Module Code	**:**	CT050-3-2-WAPP
Intake Code	**:**	APU-APD2F2211IT
Lecturer Name	**:**	Dr. Lai Ngan Kuen
Data Assigned	**:**	17th May 2023
Date Completed	**:**	28th July 2023
Group No.	**:**	1
Assignment Title	**:**	Kohedemy Web Application

Student ID	**Student Name**
TP067281	Goh Min Xuan
TP066039	Lam Wai Yan
TP067160	Melissa Khoo Yen Yin
TP059963	Yip Zi Xian

Table of Contents

1.0 Introduction

Figure 1: Company Logo

Kohedemy, an online learning platform created by a web application development team, provides students, coffee lover, or coffee aficionado of all academic levels access to digital learning materials. Registered users have extra features available such as tracking their progress if they create an account by setting up and logging in regularly to Kohedemy.

Although still under development, Kohedemy shows great promise as an educational tool for barista students worldwide. The team is eager to introduce it globally while adding new features and content. Students looking to gain knowledge in coffee brewing might benefit from Kohedemy. The website offers educational resources and is simple to navigate; for more information, do visit their website.

1.1 Objectives

Kohedemy's logo, depicting a coffee cup as well as the coffee beans, represents the knowledge and capabilities that students can acquire by studying through its website. Barista instruction is highlighted through phrases like "Learn, Brew, Barista Academy".

This website’s objective is to provide valuable information and techniques to enhance their coffee brewing skills. To achieve this, the website may include information regarding their various processes, amounts of coffee or water required and total brewing times.

Giving advice about how to fix common coffee brewing issues such as how to remedy sour, bitter or weak coffee, or offer recipes for different coffee beverages, including espresso, cappuccino and latte recipes as well as others that may be well-known among coffee drinkers would be able to fulfil the objectives as well.

1.2 Scopes

One of the scopes of this platform is that users can create new accounts using their email address, username and password to open new accounts. Registered users can then log into their accounts by providing their login info on the login screen. Logged-in users can access their personal profile details via the user profile page, including email and username information. They also have the ability to update or change these on their personal profiles.

On their course page, users can see which courses they are registered for and their progress through them. There are three categories for showing registered courses: "Enrolled," "Assessment," and "Completed." Users who register courses that require assessments can take them by clicking on the "Assessment" button on each individual course page.

Users may access and review courses they've taken previously on individual course pages as well. On each individual course page, these completed courses appear under "Completed". In addition, there is an "Log Out" button available within the navigation bar which allows them to quickly log out and thus clear their cookies and session.

On the other hand, an administrator with the username "Kohemin" can access their administrative dashboard page after signing in. Administrators can effectively oversee courses, adding new ones or altering data about existing courses. Administrators also have access to additional features and capabilities not accessible to regular users, including managing general site content, previews of courses and data analytics.

1.3 Project Plan / Schedule

1.3.1 Gantt Chart

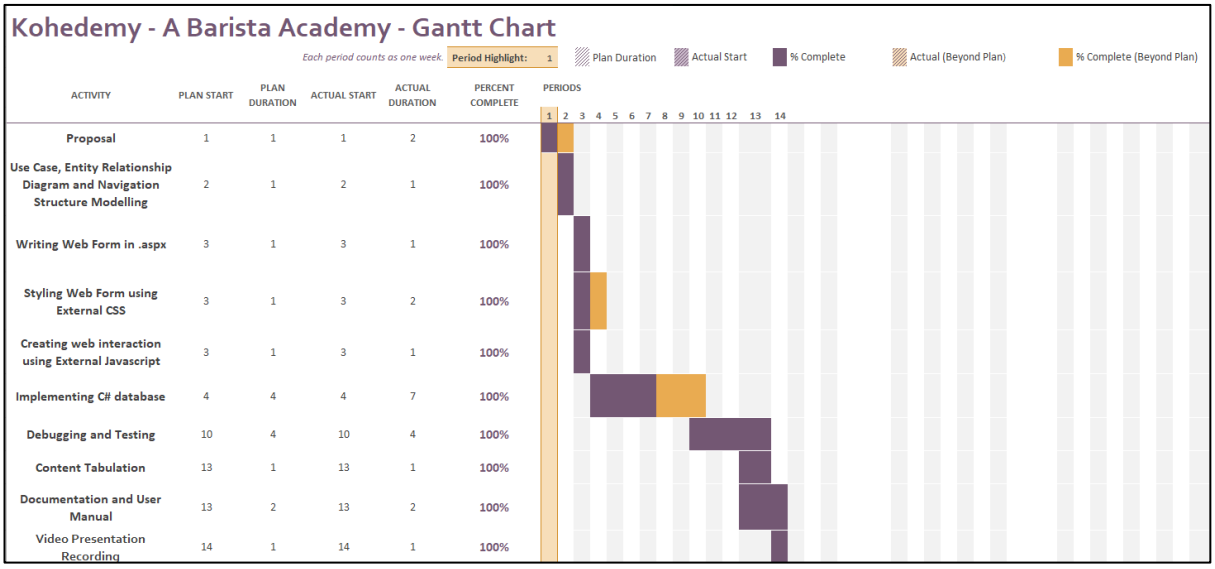


Figure 2: Gantt Chart

1.3.2 Workload Matrix

Table 1: Workload Matrix

Name / Task	Goh Min Xuan	Lam Wai Yan	Melissa Khoo Yen Yin	Yip Zi Xian
Main Tasks	Course CRUD & Web Design	Login, Register & Web Design Validation	User & Admin CRUD	Tabulation data
100%	25%	25%	25%	25%
4 Dynamic Pages	View Course Page (User side)	Admin Main Page (Admin side)	User Main Page (Admin side)	Login and Register, Course Main Page (User Side)
	25%	25%	25%	25%
4 Static Pages	25%	25%	25%	25%
CSS	25%	25%	25%	25%
SIGN OFF	GOH MIN XUAN	LAM WAI YAN	MELISSA KHOO YEN YIN	YIP ZI XIAN

2.0 Requirement Specification

2.1 Audience Modelling

People who share a passion for coffee make up the target audience. Coffee enthusiasts appreciate all aspects of it - history, brewing techniques, flavour profiles and cultural significance among others - not just casual consumers who may drink it occasionally; rather they seek out unique coffee beans, origins and experiences while being knowledgeable in roasting and brewing methods.

2.2 Use Case Diagram (Admin, Registered User and Non-Registered User)

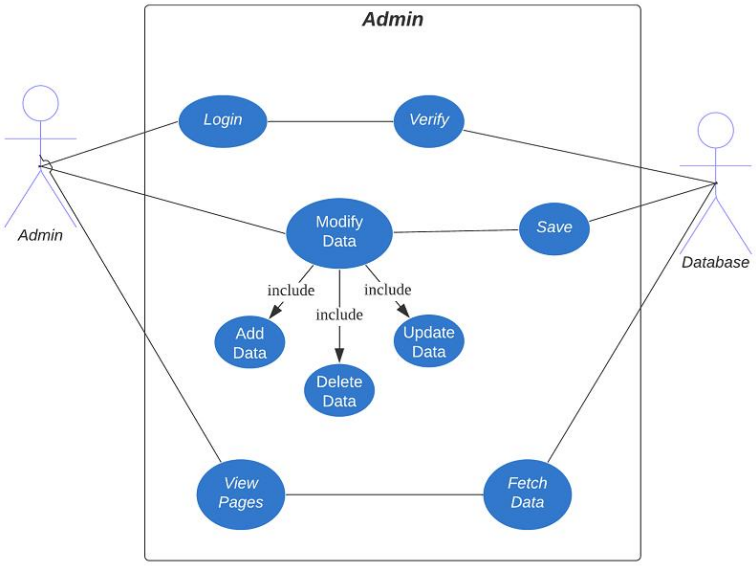


Figure 3: Admin

Based on the use case diagram above, there are 2 actors which is the administrator and the database. The admin can log onto the system. The system verification occurs for identification verification once admin has logged in. Admin can modify existing data while adding, updating, or purging other records as required by their account. Admin can browse pages within the system as well. All the data is pulled from the database and saved into the database.

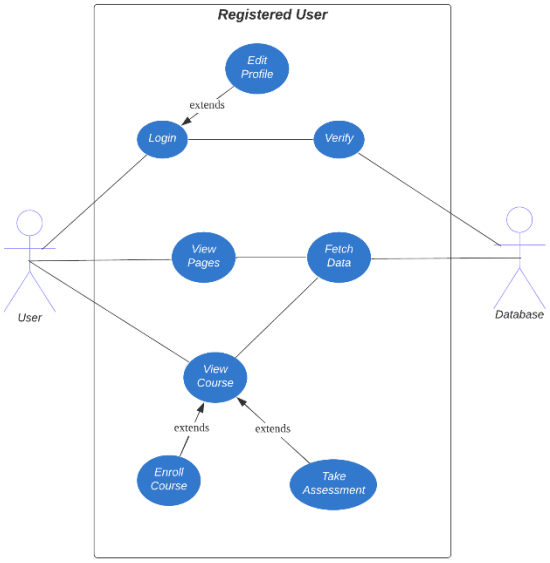


Figure 4: Registered User

Based on the figure above, the actors for the use case diagram are registered user and database. The user has to login to the system and the user's identity is verified by the system as well. The users can edit their profile once they have logged into the website. The user can also navigate between several pages while the system retrieves information from its database. Moreover, user may explore details regarding specific courses. This includes by enrolling in one or more courses and taking an exam for one or more courses as outlined. This process also pulls data from the database as well.

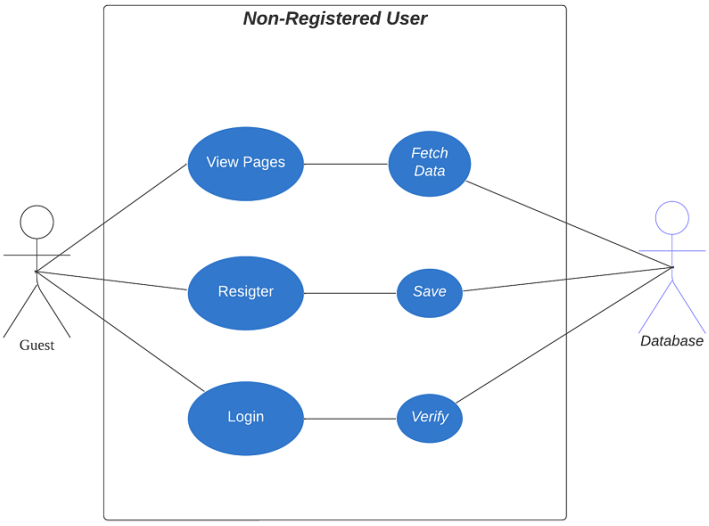


Figure 5: Non-Registered User

Figure X shows a use case diagram which actors are guest user and database. A guest user can look through the website, from its home page, course pages etc. The information on the website

will be retrieved from the database. Guest users is able to open an account and take advantage of extra services, such as enrol into a particular course, taking assessments and others. The registered data will be saved into the database, and it will be used to verify the user's identity once the guest user tries to login into its newly created account.

2.3 Flow Chart

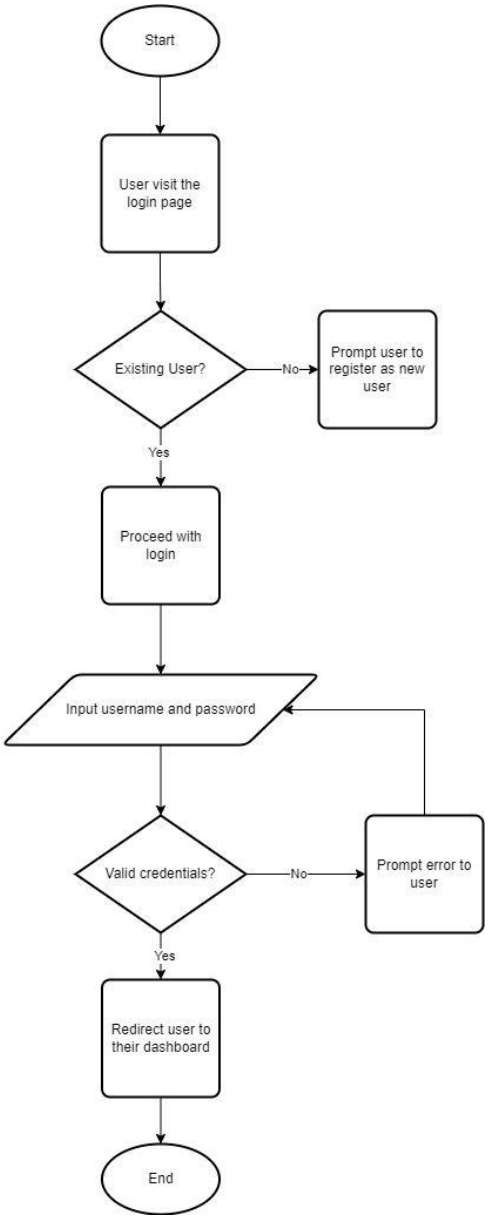


Figure 6: Login

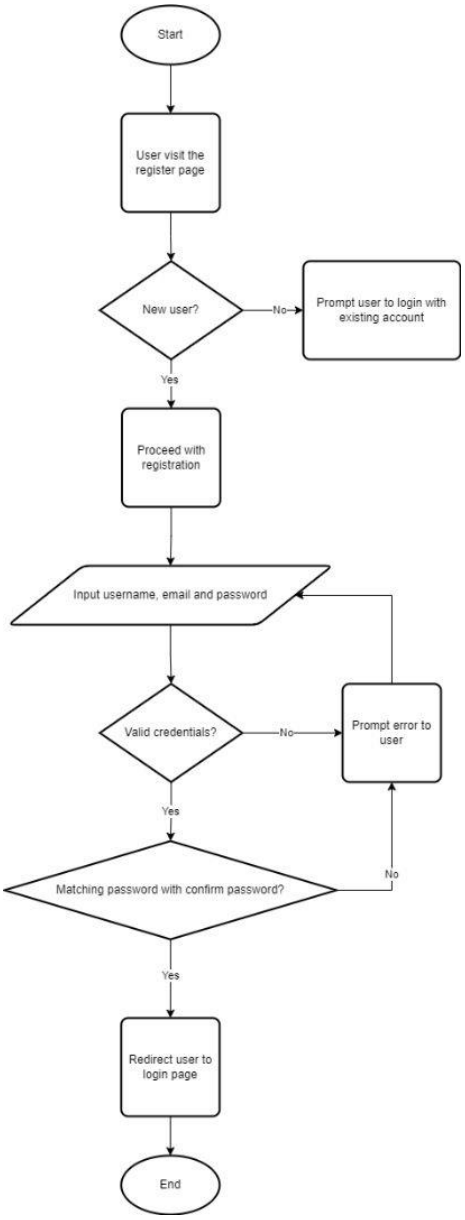


Figure 7: Register

Here are the two primary tasks Kohedemy users perform when visiting its website: logging in and registering.

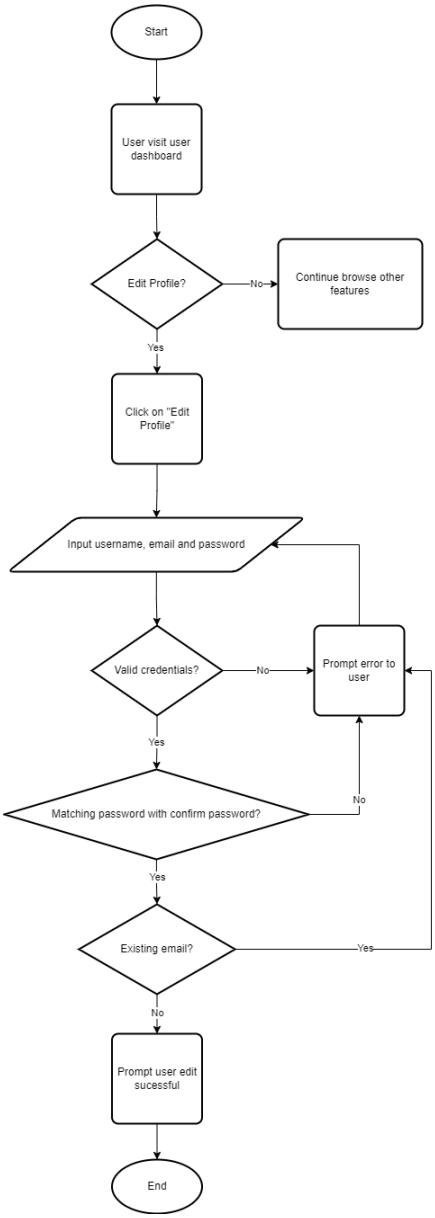


Figure 8: Edit Profile

The diagram above shows the flowchart of users editing their profile.

