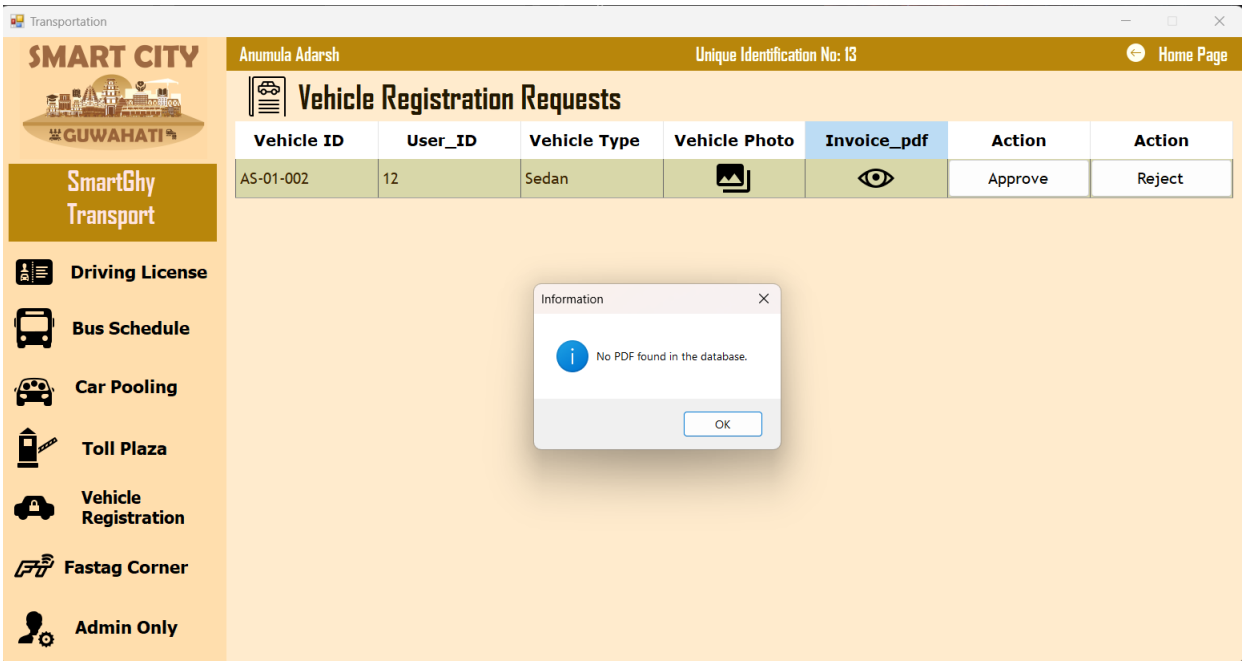

Figure 37: Error Displayed when an admin tries to access Invoice PDF but it is not in Database

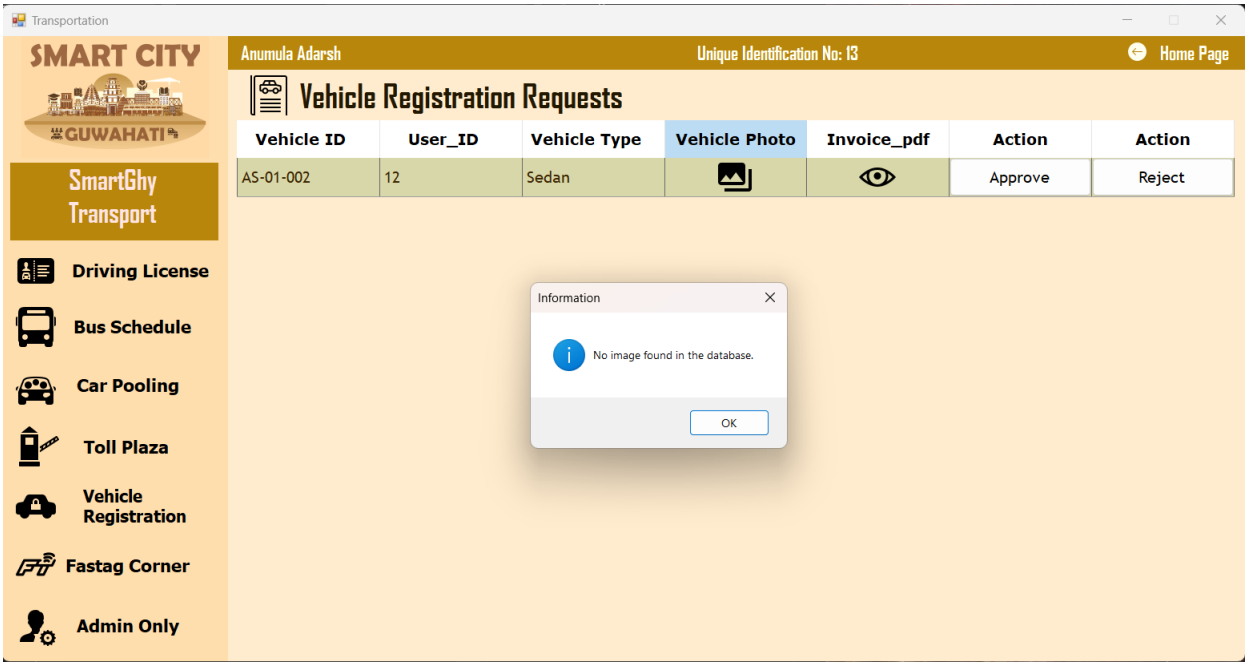

Figure 38: Error Displayed when an admin tries to access Vehicle photo but it is not in Database

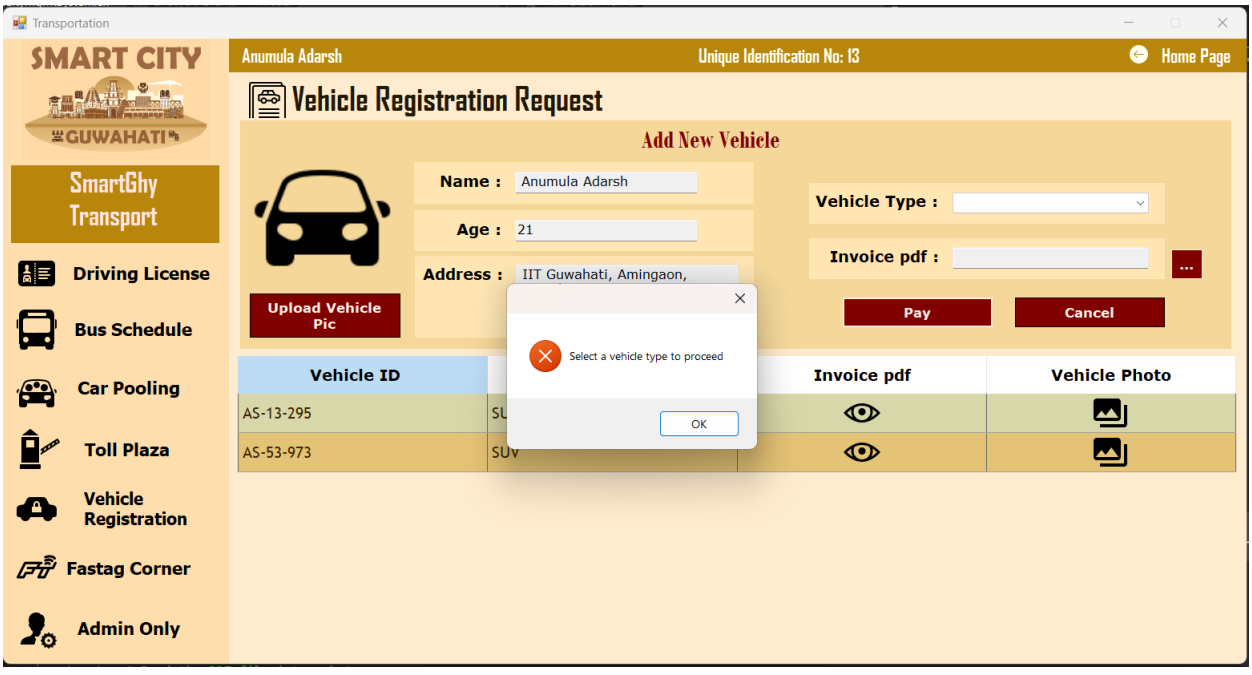

Figure 39: Error Displayed when a user did not enter all details like vehicle type, vehicle photo, invoice pdf

4.7 Festival and Event Planning Module

4.7.1 Introduction

The event management module provides a centralized platform where vendors can register and showcase their services. Customers have the convenience of browsing and booking events

with their preferred vendors, customized to their desired dates and preferences. This ensures seamless event planning and execution without any hassle. Vendors can effectively display their offerings, attracting potential customers. The module streamlines the process of event organization for both vendors and customers alike. It facilitates easy communication and coordination between vendors and customers. With the module, customers can efficiently select vendors and plan events according to their specific needs. Vendors benefit from increased visibility and accessibility to a wider customer base. Overall, the module enhances the efficiency and convenience of event planning and execution. It serves as a valuable tool for both vendors and customers in the event management process.

4.7.2 Features

1. **Customer Event Registration:** Easily register your upcoming events tailored to your preferences. Whether it's a wedding or a corporate function, explore vendors, select services, and book dates effortlessly via our user-friendly interface.

