BAIT3173 Integrative Programming

**ASSIGNMENT 202405**

Student Name : Ng Jun Yu

Student ID : 23WMR09353

Programme : RSD3

Tutorial Group : G5

System Title : Book Store Management System

Modules : Staff Management

| **Declaration**   * I confirm that I have read and complied with all the terms and conditions of Tunku Abdul Rahman University of Management and technology’s plagiarism policy. * I declare that this assignment is free from all forms of plagiarism and for all intents and purposes is my own properly derived work. |
| --- |

**Plagiarism Statement Form**

I, Name Ng Jun Yu Student ID 23WMR09353 Programme RSD3 Tutorial Group 5 confirm that the submitted work are all my own work and is in my own words.

I Ng Jun Yu acknowledge the use of AI generative technology.

Signature: 

Date: 21-9-2024

**Table of Contents**

[**1. Introduction to the System 4**](#_gjdgxs)

[**2. Module Description 4**](#_30j0zll)

[Staff Management Module 4](#_20gxsc88j6w4)

[Screenshot 5](#_uuucwdcu7x6i)

[Login Page 5](#_vfo03on9uz60)

[Forget Password Page 5](#_bfyyi7yummhs)

[Home Page 6](#_t05goqpoqmhm)

[Profile Page 6](#_8f8ax0vfphjw)

[Attendance Page 7](#_3c5b6blfeyq2)

[Staff Management - Staff List 7](#_utlrkf8zdke3)

[Register Staff 8](#_dcxb7chmpuzg)

[Change Staff Position 8](#_a0ylfyjgw9q)

[Position List 9](#_ilv31u33owfd)

[Edit Position 9](#_6psbz2devhu6)

[Staff Attendance List 10](#_4nwqukuiv604)

[**3. Entity Classes 11**](#_1fob9te)

[Staff Model 11](#_nd9sq512ax2)

[Position Model 12](#_s8uavrcvozl)

[PasswordReset Model 13](#_au21e87no2xs)

[Attendance 13](#_9xatjz2v4vf4)

[**4. Design Pattern 15**](#_3znysh7)

[4.1 Description of Design Pattern 15](#_2et92p0)

[4.2 Implementation of Design Pattern 15](#_tyjcwt)

[**5. Software Security 17**](#_3dy6vkm)

[5.1 Authentication and Password Management 17](#)

[5.1.1 Authentication Failed Response 17](#_6tbjpgl9seoo)

[5.1.2 Use only HTTP POST requests to transmit authentication credentials 17](#_1q2lpegxbbar)

[5.1.3 Store Passwords Securely 18](#_b1iwzw5q7vjv)

[5.1.4 Enforce Password Length 18](#_puorgx84gtay)

[5.2 Data Protection 19](#_t9yj4e5664l)

[5.2.1 Encryption in Sensitive Data 19](#_ren1nwuytpb)

[5.2.2 Disable autocomplete in the login form 20](#_n10h5jo4i9mv)

[5.2.3 Data Removal Support 21](#_675dpnwsj1dk)

[**6. Implementation of XML, XSLT and XPath 22**](#_4d34og8)

[6.1 XML 22](#_qik50h9wqvo3)

[6.2 XSLT (attendance\_table.xsl) 24](#_nnlt5ntohkcn)

[6.3 XPath 25](#_9vn09bhd3xzi)

[**7. Web Services 26**](#_2s8eyo1)

[7.1 Web Service, SendGrid in Laravel 26](#_9vtbqxhzjsqq)

[EmailService.php 26](#_4x9ck32mvye9)

[EmailController.php 27](#_db85rbn7py5r)

[7.2 Web Client, Java Spring Boot 29](#_rk5v41cwx4h3)

[EmailController.java 29](#_thedp2cd3x65)

[MailService.php 30](#_pbsnjhqcfzrs)

[7.3 Process Flow 32](#_791b7e1t1sol)

[**8. Conclusion 34**](#_17dp8vu)

# 

# 1. Introduction to the System

The Bookstore Management System is an integrated software solution designed to manage activities related to bookstores efficiently. It is focused on special features, including coordinating the staff and controlling inventory effectively. It aspires to contribute toward increased productivity by decreasing the administrative burden while offering higher customer service.

This will help various kinds of bookstores maintain all the records correctly, manage the staff, and give a good amount of stock to satisfy the demand of the customers. Equipped with a friendly interface and robust features, the Bookstore Management System thus allows bookstores to function much better in the competitive marketplace.

The organizations that will use this system are:

1. Independent Bookstores: Smaller bookstores that want to optimize the work of their employees and control the available inventory effectively.
2. Chain Bookstores: Larger bookstores with branches, each of which would need the same management regarding personnel and inventory.
3. E-Retailers: Online bookstores are dependent on effective employee management vis-à-vis order processing and book inventory.
4. Academic Institutions: These are school or college bookstores that need to keep a record of the staff and their academic inventory.

# 2. Module Description

## Staff Management Module

The staff management module is aimed at streamlining the staff management process in a bookstore as it enables the authorized user to register a new staff, add a new position for their bookstore if needed, change the staff’s position and so on. In this case, the default authorized users are the admin and HR manager. The HR manager data can be modified while the admin will not show its details.

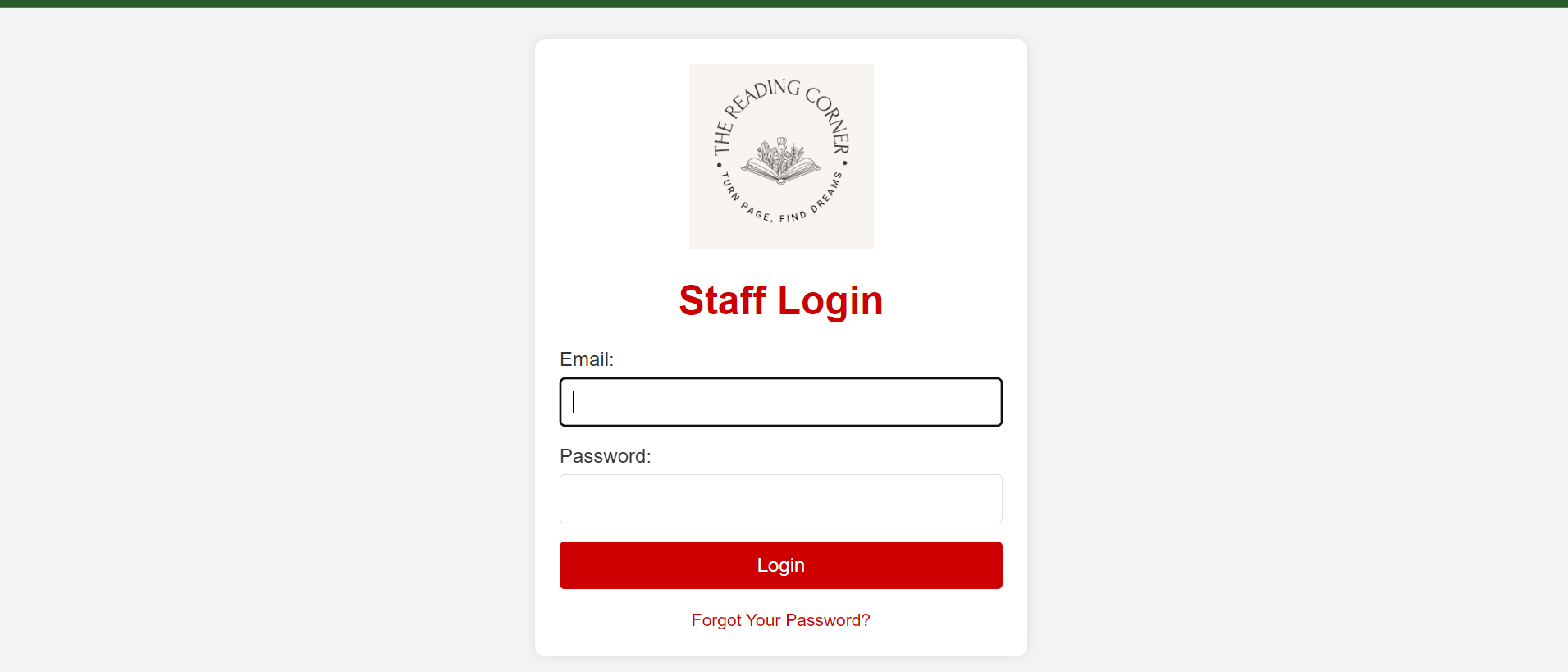
The functions of this module are:

1. Register a new staff by filling in the form with staff details. This needs to be operated by authorized users to prevent unauthorized access and maintain data integrity
2. Delete an existing staff account
3. Change a staff position
4. Normal CRUD position functions include creating a new position, reading the position and its current staff who handle this position, deleting the position but ensuring there are no staff in that position, and editing the position details.
5. Admin and HR manager can read all the staff attendance
6. Login and reset password function for staff
7. Normal staff can read their recent attendance