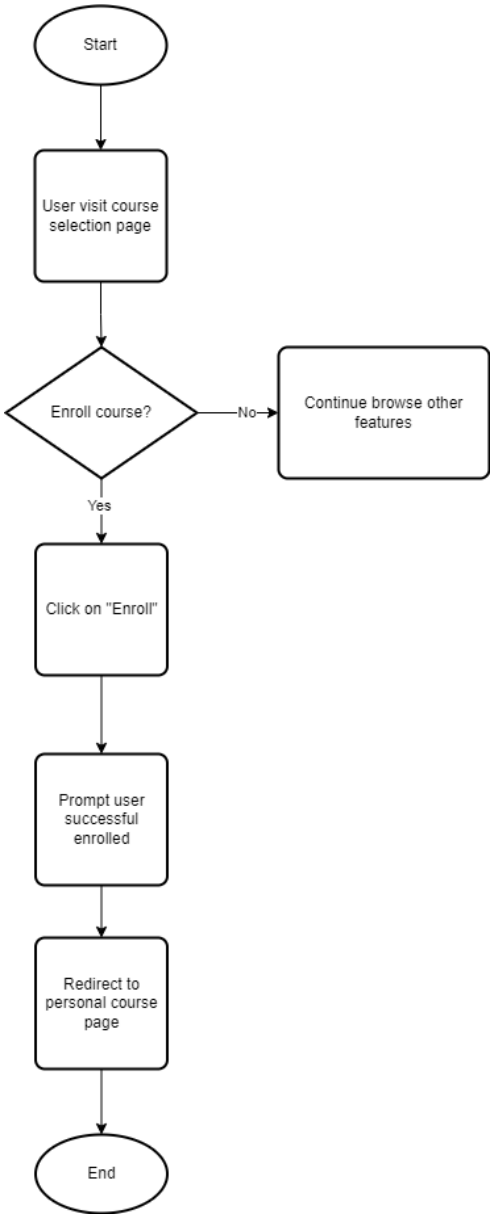


Figure 9: Enroll Course

This flowchart shows how users can enrol into a particular course that they are interested in.

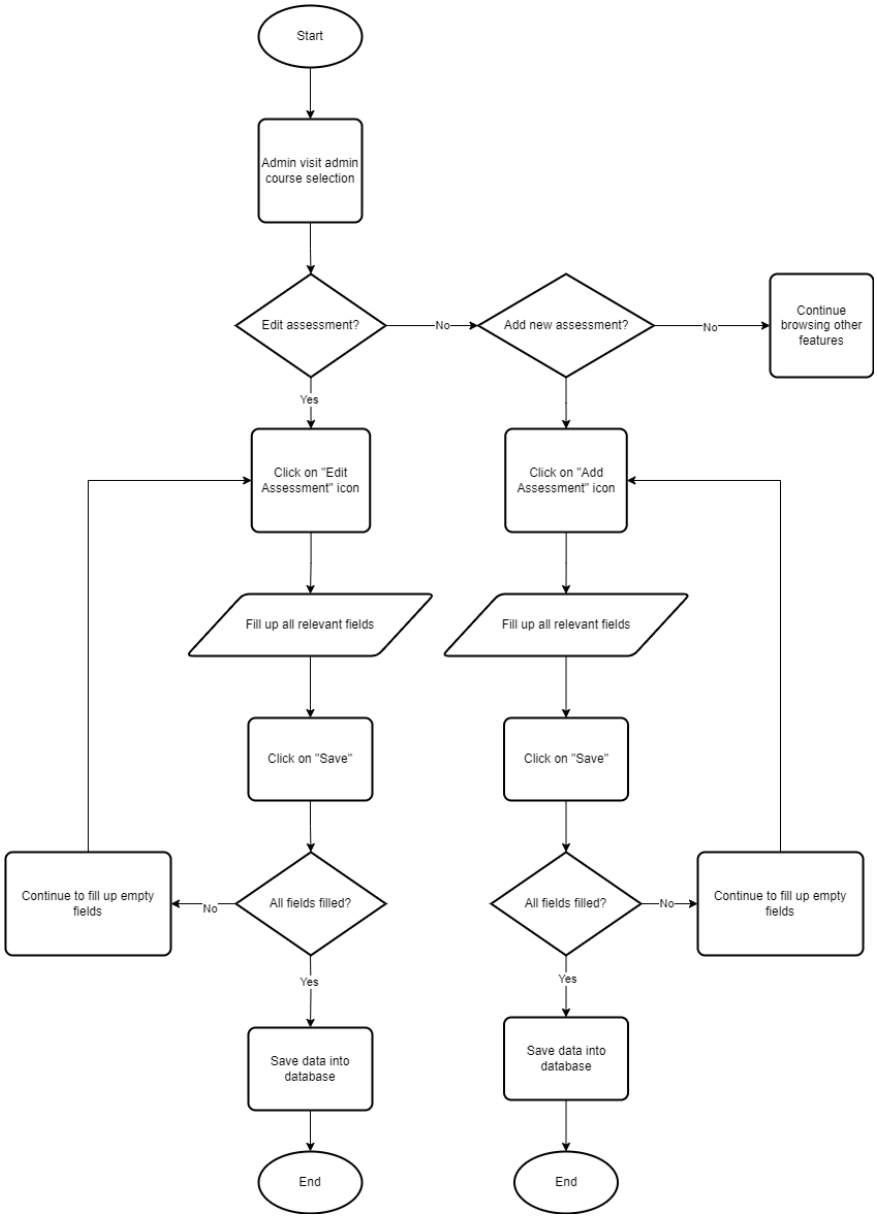


Figure 10:Manage Assessment (admin)

The process of the administrator managing the course assessment is shown on the diagram above.

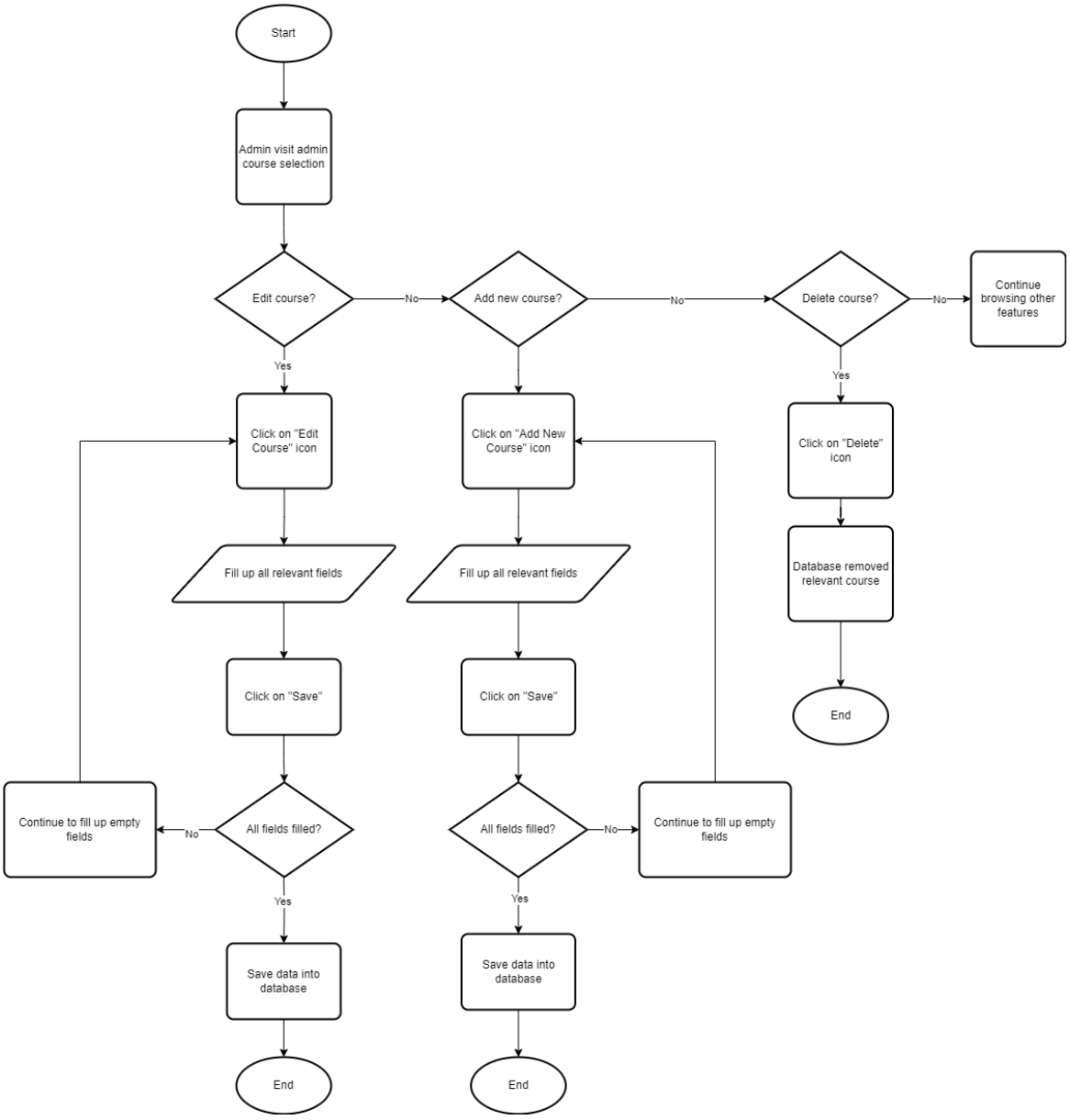


Figure 11: Manage Course (admin)

The flowchart above illustrates how an administrator manages a course.

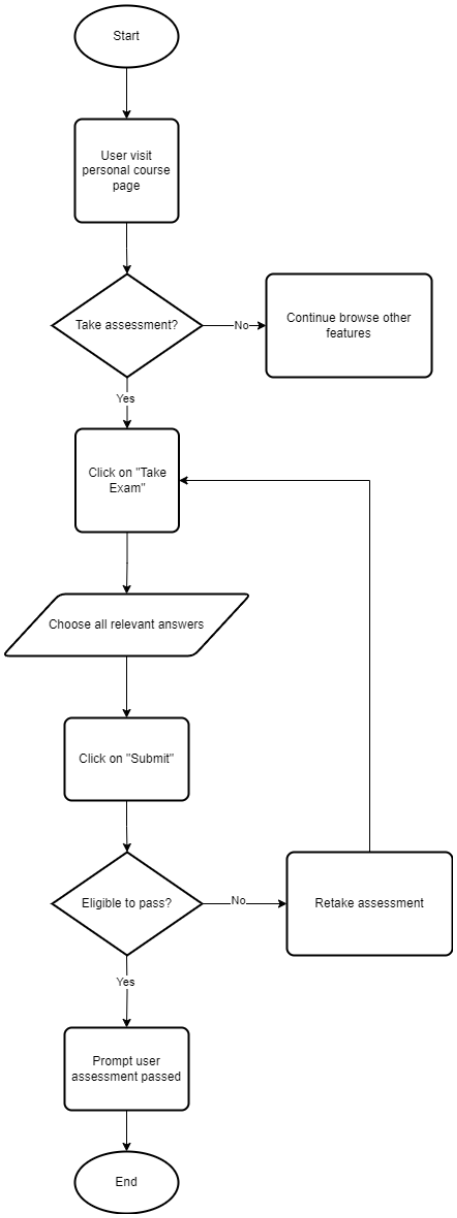


Figure 12: Take Assessment

The flowchart above outlines how users can successfully complete course-related assessments

3.0 Design and Modeling

3.1 Entity Relationship Diagram (ERD)

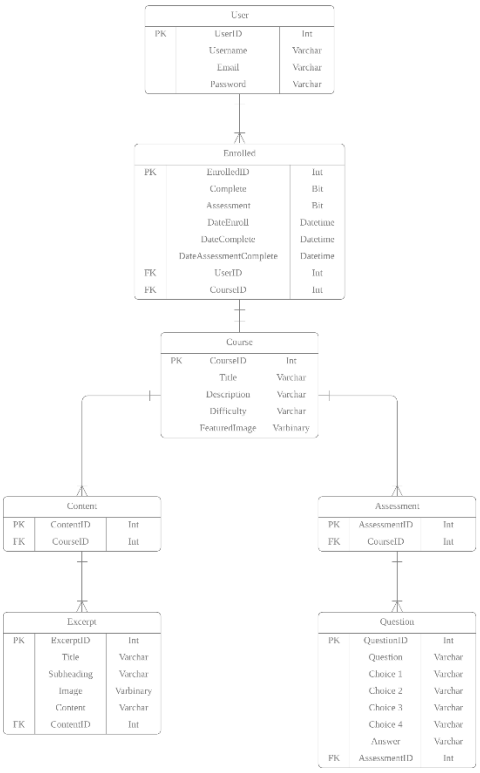


Figure 13: Entity Relationship Diagram for Kohedemy

Based on the ERD Diagram above, there are 7 different entities which are user, enrolled, course, content, excerpt, assessment, and question. User refer to those who enrolled into a course, while course are made up by content and assessment, which are also made up by excerpt and question respectively. Excerpt is a section of text and image designed to convey information on coffee brewing while question is used to assess a student's knowledge of coffee brewing.

One user can enrolled to many course while one enrolled data belongs to one user only. Meanwhile, one enrolled data refers to one course and one course can tie to enrolled course. Moving on, one course can have many contents and assessment while one content and one assessment is belonged to one course. Other than that, one content and one assessment can have many excerpts and questions respectively, while one excerpt belongs to one content and one question belongs to one assessment.

3.2 Wireframe

Home Page

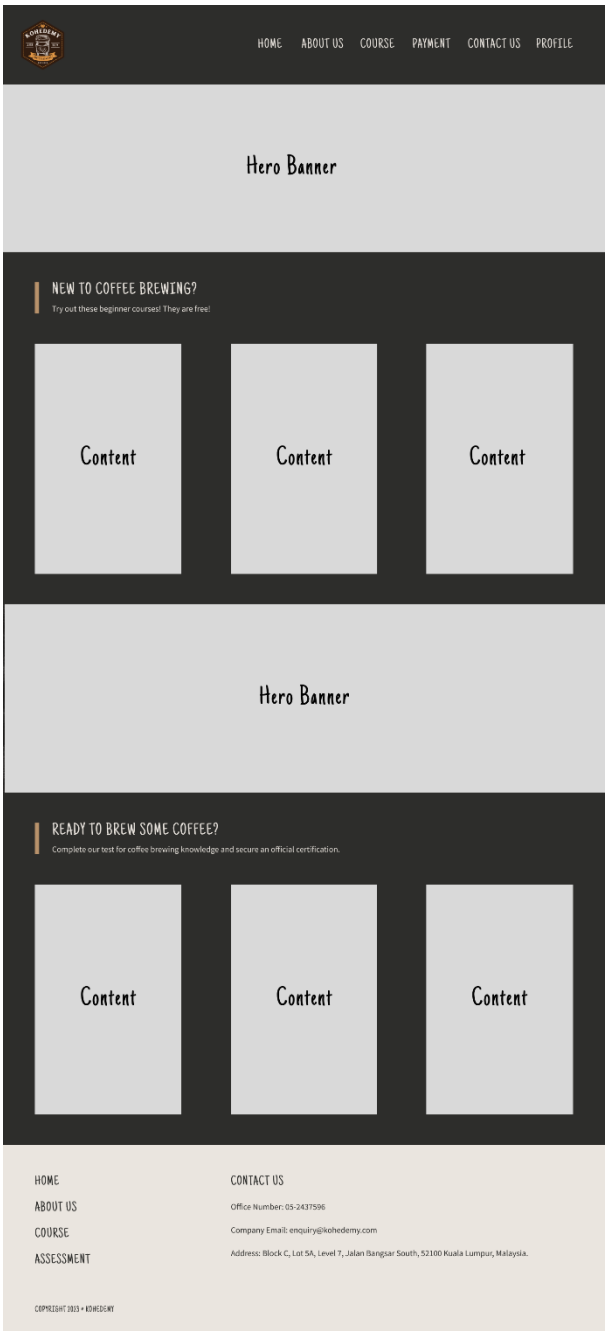


Figure 14: Wireframe for Home Page

In the home page, users will be presented with a header of the institution logo and navigation bar. While scrolling down to the web page, the user also can see two banner is being displayed and there will be three boxes which contains content for the user.