2. **Vendor Registration:** Streamline your presence to potential customers by easily registering on our platform. Provide essential details such as contact information, services offered, portfolio, and pricing to enhance visibility.

3. **Vendor Dashboard:** Upon registration, access your personalized dashboard to manage your scheduled events, bookings, and event history efficiently. Stay organized and effectively manage your commitments with ease.

4. **Customer Dashboard:** Gain convenience by logging into your account to view scheduled events and access invoices for both past and upcoming events. Track your event history and manage payments seamlessly, enhancing transparency and convenience.

5. **Invoice Generation:** The customer and vendor will be able to access the invoice for the corresponding event booking.

4.7.3 Functions

1. **Seamless Vendor Selection:** On the event registration page, customers can evaluate vendors based on their ratings, experience, and offered prices, aiding customers in making informed vendor selections that meet their desired criteria.

2. **Secure and Hassle Free Payment:** The payment process has been smoothly incorporated into the module for customer convenience, eliminating the need for redirection to third-party apps. This ensures enhanced security and reliability throughout the transaction process.

3. **Password Strength Detector:** The password strength detector allows users to assess the strength of their passwords and facilitates the creation of secure passwords.

4.7.4 Error Handling

1. **Error Handling for Event Registration Screen and Vendor Registration Screen**

   (a) **Contact Number** Contact number must be a 10 digit integer else a message prompt showing the same is displayed to the user/customer.

(b) **Password Strength Checker** Strength of the password is displayed when the password is being entered ,if password is deemed weak then a message prompting the same is displayed urging the user to reinforce his password.

(c) **Locking of username and "uid" on both registration pages to make them read only property**Thus it has been made impossible to access someone else's account by registering under a different user name from one's account for an event.

(d) Selection of Event Type has been made mandatoryFailing which an error prompt displaying the same is shown to the user.

(e) End Date must be larger than Start DateA prompt noting the same is generated.

2. **Error Handling for Event Login Screen and Vendor Login Screen**

(a) **Invalid User ID/Vendor ID and Password** User is prompted to reenter his correct user ID (vendor ID in the case of a vendor Login) and password.

(b) **Password Strength Checker** Strength of the password is displayed when the password is being entered ,if password is deemed weak then a message prompting the same is displayed urging the user to reinforce his password/

3. **Payment Gateway**

(a) **Payment Failure** The user is redirected back to his dashboard.

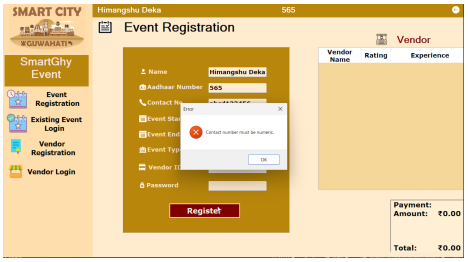

Figure 40: Popup shows that Contact No. must be numeric.

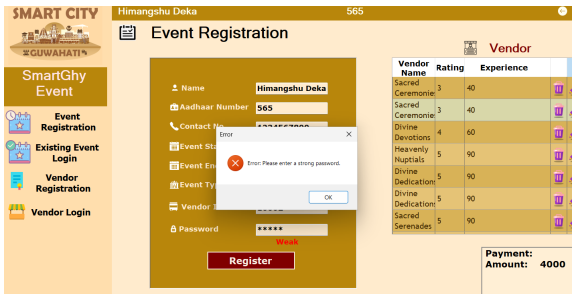

Figure 41: Popup shows that password must be strong enough.

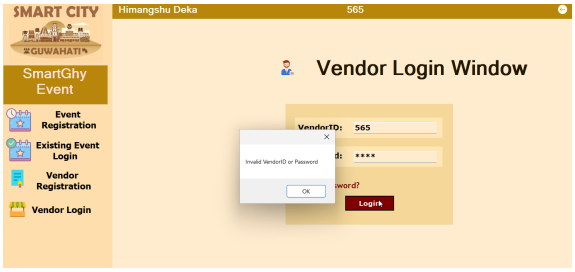

Figure 42: Popup shows Invalid userID and/or password.

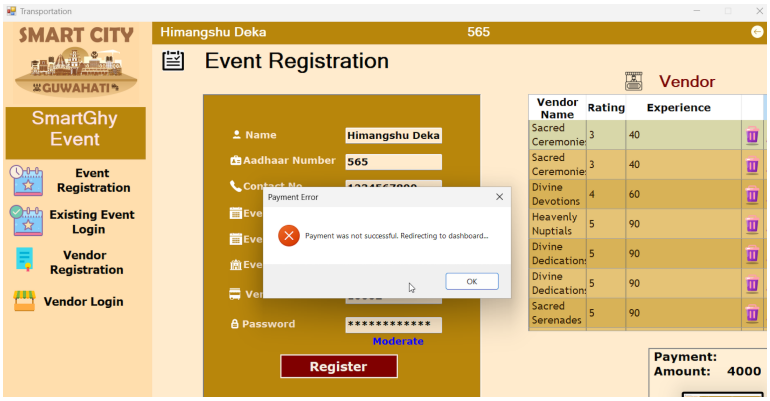

Figure 43: Payment failure redirects user back to dashboard

4.8 Complaint and Redressal Portal

4.8.1 Introduction

Welcome to the Complaint and Redressal Portal, a vital component of our commitment to building a smarter, more responsive city. Designed to address concerns swiftly and efficiently, our platform serves as a conduit for citizens to voice their grievances and seek resolution effectively. With a focus on precision and conciseness, we provide a streamlined interface for lodging complaints and tracking their progress, ensuring transparency and accountability in our governance Here, users can easily submit complaints regarding different departments like electricity and police. Department administrators promptly address these issues, ensuring efficiency and transparency

4.8.2 Features

The Complaint and Redressal Portal Portal has a variety of features which makes it a modern complaint management system. The features can be explained user-wise namely, Admin and User.

1. **Admin:**

   (a) **Dashboard:** In the Admin dashboard, admin can conveniently track the status of complaints filed by the users in that department. In dashboard, admin can find the number of complaints in progress, those resolved, and those not yet processed.

   (b) **Complaint History:** Admins now have access to a comprehensive overview of complaints filed by users, organized in a convenient table format. This table displays

the total number of complaints, categorizing them based on their current status

(c) **View Complaint:** In the Complaint History page, admins can now select a complaint to view all associated details, including the title, type of complaint, registration date, and its description. Additionally, on this page, admins have the option to take action on the complaint and view the details of the user who filed the complaint.

(d) **Take Action:** Now, admins have the capability to take action on the complaint by modifying its status, such as marking it as "in progress" or "resolved." Additionally, admins can provide remarks or comments regarding the complaint, facilitating effective communication of the resolution process.

(e) **View User Details:** Admins the see the details of user who filed the complaint.

(f) **Account Info:** The Account Settings page provides admins with the functionality to both review and edit their personal details. These details include their name, email address, user ID, physical address, pin-code, and contact number.

2. **User:**

(a) **Dashboard:** In the user dashboard, user can conveniently track the status of your complaints. In the dashboard user can find the number of complaints in progress, those resolved, and those not yet processed.

(b) **Lodge Complaint:**Users have the ability to submit new complaints through this page, providing essential details including the department (e.g., Police, Hospital), priority level (Low, Medium, High), nature of the complaint (General Query or Formal Complaint), as well as a concise title and detailed description.

(c) **Complaint History:** Users have the capability to access their historical complaints, along with the current status and remarks provided by administrators. Additionally, users can efficiently search for specific complaints and access detailed information regarding each complaint's status and progression.

(d) **Account Info:** The Account Settings page provides users with the functionality to both review and edit their personal details. These details include their name, email address, user ID, physical address, pin-code, and contact number.

4.8.3 Functions

1. **Loadcomplaintsdata():** This function retrieves data from the database tables and populates the design accordingly. It utilizes SQL queries to fetch the records and may modify certain attributes between the backend and frontend for user interaction.
In the admin interface, the function fetches complaints based on the department parameter. On the other hand, in the user interface, it retrieves complaints using the userid as a parameter.

2. **Filecomplaint():** This function facilitates the process of lodging a complaint. Upon user input of all relevant details, it automatically generates a unique complaint ID and inserts all information into a table using SQL commands, ensuring storage for future reference and processing.

3. **Takeaction():** This function empowers administrators to address complaints effectively. Administrators can add remarks and update the status of complaints directly in the

database. This enables users to track complaints easily, as the status changes are reflected in real-time.

4.8.4 Error Handling

Briefly explain how you have handled the errors in the software

1. **Exception Handling for SQL Connections:** All SQL connections are established within a try-catch block to handle exceptions gracefully, ensuring that any errors encountered during database operations are handled appropriately without disrupting the application flow.

4.9 Administrative Hierarchy Interface

4.9.1 Introduction

Administrative Hierarchy is responsible for the administration part of the Smart City which is the responsibility of the admins of each module/portal. The admin of this module is the Election Commissioner, who is assumed to be appointed by the Government. Thus, this module is responsible for creating awareness among the citizens about the ministries and the ministers and the rights they have as citizens to access information. It also conducts free and fair elections and shows the organizational structure of the Smart City.

4.9.2 Features

From Citizen side
The module offers many features to ease the process of election, gaining access to information and understanding the organizational structure of the Smart City.