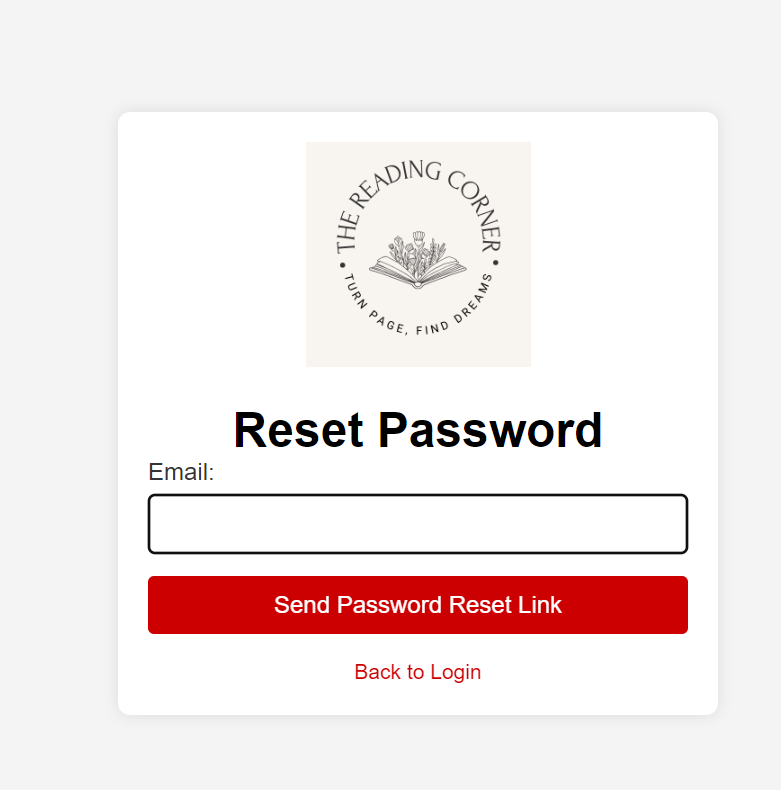
## 

## Screenshot

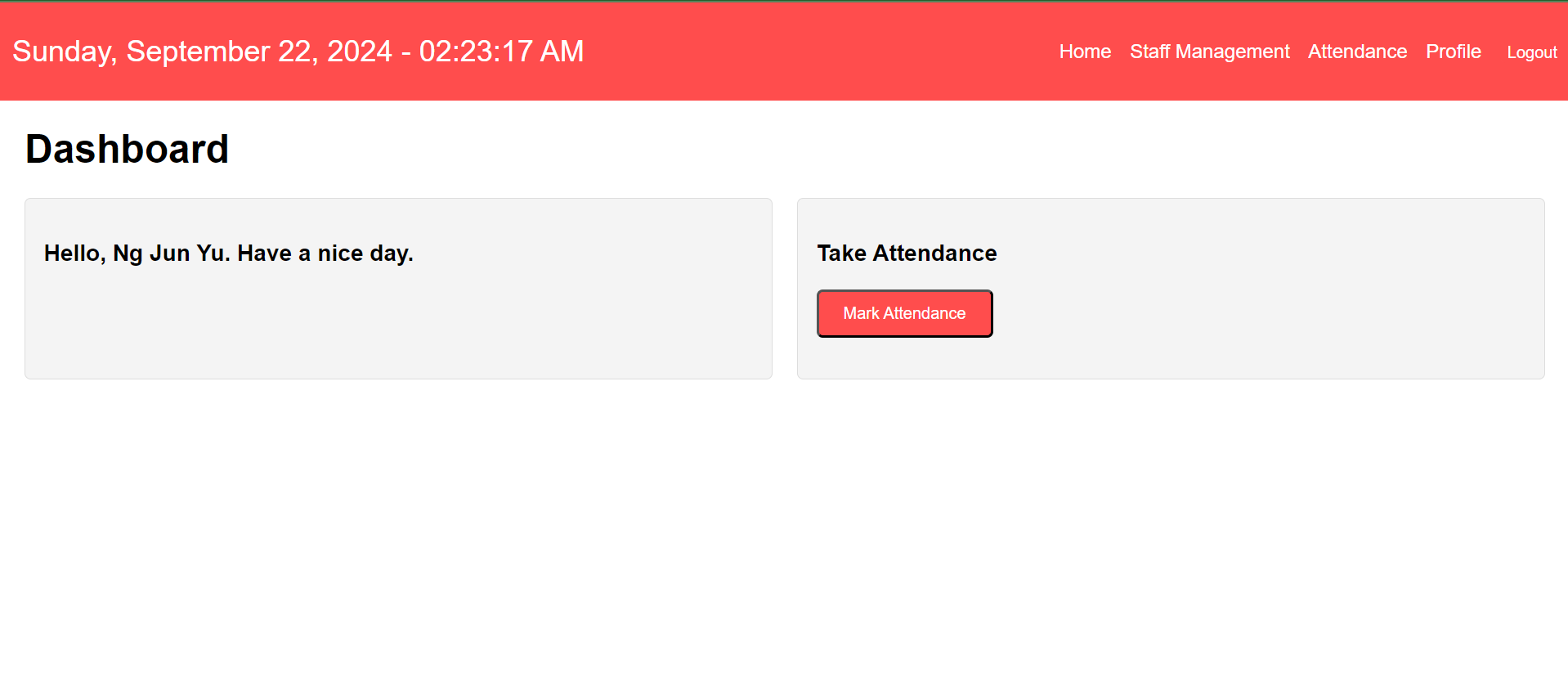
### Login Page



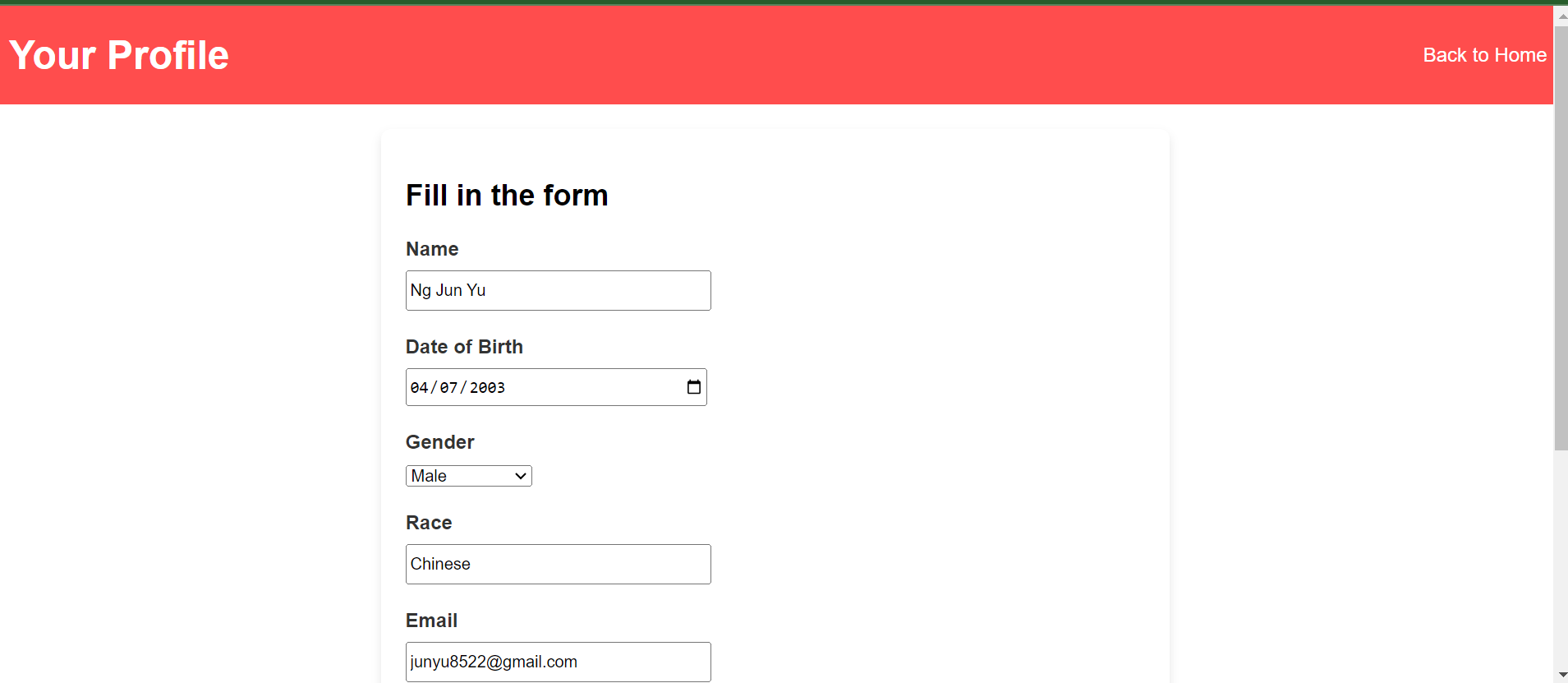
### Forget Password Page



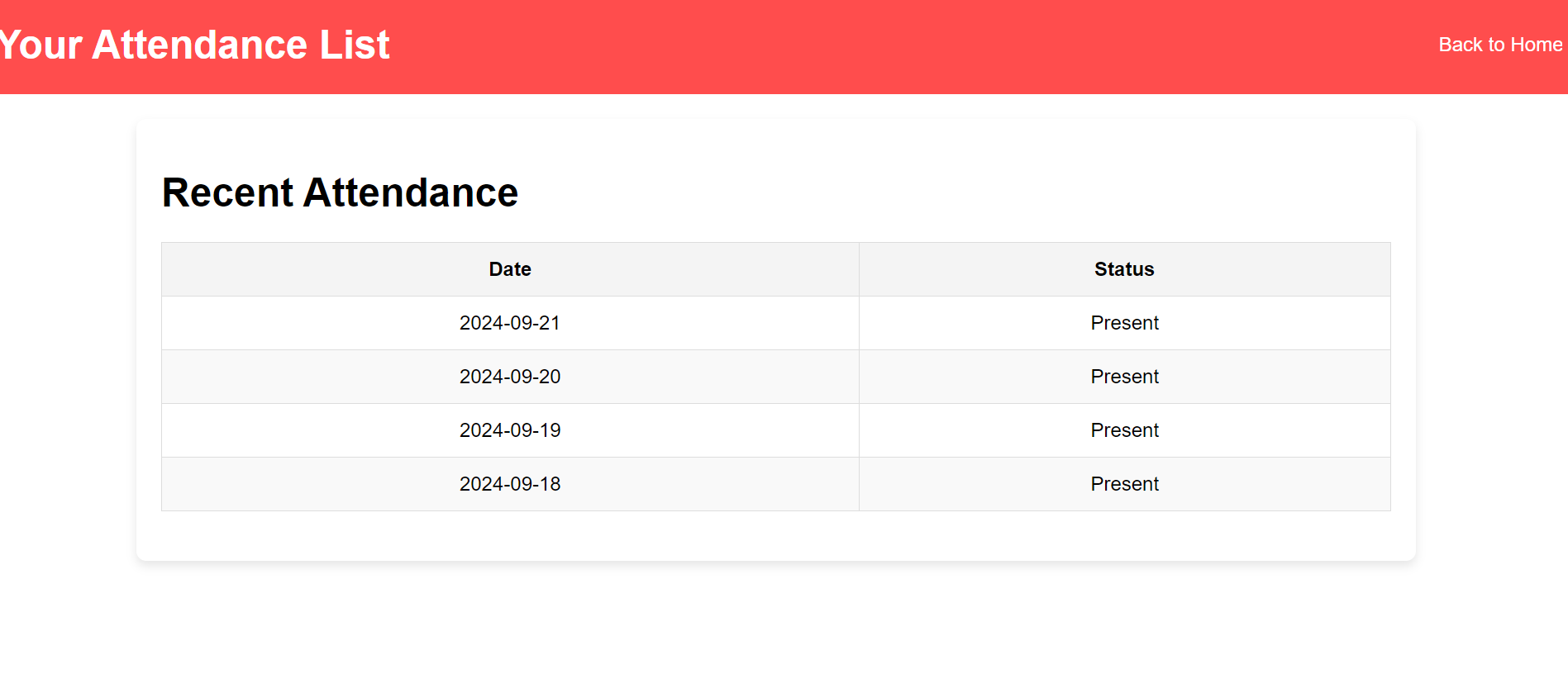