Contact Us

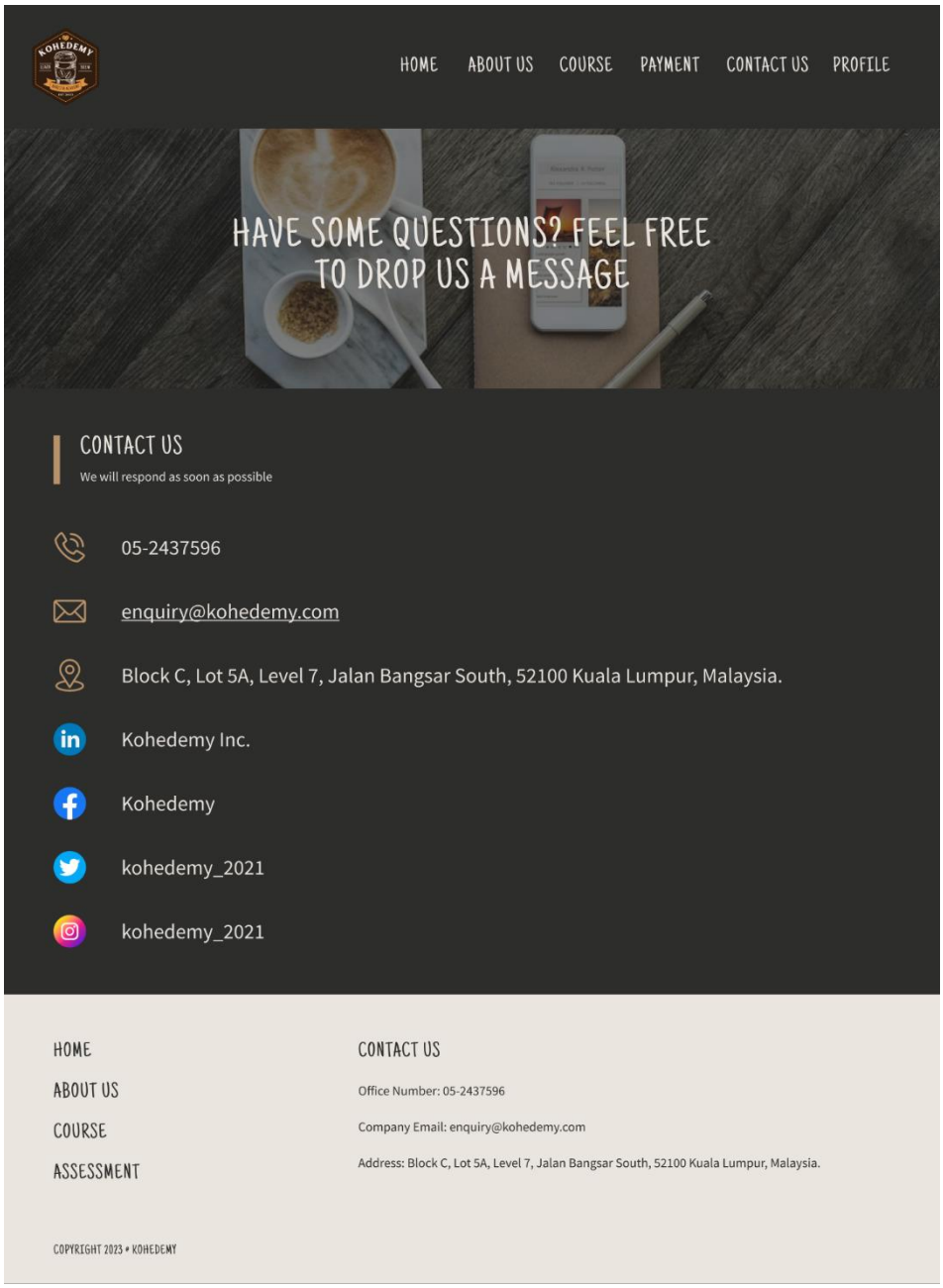


Figure 15: Wireframe for Contact Us

When user click on to the Contact Us, it will be presented with a header of the institution logo and name, followed by a contact form with phone number, email address, address, and the social media platforms.

About Us

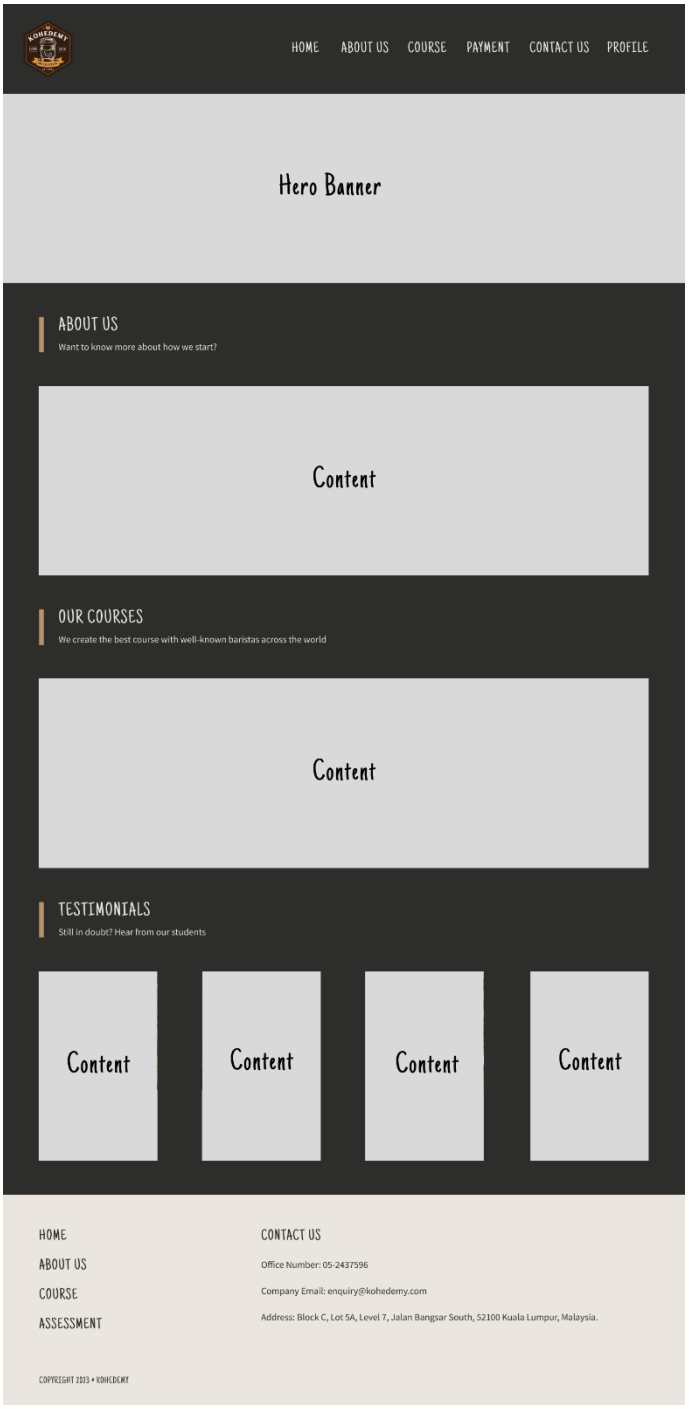


Figure 16: Wireframe for About Us

In this page, the user will be presented with the institution's background and information about this course. The web page divided it into three different section such as about us, our courses, and testimonials

Login Page

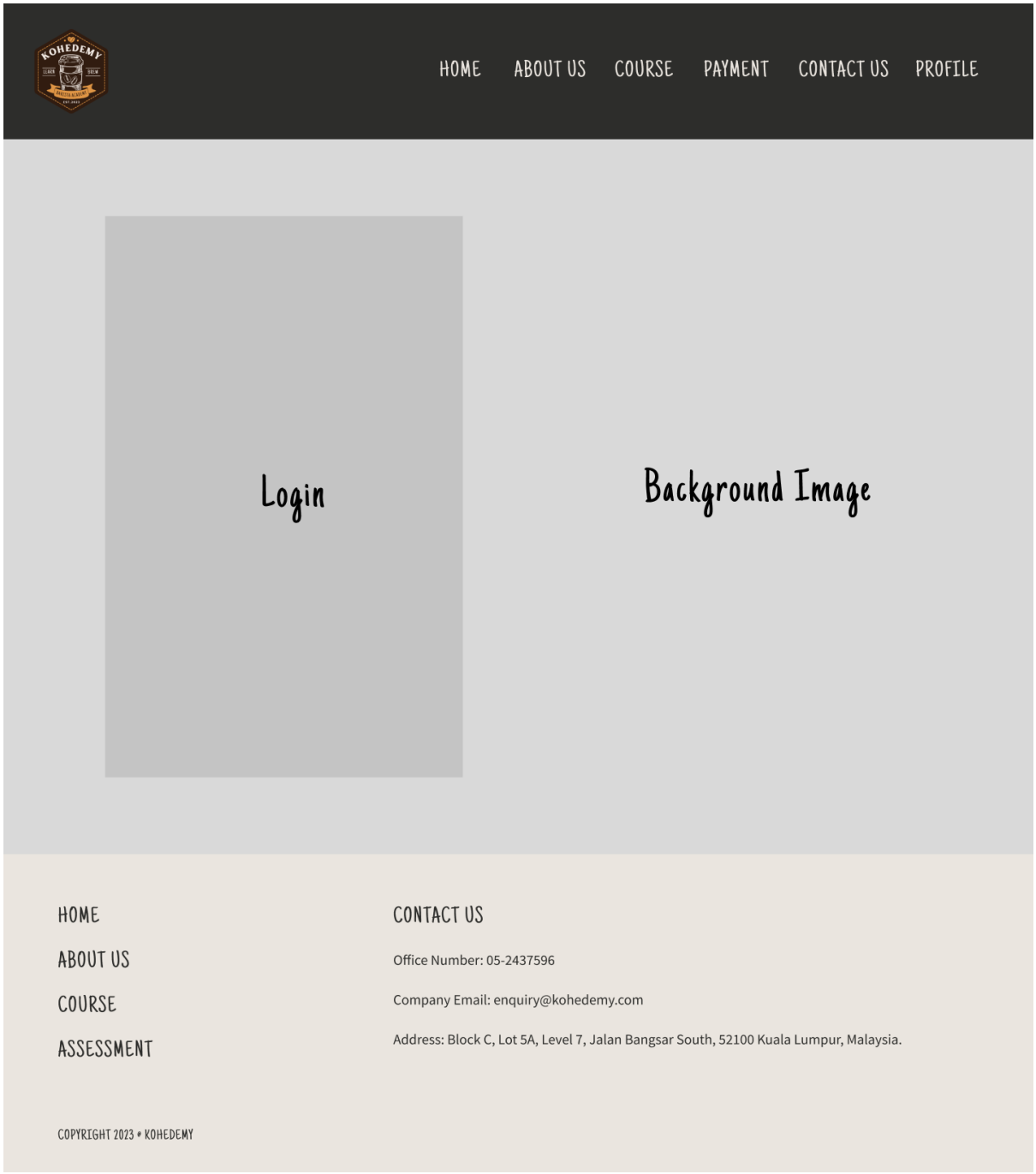


Figure 17: Wireframe for Login Page

In login page, users will be presented with a container to input their information to verify them as an existing user. Behind the container will be a supporting background image to improve the visual experience.

Register

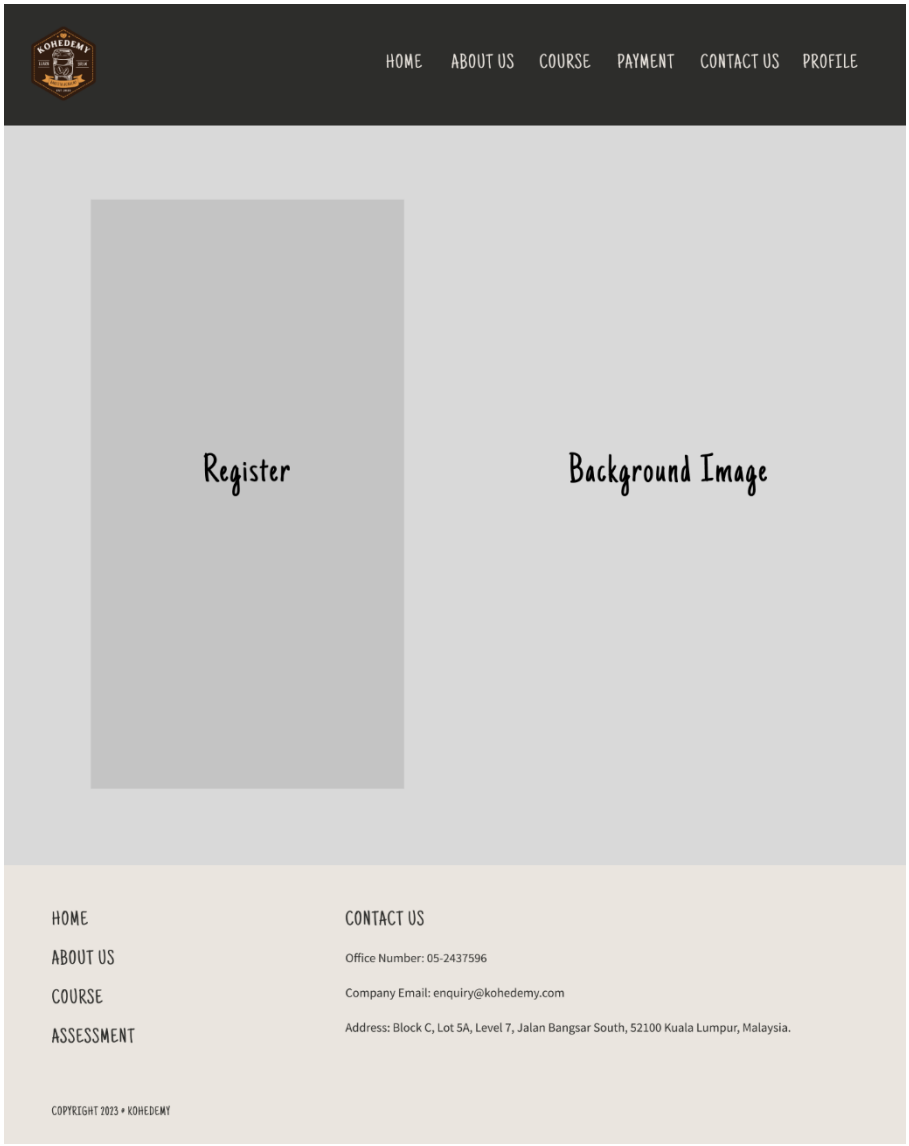


Figure 18: Wireframe for Register

In register page, users will be presented with a container to input their information to input their email address, username and password. A supportive backdrop image will be placed behind the container to enhance the visual experience

User Profile

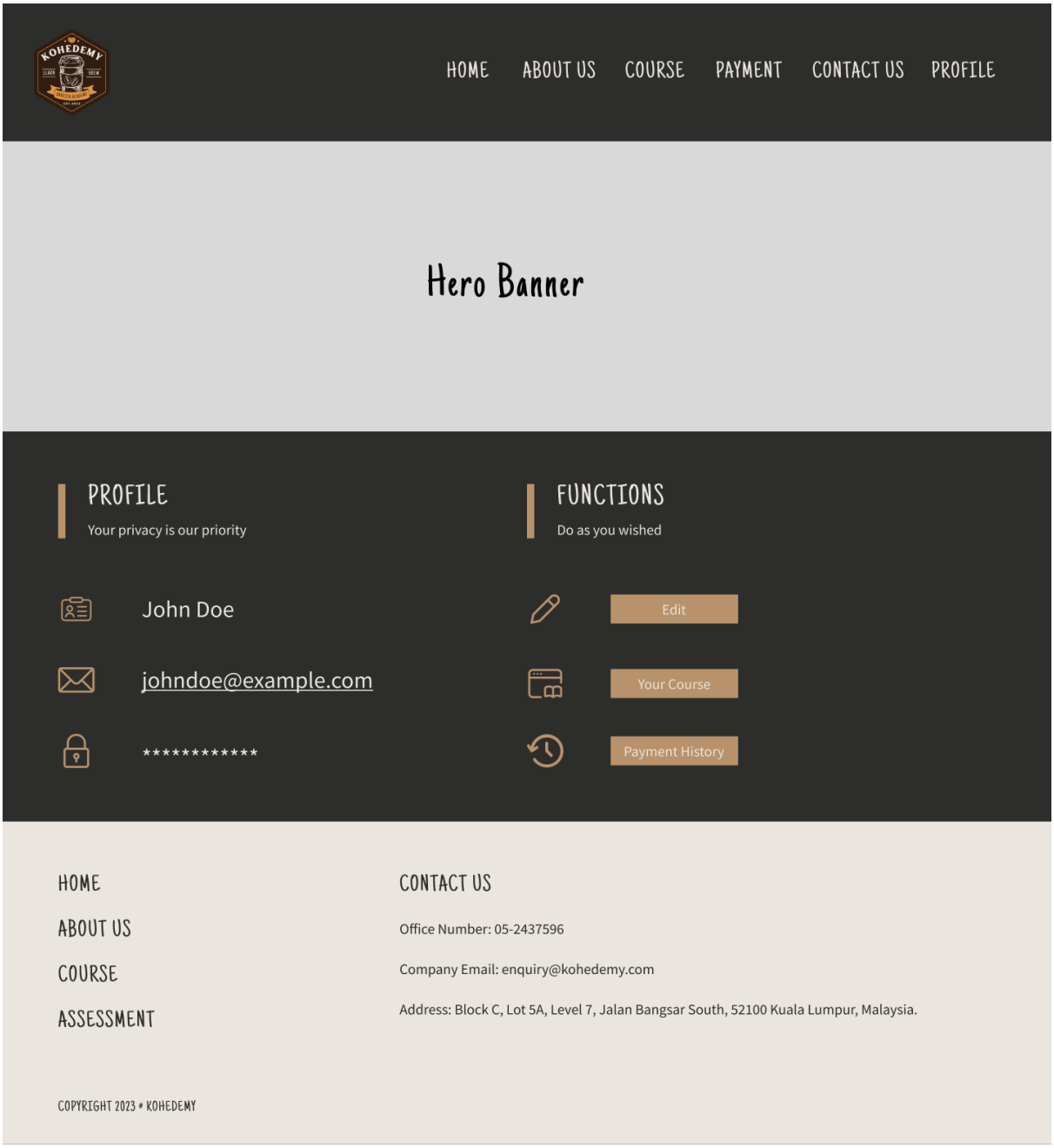


Figure 19: Wireframe for User Profile

In the user profile, the user will be presented with their personal details and there will be a section where the user can edit their account and view the courses