1. **Register as Voter:** Every citizen above the age of 18 can register himself/herself as a voter. Basic details are to be filled up and he/she will be given the chance to vote in the upcoming election.

2. **Nominate Yourself:** Any citizen can nominate themselves for any of the positions.Their nomination will be reviewed by the admin and upon acceptance, they will be able to participate in the election.

3. **Know Your Candidate:** Citizens should be aware of the candidates contesting for each position. They can find the details of all the candidates in this and can decide whom to vote for.

4. **Timeline:** Election is a very busy process. There are lots of deadlines and important dates involved. Citizen can find out about all the dates here and can keep themselves informed about the proceedings.

5. **Cast Your Vote:** Citizens can vote for each position and submit their ballot.

6. **View Results:** This feature shows the results of the recent elections.

7. **View Statistics:** In this feature, the citizen can view statistics of each election like the total turnout, turnout per ministry, gender ratio of the voters etc.

8. **Past Elections:** Citizens can view the details of the past elections.
9. **Report Violations:** Candidates who have violated the Code of Conduct can be.
10. **Organization Structure:** Citizens can view all the ministries and ministers at one place.
11. **Right To Information:** Citizens can submit their appeals to any ministry to gain access to certain information.

From Admin side

1. The admin can conduct elections and all the important activities related to it can be easily set up with the help of features offered. These features include **announcing the timeline of elections, updating code of conduct, accepting/rejecting the nominations filed, reviewing the code of conduct violations, conducting the election and releasing the list of elected candidates.**
2. After the announcement of results, the organizational structure is automatically updated with the details of the newly elected ministers.
3. Apart from elections, the admin can also send the appeals by citizen to gain access to certain information about the city to the concerned departments.

4.9.3 Functions

1. **Organizational Structure**
   (a) **LoadData():** It loads the data of from the "ministries" table in the database and presents it in the datagrid in the Organizational Structure.
   (b) **GetSolutionDirectory():** Gets the Solution Directory as a string to retrieve the logos or images of the ministries which are shown in the Organizational Structure.
2. **Election**
   (a) **AreDatesIncreasing():** Takes an array of strings of the format yyyy-mm-dd and returns true if the dates are in chronological order and false otherwise.
   (b) **AreDatesEqual():** Takes an array of strings of size two and returns true if the dates are equal and false otherwise.
   (c) **CompareUserInfo():** Takes name, uid and age as input from the user and compares with the logged in user's data and returns true if the data matches and false if it doesn't.
   (d) **LoadDates():** Loads the recent election's dates from the database into the datagrid in the Timeline of Election for the citizens to view.
   (e) **GetCoC():** Loads the Code of Conduct from the database for the current election and prints it in the RichTextBox for the citizens to view.
   (f) **helpOptionsAdd():** Iterates through all ministries and calls **OptionsAdd()** to add options to the comboboxes in the voting portal.

(g) **OptionsAdd():** Given an election ID and ministry ID, it gets the details of the approved candidates and fills the combobox corresponding to the ministry with those details.

(h) **populateComboBox():** Given an election ID, it checks if the results of that election have been announced and if yes, it adds the election ID to the combobox for the citizens to view the resulst of this election in Past Elections.

(i) **isItPastElection():** Checks if the there is no election going on.

(j) **PreviousCoC():** Gets the latest edit of the Code of Conduct and shows it as a preview for the Admin to edit it.