### Home Page



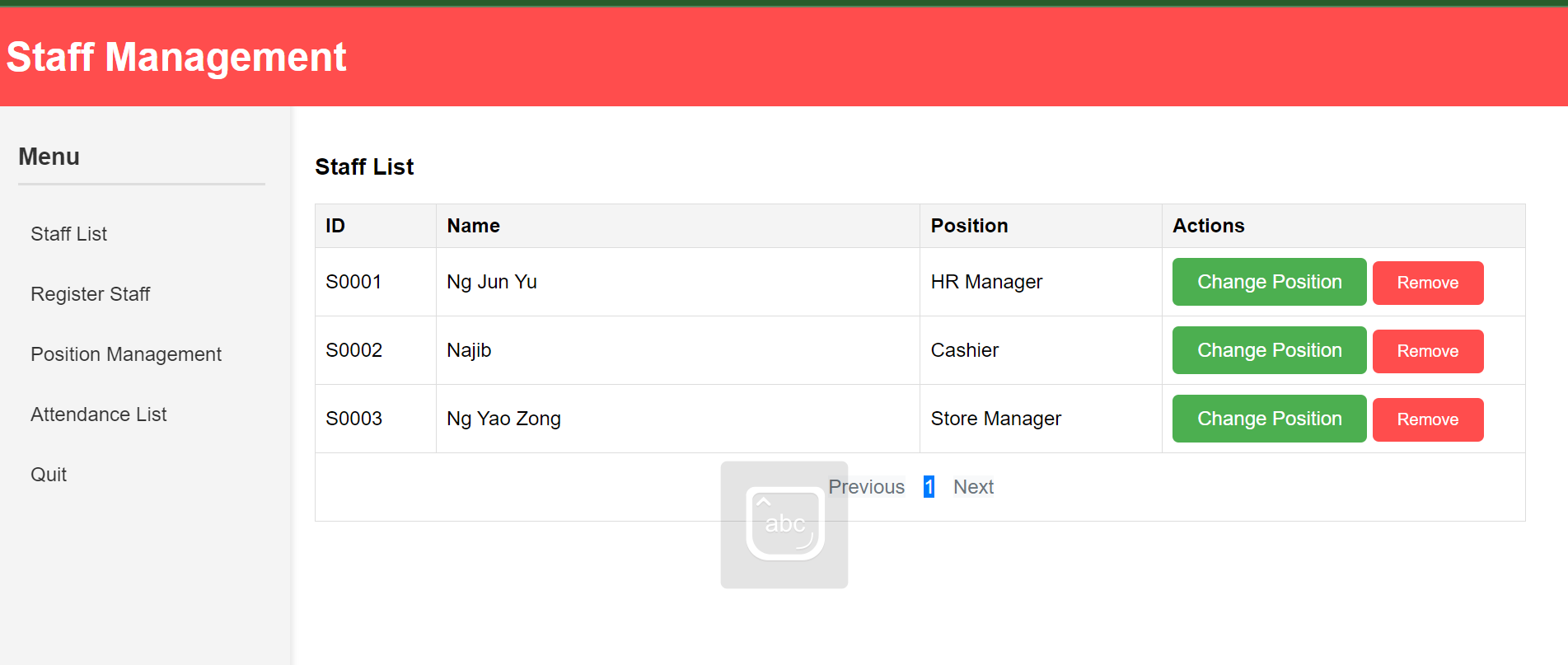
### Profile Page



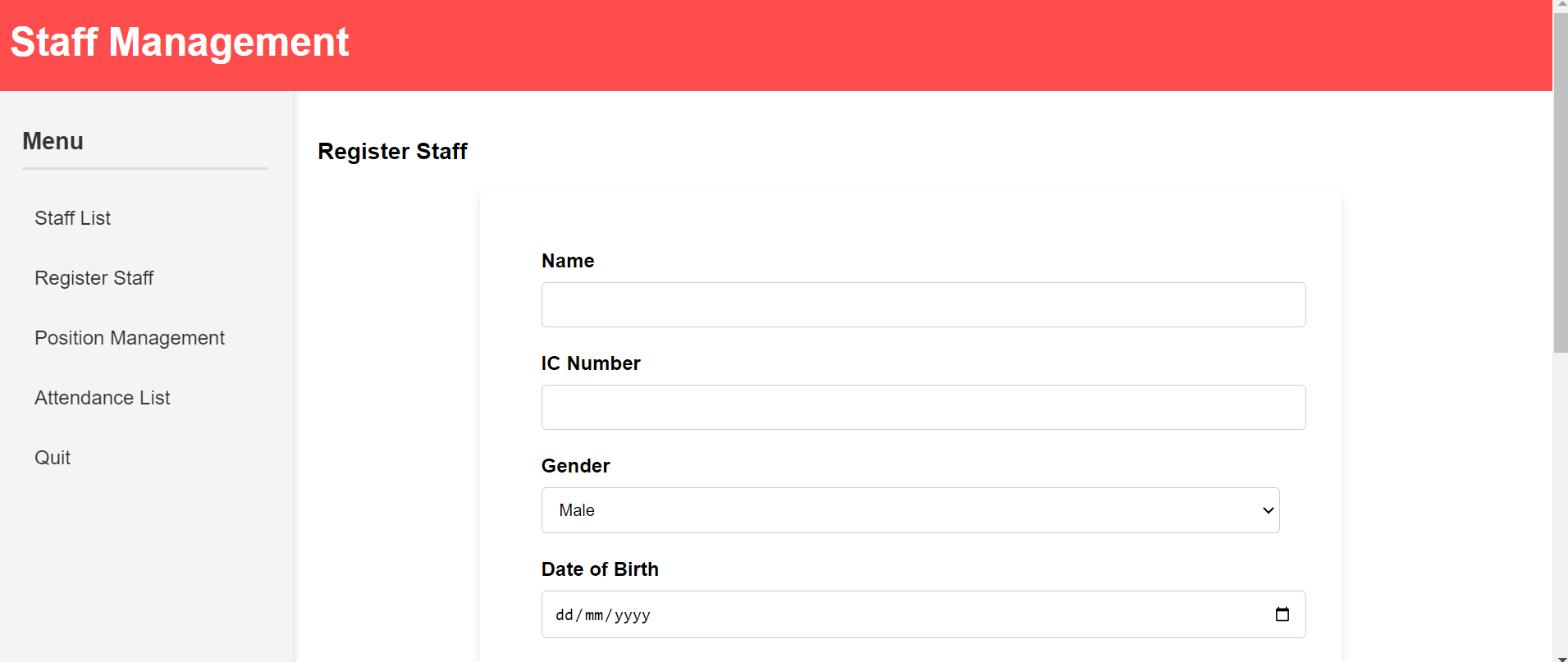
### Attendance Page



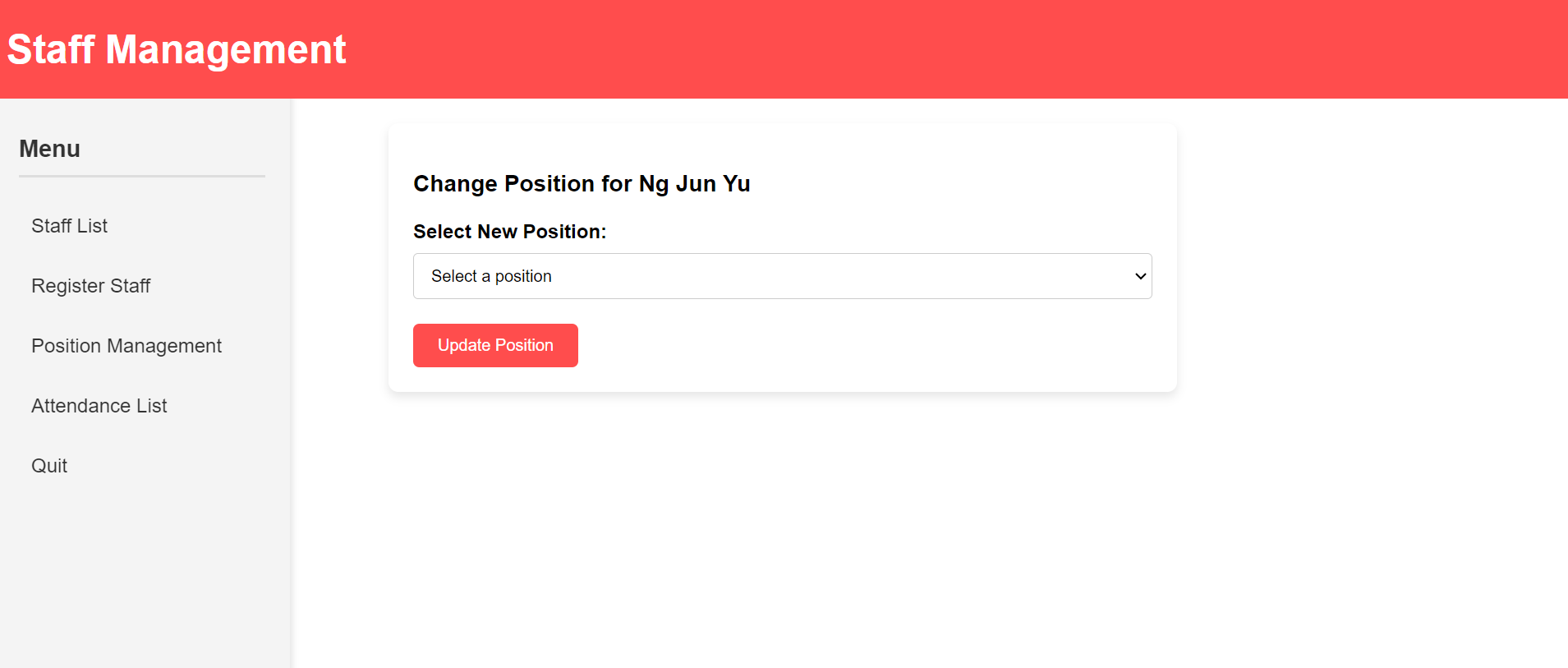
### Staff Management - Staff List



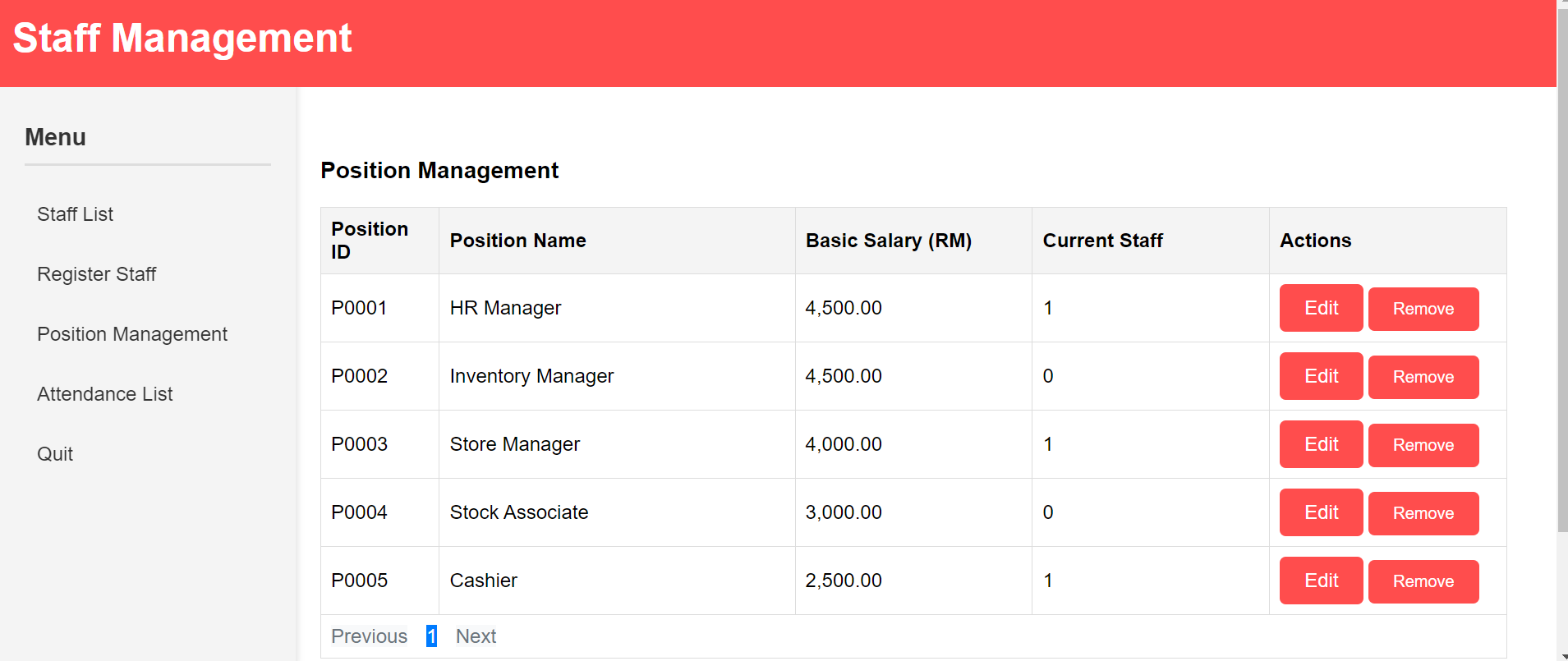