Edit Profile (Pop up)

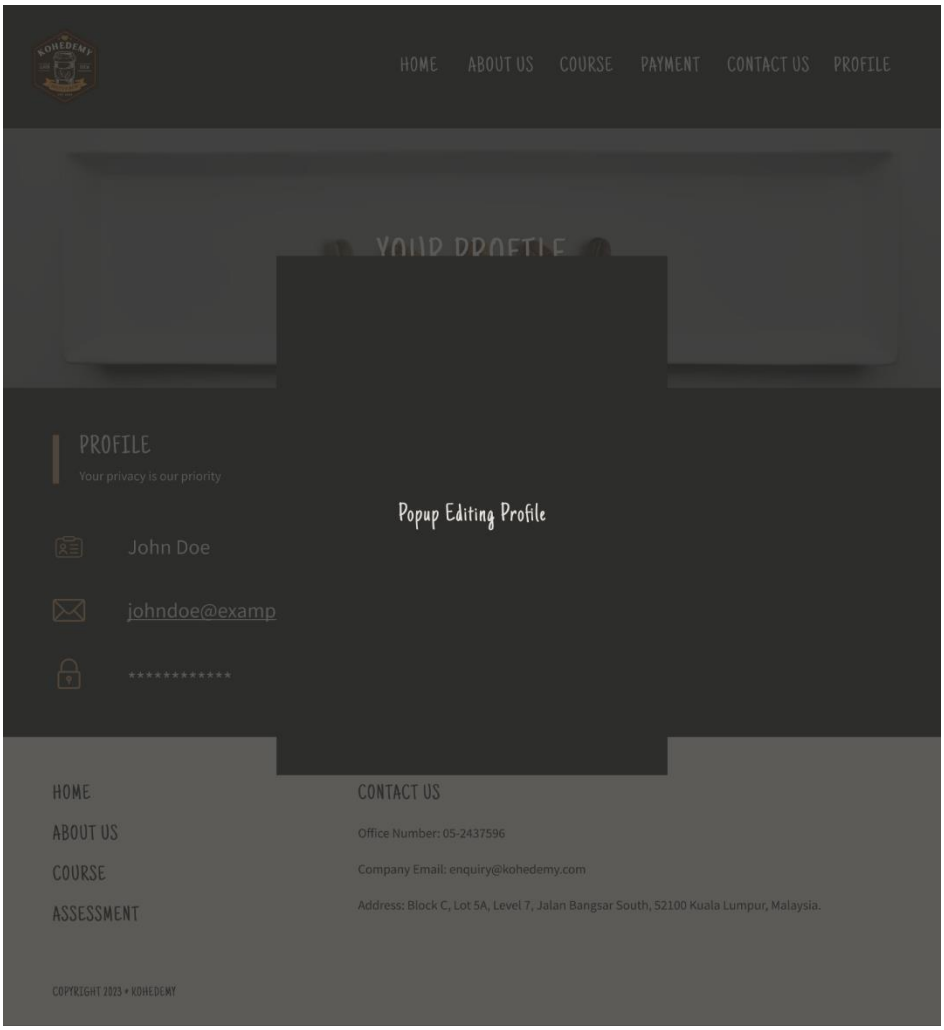


Figure 20: Wireframe for Edit Profile (pop-up)

The image depicted the edit profile, when user click on to the “Edit” button, there will be a pop-up screen for the user to change their details in the container provide as seen above the image.

Personal Course

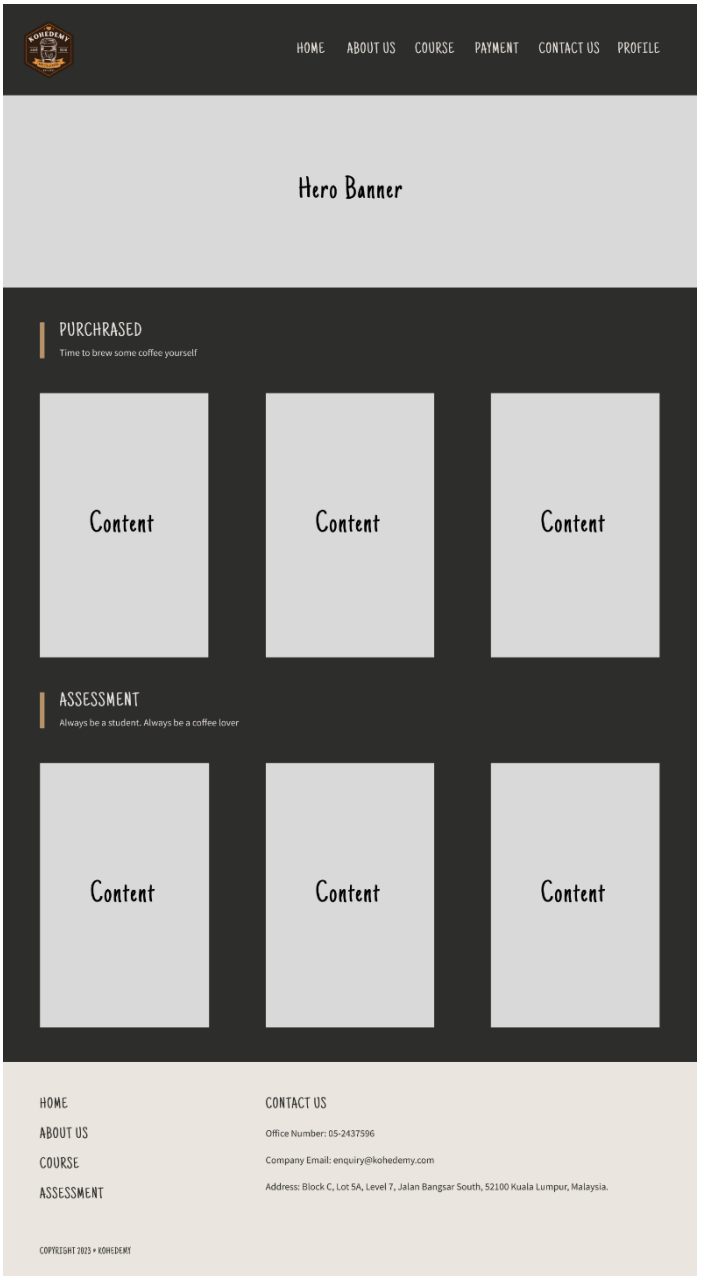


Figure 21: Wireframe for Personal Course

In personal course, the user will be able to view and track their progression in taking the courses. There will be divide it into three different sections such as enrolled, assessment, and completed. The content will contains the courses that the user attended and complete the assessment.

Course Selection Page

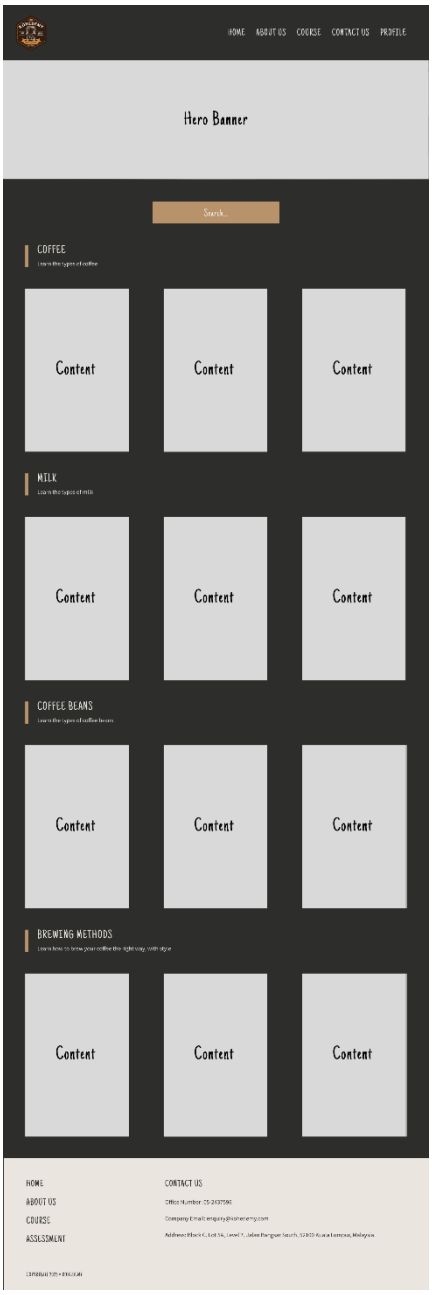


Figure 22: Wireframe for Course Selection Page

In this page, the header will display “Course selection” while the body section displayed few sections about the courses available to the user. Each section has a heading and subheading provided with the content that allow the user to select the courses that the user wish to attempt.

Single Course

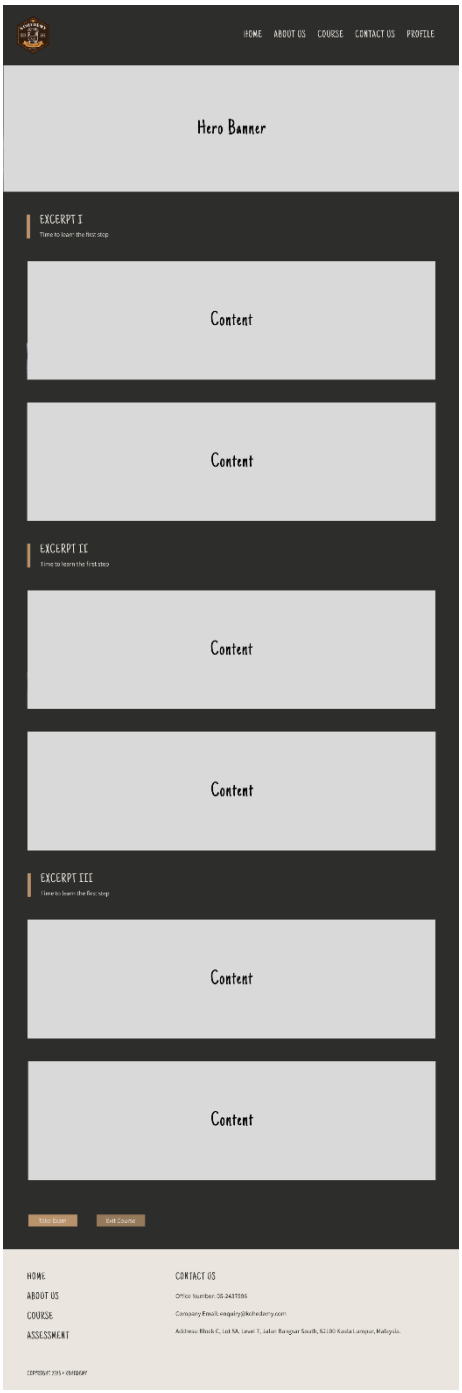


Figure 23: Wireframe for Single Course

In this single course page, the user will be presented with the course that the user enrolled into. The body section will be displayed the learning materials that the institution provided for the user to self-reading. Once completed the course, the body section below provided two push button which is “Take Exam” or “Exit Course”.

Course Assessment

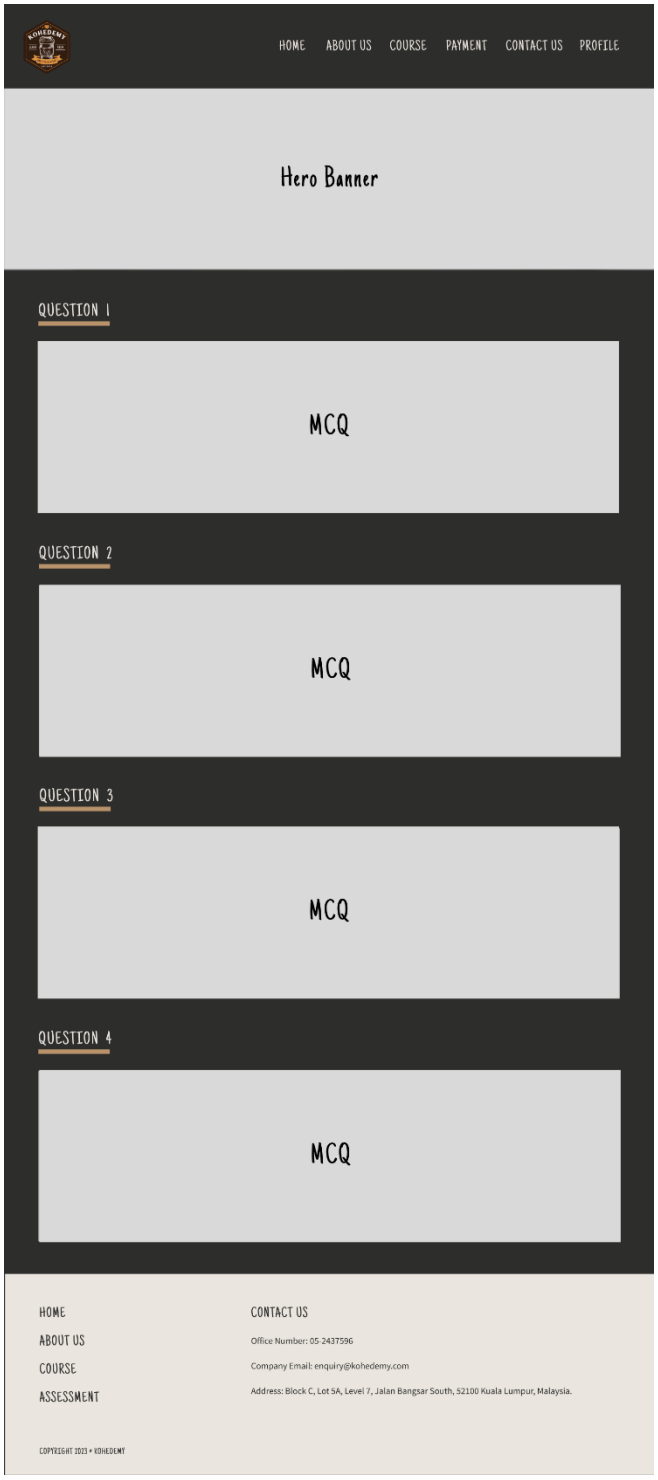


Figure 24: Wireframe for Course Assessment

Upon completion of the course materials, the image above illustrates the user course assessment. In this section, there will be several questions provided to the user to answer in order for them to complete the entire course

Course Selection Page (Admin)

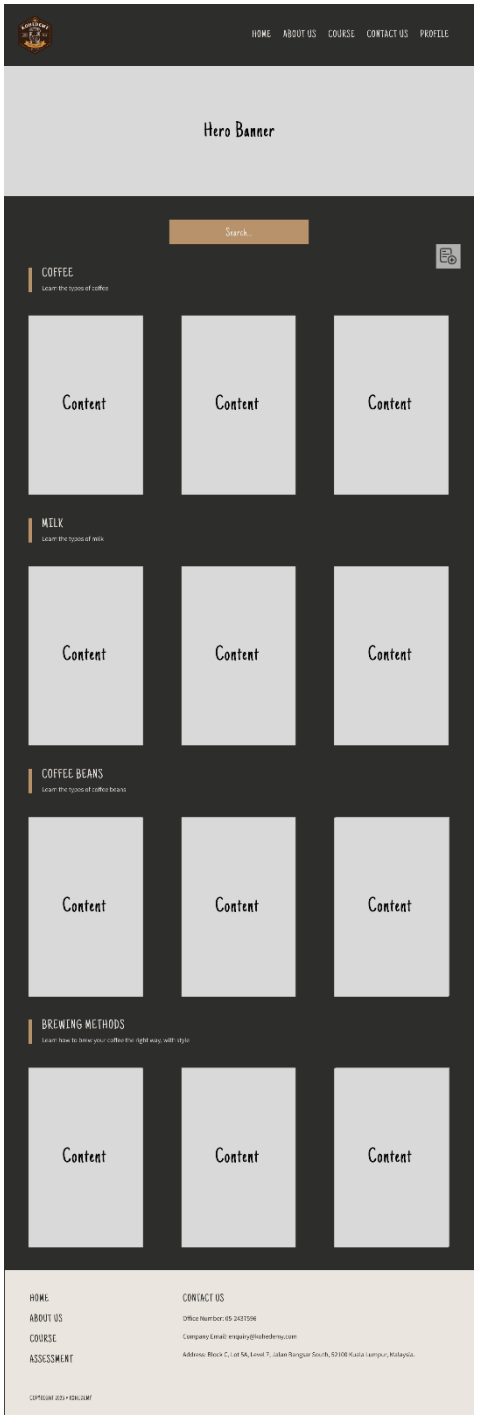


Figure 25: Wireframe for Course Selection Page (Admin)

This is admin version of course selection page, the header will display as “Course Management” while the body section displayed few sections about the courses available to the user. Each section has a heading and subheading provided with the content that allow the user to select the courses that the user wish to attempt. In addition, the admin enables to edit, create and delete the courses.