4.9.4 Error Handling

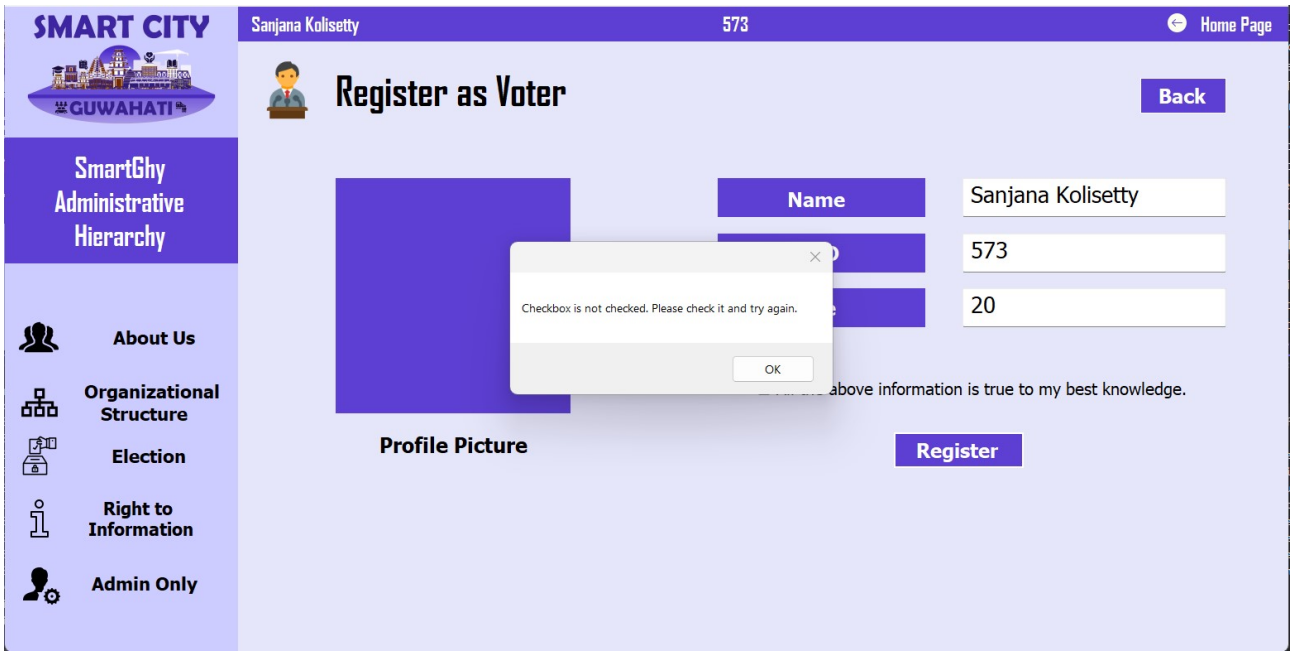

Figure 44: Registering without ticking the checkbox

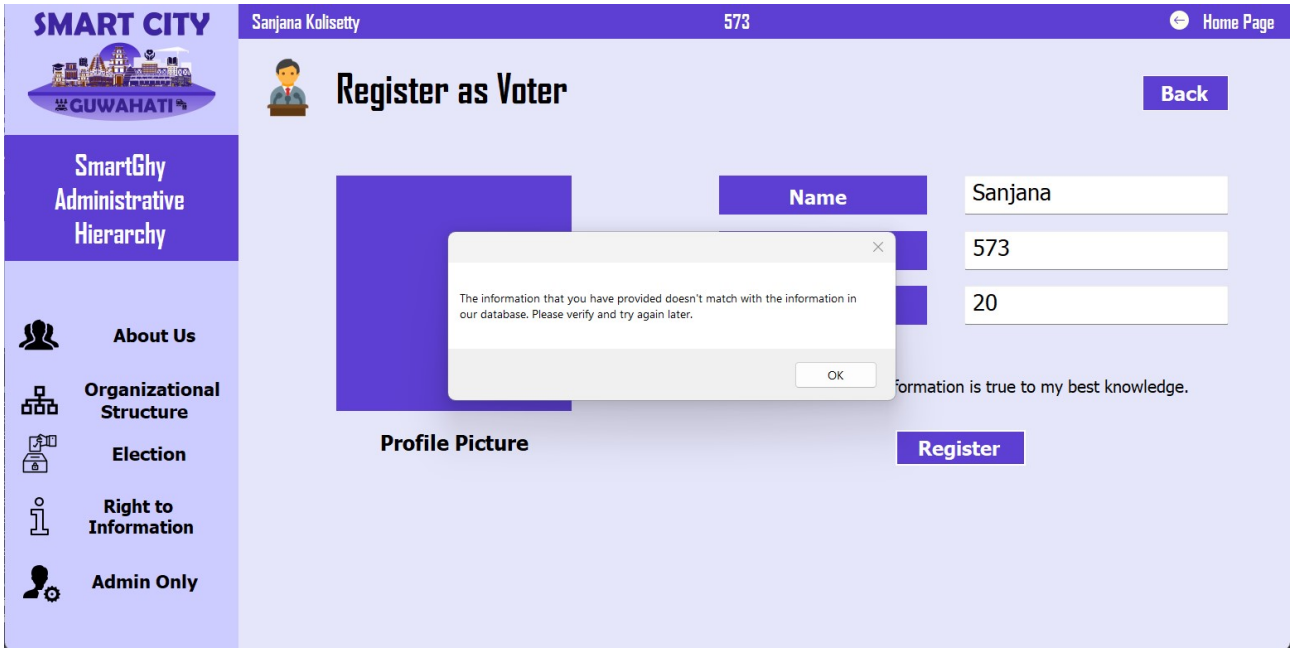

Figure 45: Incorrect information

Figure 46: When it's past the nomination date, nominations aren't accepted

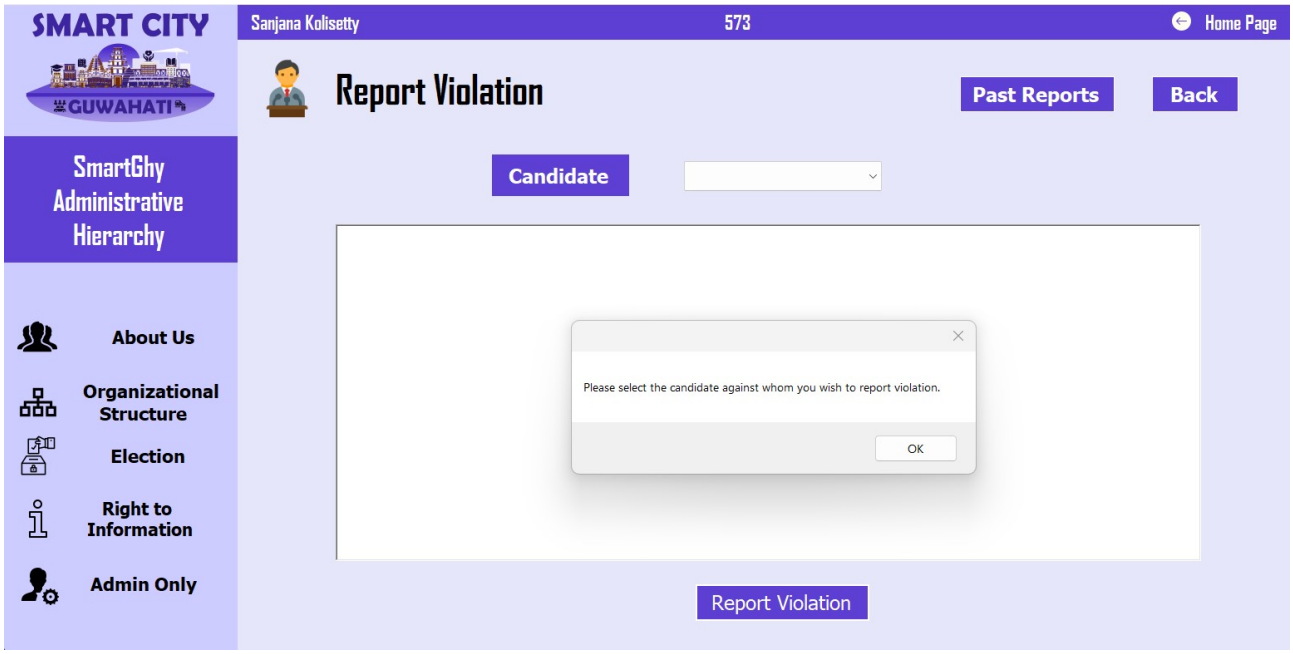

Figure 47: Candidate has to be selected for reporting violation

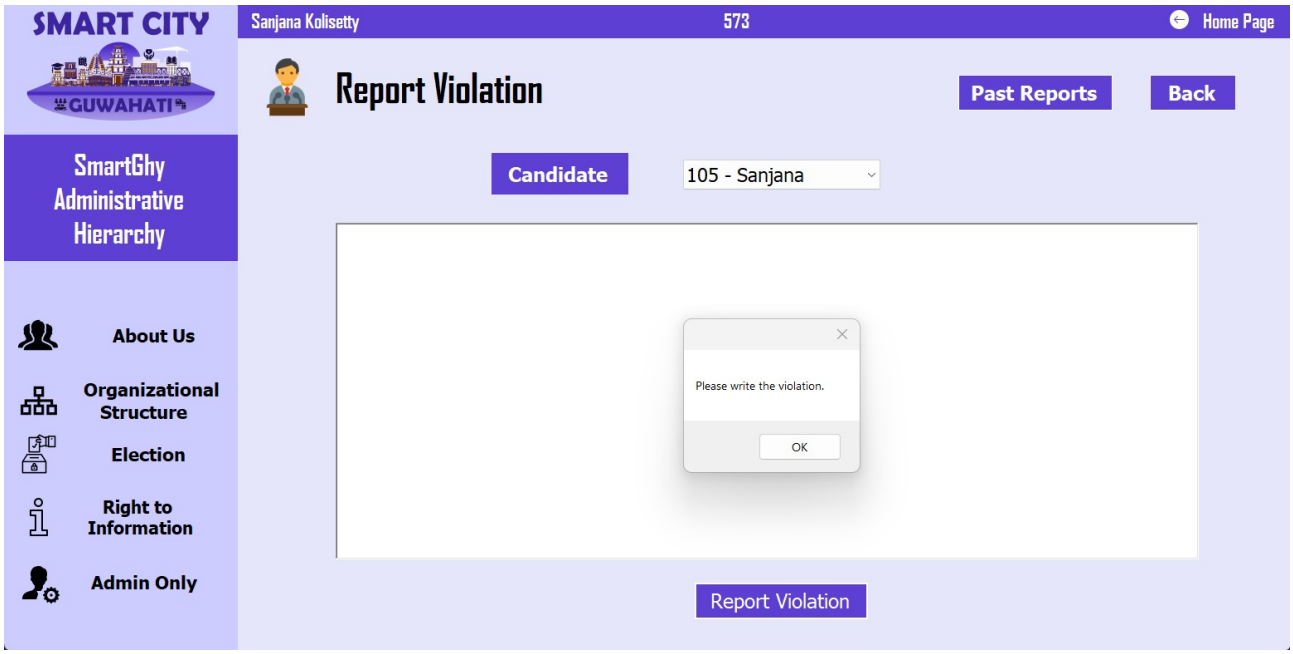

Figure 48: Empty violation report

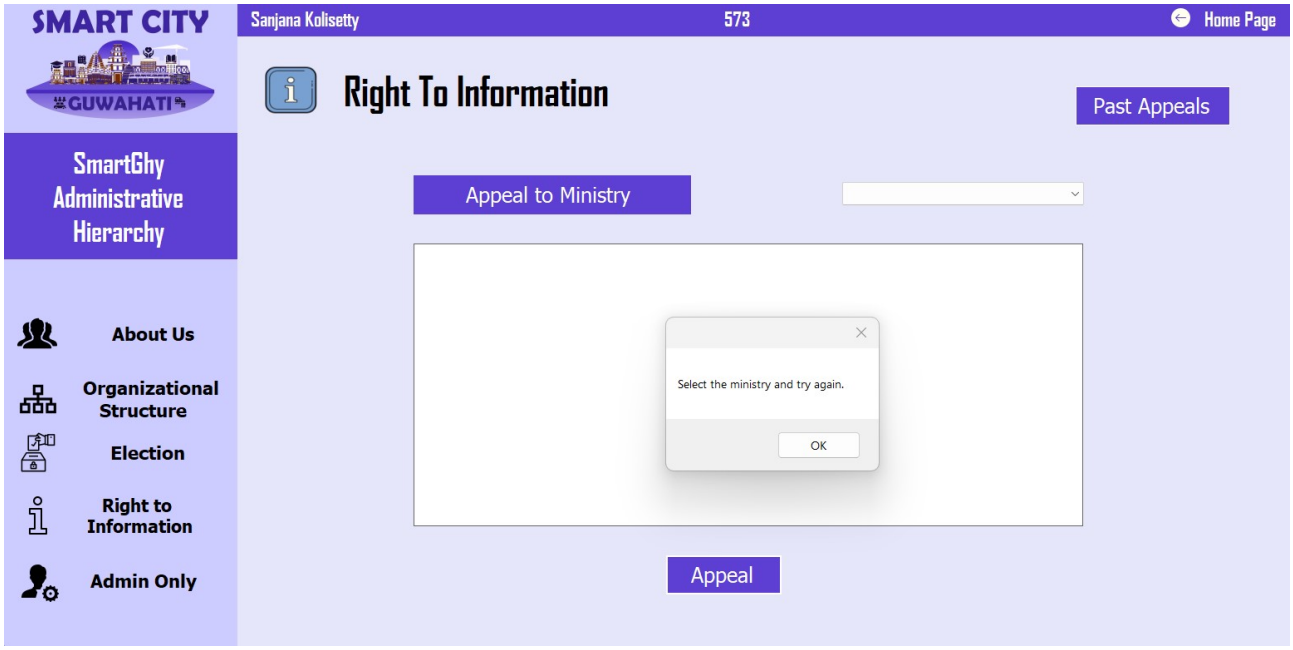

Figure 49: Select ministry to submit RTI appeal

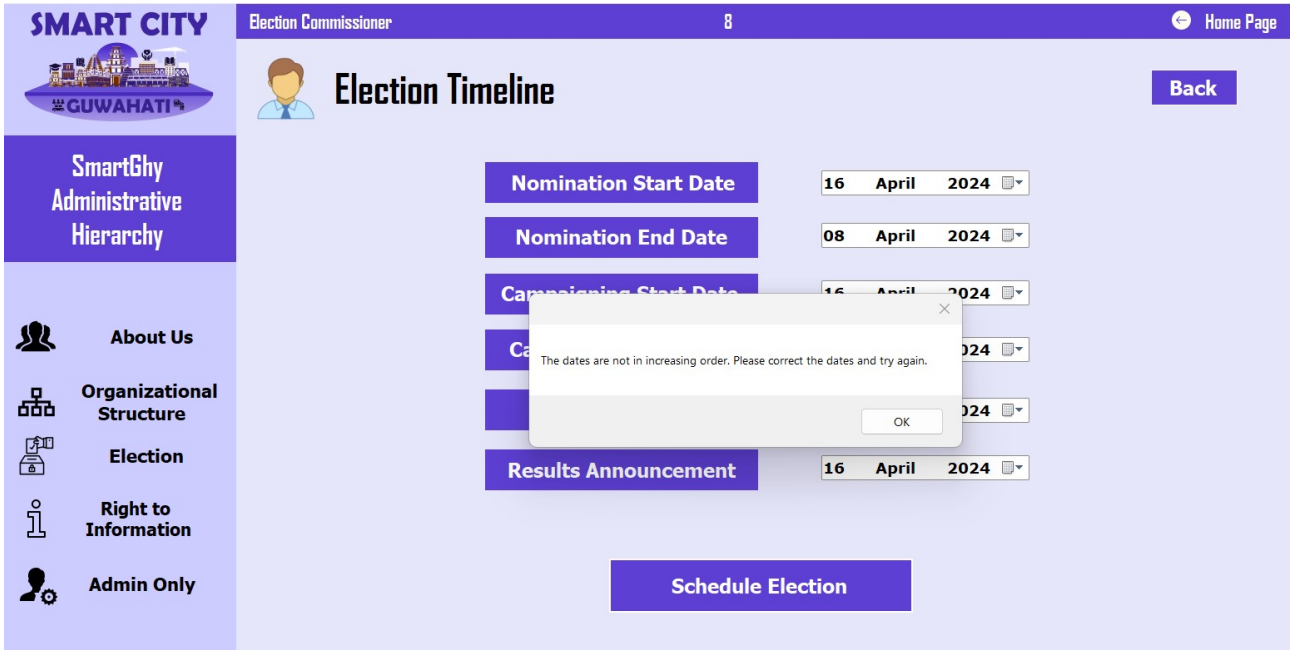

Figure 50: Chronological order needs to be maintained

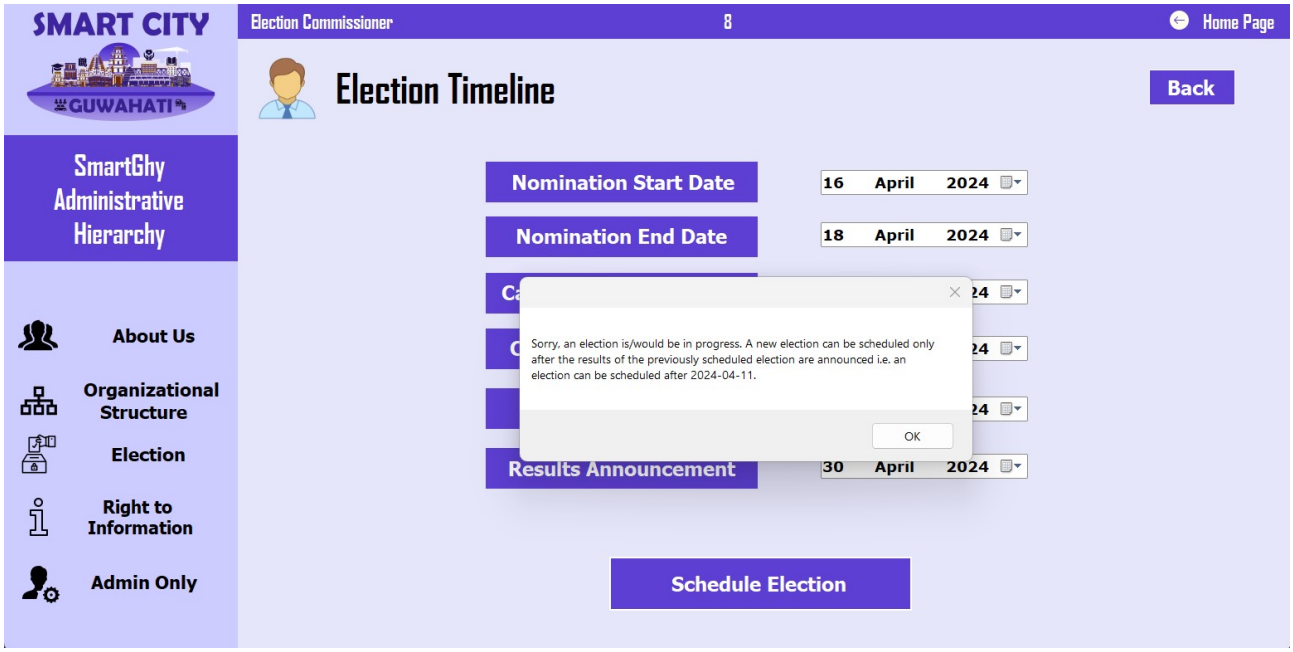

Figure 51: An election can't be scheduled till the results of the ongoing election are released

4.10 Library

4.10.1 Introduction

Welcome to the central online portal for Guwahati's library system, where knowledge meets convenience at your fingertips. Administered with precision and care, our platform offers a diverse range of digital resources, from e-books and academic journals to multimedia collections, catering to the intellectual pursuits of students, researchers, and enthusiasts alike. Join us in unlocking the wealth of information and fostering a culture of learning in the vibrant city of Guwahati.

4.10.2 Features

The Central Library Portal has a variety of features which makes it a modern library manage ment system. The features can be explained user-wise namely, Admin and User

1. **Admin:**
   (a) **Dashboard:** In the dashboard, We get an overview of key statistics related to the library system such as Total Books in the library, Borrowed Books, Overdue Books, fines due and fines collected.
   (b) **Requested Books:** This screen Shows all the requested Books that are requested by users to get issued by the admin and out of these admin can check the status of whether it was issued or not before.
   (c) **Book Management:** Admin can add a book Adds a new book to the library database using the book details provided. Admin can update a book that modifies the details of an existing book with a specified book ID in the database. Admin can remove a book with a specified book ID from the library database.

(d) **Manual Transaction:** In this window: Admin can issue a book with a specified book ID to a user with a specified user ID. Admin can return a book with a specified book ID from a user with a specified user ID to the library. Admin can renew a book with a specified book ID to a user with the specified ID. Admin can pay a fine of a user. It will be cut from the user's balance. Admin can add money to the balance of a user

2. **User:**

(a) **Overdue Books:** This feature allows users to view a list of books that they have borrowed but have not returned within the specified timeframe. It serves as a re minder for users to return borrowed items promptly to avoid late fees or penalties.