### Register Staff



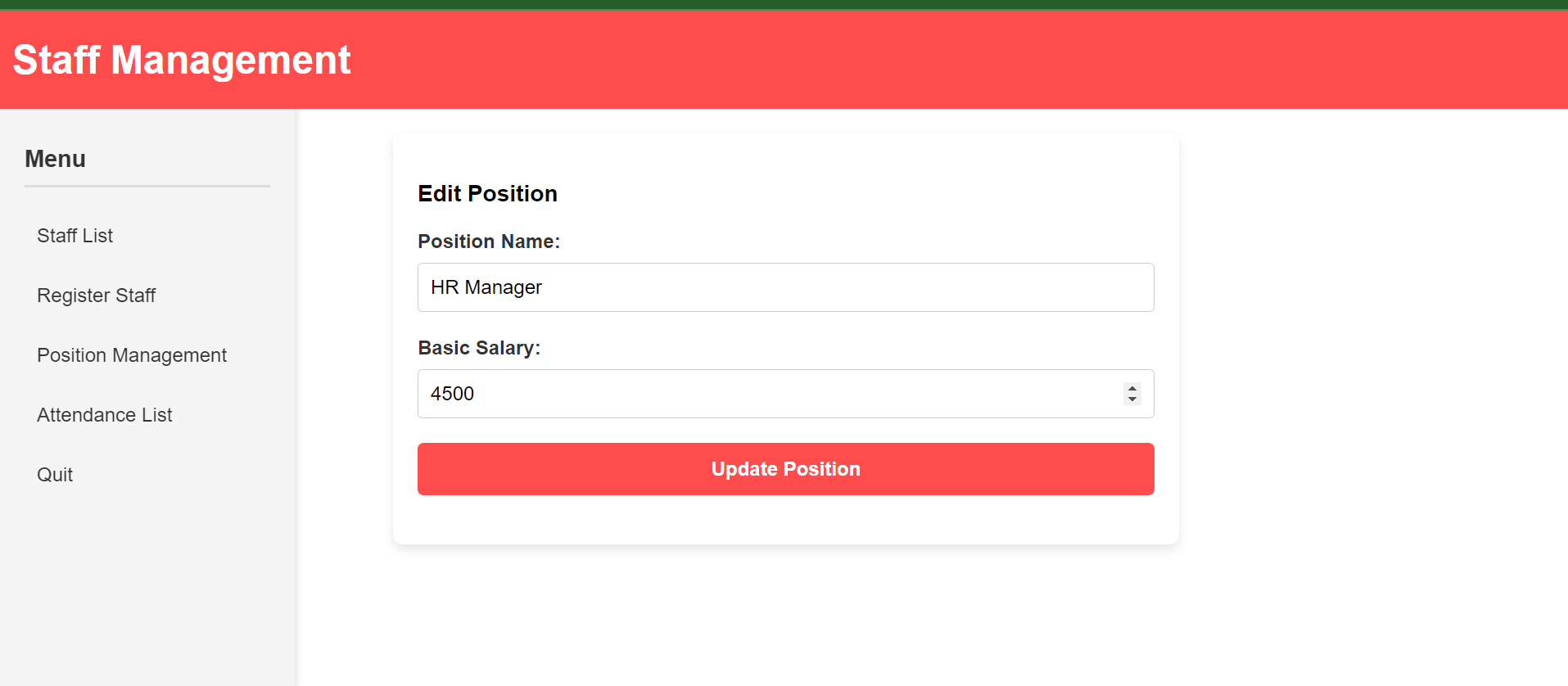
### Change Staff Position



### Position List



### Edit Position



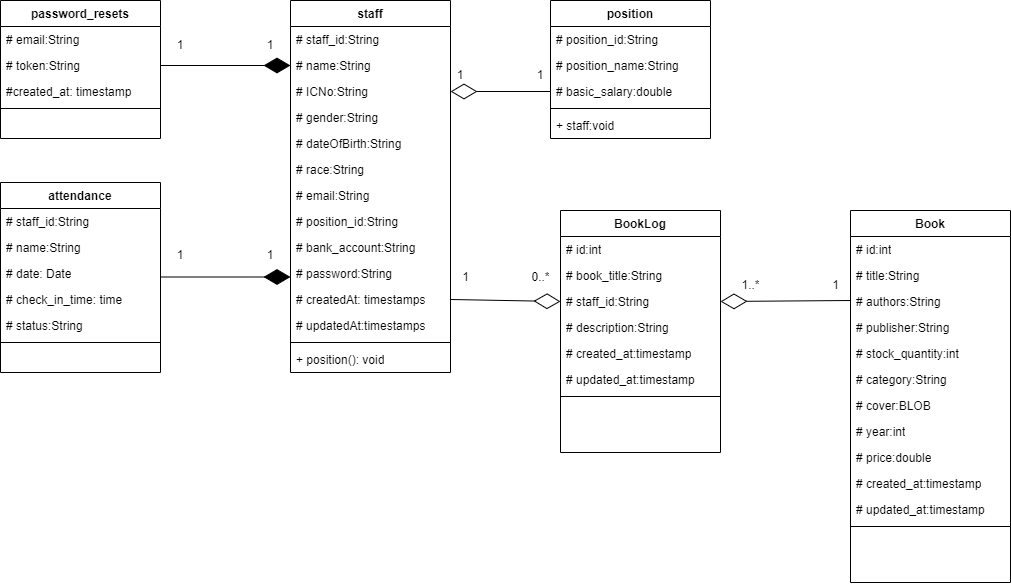
### Staff Attendance List

### 

# 

# 3. Entity Classes

a) Entity class diagram of the system.