Single Course (Admin)

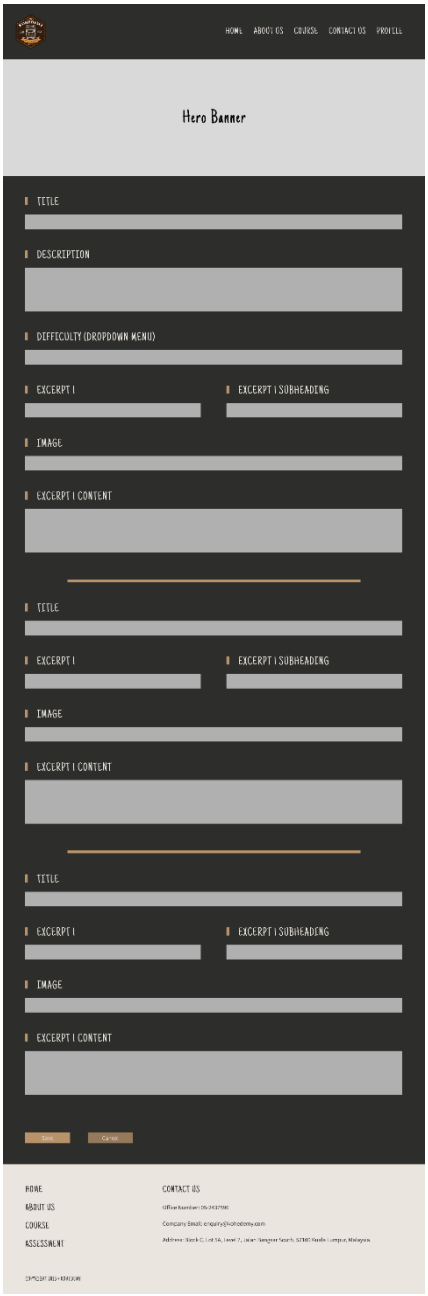


Figure 26: Wireframe for Single Course (Admin)

In single course, the admin will be presented with a hero banner for create or edit the course. The course content section is where the admin will add text, images, and other media that make up the course.

Single Assessment (Admin)

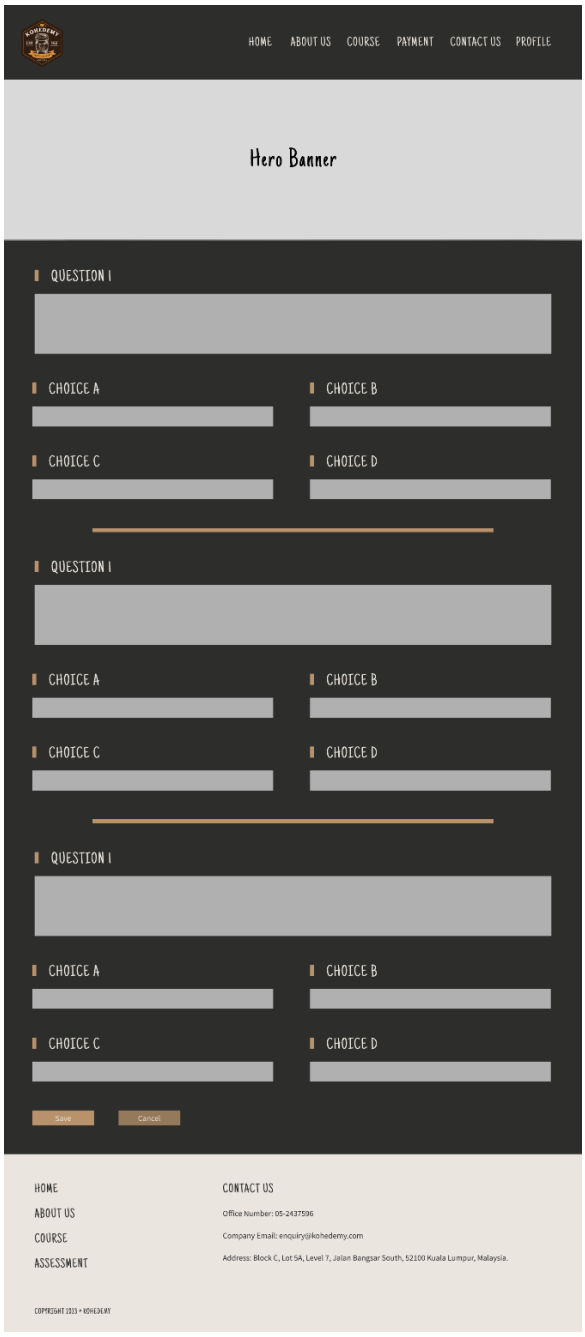


Figure 27: Wireframe for Single Assessment (Admin)

In this section, the admin is required to create and edit the course assessment for the user. For every question, there will be heading and subheading for the multiple choices. The admin is required to input the question and answers in the text field. Besides that, there are two push buttons for admin to select either save or cancel the assessment page.

Admin Dashboard - Extra

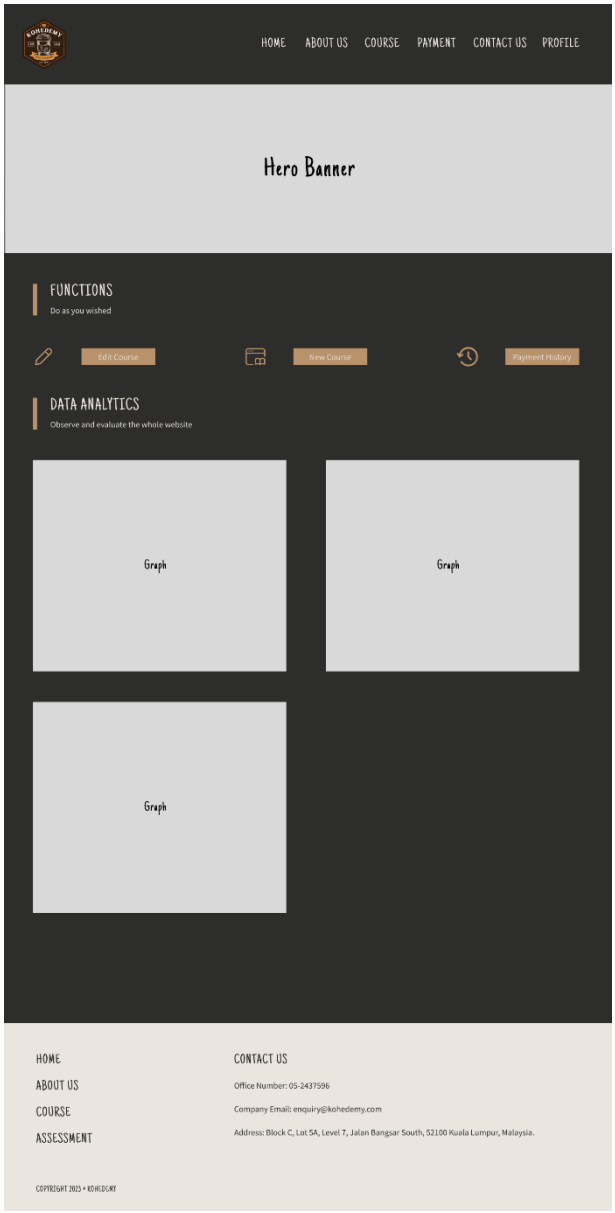


Figure 28: Wireframe for Admin Dashboard

In the admin dashboard, the admin will be presented with two different section which is the functions and data analytics under the body of the wireframe. In the function section, the admin can edit and create courses, while data analytics for each containers display the graphs and analysis of the number of users sign up and user attempted the assessments.

3.3 Website Navigational Structure

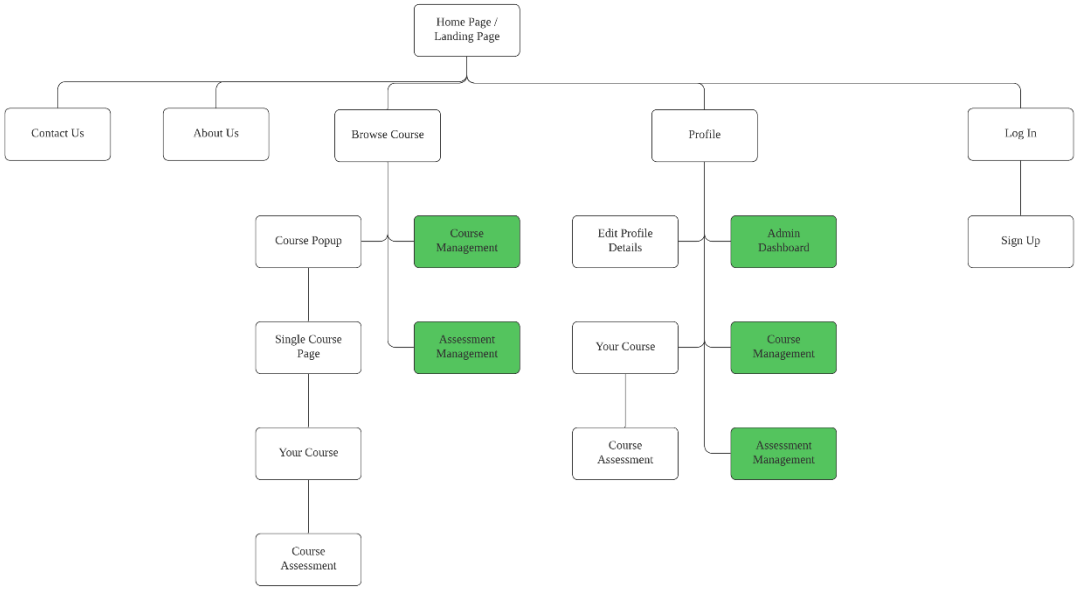


Figure 29: Website Navigational Structure of Kohedemy

According to the website navigational structure above, the home page or landing page of Kohedemy serve as the starting point. Users can navigate from the home page by clicking any of the following links: “Contact us”, “About us”, “Browse Course”, “Profile”, or “Log In.” They will take them directly to other sections. The “Contact us", “About us", and “Browse Course” pages are to provide users (both admin and user) methods to contact the team, details about the website, as well as browsing for courses that are offered on the website.

Under the page of “Browse Course” and “Profile”, it is split into two different categories one for registered users, and one for administrators. The sub-pages of “Browse Course” include Course Management which is typically used by administrators to oversee courses; Assessment Management meanwhile allows assessors to manage assessments. For user, if necessary, pop-up windows containing course-related information may appear, and the Single Course Page is likely to provide all the information about an individual course. Meanwhile, Your Course are to provide information about the course that the user is registered to take; Exams related to the enrolled course are under the Course Assessment Page.

Next, the “Profile” page contain a user profile data, and users can update their profile details using the Edit Profile Details feature. On the other hand, administrators profile page are presented as Admin Dashboard page. The remaining pages Course Management, Assessment Management, Your Course, and Course Assessment are the same as the one which are under the “Browse Course” page. Lastly, user can log in to the website once they have registered an account, else, they can register one on the Sign-Up page.

4.0 Implementation

4.1 Source codes

Admin Course Selection Page

```
<asp:LinkButton ID="LogOutButton" runat="server" OnClick="LogOutButton_Click">Log Out</asp:LinkButton>
```

Figure 30: server-side link button

Here, a server-side link button with the text "Log Out" is created using the *<asp:LinkButton>* and given an identification via its ID attribute. Likewise, its *runat="server"* attribute enables server-side handling. The *OnClick* attribute specifies the server-side event handler that will be invoked whenever a button is pressed and defines when this event handler should run.

```
<!--Flex Cards-->
<div class="flex-card-container">
  <asp:Repeater ID="BeginnerRepeater" runat="server">
    <ItemTemplate>
      <div class="cards" data-id="<%#Eval("CourseId") %>" data-description="<%# Eval("CourseDescription") %>" data-difficulty="<%# Eval("CourseDifficulty") %
        <div class="card-image-container">
          <img class="card-image" src="<%# Convert.ToBase64String((byte[])Eval("CourseImage")) %>" />
        </div>

        <h4><%# Eval("CourseTitle") %></h4>

        <div class="edit-delete-container">
          <button onclick="return false" class="card-button">
            <p>Preview</p>
          </button>

          <div class="inner-edit-delete">
            <asp:ImageButton CommandArgument='<%# Eval("CourseId") %>' ImageUrl="../Assets/icons/edit.png" CssClass="inner-button" ID="EditButton" runat="ser
            <asp:ImageButton CommandArgument='<%# Eval("CourseId") %>' ImageUrl="../Assets/icons/check-list.png" CssClass="inner-button" ID="AssessmentButton
            <asp:ImageButton CommandArgument='<%# Eval("CourseId") %>' ImageUrl="../Assets/icons/delete.png" CssClass="inner-button" ID="DeleteButton" runat=
          </div>
        </div>
      </div>
    </ItemTemplate>
  </asp:Repeater>
</div>
```

Figure 31: Source code for display course cards

For the purposes of displaying course cards, an *<asp:Repeater>* repeating structure has been devised. A template for every repeated item is detailed under *<ItemTemplate>*. <asp:ImageButton> provides an image button control in the template. The *"CourseId"* field from the data source is linked with the *CommandArgument* attribute. *ImageUrl* specifies the image source for the button. *CssClass* gives the button a CSS class. The rest of the ID, *runat="server"*, and the *OnClick* attribute is all the same as the above.

The "*AdminCourseSelection.aspx.cs*" code-behind file corresponds to server-side event handlers set in the *OnClick* attribute of various *<asp:>* controls. When one of their buttons is clicked, these event handlers go into action based on user interactions such as logging out, editing courses, rating courses, or deleting courses.

```
protected void Page_Load(object sender, EventArgs e)
{
  if (Session["Username"] as string == "Kohemin")
  {
    try
    {
      SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
      con.Open();

      // For Beginner Course
      string beginnerQuery = "SELECT * from [Course] WHERE Difficulty = 'Beginner'";
      SqlCommand beginnerCmd = new SqlCommand(beginnerQuery, con);

      SqlDataReader sdr = beginnerCmd.ExecuteReader();

      List<CourseData> beginnerCourses = new List<CourseData>();

      while (sdr.Read())
      {
        CourseData beginnerCourse = new CourseData
        {
          CourseId = Convert.ToInt32(sdr["CourseID"]),
          CourseTitle = sdr["Title"].ToString(),
          CourseDescription = sdr["Description"].ToString(),
          CourseDifficulty = sdr["Difficulty"].ToString(),
          CourseImage = (byte[])sdr["FeaturedImage"]
        };

        beginnerCourses.Add(beginnerCourse);
      }

      sdr.Close();
```

Figure 32: Source code for loading page

Before loading any page, our code checks Session*["Username"]* value to determine if the user possesses necessary authorization before connecting to the database and retrieving course information for four levels of difficulty: Beginner, Intermediate, Advanced and Masterclass using SQL queries. Afterwards, this data is saved into various lists - *beginnerCourses*, *intermediateCourses* etc - providing essential data required for course card generation on our website.

```
protected void EditButton_Click(object sender, System.Web.UI.ImageClickEventArgs e)
{
  ImageButton editButton = (ImageButton)sender;
  string courseId = editButton.CommandArgument;

  try
  {
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
    con.Open();

    string excerptQuery = @"
                          SELECT e.ExcerptID, ct.ContentID, c.CourseID FROM [Excerpt] AS e
                          INNER JOIN [Content] AS ct ON e.ContentID = ct.ContentID
                          INNER JOIN [Course] AS c ON ct.CourseID = c.CourseID
                          WHERE c.CourseID = @CourseID
                          ";
    SqlCommand excerptCmd = new SqlCommand(excerptQuery, con);
    excerptCmd.Parameters.AddWithValue("@CourseID", courseId);

    SqlDataReader sdr = excerptCmd.ExecuteReader();

    StringBuilder sb = new StringBuilder("CreateCourse.aspx?CourseId=" + courseId);
    int count = 1;

    while (sdr.Read())
    {
      int excerptId = (int)sdr["ExcerptID"];
      sb.Append("&ExcerptId" + count + "=" + excerptId);
      count++;
    }

    Response.Redirect(sb.ToString());

    sdr.Close();

    con.Close();
  }
  catch (Exception ex)
  {
    Debug.WriteLine(ex.Message);
  }
}
```

Figure 33: Source code for Edit course

The code also accommodates user interactions on the page, recording course ID and retrieving excerpt data when an "Edit" button on a course card is clicked, to create a URL leading directly to an editing page with all necessary data already filled out for them.

```
protected void AssessmentButton_Click(object sender, System.Web.UI.ImageClickEventArgs e)
{
  ImageButton assessmentButton = (ImageButton)sender;
  string courseId = assessmentButton.CommandArgument;

  try
  {
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
    con.Open();

    string checkQuestion = "SELECT COUNT(*) FROM [Assessment] WHERE CourseID = @CourseID";
    SqlCommand cmdCheck = new SqlCommand(checkQuestion, con);
    cmdCheck.Parameters.AddWithValue("@CourseID", courseId);

    int check = Convert.ToInt32(cmdCheck.ExecuteScalar());

    if (check != 1)
    {
      StringBuilder sb = new StringBuilder("CreateAssessment.aspx?CourseId=" + courseId);

      Response.Redirect(sb.ToString());
    }
    else
    {
      string getQuestion = @"
                            SELECT q.QuestionID, a.AssessmentID, c.CourseID FROM [Question] AS q
                            INNER JOIN [Assessment] AS a ON q.AssessmentID = a.AssessmentID
                            INNER JOIN [Course] AS c ON c.CourseID = a.CourseID
                            WHERE c.CourseID = @CourseID
                            ";
      SqlCommand getQuestionCmd = new SqlCommand(getQuestion, con);
      getQuestionCmd.Parameters.AddWithValue("@CourseID", courseId);
```

```
      getQuestionCmd.Parameters.AddWithValue("@CourseID", courseId);

      SqlDataReader sdr = getQuestionCmd.ExecuteReader();

      StringBuilder sb = new StringBuilder("CreateAssessment.aspx?CourseId=" + courseId);
      int count = 1;

      while (sdr.Read())
      {
        int questionId = (int)sdr["QuestionID"];
        sb.Append("&QId" + count + "=" + questionId);
        count++;
      }

      Response.Redirect(sb.ToString());

      sdr.Close();
    }

    con.Close();
  }
  catch (Exception ex)
  {
    Debug.WriteLine(ex.Message);
  }
}
```