(b) **Search Books:** Users can utilize this feature to search for specific books within the library's collection. With a robust search functionality, users can find books by title, author, genre, or keywords, streamlining the process of locating desired reading material

(c) **E-Books:** Access to a digital library of e-books provides users with the convenience of reading electronic versions of books on various devices. This feature expands access to literature and educational resources beyond the physical constraints of the library, offering flexibility and convenience to users.

(d) **Borrowed Books:** Users can easily track the books they have borrowed through this feature. It provides information on the titles, due dates, and status of borrowed items, empowering users to manage their borrowing activities efficiently and ensuring timely returns.

(e) **Requested Books:** Users can request books that are currently unavailable or in high demand through this feature. By submitting a request, users express their interest in specific titles, allowing the library to prioritize acquisitions and fulfil user demands, enhancing the overall user experience.

4.10.3 Functions

1. **LoadrequestBooks():** This function fetches the data from the database(tables) and feed it to the design. The data/ Records are fetched using sql Query. Also this function modifies some attributes between backend and frontend to user.

2. **PopulateTable():** This PopulateTable function is designed to populate a table with data about requested books. It checks if there are any books to display. If so, it iterates through each book, creating and adding labels for the book's status, author, title, and return date to specific cells within the table. The function also sets the alignment and appearance of these labels based on the book's status (e.g., color-coding status labels). Finally, it adds an empty label at the end to ensure proper layout.

3. **show()/ Routing:** This function is basically used for routing between pages. It is used with forms to display them to the user. When we call .Show() on a form object, it makes that form visible on the screen, allowing the user to interact with it.

4.10.4 Error Handling

Briefly explain how you have handled the errors in the software

1. **Exception Handling for SQL Connections:** All SQL connections are established within a try-catch block to handle exceptions gracefully, ensuring that any errors encountered during database operations are handled appropriately without disrupting the application flow.

2. **Book Issuance:** Ensure that the book being issued is available in the library's inventory. If the book is not available, inform the admin with a message like "The requested book is currently not available for issuance". After issuing the book, update the user's account with the book details, including book ID, issue date, and return due date.