* 

b) Entity classes source code(Models)

### Staff Model

| <?php  namespace App\Models;  use Illuminate\Database\Eloquent\Factories\HasFactory;  use Illuminate\Database\Eloquent\Model;  class Staff extends Model  {  use HasFactory;  protected $primaryKey = 'staff\_id';  public $incrementing = false;  protected $keyType = 'string';  protected $table = 'staff';  protected $fillable = [  'staff\_id',  'name',  'ICNo',  'gender',  'dateOfBirth',  'race',  'email',  'position\_id',  'bank\_account',  'password',  ];  public function position()  {  *return* $this->belongsTo(Position::class, 'position\_id', 'position\_id');  }  } |
| --- |

### Position Model

| <?php  namespace App\Models;  use Illuminate\Database\Eloquent\Factories\HasFactory;  use Illuminate\Database\Eloquent\Model;  class Position extends Model  {  use HasFactory;  protected $table = "position";  protected $primaryKey = 'position\_id';  public $incrementing = false;  protected $keyType = 'string';  public $timestamps = false;  protected $fillable = [  'position\_id',  'position\_name',  'basic\_salary'  ];  public function staff()  {  *return* $this->hasMany(Staff::class, 'position\_id', 'position\_id');  }  } |
| --- |

### PasswordReset Model

| <?php  namespace App\Models;  use Illuminate\Database\Eloquent\Factories\HasFactory;  use Illuminate\Database\Eloquent\Model;  class PasswordReset extends Model  {  use HasFactory;  protected $table = 'password\_resets';  public $timestamps = false;  protected $primaryKey = null;  protected $fillable = [  'email',  'token',  ];  } |
| --- |

### Attendance

For the attendance model, there is no attendance model in this project as it has not implemented a database table. This is due to the reason that XML is a lightweight solution for storing simple data and attendance data does not need the complexity and overhead of a relational database. However, below code will show how the XML file saves data.

| //ensure the xml file is existed, if not the system will create a new one  protected function loadOrCreateXML()  {  *if* (file\_exists($this->xmlFilePath)) {  *return* simplexml\_load\_file($this->xmlFilePath);  } *else* {  *return* new SimpleXMLElement('<attendance></attendance>');  }  }  //save the attendance data in xml file  public function storeAttendance($staffId, $name, $status, $checkInTime)  {  $xml = $this->loadOrCreateXML();  $newRecord = $xml->addChild('record');  $newRecord->addChild('staff\_id', $staffId);  $newRecord->addChild('name', $name);  $newRecord->addChild('date', date('Y-m-d'));  $newRecord->addChild('check\_in\_time', $checkInTime ?? now()->format('H:i:s'));  $newRecord->addChild('status', $status);  $xml->asXML($this->xmlFilePath);  } |
| --- |

# 

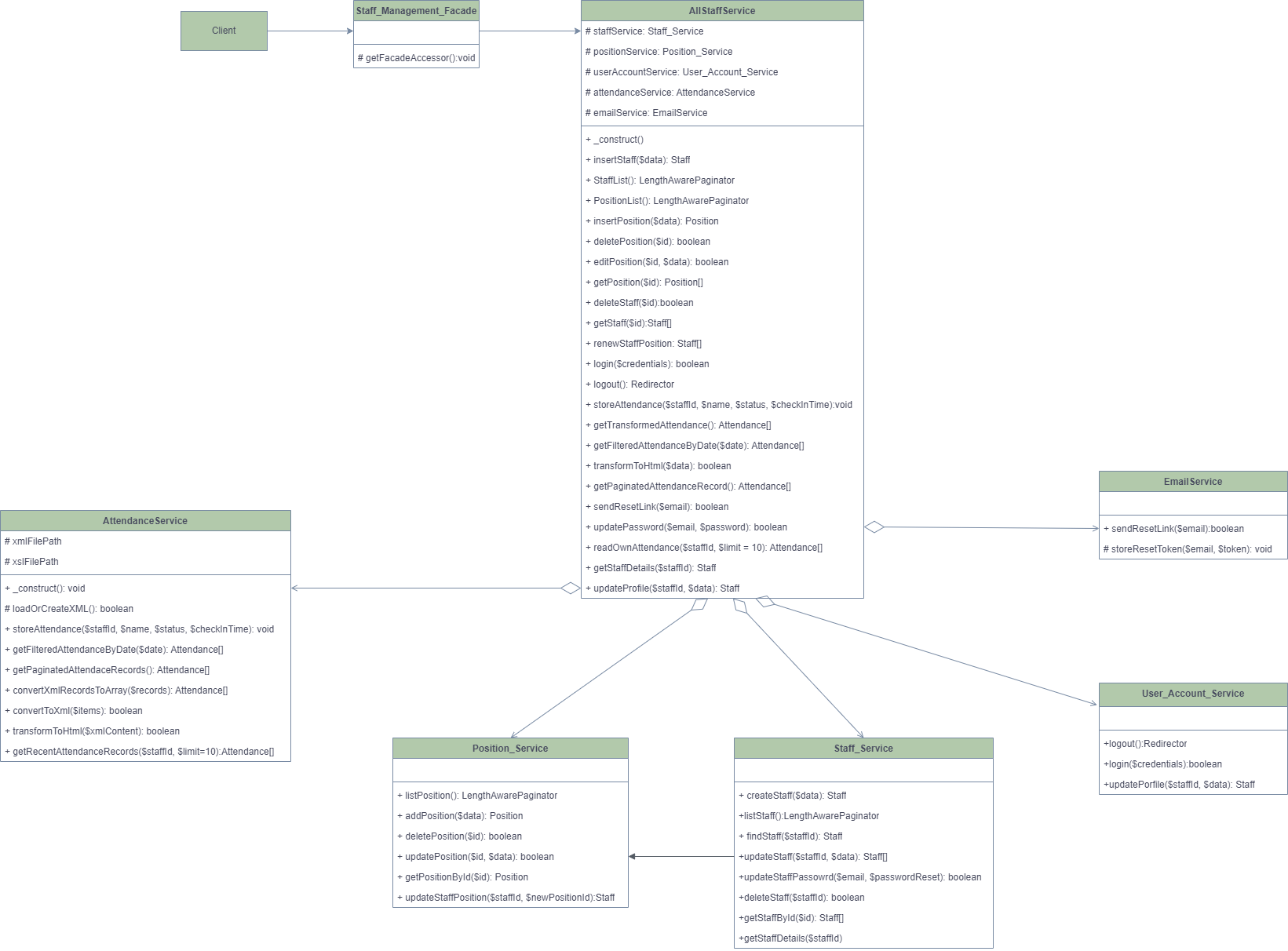
# 4. Design Pattern

## 4.1 Description of Design Pattern

The Facade design pattern is applied when it's required to give an easily understandable interface to some more complex system, whereby doing so, access to some underlying functionality of the system becomes much easier for clients. In this case, the project had applied the Facade design pattern on the Staff Management module in the Laravel application.

* Objective:
* Accordingly, the Facade design pattern in your Staff Management module will ease the work of different staff functionalities that may relate to adding a position, fetching staff information, or working with attendance.
* However, it provides a unified interface StaffManagement Facade-from which to access those complex operations that otherwise would have needed to interact with several services, repositories, or controllers.

## 4.2 Implementation of Design Pattern



Justification:

1. Simplify Interface: Facade Design Pattern is chosen because it provides a single, unified interface to a set of interfaces in some subsystems. Consequently, this will make it easier for users of the system to interact with major operations without interest in the underlying complexities.
2. Decoupling: Can decouple the higher-level modules from the underlying implementations by using the facade. This reduces the dependencies throughout different parts of the application, and by doing so, it makes management and maintenance easier.
3. Improved Readability of Code: The Facade pattern cleans up the code as the complex logic is moved to the background of a simple, easily usable interface. This makes it easier to read for other developers.
4. Easier Testing: An interface can be easily mocked or stubbed while testing. It lets the developers test components independently with no need to bother about the complexities of the underlying system.
5. Encapsulating Changes: The developers might want to change their subsystem's implementation without changing the higher-level code using the facade. This encapsulation of changes makes your application resistant to changes.

# 

# 5. Software Security

## 5.1 Authentication and Password Management

### 5.1.1 Authentication Failed Response

Authentication failure responses should not indicate which part of the authentication data was incorrect. For example, instead of "Invalid username" or "Invalid password", just use "Invalid username and/or password" for both. Error responses must be truly identical in both display and source code. This can ensure that the hacker tries to login into the account but cannot get a clear error message to increase the success rate of hacking into the account.

| public function login(Request $request)  {  $credentials = $request->only('email', 'password');  $loginSuccess = StaffManagement::login($credentials);  *if* ($loginSuccess) {  *return* redirect('/home')->with('success', 'Logged in successfully');  } *else* {  *return* redirect()->back()->withErrors(provider: ['error' => 'Invalid email or password']);  }  } |
| --- |

### 5.1.2 Use only HTTP POST requests to transmit authentication credentials

POST requests send data in the request body rather than the URL. This can make it less visible in logs, browser history, and network monitoring tools. This reduces the risk of sensitive information being exposed.

| <form method="POST" action="{{ route('loginAccount') }}" autocomplete="off">  @csrf  <div class="form-group">  <label for="email">Email:</label>  <input type="email" id="email" name="email" value="{{ old('email') }}" required autofocus autocomplete="off" >  </div>  <div class="form-group">  <label for="password">Password:</label>  <input type="password" id="password" name="password" required autocomplete="off">  </div>  <button type="submit">Login</button>  </form> |
| --- |

### 5.1.3 Store Passwords Securely

If an organization's database is compromised, securely stored passwords will make it harder for attackers to exploit compared to plaintext passwords. The encryption way can refer to 5.2.1 Encryption in Sensitive Data.

| password' => $dataProtection->encryptData($data['email']) |
| --- |

### 5.1.4 Enforce Password Length

Require a minimum password length during resetting the password.

| $validator = Validator::make($request->all(), [  'password' => 'required|min:8',  'token' => 'required'  ]);  *if* ($validator->fails()) {  *return* back()->withErrors(["error" => "Your password must at least 8."]);  } |
| --- |

## 

## 5.2 Data Protection

### 5.2.1 Encryption in Sensitive Data

In the staff table, there are some sensitive data such as bank account, password and so on. Therefore, it needs a strong encryption algorithm to protect the data.