Figure 34: Source code for Assessment Button Click

When clicking "Assessment," the code first checks if any assessment questions already exist for a chosen course, then directs users either towards adding new questions or editing existing ones related to that course. If no questions exist yet, users will either be directed towards creating new ones via an add/modify page, or they are taken directly to where they can change current questions on that course page.

```
protected void DeleteButton_Click(object sender, System.Web.UI.ImageClickEventArgs e)
{
  ImageButton deleteButton = (ImageButton)sender;
  string courseId = deleteButton.CommandArgument;

  try
  {
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
    con.Open();

    string deleteQuery = @"
                          DELETE e FROM [Excerpt] AS e
                          JOIN [Content] AS ct ON ct.ContentID = e.ContentID
                          JOIN [Course] AS c ON c.CourseID = ct.CourseID
                          WHERE c.CourseID = @CourseID

                          DELETE q FROM [Question] AS q
                          JOIN [Assessment] AS a ON a.AssessmentID = q.AssessmentID
                          JOIN [Course] AS c ON c.CourseID = a.CourseID
                          WHERE c.CourseID = @CourseID

                          DELETE ct FROM [Content] AS ct
                          JOIN [Course] AS c ON c.CourseID = ct.CourseID
                          WHERE c.CourseID = @CourseID

                          DELETE a FROM [Assessment] AS a
                          JOIN [Course] AS c ON c.CourseID = a.CourseID
                          WHERE c.CourseID = @CourseID

                          DELETE FROM [Course] WHERE CourseID = @CourseID
                          ";
    SqlCommand cmdDelete = new SqlCommand(deleteQuery, con);
    cmdDelete.Parameters.AddWithValue("@CourseID", courseId);

    cmdDelete.ExecuteNonQuery();
```

Figure 35: Source code for Delete Button Click

The code oversees a complete procedure when an administrator chooses to delete a course by clicking "Delete"; to erase data like excerpts, questions, assessments, and material in addition to deleting the entire course itself via SQL delete requests. Once successful, users are returned to a screen where they can select their course(s).

Admin Dashboard Page

```
protected void Page_Load(object sender, EventArgs e)
{
  if (Session["Username"] as string == "Kohemin")
  {
    try
    {
      SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
      con.Open();

      // Total Trainee
      string totalTrainee = "SELECT count(*) FROM [User]";
      SqlCommand totalTraineeCmd = new SqlCommand(totalTrainee, con);
      int totalTraineeCount = Convert.ToInt32(totalTraineeCmd.ExecuteScalar().ToString());

      TotalTrainee.Text = totalTraineeCount.ToString();

      // Total Course based on Difficulty
      string totalBegin = "SELECT count(*) FROM [Course] WHERE Difficulty = 'Beginner'";
      SqlCommand totalBeginCmd = new SqlCommand(totalBegin, con);
      int totalBeginCount = Convert.ToInt32(totalBeginCmd.ExecuteScalar().ToString());

      TotalBegin.Text = totalBeginCount.ToString();

      string totalInter = "SELECT count(*) FROM [Course] WHERE Difficulty = 'Intermediate'";
      SqlCommand totalInterCmd = new SqlCommand(totalInter, con);
      int totalInterCount = Convert.ToInt32(totalInterCmd.ExecuteScalar().ToString());

      TotalInter.Text = totalInterCount.ToString();

      string totalAdv = "SELECT count(*) FROM [Course] WHERE Difficulty = 'Advanced'";
      SqlCommand totalAdvCmd = new SqlCommand(totalAdv, con);
      int totalAdvCount = Convert.ToInt32(totalAdvCmd.ExecuteScalar().ToString());

      TotalAdv.Text = totalAdvCount.ToString();

      string totalMaster = "SELECT count(*) FROM [Course] WHERE Difficulty = 'Masterclass'";
```

Figure 36: Source code for loading page

This code focuses on Gathering and Displaying Statistics for the Learning Platform. In order to gather necessary data, a *SqlConnection* object connects to the database via SQL queries that return information such as enrollment counts as well as course counts in Beginner, Intermediate, Advanced, and Masterclass categories. Once collected, labels matching these counts can be displayed on a dashboard page.

```
// Most Popular Course
string popular = @"
                SELECT count(*) AS EnrolledCount, c.CourseId, c.Title
                FROM Enrolled AS en
                INNER JOIN Course AS c ON en.CourseID = c.CourseID
                GROUP BY c.CourseID, c.Title
                ORDER BY EnrolledCount DESC
                ";
SqlCommand popularCmd = new SqlCommand(popular, con);
SqlDataReader sdr = popularCmd.ExecuteReader();

if (sdr.Read())
{
  MostPopular.Text = sdr["Title"].ToString();
}

sdr.Close();
```

Figure 37: Source code for dashboard

The dashboard's ability to identify the Most Popular Course based on enrollment numbers is one of its key advantages. A SQL query runs to aggregate enrollment information across courses and identify those with the greatest student enrollment numbers - these popular classes will then be prominently displayed on the dashboard allowing administrators to instantly identify which are generating most student enthusiasm.

```
// Number of Trainee Enrolled
string totalEnrolled = "SELECT count(*) FROM [Enrolled]";
SqlCommand totalEnrolledCmd = new SqlCommand(totalEnrolled, con);
int totalEnrolledCount = Convert.ToInt32(totalEnrolledCmd.ExecuteScalar().ToString());

TotalEnrolled.Text = totalEnrolledCount.ToString();

// Number of Trainee Completed
string totalCompleted = "SELECT count(*) FROM [Enrolled] WHERE Complete = 1";
SqlCommand totalCompletedCmd = new SqlCommand(totalCompleted, con);
int totalCompletedCount = Convert.ToInt32(totalCompletedCmd.ExecuteScalar().ToString());

TotalCompleted.Text = totalCompletedCount.ToString();

// Number of Trainee Assessed
string totalAssessed = "SELECT count(*) FROM [Enrolled] WHERE Complete = 1 AND Assessment = 1";
SqlCommand totalAssessedCmd = new SqlCommand(totalAssessed, con);
int totalAssessedCount = Convert.ToInt32(totalAssessedCmd.ExecuteScalar().ToString());

TotalAssessed.Text = totalAssessedCount.ToString();
```

Figure 38: Source code for SQL data

The dashboard also provides information about trainee activities. The number of enrollees and those who successfully complete courses is calculated and displayed, providing administrators with better insight into engagement rates and completion rates of students enrolled in training courses. Furthermore, administrators can monitor how many of these learners' complete assessments successfully as an indicator of future participation levels in assessments related to course completions.

```
        // Pending Assessment Course
        string pending = @"
                        SELECT * FROM [Course] AS c
                        LEFT JOIN [Assessment] AS a ON a.CourseID = c.CourseID
                        WHERE a.CourseID IS NULL;
                        ";
        SqlCommand pendingCmd = new SqlCommand(pending, con);

        DataTable pendingAssessmentsTable = new DataTable();

        SqlDataReader sdr2 = pendingCmd.ExecuteReader();
        pendingAssessmentsTable.Load(sdr2);

        if (pendingAssessmentsTable.Rows.Count == 0)
        {
            NoPendingAssessment.Visible = true;
            PendingAssessments.Visible = false;
        }
        else
        {
            PendingAssessments.DataSource = pendingAssessmentsTable;
            PendingAssessments.DataBind();
        }

        sdr2.Close();

        con.Close();
    }
    catch (Exception ex)
    {
        Debug.WriteLine(ex.Message);
    }
}
```

Figure 39: Source code for pending assessment course

The dashboard also provides a listing of courses requiring attention. Courses without associated assessments are shown under Pending Assessment Courses; using SQL queries, code fetches this list and connects it to a repeater control for easier assessment management processes by quickly allowing administrators to identify which courses need assessments added to them.

Course

```
<asp:Label ID="BannerQuote" runat="server" Text=""></asp:Label>
```

Figure 40: Source code for display banner quote

An <*asp:Label*> element with ID *BannerQuote* dynamically displays a banner quote related to the course using engaging information pulled directly from its database and placed into its Text field in *Course.aspx.cs* code-behind (*Course.aspx.cs*). This engaging information enhances aesthetic appeal on course pages.

```
<% if (Session["Username"] as string != "Kohemin") { %>
  <div class="assessment-container">
    <div class="cancel-save-container">
      <asp:Button CssClass="cancel-save-button" ID="FinishCourseButton" runat="server" Text="Finish Course" OnClick="FinishC
    </div>
  </div>
<% } %>
```

Figure 41: Source code for assessment-related container

Presentation of an assessment-related container is determined by a conditional logic block included within a div element. If the user is not "*Kohemin*," an assessment container with an <*asp:Button*> with ID "*FinishCourseButton*" will be displayed so regular users may click it to indicate when their course has completed, initiating evaluation procedures.

```
protected void Page_Load(object sender, EventArgs e)
{
  if (Session["Username"] as string != null)
  {
    int courseId = Convert.ToInt32(Request.QueryString["CourseId"]);

    try
    {
      SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
      con.Open();

      string courseQuery = "SELECT Title FROM [Course] WHERE CourseID = @CourseID";
      SqlCommand courseCmd = new SqlCommand(courseQuery, con);
      courseCmd.Parameters.AddWithValue("@CourseID", courseId);

      SqlDataReader sdr = courseCmd.ExecuteReader();

      while (sdr.Read())
      {
        BannerQuote.Text = sdr["Title"].ToString();
      }

      sdr.Close();

      string excerptQuery = @"
                            SELECT e FROM [Excerpt] AS e
                            INNER JOIN [Content] AS ct ON e.ContentID = ct.ContentID
                            INNER JOIN [Course] AS c ON ct.CourseID = c.CourseID
                            WHERE c.CourseID = @CourseID
                            ";
      SqlCommand excerptCmd = new SqlCommand(excerptQuery, con);
      excerptCmd.Parameters.AddWithValue("@CourseID", courseId);

      excerptCmd.Parameters.AddWithValue("@CourseID", courseId);

      SqlDataReader sdr2 = excerptCmd.ExecuteReader();

      List<ExcerptData> excerptDatas = new List<ExcerptData>();

      while(sdr2.Read())
      {
        ExcerptData excerptData = new ExcerptData
        {
          ExcerptId = Convert.ToInt32(sdr2["ExcerptID"]),
          ExcerptTitle = sdr2["Title"].ToString(),
          ExcerptSubheading = sdr2["Subheading"].ToString(),
          ExcerptContent = sdr2["Content"].ToString(),
          ExcerptImage = (byte[])sdr2["Image"]
        };

        excerptDatas.Add(excerptData);
      }

      sdr2.Close();

      con.Close();

      CourseRepeater.DataSource = excerptDatas;
      CourseRepeater.DataBind();
    }
    catch(Exception ex)
    {
      Debug.WriteLine(ex.Message);
    }
  }
```

Figure 42: Source code for loading page

This process begins as soon as the page loads and uses *Session["Username"]* value to identify whether a user is currently logged in. Once verified, *CourseId* from query string is extracted, database connection established, and course name/excerpts obtained - *BannerQuote* label filled with course name; *ExcerptData* objects generated for excerpts placed into *ExcerptData* list created from excerpt data are bound using *CourseRepeater* control so they may be displayed within HTML markup.

```
public class ExcerptData
{
    1 reference | NightfuryEquinn, 29 days ago | 1 author, 1 change
    public int ExcerptId { get; set; }
    1 reference | NightfuryEquinn, 29 days ago | 1 author, 1 change
    public string ExcerptTitle { get; set; }
    1 reference | NightfuryEquinn, 29 days ago | 1 author, 1 change
    public string ExcerptSubheading { get; set; }
    1 reference | NightfuryEquinn, 29 days ago | 1 author, 1 change
    public string ExcerptContent { get; set; }
    1 reference | NightfuryEquinn, 29 days ago | 1 author, 1 change
    public byte[] ExcerptImage { get; set; }
}
```

Figure 43: Source code for store excerpts

This nested class defines a data structure for storing excerpts. This class's attributes include *ExcerptId*, *ExcerptTitle*, *ExcerptSubheading*, *ExcerptContent*, and *ExcerptImage*; when traversing acquired excerpt data instances of this class are produced as instances and used populate *CourseRepeater* instances with new *excerptDatas* list items.

```
protected void FinishCourseButton_Click(object sender, EventArgs e)
{
  int courseId = Convert.ToInt32(Request.QueryString["CourseId"]);

  try
  {
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
    con.Open();

    string getUserID = "SELECT * FROM [User] WHERE Username = @Username";
    SqlCommand getUserIDCmd = new SqlCommand(getUserID, con);
    getUserIDCmd.Parameters.AddWithValue("@Username", Session["Username"].ToString());

    SqlDataReader sdr = getUserIDCmd.ExecuteReader();
    int theUserID = 0;

    if (sdr.Read())
    {
      theUserID = Convert.ToInt32(sdr["UserID"]);
    }

    sdr.Close();

    string checkFinish = "SELECT count(*) FROM [Enrolled] WHERE DateComplete IS NULL AND DateAssessmentComplete IS NULL AND UserID = @UserID1 A
    SqlCommand checkFinishCmd = new SqlCommand(checkFinish, con);
    checkFinishCmd.Parameters.AddWithValue("@UserID1", theUserID);
    checkFinishCmd.Parameters.AddWithValue("@CourseID1", courseId);

    string checkFinishBoth = "SELECT count(*) FROM [Enrolled] WHERE DateComplete IS NOT NULL AND DateAssessmentComplete IS NULL AND UserID = @U
    SqlCommand checkFinishBothCmd = new SqlCommand(checkFinishBoth, con);
    checkFinishBothCmd.Parameters.AddWithValue("@UserID2", theUserID);
    checkFinishBothCmd.Parameters.AddWithValue("@CourseID2", courseId);
```

```
    int checkFinishCount = Convert.ToInt32(checkFinishCmd.ExecuteScalar());
    int checkFinishBothCount = Convert.ToInt32(checkFinishBothCmd.ExecuteScalar());

    if (checkFinishCount == 1)
    {
      string saveFinish = @"
                          UPDATE [Enrolled]
                          SET
                          Complete = 1, DateComplete = CURRENT_TIMESTAMP
                          WHERE UserID = @UserID3 AND CourseID = @CourseID3
                          ";
      SqlCommand saveFinishCmd = new SqlCommand(saveFinish, con);
      saveFinishCmd.Parameters.AddWithValue("@UserID3", theUserID);
      saveFinishCmd.Parameters.AddWithValue("@CourseID3", courseId);

      saveFinishCmd.ExecuteNonQuery();

      Response.Write(
        "<script>alert('Congrats. Now, you take your assessment'); document.location.href='./PersonalCourse.aspx'</script>"
      );
    }
    else if(checkFinishBothCount == 1)
    {
      Response.Write(
        "<script>alert('You have completed the course but not the assessment. Please take your assessment.'); document.location.href='./Persona
      );
    }
    else
    {
      Response.Write(
        "<script>alert('You have completed the course and assessment in the past.'); document.location.href='./PersonalCourse.aspx'</script>"
      );
    }
```

Figure 44: Source code for Finish Course

When clicking "Finish Course," this function is activated. To interact with the database further, it retrieves *CourseId* from query string; using their username from current session username they retrieve their *UserID* and retrieve whether or not user has completed course, test or both; depending on this result appropriate action are taken: Once enrolled is changed to reflect completion and timestamp saved if user has completed course but not evaluation process, success message appears and notice prompting to take assessment is shown on screen. Notification will appear if a user has completed both the course and assessment.

