**Automated Daily Update Posting Application**

**1. Introduction**

This application automates the process of posting daily updates to a web application. It retrieves the daily message (task name, hours worked, and any additional notes) from a **Google Sheet**, formats the message, and posts it to a specified web application using **Selenium WebDriver**. The system is scheduled to run automatically every day, fetching the latest data from the Google Sheet and posting the message.

**2. Functional Requirements**

* **Login to Web Application from .net Console Background Application**
  + Launch the browser using Selenium.
  + Login to the web application using valid credentials stored securely.
* **Read Task Data from Api With Cookies**
  + The application will read data from a Google Sheet.
  + The Google Sheet will have **one row**, containing 3 columns:
    1. **Task Name**
    2. **Hours Worked**
    3. **Notes**
  + The latest row of data will always be read and used for posting the daily message.
* **Post the Message**
  + The task data will be formatted into a message string.
  + The application will navigate to the required page in the web application.
  + It will input the formatted message into the required field and click the submit button.
* **Take a Screenshot**
  + A screenshot of the confirmation page will be captured after the post is successful.
* **Send Email Notification**
  + The screenshot will be attached to an email.
  + The email will be sent to the configured email address with a confirmation message.
* **Automatic Scheduling**
  + The application will be scheduled to run daily using Windows Task Scheduler.
  + It will execute automatically without manual intervention.

**3. Non-Functional Requirements**

* **Security**
  + Store sensitive credentials (e.g., username, password) securely in an encrypted configuration file or environment variables.
* **Performance**
  + The application should execute within 2 minutes for each run, including logging in, posting the message, and sending the email.
* **Reliability**
  + The application should retry the post in case of failure (e.g., network error, site not responding).
  + Log errors and email them for quick resolution.
* **Scalability**
  + The architecture should allow easy updates to the logic without major rework.

**4. Technology Stack**

* **.NET Core**: The primary framework used to build the console application.
* **Selenium WebDriver**: Used for automating the browser interactions (login, post message, etc.).
* **Google Sheets API**: Fetches data from a Google Sheet to retrieve the daily message.
* **SMTP (Simple Mail Transfer Protocol)**: Sends an email with the screenshot.
* **Windows Task Scheduler**: Schedules the application to run automatically at a specified time daily.

**6. Development Steps**

**Step 1: Create Google Sheet and Configure API**

1. Create a Google Sheet with a structure like this:

|  |  |  |
| --- | --- | --- |
| Project | Hour | TaskName |
| Fe4-Enricher | 6 | Unit test Cases for Event Name Mapping |
| NTC | 2 | Room Assignment Analysis |

1. Share the Google Sheet with the service account email.

**Step 2: Build the .NET Console Background Application**

1. **Create Console Application**:
   * Create a new .NET Console Background Application using Visual Studio or the CLI.
2. **Install Required Packages**:
   * Install **Selenium WebDriver**, **Google Sheets API**, and **SMTP**:

dotnet add package Selenium.WebDriver

dotnet add package Google.Apis.Sheets.v4

dotnet add package System.Net.Mail

1. **Google Sheets Integration**:
   * Authenticate the service account.
   * Fetch the required data from the Google Sheet:

var range = "DailyUpdate!A2:C2"; // Fixed row range for one message

1. **Selenium Setup**:
   * Automate browser interactions to log in and post the message.
   * Take a screenshot after posting the message.
2. **Email Notification**:
   * After the task is posted, capture a screenshot and send it via email.

**7. Deployment & Execution**

**Deploy it to the server by Dockerizing the application with docker hub**

**Step 3: Monitoring and Logging**

1. **Log Execution**:
   * Implement basic logging (e.g., using **Serilog** or **NLog**) to monitor the application’s activity.
2. **Check Email Reports**:
   * Ensure email notifications are received daily with the screenshot.

**8. Security Considerations**

* **Sensitive Data**: Store credentials and sensitive data (e.g., Google Sheets API credentials) securely. Use encrypted configuration files or environment variables.
* **Access Control**: Ensure that only authorized users or systems can update the Google Sheet and interact with the application.

**Step 9: Test the Application**

* Run the console application manually to ensure that:
  1. The application logs in correctly.
  2. The message is posted successfully.
  3. The screenshot is captured.
  4. The email is sent.