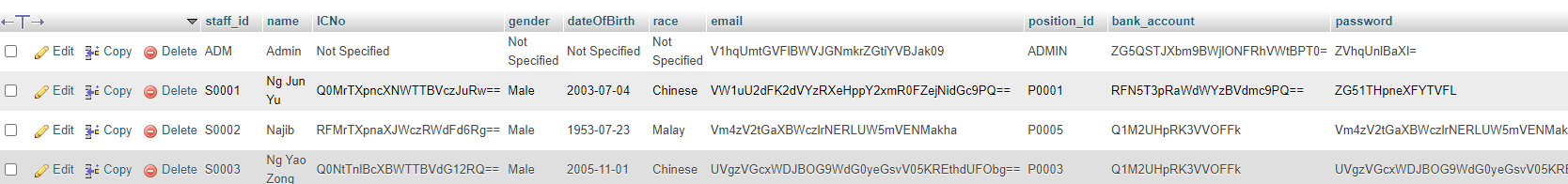
In DataProtection.php, this class provides a simple interface for encrypting and decrypting data using the AES-256-CTR algorithm. It securely generates a key and IV based on environment variables and a fixed string, making it suitable for protecting sensitive data.

The protection key is saved and configured in .env file which is “DATA\_PROTECTION\_KEY=”BOOKSTORE-MANAGEMENT-SYSTEM-DEMO””

Source code:

| ?php  namespace App\Security;  class DataProtection {  private $secretKey;  private $ciphering = "AES-256-CTR";  private $options = 0;  private $iv;  public function \_\_construct() {  $this->secretKey = substr(hash('sha256', env('DATA\_PROTECTION\_KEY')), 0, 32);  $this->iv = substr(hash('sha256', 'bookstore-management-system-iv'), 0, openssl\_cipher\_iv\_length($this->ciphering));  }  public function encryptData($data) {  $encryptedData = openssl\_encrypt($data, $this->ciphering, $this->secretKey, $this->options, $this->iv);  *return* base64\_encode($encryptedData);  }  public function decryptData($encryptedData) {  $encryptedData = base64\_decode($encryptedData);  *return* openssl\_decrypt($encryptedData, $this->ciphering, $this->secretKey, $this->options, $this->iv);  }  } |
| --- |

The encrypted result will be:



### 5.2.2 Disable autocomplete in the login form

This can prevent browsers from storing and auto-filling sensitive information. In this case, is email and password. This can help protect against unauthorized access if someone gains access to a user's device.

Below is the form on the login page, it will prevent the user from automatically filling in the form

| <form method="POST" action="{{ route('loginAccount') }}" autocomplete="off">  @csrf  <div class="form-group">  <label for="email">Email:</label>  <input type="email" id="email" name="email" value="{{ old('email') }}" required autofocus autocomplete="off" >  </div>  <div class="form-group">  <label for="password">Password:</label>  <input type="password" id="password" name="password" required autocomplete="off">  </div>  <button type="submit">Login</button>  </form> |
| --- |

### 5.2.3 Data Removal Support

Data deletion reduces an organization's liability in case of data breaches because less sensitive information is stored, meaning a lower risk in case of a breach. In this case, the reset password link token will be deleted when used so the users need to request a new password reset link. This can be avoided if others get the user’s device to reset the password

| public function storeResetToken($email, $resetToken)  {  PasswordReset::where('email', $email)->delete();  PasswordReset::insert([  'email' => $email,  'token' => Hash::make($resetToken),  'created\_at' => now(),  ]);  } |
| --- |

# 

# 6. Implementation of XML, XSLT and XPath

For this project, the XML, XSLT and XPath are used to store attendance list and print them out to the normal users and admin.

## 6.1 XML

The XML is been used to store the attendance from the staff. Thus, the AttendanceService.php will be implemented to create a new XML file if the file named staff\_attendance in storage/app/xml folder.

AttendanceService.php which store data in XML file:

| <?php  namespace App\Services;  use Illuminate\Support\Facades\Facade;  use SimpleXMLElement;  use DOMDocument;  use XSLTProcessor;  use Illuminate\Pagination\LengthAwarePaginator;  class AttendanceService extends Facade  {  protected $xmlFilePath;  protected $xslFilePath;  public function \_\_construct()  {  $this->xmlFilePath = storage\_path('app/xml/staff\_attendance.xml');  $this->xslFilePath = storage\_path('app/xml/attendance\_table.xsl');  }  protected function loadOrCreateXML()  {  *if* (file\_exists($this->xmlFilePath)) {  *return* simplexml\_load\_file($this->xmlFilePath);  } *else* {  *return* new SimpleXMLElement('<attendance></attendance>');  }  }  public function storeAttendance($staffId, $name, $status, $checkInTime)  {  $xml = $this->loadOrCreateXML();  $newRecord = $xml->addChild('record');  $newRecord->addChild('staff\_id', $staffId);  $newRecord->addChild('name', $name);  $newRecord->addChild('date', date('Y-m-d'));  $newRecord->addChild('check\_in\_time', $checkInTime ?? now()->format('H:i:s'));  $newRecord->addChild('status', $status);  $xml->asXML($this->xmlFilePath);  }  //other functions…  } |
| --- |

Result of staff\_attendance.xml:

| <?xml version="1.0"?>  <attendance>  <record>  <staff\_id>S0001</staff\_id>  <name>Ng Jun Yu</name>  <date>2024-09-18</date>  <check\_in\_time>11:22</check\_in\_time>  <status>Present</status>  </record>  <record>  <staff\_id>S0001</staff\_id>  <name>Ng Jun Yu</name>  <date>2024-09-19</date>  <check\_in\_time>01:50</check\_in\_time>  <status>Present</status>  </record>  <record>  <staff\_id>S0001</staff\_id>  <name>Ng Jun Yu</name>  <date>2024-09-20</date>  <check\_in\_time>11:50</check\_in\_time>  <status>Present</status>  </record>  <record>  <staff\_id>S0003</staff\_id>  <name>Ng Yao Zong</name>  <date>2024-09-20</date>  <check\_in\_time>18:35</check\_in\_time>  <status>Present</status>  </record>  <record>  <staff\_id>S0002</staff\_id>  <name>Najib</name>  <date>2024-09-20</date>  <check\_in\_time>18:35</check\_in\_time>  <status>Present</status>  </record>  <record>  <staff\_id>S0001</staff\_id>  <name>Ng Jun Yu</name>  <date>2024-09-21</date>  <check\_in\_time>02:52</check\_in\_time>  <status>Present</status>  </record>  <record>  <staff\_id>S0001</staff\_id>  <name>Ng Jun Yu</name>  <date>2024-09-22</date>  <check\_in\_time>02:24</check\_in\_time>  <status>Present</status>  </record>  </attendance> |
| --- |

## 6.2 XSLT (attendance\_table.xsl)

Next, Once the XML is generated, it’s transformed into a readable HTML table using XSLT. The XSL file defines how to display the XML data.

| <?xml version="1.0" encoding="UTF-8"?>  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  <xsl:template match="/">  <table class="attendance-list">  <tr class="header-row">  <th class="id">Staff ID</th>  <th class="name">Name</th>  <th class="date">Date</th>  <th class="check-in-time">Check-In Time</th>  <th class="status">Status</th>  </tr>  <xsl:for-each select="attendance/record">  <tr>  <td class="id">  <xsl:value-of select="staff\_id" />  </td>  <td class="name">  <xsl:value-of select="name" />  </td>  <td class="date">  <xsl:value-of select="date" />  </td>  <td class="check-in-time">  <xsl:value-of select="check\_in\_time" />  </td>  <td class="status">  <xsl:value-of select="status" />  </td>  </tr>  </xsl:for-each>  </table>  </xsl:template>  </xsl:stylesheet> |
| --- |

## 6.3 XPath

It is a query language used to navigate and select nodes from an XML document. It provides a way to access and manipulate the content and structure of XML data. The below XPath is used to call the attendance record.

| <xsl:for-each select="attendance/record"> |
| --- |

# 

# 7. Web Services

The web service will be implemented using Java Spring Boot, whereby it sends an email with a link to reset the password via SendGrid. Such a service is to be exposed as a RESTful API, which this application can consume, and the application itself is at Laravel-based.

Objective: To implement a secure and scalable architecture for the web service in sending the password reset link using SendGrid via email. The service is in Java Spring Boot to take full advantage of the power of its backend, while the consumer Laravel application in PHP listens for user requests to reset the password. This here shows cross-language integrations where the service and consumer-run under different programming environments. The purpose is to get the email delivered reliably to enhance security by handling user credentials correctly, and also make the password reset feature more maintainable and modular.

Separating the email service from the main application can ensure that email delivery will be highly available and fault-tolerant, security will be enhanced by centralizing the email-sending logic, and integration with other modules or applications in the future will be easy irrespective of their programming language. Thus, user authentication can be protected as it can verify the email is real and others without email accounts cannot reset the password easily.

## 7.1 Web Service, SendGrid in Laravel

### EmailService.php

The EmailService class handles sending password reset links to users. It generates a token then saves it in the database, and sends the reset link via an external Java service. It also validates the token when users attempt to reset their passwords.

| <?php  namespace App\Services;  use Illuminate\Support\Facades\Http;  use App\Models\PasswordReset;  use Illuminate\Support\Str;  use Illuminate\Support\Facades\Hash;  class EmailService  {  public function sendResetLink($email)  {  $resetToken = Str::random(60);  $resetLink = url('/reset-password?token=' . base64\_encode($resetToken));  $this->storeResetToken($email, $resetToken);  $url = env('JAVA\_SERVICE\_URL');  $response = Http::post($url, [  'email' => $email,  'resetLink' => $resetLink,  ]);  *return* $response->successful();  }  public function storeResetToken($email, $resetToken)  {  PasswordReset::where('email', $email)->delete();  PasswordReset::insert([  'email' => $email,  'token' => Hash::make($resetToken),  'created\_at' => now(),  ]);  }  public function validateResetToken($email, $token)  {  $decodedToken = base64\_decode($token);  $resetRecord = PasswordReset::where('email', $email)->first();  *if* ($resetRecord && Hash::check($decodedToken, $resetRecord->token) && $resetRecord->created\_at->addHours(1)->isFuture()) {  *return* true;  }  *return* false;  }  } |
| --- |

### EmailController.php

The EmailController sends password reset emails and updates the reset token in the database, providing feedback on success or failure.