Course Assessment Page

```
<div class="question-selection">
  <asp:RadioButton AutoPostBack="true" EnableViewState="true" GroupName='<%# Eval("QuestionID") %>' CssClass="radio-button" ID="ChoiceA" Text='<%# Ev
  <asp:RadioButton AutoPostBack="true" EnableViewState="true" GroupName='<%# Eval("QuestionID") %>' CssClass="radio-button" ID="ChoiceB" Text='<%# Ev
  <asp:RadioButton AutoPostBack="true" EnableViewState="true" GroupName='<%# Eval("QuestionID") %>' CssClass="radio-button" ID="ChoiceC" Text='<%# Ev
  <asp:RadioButton AutoPostBack="true" EnableViewState="true" GroupName='<%# Eval("QuestionID") %>' CssClass="radio-button" ID="ChoiceD" Text='<%# Ev
```

Figure 45: Source code for radio buttons

Radio buttons are created using the *<asp:RadioButton>* element for every option in a multiple-choice question, with only one choice allowed per question due to *GroupName* attribute grouping of radio buttons. Once an option is chosen, its *AutoPostBack* attribute initiates a server-side postback allowing interaction with server-side functionality; when selected again the *Text* property populates text for radio button choices from its data source.

```
protected void Page_Load(object sender, EventArgs e)
{
  if (Session["Username"] as string != null)
  {
    try
    {
      SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
      con.Open();

      int theCourseId = Convert.ToInt32(Request.QueryString["CourseId"]);

      string fetchQuestionQuery = @"
                                  SELECT * FROM [Question] as q
                                  INNER JOIN [Assessment] as a ON q.AssessmentID = a.AssessmentID
                                  INNER JOIN [Course] as c ON a.CourseID = c.CourseID
                                  WHERE c.CourseID = @CourseID;
                                  ";
      SqlCommand fetchQuestionQueryCmd = new SqlCommand(fetchQuestionQuery, con); ;
      fetchQuestionQueryCmd.Parameters.AddWithValue("@CourseID", theCourseId);

      SqlDataReader sdr = fetchQuestionQueryCmd.ExecuteReader();

      List<McqData> mcqQuestions = new List<McqData>();
      int count = 1;

      while(sdr.Read())
      {
        McqData mcqQuestion = new McqData
        {
          QuestionID = Convert.ToInt32(sdr["QuestionID"]),
          QuestionNumber = count.ToString(),
          Question = sdr["Question"].ToString(),
          ChoiceA = sdr["ChoiceA"].ToString(),
          ChoiceB = sdr["ChoiceB"].ToString(),
          ChoiceC = sdr["ChoiceC"].ToString(),
          ChoiceD = sdr["ChoiceD"].ToString(),
        };

        count++;
        mcqQuestions.Add(mcqQuestion);
      }

      sdr.Close();

      con.Close();

      if (!IsPostBack)
      {
        MCQRepeater.DataSource = mcqQuestions;
        MCQRepeater.DataBind();
      }
    }
    catch (Exception ex)
    {
      Debug.WriteLine(ex.Message);
    }
  }
  else
  {
    Response.Write(
      "<script>alert('Please register a Kohedemy account or login to Kohedemy before taking your assessment'); document.location.href='./Login.
    );
  }
}
```

Figure 46: Source code for loading page

This page retrieves evaluation questions relevant to the chosen course from the database and adds them to the *MCQRepeater* repeater control, offering multiple-choice responses for each question. When the page first loads, these fetched questions are stored as *McqData* objects linked directly with its repeater control.

```
protected void FinishAssessmentButton_Click(object sender, EventArgs e)
{
  try
  {
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
    con.Open();

    int theCourseId = Convert.ToInt32(Request.QueryString["CourseId"]);

    int totalCorrectAnswers = 0;

    foreach (RepeaterItem item in MCQRepeater.Items)
    {
      RadioButton ChoiceA = item.FindControl("ChoiceA") as RadioButton;
      RadioButton ChoiceB = item.FindControl("ChoiceB") as RadioButton;
      RadioButton ChoiceC = item.FindControl("ChoiceC") as RadioButton;
      RadioButton ChoiceD = item.FindControl("ChoiceD") as RadioButton;

      int theQuestionID = Convert.ToInt32(ChoiceA.GroupName.ToString());
      string correctAnswer = GetCorrectAnswerFromDatabase(theQuestionID);

      if (ChoiceA.Checked && correctAnswer == "A")
      {
        totalCorrectAnswers++;
      }
      else if (ChoiceB.Checked && correctAnswer == "B")
      {
        totalCorrectAnswers++;
      }
      else if (ChoiceC.Checked && correctAnswer == "C")
      {
        totalCorrectAnswers++;
      }
      else if (ChoiceD.Checked && correctAnswer == "D")
      {
        totalCorrectAnswers++;
      }
    }

    if (totalCorrectAnswers > 7)
    {
      string getUserID = "SELECT * FROM [User] WHERE Username = @Username";
      SqlCommand getUserIDCmd = new SqlCommand(getUserID, con);
      getUserIDCmd.Parameters.AddWithValue("@Username", Session["Username"].ToString());

      SqlDataReader sdr = getUserIDCmd.ExecuteReader();
      int theUserId = 0;

      if (sdr.Read())
      {
        theUserId = Convert.ToInt32(sdr["UserID"]);
      }

      sdr.Close();

      string saveAssessmentFinish = @"
                                    UPDATE [Enrolled]
                                    SET
                                    Assessment = 1, DateAssessmentComplete = CURRENT_TIMESTAMP
                                    WHERE UserID = @UserID AND CourseID = @CourseID
                                    ";
      SqlCommand saveAssessmentFinishCmd = new SqlCommand(saveAssessmentFinish, con);
      saveAssessmentFinishCmd.Parameters.AddWithValue("@UserID", theUserId);
      saveAssessmentFinishCmd.Parameters.AddWithValue("@CourseID", theCourseId);

      saveAssessmentFinishCmd.ExecuteNonQuery();

      Response.Write(
        "<script>alert('You have passed the assessment. Total Correct Answers: " + totalCorrectAnswers + "'); document.location.href = './Perso
      );
    }
    else
    {
      Response.Write(
        "<script>alert('You have failed the assessment. Please retake. Total Correct Answers: " + totalCorrectAnswers + "'); document.location.
      );
    }

    con.Close();
  }
  catch (Exception ex)
  {
    Debug.WriteLine(ex.Message);
  }
}
```

Figure 47: Source code for Finish Assessment

As soon as the "*Submit Assessment*" button is clicked, the *FinishAssessmentButton_Click* function kicks in and assesses user responses to assessment questions, calculates how many correct responses overall there were, and then makes its decision regarding passing or failing an assessment. Once a decision has been made for one user's assessment (such as 7 correct answers exceeding threshold), their database record is updated and labeled complete or

otherwise directed them as required (with failure warning if their right responses fall below that threshold).

```
public string GetCorrectAnswerFromDatabase(int theQuestionId)
{
  try
  {
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
    con.Open();

    string correctAnswer = "";

    int theCourseId = Convert.ToInt32(Request.QueryString["CourseId"]);

    string getCorrectQuery = @"
                              SELECT Answer FROM [Question] as q
                              INNER JOIN [Assessment] as a ON q.AssessmentID = a.AssessmentID
                              INNER JOIN [Course] as c ON a.CourseID = c.CourseID
                              WHERE c.CourseID = @CourseID AND q.QuestionID = @QuestionID
                              ";
    SqlCommand getCorrectQueryCmd = new SqlCommand(getCorrectQuery, con);
    getCorrectQueryCmd.Parameters.AddWithValue("@CourseID", theCourseId);
    getCorrectQueryCmd.Parameters.AddWithValue("@QuestionID", theQuestionId);

    correctAnswer = (string) getCorrectQueryCmd.ExecuteScalar();

    con.Close();

    return correctAnswer;
  }
  catch (Exception ex)
  {
    Debug.WriteLine(ex.Message);
    throw;
  }
 }
}
```

Figure 48: Source code for Correct Answer from Database

Based on a supplied question ID, *GetCorrectAnswerFromDatabase* retrieves an accurate response for an assessment question from the database using SQL queries and the *FinishAssessmentButton_Click* function compares responses against each other to ensure their accuracy.

Course Selection Page

```
protected void EnrollButton_Click(object sender, EventArgs e)
{
  try
  {
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
    con.Open();

    if (Session["Username"] != null)
    {
      string theUsername = Session["Username"].ToString();
      Button enrollButton = (Button)sender;
      int theCourse = Convert.ToInt32(enrollButton.CommandArgument);

      Debug.WriteLine(theCourse);

      string checkCourse = @"
                            SELECT count(*) FROM [Enrolled] AS en
                            INNER JOIN [User] AS u ON u.UserID = en.UserID
                            INNER JOIN [Course] AS c On c.CourseID = en.CourseID
                            WHERE u.Username = @Username AND c.CourseID = @CourseID
                            ";
      SqlCommand checkCourseCmd = new SqlCommand(checkCourse, con);
      checkCourseCmd.Parameters.AddWithValue("@Username", theUsername);
      checkCourseCmd.Parameters.AddWithValue("@CourseID", theCourse);

      int check = Convert.ToInt32(checkCourseCmd.ExecuteScalar().ToString());

      if (check > 0)
      {
        Response.Write(
          "<script>alert('You have already enrolled in this course'); document.location.href = './PersonalCourse.aspx'</script>"
        );
      }
      else
      {
        string getTheUserID = "SELECT UserID FROM [User] WHERE Username = @Username";
```

```
    SqlCommand getTheUserIDCmd = new SqlCommand(getTheUserID, con);
    getTheUserIDCmd.Parameters.AddWithValue("@Username", theUsername);

    SqlDataReader sdr = getTheUserIDCmd.ExecuteReader();
    int theUserID = 0;

    if (sdr.Read())
    {
      theUserID = Convert.ToInt32(sdr["UserID"].ToString());
    }

    string insertCourse = @"
                          INSERT INTO [Enrolled] (DateEnroll, UserID, CourseID)
                          VALUES (CURRENT_TIMESTAMP, @UserID, (SELECT CourseID FROM [Course] WHERE CourseID = @CourseID))
                          ";
    SqlCommand insertCourseCmd = new SqlCommand(insertCourse, con);
    insertCourseCmd.Parameters.AddWithValue("@UserID", theUserID);
    insertCourseCmd.Parameters.AddWithValue("@CourseID", theCourse);

    sdr.Close();

    insertCourseCmd.ExecuteNonQuery();

    Response.Write(
      "<script>alert('You have sucessfully enrolled. You can start anytime now.'); document.location.href = './PersonalCourse.aspx'</script
    );
  }
}
else
{
  Response.Write(
    "<script>alert('Please register a Kohedemy account or login to Kohedemy before enrolling to a course.'); document.location.href='./Logi
  );
}

con.Close();
}
```

Figure 49: Source code for enrolment

Once the "*Enroll*" button for a course has been clicked, *EnrollButton_Click* becomes active and takes care of enrolling the user in that course. It does so by querying the database to see whether an authenticated session exists (*session "Username"*) for that user; otherwise, a user ID will be retrieved and an enrollment record with most recent date added to *Enrolled* table; an appropriate alert will then be displayed if user was previously enrolled. To determine whether a user has been authenticated, this code also employs session management. It utilizes SQL queries (*SELECT* and *INSERT*) to communicate with its database to retrieve enrollment-related data for storage purposes.

Create Assessment Page

```
<asp:TextBox TextMode="MultiLine" CssClass="management-textarea" ID="Question1" runat="server"></asp:TextBox>
```

Figure 50: Source code for text box

```
<asp:TextBox CssClass="management-textbox" ID="Question1A" runat="server"></asp:TextBox>
```

Users can enter text into input fields created with an *<asp:TextBox>* tag, with properties such as *TextMode* and *CssClass* controlling how it looks and behaves.

```
<asp:DropDownList CssClass="management-textbox" ID="Question1Answer" runat="server">
  <asp:ListItem Value="A">A</asp:ListItem>
  <asp:ListItem Value="B">B</asp:ListItem>
  <asp:ListItem Value="C">C</asp:ListItem>
  <asp:ListItem Value="D">D</asp:ListItem>
</asp:DropDownList>
```

Figure 51: source code for dropdown menus

<asp:ListItem> tags are used to define each choice in dropdown menus created with *<asp:DropDownList>* tags. Individual dropdown items are defined via the *<asp:ListItem>* tag, which also specifies user-visible text and data binding parameters such as *Value*.

```
protected void AssessmentSaveButton_Click(object sender, EventArgs e)
{
  int courseId = Convert.ToInt32(Request.QueryString["CourseId"]);

  try
  {
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
    con.Open();

    if(Request.QueryString["QId1"] == null)
    {
      for (int i = 0; i < 10; i++)
      {
        string question = ((TextBox)Page.FindControl($"Question{i + 1}")).Text;
        string choiceA = ((TextBox)Page.FindControl($"Question{i + 1}A")).Text;
        string choiceB = ((TextBox)Page.FindControl($"Question{i + 1}B")).Text;
        string choiceC = ((TextBox)Page.FindControl($"Question{i + 1}C")).Text;
        string choiceD = ((TextBox)Page.FindControl($"Question{i + 1}D")).Text;
        string answer = ((DropDownList)Page.FindControl($"Question{i + 1}Answer")).SelectedValue;

        if (string.IsNullOrEmpty(question) || string.IsNullOrEmpty(choiceA) ||
            string.IsNullOrEmpty(choiceB) || string.IsNullOrEmpty(choiceC) ||
            string.IsNullOrEmpty(choiceD) || string.IsNullOrEmpty(answer))
        {
          // Handle the case where a required field is empty or null
          Response.Write("<script>alert('Please fill in all input fields.'); document.location.href='./AdminCourseSelection.aspx'</script>");
          return; // Stop further processing
        }
      }

      string createQuery = @"
                          INSERT INTO [Assessment] (CourseID) VALUES (@CourseID)
```

```
else
{
  int q1 = Convert.ToInt32(Request.QueryString["QId1"]);
  int q2 = Convert.ToInt32(Request.QueryString["QId2"]);
  int q3 = Convert.ToInt32(Request.QueryString["QId3"]);
  int q4 = Convert.ToInt32(Request.QueryString["QId4"]);
  int q5 = Convert.ToInt32(Request.QueryString["QId5"]);

  int q6 = Convert.ToInt32(Request.QueryString["QId6"]);
  int q7 = Convert.ToInt32(Request.QueryString["QId7"]);
  int q8 = Convert.ToInt32(Request.QueryString["QId8"]);
  int q9 = Convert.ToInt32(Request.QueryString["QId9"]);
  int q10 = Convert.ToInt32(Request.QueryString["QId10"]);

  for (int i = 0; i < 10; i++)
  {
    string question = ((TextBox)Page.FindControl($"Question{i + 1}")).Text;
    string choiceA = ((TextBox)Page.FindControl($"Question{i + 1}A")).Text;
    string choiceB = ((TextBox)Page.FindControl($"Question{i + 1}B")).Text;
    string choiceC = ((TextBox)Page.FindControl($"Question{i + 1}C")).Text;
    string choiceD = ((TextBox)Page.FindControl($"Question{i + 1}D")).Text;
    string answer = ((DropDownList)Page.FindControl($"Question{i + 1}Answer")).SelectedValue;

    if (string.IsNullOrEmpty(question) || string.IsNullOrEmpty(choiceA) ||
        string.IsNullOrEmpty(choiceB) || string.IsNullOrEmpty(choiceC) ||
        string.IsNullOrEmpty(choiceD) || string.IsNullOrEmpty(answer))
    {
      // Handle the case where a required field is empty or null
      Response.Write("<script>alert('Please fill in all input fields.'); document.location.href='./AdminCourseSelection.aspx'</script>");
      return; // Stop further processing
    }
```

Figure 52: Source code for save assessment

When clicking the *Save* button of an assessment, the *AssessmentSaveButton_Click* method is invoked, which handles the logic to either create new assessments or modify existing ones.

Request.QueryString["QId1"] will return null if the page is in "add assessment" mode. From its associated textboxes and dropdown menus, an approach extracts data for up to 10 questions with text, alternatives, and correct responses, before verifying all necessary fields have been filled in and processed appropriately. An alert is displayed if any necessary fields are missing while, otherwise, all necessary data is entered directly into SQL queries within minutes if all required fields have been filled out properly.

Request.QueryString["QId1"] indicates "edit assessment" mode on a page if it does not return as null, whereby question information and input fields are retrieved using similar techniques as "add assessment". These details of current questions will then be updated in the database using SQL queries. The code interacts with multiple textboxes, dropdown lists and button

ASP.NET server controls; users provide input by accessing these controls using *Page.FindControl* function.

Create Course Page

```
<asp:FileUpload CssClass="management-file-upload" ID="FeaturedImage" runat="server" />
```

Figure 53: Source code for upload file

Users can select which files to upload using the file input element created by <asp:FileUpload> control. They can browse their local file system for files they would like to upload as well. Furthermore, this control has *runat*, *ID* and *CSSClass* attributes.

```
protected void CourseSaveButton_Click(object sender, EventArgs e)
{
    if (Request.QueryString["CourseId"] != null)
    {
        int courseID = Convert.ToInt32(Request.QueryString["CourseId"]);
        int excerpt1 = Convert.ToInt32(Request.QueryString["ExcerptId1"]);
        int excerpt2 = Convert.ToInt32(Request.QueryString["ExcerptId2"]);
        int excerpt3 = Convert.ToInt32(Request.QueryString["ExcerptId3"]);
        int excerpt4 = Convert.ToInt32(Request.QueryString["ExcerptId4"]);

        try
        {
            SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
            con.Open();

            byte[] featuredImageByte = FeaturedImage.FileBytes;
            byte[] image1Byte = ExcerptImage1.FileBytes;
            byte[] image2Byte = ExcerptImage2.FileBytes;
            byte[] image3Byte = ExcerptImage3.FileBytes;
            byte[] image4Byte = ExcerptImage4.FileBytes;

            if (!string.IsNullOrWhiteSpace(CourseTitle.Text) && !string.IsNullOrWhiteSpace(CourseDescription.Text)
                && !string.IsNullOrWhiteSpace(CourseDifficulty.SelectedValue)
                && featuredImageByte != null
                && !string.IsNullOrWhiteSpace(Excerpt1.Text) && !string.IsNullOrWhiteSpace(ExcerptSubheading1.Text)
                && !string.IsNullOrWhiteSpace(ExcerptContent1.Text) && image1Byte != null
                && !string.IsNullOrWhiteSpace(Excerpt2.Text) && !string.IsNullOrWhiteSpace(ExcerptSubheading2.Text)
                && !string.IsNullOrWhiteSpace(ExcerptContent2.Text) && image2Byte != null
                && !string.IsNullOrWhiteSpace(Excerpt3.Text) && !string.IsNullOrWhiteSpace(ExcerptSubheading3.Text)
                && !string.IsNullOrWhiteSpace(ExcerptContent3.Text) && image3Byte != null
                && !string.IsNullOrWhiteSpace(Excerpt4.Text) && !string.IsNullOrWhiteSpace(ExcerptSubheading4.Text)
                && !string.IsNullOrWhiteSpace(ExcerptContent4.Text) && image4Byte != null)
            {
```

Figure 54: Source code for save course

Similarly, as soon as a website visitor clicks the "*Save*" button, the *CourseSaveButton_Click* method acts as an event handler for this event and updates or preserves course and excerpt data in their database.