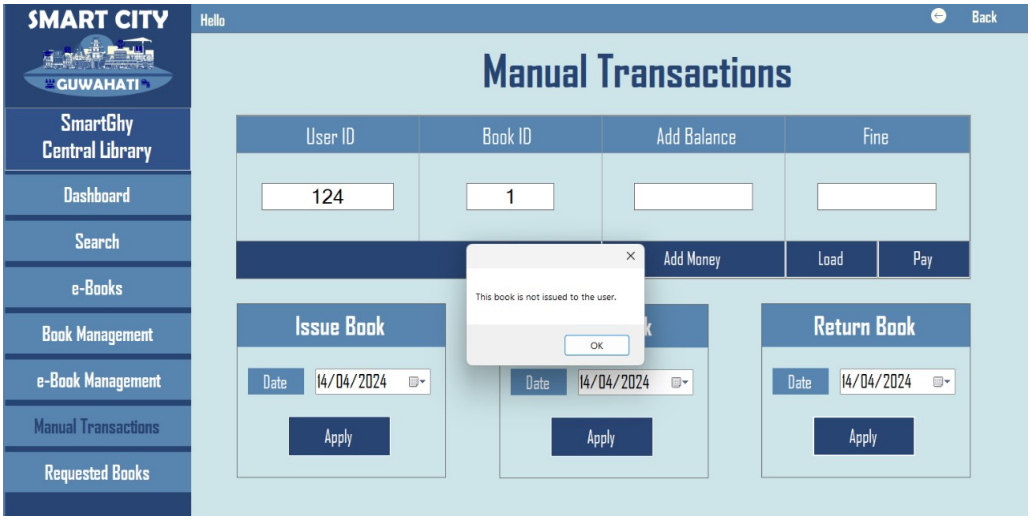

Figure 52: Popup shows whether Book Issued successfully or not.

3. **Book Return:** When a user returns a book, authenticate the user to ensure it's the correct user returning the book. Update the book's status in the library inventory to mark it as returned.

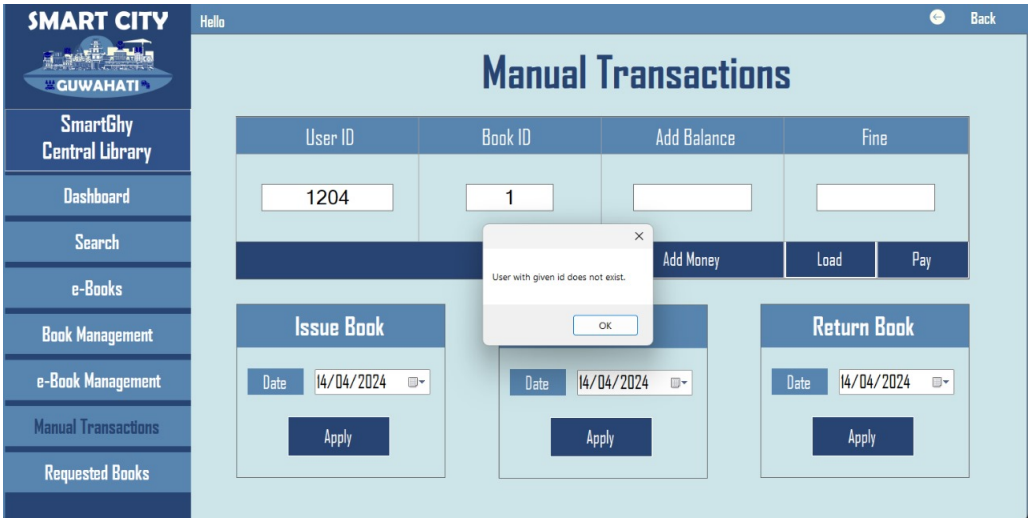

Figure 53: Only permitted users are allowed for Book issuance.

4. **Book Renewal:** Check if the book is eligible for renewal based on library policies (e.g., no overdue fines, maximum renewal limit not reached). If the book is not eligible for renewal, inform the user with a message like "This book cannot be renewed. Please return it to the library."

5. **Memory Management (BC2004):** The application monitors memory usage and handles the BC2004 error, indicating an out-of-memory situation. It accomplishes this by tracking the forms that are opened and closing them when they are no longer needed, effectively managing memory usage and preventing memory-related errors.

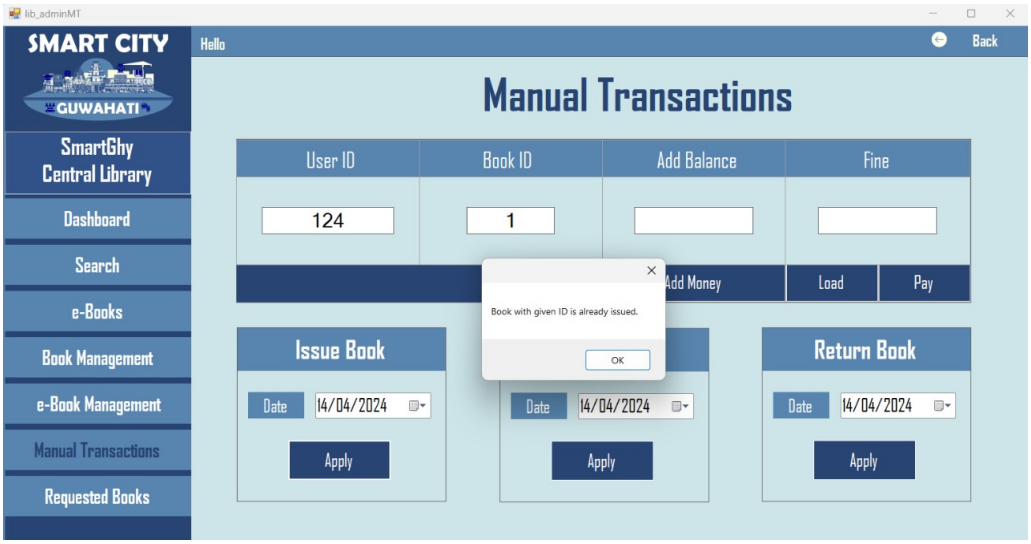

Figure 54: Popup shows whether the given Book with Book ID is available or not.

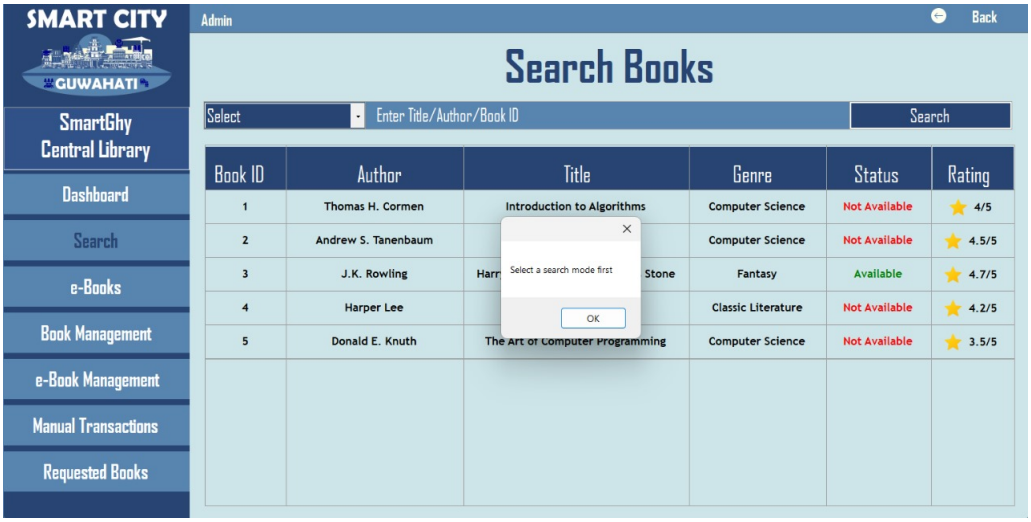

Figure 55: Popup shows whether Search Mode is right or not.

4.11 Banking

4.11.1 Introduction

The Banking module within the Smart City Management System offers users a convenient and secure platform to manage their financial activities. From creating a new bank account to accessing account information and performing transactions, this module ensures seamless banking services for users within the smart city ecosystem.

4.11.2 Features

1. **Create Account**: Allows users to set up a new account within the banking system, providing necessary personal and account information to initiate the process.
2. **View Account Profile**: Enables users to access and review their account profile details, including personal information and preferences.
3. **Bank Account Dashboard**: Provides users with an overview of their bank account, including summary information such as account balance, recent transactions, and account status.
4. **Check Balance**: Allows users to quickly check the balance of their bank account to stay informed about their available funds.
5. **Cash Withdraw and Credit**: Enables users to withdraw cash from their bank account, as well as the ability to deposit funds into their account.
6. **View Transactions**: Provides users with access to their transaction history, allowing them to review past transactions and monitor their spending patterns.
7. **Make Payments**: Allows users to initiate payments to merchants or individuals, including transfers to other bank accounts, bill payments, and peer-to-peer transactions.
8. **Take a Loan**: Offers users the option to apply for and receive loans based on their eligibility criteria, providing necessary funds for personal or business purposes.
9. **Pay Bills**: Facilitates the payment of various bills, including house hold bills, subscriptions, and other services, directly from the user's bank account.
10. **Admin Transaction Management:** In addition to the above features, the Finance minister has the authority to access and oversee the list of all user transactions. The minister can also filter transactions based on either an account ID or a specified transaction amount threshold. Such capabilities give the administrator greater control over monitoring and managing financial activities within the system.

4.11.3 Functions

1. **ProcessPayment():**The `ProcessPayment`() function is designed to facilitate secure and efficient payment transactions within the system. Inputs required for the function include the receiver account ID, transaction amount, a message, and the user's bank account password. Upon invocation, the function conducts a series of checks to validate the provided information: it verifies the validity of the user ID, ensures that a valid amount

is entered, confirms that the user's account balance is sufficient for the transaction, and validates the user's bank account password using data from the `accounts` table. Following successful verification, the function proceeds to execute the transaction by adding a new entry to the `transactions` table and updating the account balance accordingly. For scenarios where the receiver account ID, transaction amount, and purpose of payment remain constant, the readonly_prop property can be set to true, enabling the fixation of textbox values to streamline the payment process.