# 

| **<?php**  **namespace App\Http\Controllers;**  **use Illuminate\Http\Request;**  **use App\Models\PasswordReset;**  **use StaffManagement;**  **class EmailController extends Controller**  **{**  **public function sendResetEmail(Request $request)**  **{**  **$email = $request->input('email');**  ***if* (StaffManagement::sendResetLink($email)) {**  ***return* back()->with('status', 'Password reset link sent!');**  **}**  ***return* back()->withErrors(['email' => 'Failed to send reset link. Please try again.']);**  **}**  **protected function storeResetToken($email, $token)**  **{**  **PasswordReset::where('email', $email)->delete();**  **PasswordReset::updateOrInsert([**  **'email' => $email,**  **'token' => $token**  **]);**  **}**  **}** |
| --- |

## 

## 

## 7.2 Web Client, Java Spring Boot

### EmailController.java

The EmailController in this Spring Boot application handles sending password reset emails. It listens for POST requests with an email and reset link, then calls the MailService to send the email. If successful, it returns a success message; otherwise, it handles and returns any errors.

| <?php  namespace App\Services;  use Illuminate\Support\Facades\Http;  use App\Models\PasswordReset;  use Illuminate\Support\Str;  use Illuminate\Support\Facades\Hash;  class EmailService  {  public function sendResetLink($email)  {  $resetToken = Str::random(60);  $resetLink = url('/reset-password?token=' . base64\_encode($resetToken));  $this->storeResetToken($email, $resetToken);  $url = env('JAVA\_SERVICE\_URL');  $response = Http::post($url, [  'email' => $email,  'resetLink' => $resetLink,  ]);  *return* $response->successful();  }  public function storeResetToken($email, $resetToken)  {  PasswordReset::where('email', $email)->delete();  PasswordReset::insert([  'email' => $email,  'token' => Hash::make($resetToken),  'created\_at' => now(),  ]);  }  public function validateResetToken($email, $token)  {  $decodedToken = base64\_decode($token);  $resetRecord = PasswordReset::where('email', $email)->first();  *if* ($resetRecord && Hash::check($decodedToken, $resetRecord->token) && $resetRecord->created\_at->addHours(1)->isFuture()) {  *return* true;  }  *return* false;  }  } |
| --- |

### MailService.php

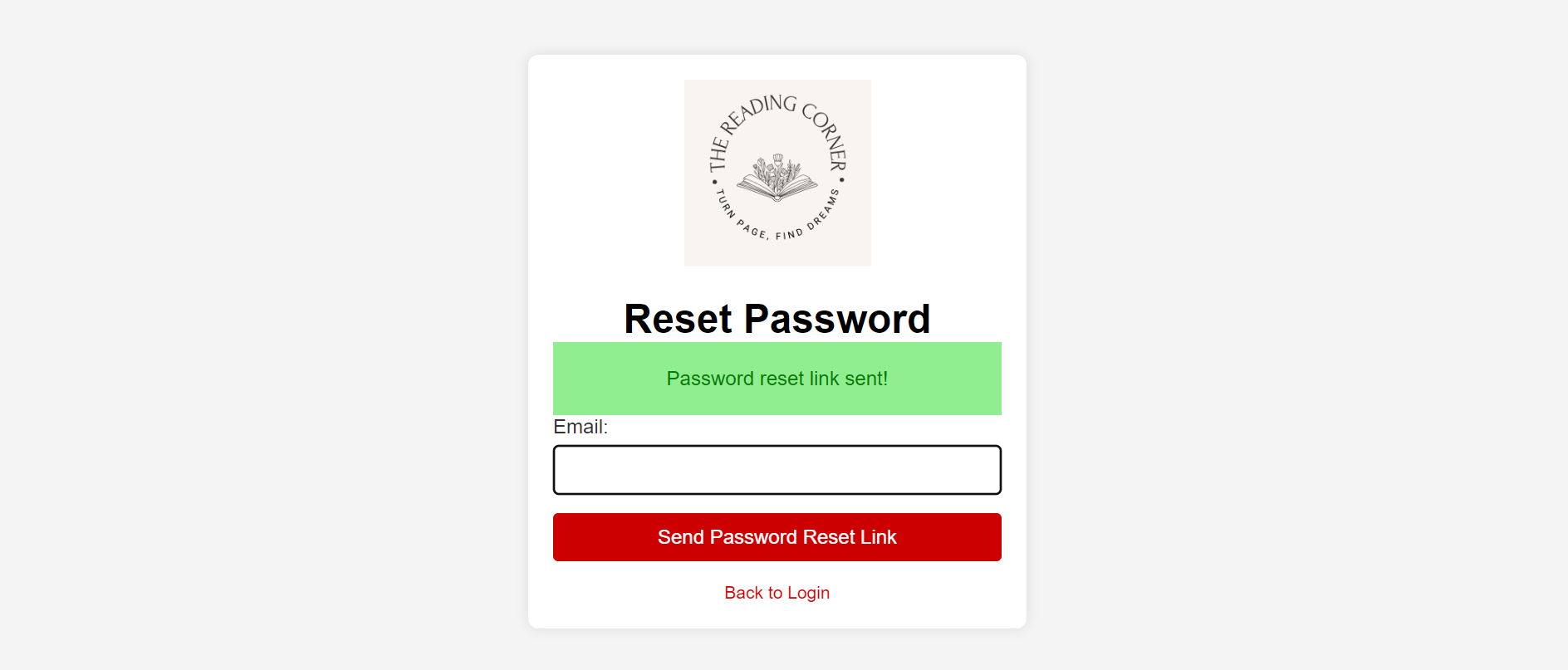
The EmailService class manages the password reset functionality. It generates a reset token, creates a password reset link, and sends it via an external Java service. The service also stores the reset token securely in the database and validates the token when a user attempts to reset their password.

| **package com.example.sendgrid\_email\_service.Service;**  **import java.io.IOException;**  **import org.slf4j.Logger;**  **import org.slf4j.LoggerFactory;**  **import org.springframework.stereotype.Service;**  **import com.sendgrid.Content;**  **import com.sendgrid.Email;**  **import com.sendgrid.Mail;**  **import com.sendgrid.Method;**  **import com.sendgrid.Request;**  **import com.sendgrid.Response;**  **import com.sendgrid.SendGrid;**  **@Service**  ***public* *class* MailService {**  ***private* *static* *final* Logger logger = LoggerFactory.getLogger(MailService.*class*);**    ***public* String sendTextEmail(String *ReceiverEmail*, String *ResetLink*) throws IOException {**  **Email from = *new* Email("junyu8522@gmail.com");**  **String subject = "Reset your password";**  **Email to = *new* Email(*ReceiverEmail*);**  **Content content = *new* Content("text/plain", "Click here to reset your password: " + *ResetLink*);**  **Mail mail = *new* Mail(from, subject, to, content);**    **SendGrid sg = *new* SendGrid("SG.b6zffdbJT2iuSh\_f5q-NQA.zzNvc7AotuBnDD06nPkwyt31buO7mxPyI3sA-WdXkio");**  **Request request = *new* Request();**  ***try* {**  **request.setMethod(Method.POST);**  **request.setEndpoint("mail/send");**  **request.setBody(mail.build());**  **Response response = sg.api(request);**  **logger.info(response.getBody());**  ***return* response.getBody();**  **} *catch* (IOException *ex*) {**  ***throw* ex;**  **}**  **}**  **}** |
| --- |