2. **LoanPage():** Users can apply for loans through the ApplyForLoan form, where eligibility is determined based on age and occupation. Upon applying, loan details such as account number, start date, status, and installment amount are stored in the database. Users can view their loan details, including the remaining months, installment amount, and last payment month, on the form. The system allows users to make monthly payments through the PaymentGateway form. Upon successful payment, the system updates the last payment month and the amount paid in the database. Additionally, if a user fails to pay the loan bill for a month, their status changes to "defaulters", indicating non-compliance. Conversely, when the loan is fully paid, the status returns to null. This system provides users with a streamlined process for managing their loans while ensuring timely payments and status updates.

3. **Paybills():** Bill payment module named PayBills, designed to manage various household bills. Upon loading, the form checks the last payment month for each bill type, such as electricity, internet, water, gas, and TV. It compares these with the current month to determine payment status and updates the buttons accordingly. Users can initiate bill payments by clicking on the respective buttons, which open a payment gateway form. Upon successful payment, the current month is stored as the last payment month for the corresponding bill type in the database. Conversely, if payment fails, an error message is displayed. The system employs MySQL database integration for storing and retrieving bill payment data, ensuring accurate tracking and management of household expenses.

4.11.4 Error Handling

Password strength checking: The password strength checking feature ensures that users create secure passwords by evaluating their complexity, including factors like length, character variety, and special characters.

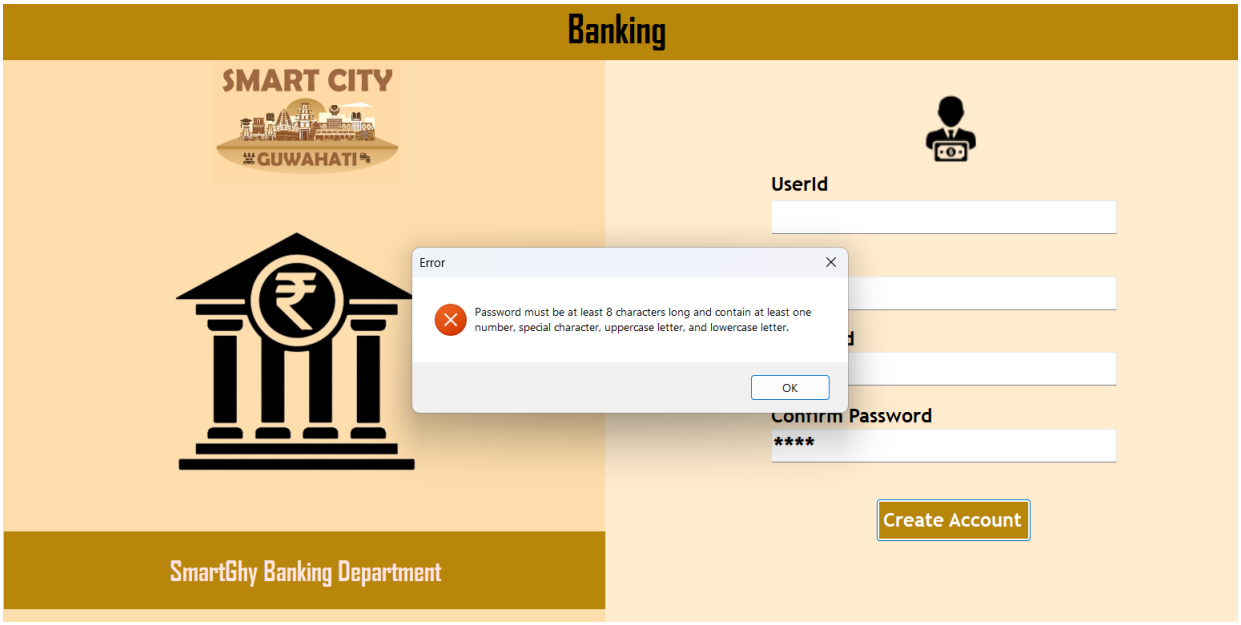

Figure 56: Password strength checking

Forget password functionality: For the forget password functionality, users need to provide their unique user ID as a security measure to verify their identity before initiating the password reset process.

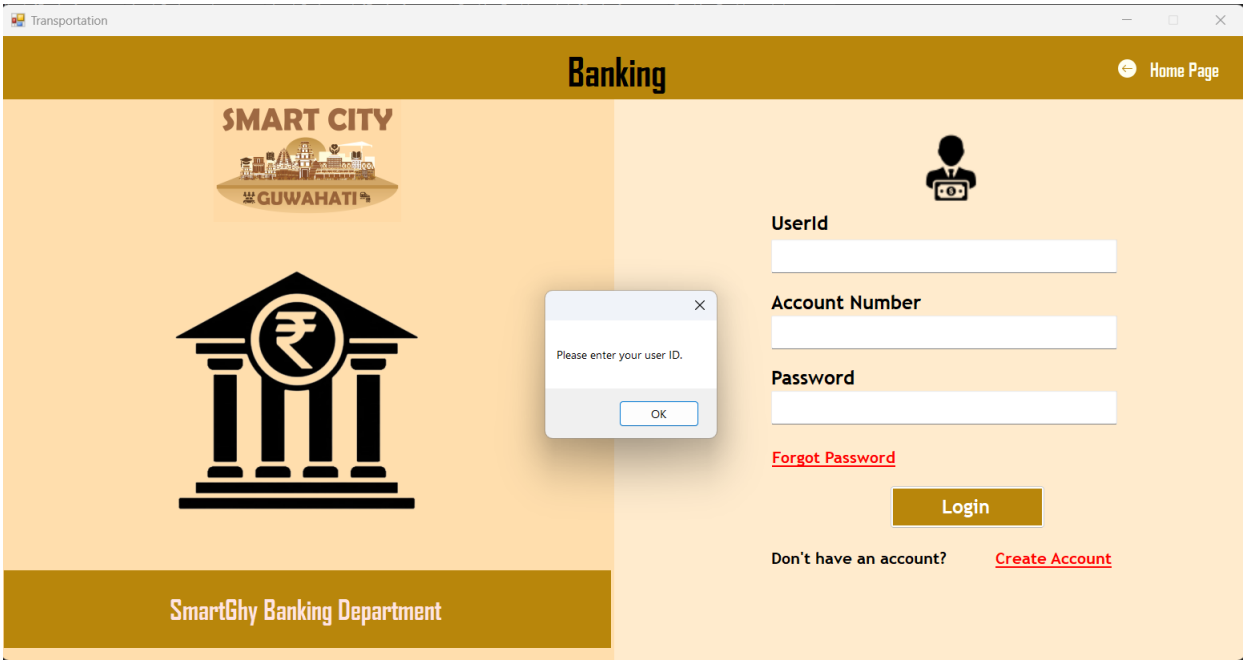

Figure 57: Forget password

Wrong Login Information: In case of login credential mismatch, the system provides an error message to alert the user about the incorrect input, prompting them to re-enter the correct username and password combination.

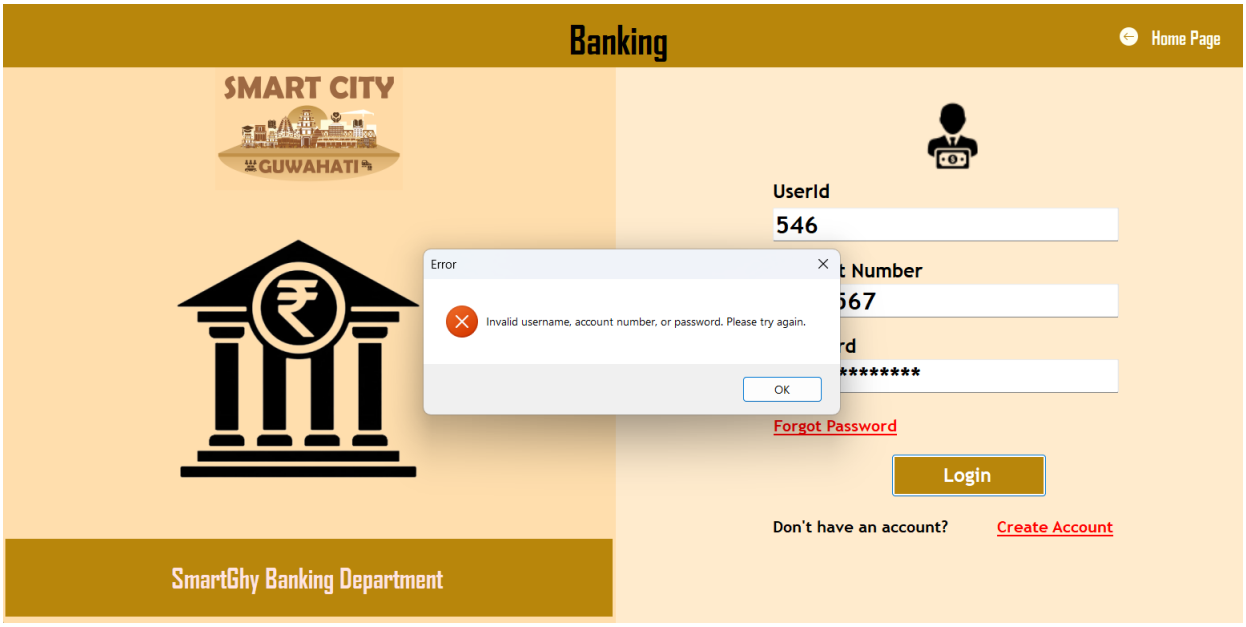

Figure 58: Incorrect information warning

Pay bill in a month only once: Preventing users from paying bills multiple times in the same month enhances system integrity and ensures accurate payment records, reducing the risk of duplicate payments and potential errors.