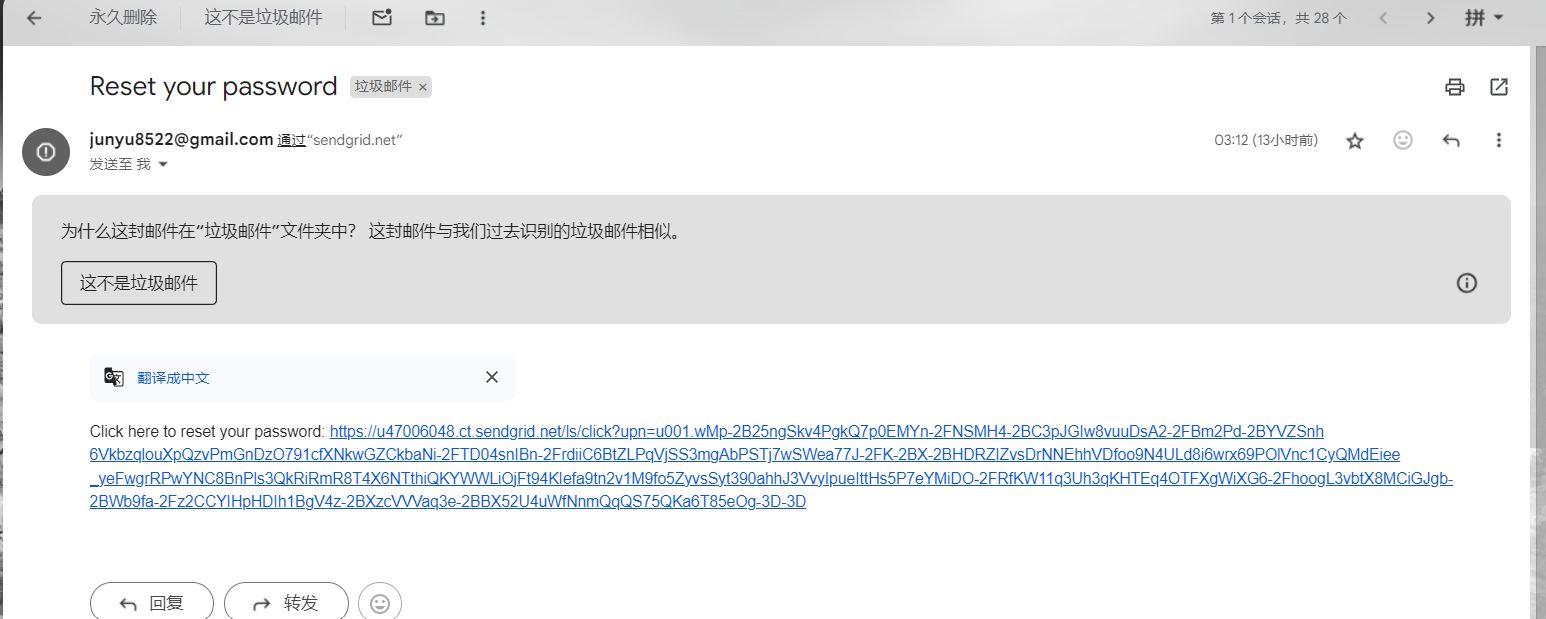
## 

## 7.3 Process Flow

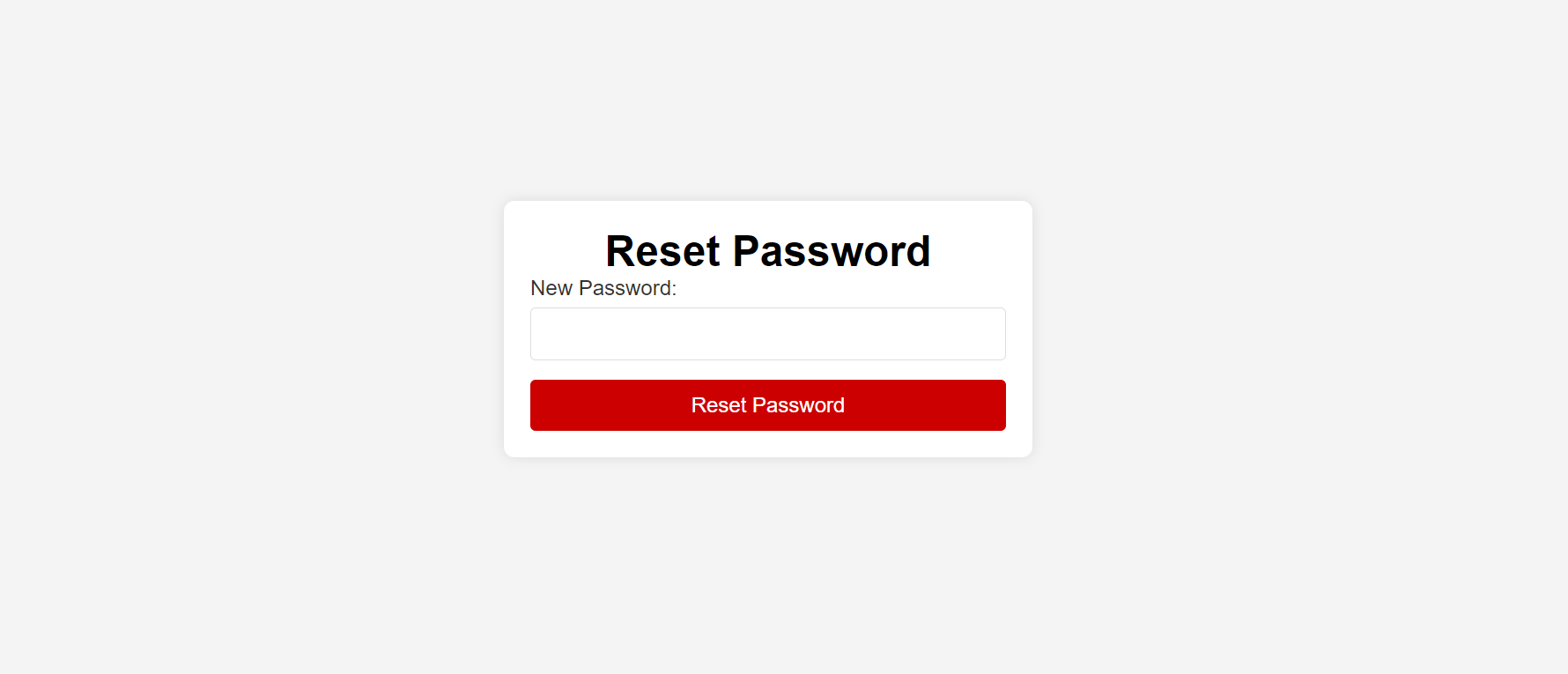
The user needs to enter the email address that he/she wants to reset the password. If the password reset link is sent successfully, the success message will be prompted.



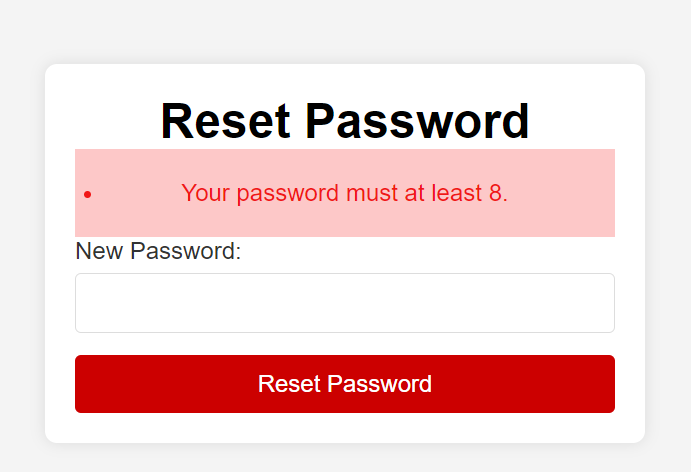
Then, the user will receive the email with the reset link.



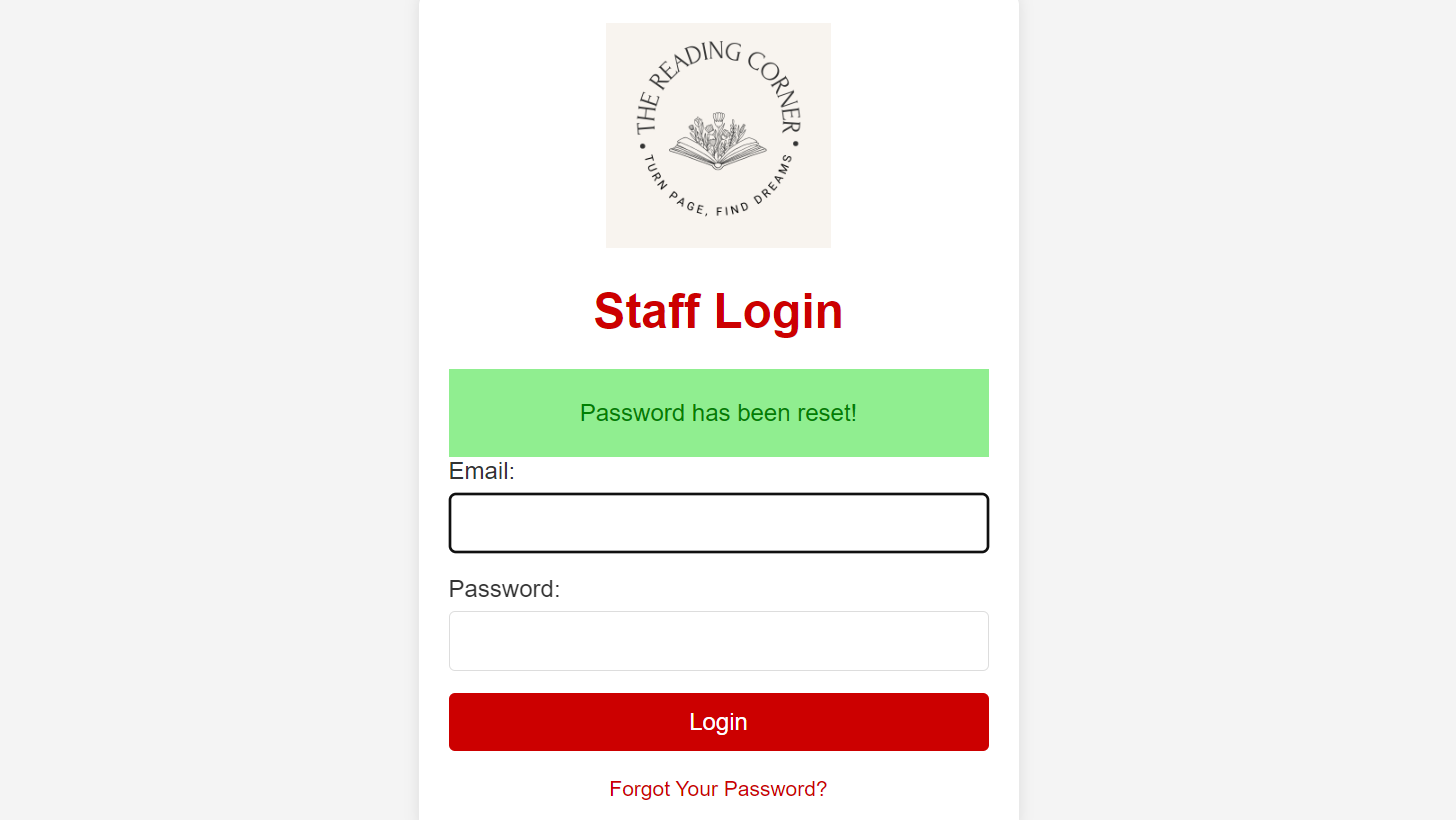
The user will see the below page when clicking the link.



When the user enters an invalid length of password, it will prompt an error message. (Refer to 5.1.4 Enforce Password Length)



If the password is successfully reset, it will redirect back to the login page and show a success message.



# 

# 8. Conclusion

The Facade Design Pattern simplifies complex systems by providing a unified interface, improving modularity and readability of code. It does this by making it more maintainable: changes can be made behind the facade without affecting the rest of the system, especially for complex authentication, password management, and data protection logic.

With secure authentication and password management, code practices are put in place to provide safety for user data. Thus, the system minimizes these types of security risks, such as unauthorized access or data breaches, by encrypting sensitive information and following best practices for session handling, password storage, and HTTP requests.

This is further enhanced by mechanisms for data protection, such as encryption on sensitive data which ensures sensitive information is only accessible or decrypted by those authorized to do so. This becomes important in securing the user credentials and other private data within the system.

XSL, XPath, and XML can be used to deal with data, providing flexible and platform-independent ways of storing, querying, and transforming data. In particular, these technologies are useful for storing structured data, such as attendance records, in a uniform and queryable format.

Finally, SendGrid web service provides reliable and scalable e-mail delivery; most importantly, password reset links. This cloud service provides an easy way to handle e-mail infrastructure; hence, it ensures that users receive important communication without dealing with e-mail servers in the organization. Most importantly, web services will further facilitate cross-language integrations where systems written in different languages can communicate effectively.