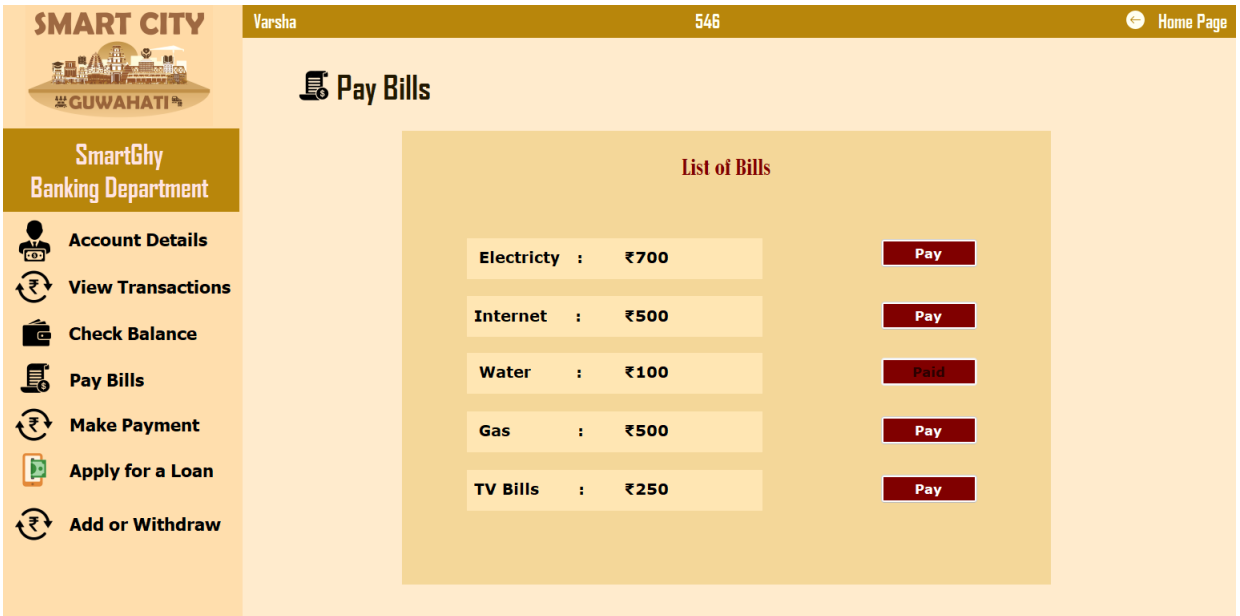

Figure 59: Pay bill check

Loan bill in a month only once: Similar to bill payments, the system restricts users from paying their loan bill multiple times within the same month, maintaining consistency and preventing financial discrepancies.

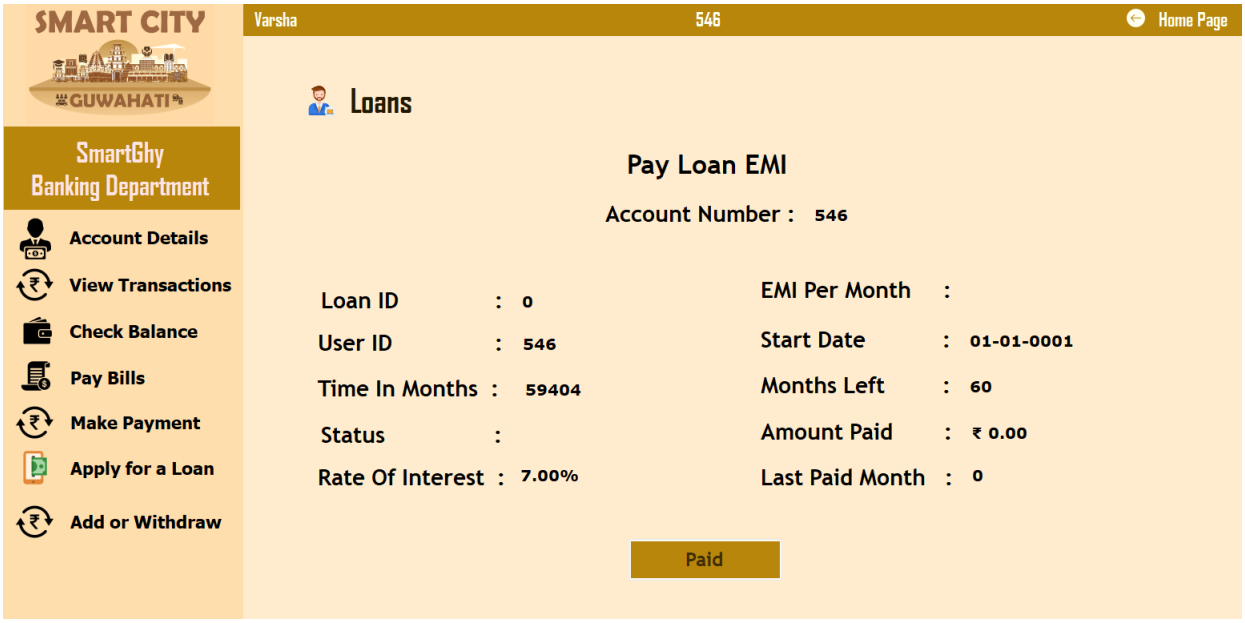

Figure 60: Loan payment check

Loan eligibility: Loan eligibility is determined by specific criteria, including an age requirement of at least 18 years and an occupation status that is not designated as "student" or left unspecified (null). This ensures that only financially stable individuals are considered for loans, minimizing default risks.

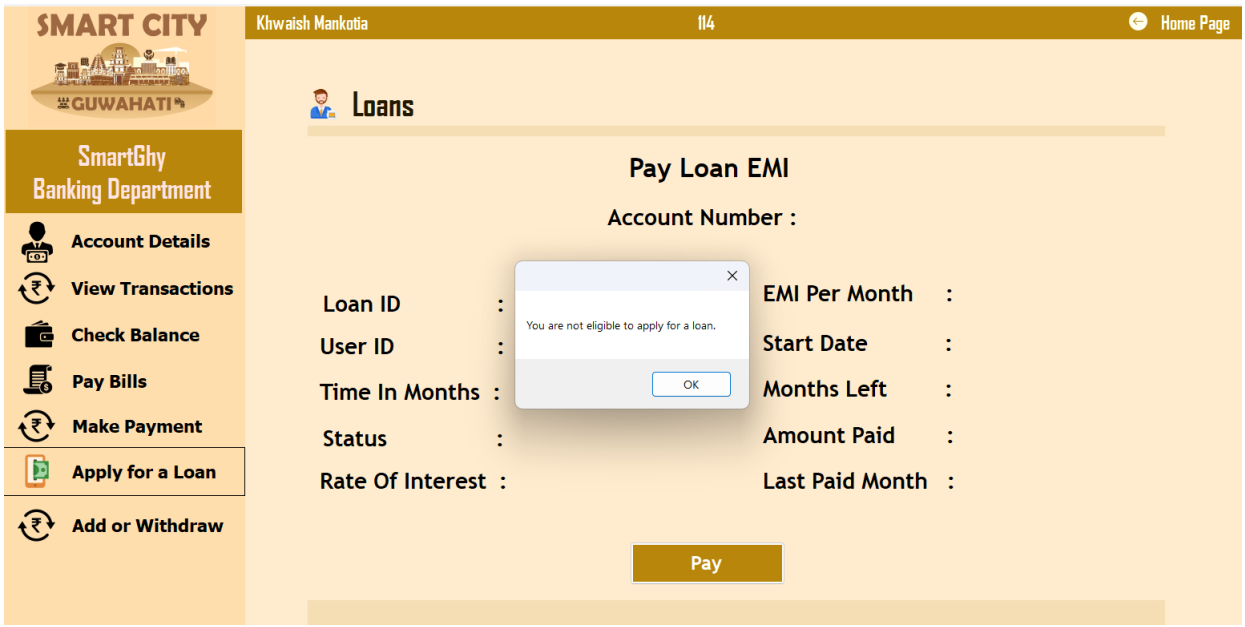

Figure 61: Loan eligibility check

Error handling for SQL Query: In the code, each SQL query function, namely IsUserPresentInLoansTable, IsEligibleForLoan, GetLastPaymentMonth, and StoreLastPaymentMonth, is encapsulated within a try-catch block, ensuring robust error handling. These blocks effectively manage potential exceptions that may arise during SQL query execution, such as database connectivity issues or query failures. By incorporating error management at the database in-

teraction level, the code enhances reliability and maintains data integrity, contributing to a more stable and resilient system architecture.

5 Conclusion

The Smart City Management Platform consists of several modules that put together creates a software which can be of great help to the citizens in their daily life. Implementing these features and integrating all the modules is going to be a hard task but definitely possible. We aim to create a software that is efficient and easy to use.