This page allows users to modify an existing course if the "*CourseId*" URL query string argument is available. In such a situation, several query string parameters like "*CourseId*," "*ExcerptId1*," "*ExcerptId2*," etc. are retrieved before connecting to the database and issuing SQL *UPDATE* statements against its records in both "*Course*" and "*Excerpt*" databases for updates. If the "*CourseId*" query string parameter is blank, a new course is generated with SQL *INSERT* commands by adding records into "*Course*," "*Content*," and "*Excerpt*" tables.

This method performs various validation checks to ensure all required fields - including images for courses and excerpts- are filled out correctly. If all validations pass successfully, relevant SQL queries are run and JavaScript alerts used to display success messages. Should one or

more errors arise while running this process, an error message will be published to debug output for display.

Login Page

```
protected void LoginButton_Click(object sender, EventArgs e)
{
  try
  {
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
    con.Open();

    SqlCommand cmd = new SqlCommand("SELECT count(*) FROM [User] WHERE EmailAddress = @Email AND Password = @Password", con);
    cmd.Parameters.AddWithValue("@Email", EmailBox.Text);
    cmd.Parameters.AddWithValue("@Password", PasswordBox.Text);

    int check = Convert.ToInt32(cmd.ExecuteScalar());

    if (check == 1)
    {
      SqlCommand cmdCheck = new SqlCommand("SELECT Username FROM [User] WHERE EmailAddress = @Email AND Password = @Password", con);
      cmdCheck.Parameters.AddWithValue("@Email", EmailBox.Text);
      cmdCheck.Parameters.AddWithValue("@Password", PasswordBox.Text);

      SqlDataReader sdr = cmdCheck.ExecuteReader();

      string username = "";

      while (sdr.Read())
      {
        username = sdr["Username"].ToString().Trim();
      }

      sdr.Close();

      Session["Username"] = username;

      if (Session["Username"] as string != "Kohemin")
```

```
      if (Session["Username"] as string != "Kohemin")
      {
        Response.Redirect("UserProfile.aspx");
      }
      else
      {
        Response.Redirect("AdminDashboard.aspx");
      }
    }
    else
    {
      Response.Write(
        "<script>alert('Invalid credentials. Please try again.'); document.location.href='./Login.aspx'</script>"
      );
    }

    con.Close();
  }
  catch (Exception ex)
  {
    Debug.WriteLine(ex.Message);
  }
 }
}
}
```

Figure 55: Source code for Login button

The *LoginButton_Click* function is called whenever a user interacts with the login form. By connecting to the database using its connection string specified during setup, this function creates a SQL query to see if there exists a user matching their email address and password in its database.

This function adds parameters to the query by using data from *EmailBox* and *PasswordBox* input fields, in order to prevent SQL injection. *ExecuteScalar()* method is then utilized by this function in order to retrieve how many rows were returned by query; counting 1 is indicative of successful login attempt.

After successfully logging in, an additional query is run to retrieve the user's username from the database and store it as *Session["Username"]* within their session variable. Once their

username is known, they are taken directly to either their profile page (*UserProfile.aspx*) or admin dashboard (*AdminDashboard.aspx*), depending on their role as indicated by their username. Otherwise, the user remains on login page and receives an alert if their attempt at login fails due to incompatible user credentials being identified.

Personal Course Page

```
protected void Page_Load(object sender, EventArgs e)
{
  try
  {
    if (Session["Username"] as string != null)
    {
      SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
      con.Open();

      string getUserID = "SELECT * FROM [User] WHERE Username = @Username";
      SqlCommand getUserIDCmd = new SqlCommand(getUserID, con);
      getUserIDCmd.Parameters.AddWithValue("@Username", Session["Username"].ToString());

      SqlDataReader sdr = getUserIDCmd.ExecuteReader();
      int theUserID = 0;

      if (sdr.Read())
      {
        theUserID = Convert.ToInt32(sdr["UserID"]);
      }

      sdr.Close();

      string courseQuery = @"
                            SELECT * FROM [Enrolled] AS en
                            INNER JOIN [Course] AS c ON c.CourseID = en.CourseID
                            WHERE en.DateComplete IS NULL AND en.DateAssessmentComplete IS NULL AND en.UserID = @UserID1
                            ";
      SqlCommand courseQueryCmd = new SqlCommand(courseQuery, con);
      courseQueryCmd.Parameters.AddWithValue("@UserID1", theUserID);

      SqlDataReader sdr2 = courseQueryCmd.ExecuteReader();

      List<PersonalCourseData> newCourses = new List<PersonalCourseData>();
```

```
while (sdr2.Read())
{
  PersonalCourseData newCourse = new PersonalCourseData
  {
    CourseId = Convert.ToInt32(sdr2["CourseID"]),
    CourseTitle = sdr2["Title"].ToString(),
    CourseDifficulty = sdr2["Difficulty"].ToString(),
    CourseImage = (byte[])sdr2["FeaturedImage"]
  };

  newCourses.Add(newCourse);
}

sdr2.Close();

string assessmentQuery = @"
                          SELECT * FROM [Enrolled] AS en
                          INNER JOIN [Course] AS c ON c.CourseID = en.CourseID
                          WHERE en.DateComplete IS NOT NULL AND en.DateAssessmentComplete IS NULL AND en.UserID = @UserID2
                          ";
SqlCommand assessmentQueryCmd = new SqlCommand(assessmentQuery, con);
assessmentQueryCmd.Parameters.AddWithValue("@UserID2", theUserID);

SqlDataReader sdr3 = assessmentQueryCmd.ExecuteReader();

List<PersonalCourseData> assessmentCourses = new List<PersonalCourseData>();

while (sdr3.Read())
{
  PersonalCourseData assessmentCourse = new PersonalCourseData
  {
    CourseId = Convert.ToInt32(sdr3["CourseID"]),
    CourseTitle = sdr3["Title"].ToString(),
    CourseDifficulty = sdr3["Difficulty"].ToString(),
    CourseImage = (byte[])sdr3["FeaturedImage"]
  };
```

```
            assessmentCourses.Add(assessmentCourse);
        }

        sdr3.Close();

        string reviseQuery = @"
                            SELECT * FROM [Enrolled] AS en
                            INNER JOIN [Course] AS c ON c.CourseID = en.CourseID
                            WHERE en.DateComplete IS NOT NULL AND en.DateAssessmentComplete IS NOT NULL AND en.UserID = @UserID3
                            ";
        SqlCommand reviseQueryCmd = new SqlCommand(reviseQuery, con);
        reviseQueryCmd.Parameters.AddWithValue("@UserID3", theUserID);

        SqlDataReader sdr4 = reviseQueryCmd.ExecuteReader();

        List<PersonalCourseData> reviseCourses = new List<PersonalCourseData>();

        while (sdr4.Read())
        {
            PersonalCourseData reviseCourse = new PersonalCourseData
            {
                CourseId = Convert.ToInt32(sdr4["CourseID"]),
                CourseTitle = sdr4["Title"].ToString(),
                CourseDifficulty = sdr4["Difficulty"].ToString(),
                CourseImage = (byte[])sdr4["FeaturedImage"]
            };

            reviseCourses.Add(reviseCourse);
        }

        sdr4.Close();

        con.Close();

        EnrolledRepeater.DataSource = newCourses;
        EnrolledRepeater.DataBind();


            AssessmentRepeater.DataSource = assessmentCourses;
            AssessmentRepeater.DataBind();

            CompletedRepeater.DataSource = reviseCourses;
            CompletedRepeater.DataBind();
        }
        else
        {
            Response.Write(
              "<script>alert('Unable to access your profile. Please log in again.'); document.location.href='./Login.aspx'</script>"
            );
        }
    }
    catch (Exception ex)
    {
        Debug.WriteLine(ex.Message);
    }
}
```

Figure 56: Source code for loading page

Once *Page_Load* has been called, *PersonalCourse.aspx* page will be loaded. Its functions include check if a user is authorized by using *Session["Username"]*. When verified, establishes a connection to the database using their ID to retrieve course-related data.

Additionally, it creates three lists of *PersonalCourseData* objects--one each for enrolled, assessments, and completed courses--and divides each list into three categories for dynamic display on the website. Finally, these three lists are connected with relevant *<asp:Repeater>* controls such as *EnrolledRepeater*, *AssessmentRepeater* and *CompletedRepeater* in order to display course details as they change.

```
protected void GetStartedButton_Click(object sender, EventArgs e)
{
  try
  {
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
    con.Open();

    string courseId = "";
    Button startButton = (Button)sender;
    if (startButton.ID == "GetStartedButton")
    {
      courseId = startButton.CommandArgument;
    }

    StringBuilder sb = new StringBuilder("Course.aspx?CourseId=" + courseId);

    Response.Redirect(sb.ToString());

    con.Close();
  }
  catch(Exception ex)
  {
    Debug.WriteLine(ex.Message);
  }
}
```

Figure 57: Source code for start button

```
protected void TakeAssessmentButton_Click(object sender, EventArgs e)
{
  try
  {
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
    con.Open();

    string courseId = "";
    Button takeButton = (Button)sender;
    if (takeButton.ID == "TakeAssessmentButton")
    {
      courseId = takeButton.CommandArgument;
    }

    StringBuilder sb = new StringBuilder("CourseAssessment.aspx?CourseId=" + courseId);

    Response.Redirect(sb.ToString());

    con.Close();
  }
  catch (Exception ex)
  {
    Debug.WriteLine(ex.Message);
  }
}
```

Figure 58: Source code for take assessment button

```
protected void ReviseButton_Click(object sender, EventArgs e)
{
  try
  {
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
    con.Open();

    string courseId = "";
    Button reviseButton = (Button)sender;
    if (reviseButton.ID == "ReviseButton")
    {
      courseId = reviseButton.CommandArgument;
    }

    StringBuilder sb = new StringBuilder("Course.aspx?CourseId=" + courseId);

    Response.Redirect(sb.ToString());

    con.Close();
  }
  catch (Exception ex)
  {
    Debug.WriteLine(ex.Message);
  }
 }
}
```

Figure 59: Source code for revision button

Once an assessment and review have been taken, three buttons need to be clicked: Get Started, Assess, and Revise. The functions (*GetStartedButton_Click, TakeAssessmentButton_Click, ReviseButton_Click*) serve as event handlers for these three buttons on repeaters.

The functions can generate a URL using the obtained course ID, use *Response.Redirect* for client-side redirection and collect the associated course ID from clicked buttons' *CommandArgument* attributes to access its associated course or assessment page by redirecting.

Register Page

```
protected void RegisterButton_Click(object sender, EventArgs e)
{
  try
  {
    if((UsernameBox.Text != "") && (PasswordBox.Text != "") && (EmailBox.Text != ""))
    {
      if(PasswordBox.Text == ConfirmBox.Text)
      {
        if (PasswordBox.Text.Length > 9)
        {
          SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
          con.Open();

          string query = "SELECT count(*) from [User] where EmailAddress = @Email";
          SqlCommand cmd = new SqlCommand(query, con);
          cmd.Parameters.AddWithValue("@Email", EmailBox.Text);

          int check = Convert.ToInt32(cmd.ExecuteScalar().ToString());

          if (check > 0)
          {
            Response.Write("<script>alert('Email address already in use.');</script>");
          }
          else
          {
            string insertQuery = "INSERT INTO [User] (Username, EmailAddress, Password) VALUES (@Username, @Email, @Password)";
            SqlCommand cmd1 = new SqlCommand(insertQuery, con);
            cmd1.Parameters.AddWithValue("@Username", UsernameBox.Text);
            cmd1.Parameters.AddWithValue("@Email", EmailBox.Text);
            cmd1.Parameters.AddWithValue("@Password", PasswordBox.Text);

            cmd1.ExecuteNonQuery();

            Response.Write("<script>alert('Your account is created. Please login.'); document.location.href = './Login.aspx'</script>");

            con.Close();
```

Figure 60: Source code for register button

As soon as a user clicks on the "Register" button on a registration form, an event handler named *RegisterButton_Click* is activated. This method is responsible for handling user input and performing registration based on that data. First, the function checks whether all three fields of *UsernameBox.Text*, *PasswordBox.Text* and *EmailBox.Text* are not empty before proceeding further with registration if all three appear so.

When providing a password, it will first be compared with that which has been verified (*PasswordBox.Text == ConfirmBox.Text*), with registration proceeding if the length exceeds 9 characters (and therefore meeting certain requirements). Once connected to a database using its connection string from configuration.

Step two is running a SQL query (*SELECT count(*) from [User] where EmailAddress = @Email*) to determine if the supplied email address already exists in the database, displaying an alert if it does. Otherwise, an *INSERT INTO* query may be used to add new user's data into it if distinct email exists.

Once registration has been successful, a notification confirming account creation will appear and users will be forwarded to the *Login.aspx* page.

User Profile Page

```
protected void SaveButton_Click(object sender, EventArgs e)
{
  try
  {
    if ((editUsername.Text != "") && (editPassword.Text != "") && (editEmail.Text != ""))
    {
      if (editPassword.Text == editConfirm.Text)
      {
        if (editPassword.Text.Length > 9)
        {
          SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
          con.Open();

          SqlCommand cmdCheck = new SqlCommand("SELECT * FROM [User] WHERE Username = '" + Session["Username"] + "'", con);

          SqlDataReader sdr = cmdCheck.ExecuteReader();

          string compareEmail = "";

          while (sdr.Read())
          {
            compareEmail = sdr["EmailAddress"].ToString().Trim();
          }

          sdr.Close();

          string query = "SELECT count(*) from [User] where EmailAddress = @Email";
          SqlCommand cmd = new SqlCommand(query, con);
          cmd.Parameters.AddWithValue("@Email", editEmail.Text);

          int check = Convert.ToInt32(cmd.ExecuteScalar().ToString());

          if (check > 0 && (ProfileEmail.Text != compareEmail))
          {
            Response.Write("<script>alert('Email address already in use.');</script>");
          }
          else
          {
            SqlCommand cmdSave = new SqlCommand("UPDATE [User] SET Username ='" + editUsername.Text + "', EmailAddress = '" + editEmail.Text +
            cmdSave.ExecuteNonQuery();

            con.Close();

            Session["Username"] = editUsername.Text;

            Response.Write(
              "<script>alert('Your personal detail has been updated successfully.'); document.location.href='./UserProfile.aspx'</script>"
            );
          }
        }
        else
        {
          Response.Write("<script>alert('Password must be more than 10 characters'); document.location.href = './UserProfile.aspx'</script>");
        }
      }
      else
      {
        Response.Write("<script>alert('Password and confirm password are not the same.'); document.location.href = './UserProfile.aspx'</script
      }
    }
    else
    {
      Response.Write("<script>alert('Please fill in all input fields.'); document.location.href = './UserProfile.aspx'</script>");
    }
  }
  catch (Exception ex)
  {
    Debug.WriteLine(ex.Message);
  }
 }
}
```

Figure 61: Source code for save

As soon as a user clicks the "Save" button (inside an edit modal), the *SaveButton_Click* event handler is activated and ensures all input fields are valid before updating user profiles with valid entries such as new username, email address and password (which meet requirements) being submitted.

An email check is then performed to see if that email address--other than what currently being used by that particular user--is already being utilized by another. If the email is genuine, a database record for that user will be updated. An alert will be displayed if the password is shorter than 10 characters; additionally, an alert will appear if both password and confirm password differ, and an additional warning appears if not all fields have been filled out correctly.

Once successful updates are applied, a user alert is displayed and username updates take effect for this session.

4.1.1 CSS
External CSS

```
<!--Link to CSS-->
<link rel = "stylesheet" href = "../Css/main.css" />
```

Figure 62: Source code for external CSS

The <head> section of HTML code contains an external CSS link. This line of code connects the website with an external "*main.css*" CSS file, relative to where the HTML file resides. Specifically, its *href* property indicates the path leading directly to that CSS file - commonly found one level up called "Css", according to its path: *../Css/main.css*. This can style and format the HTML components on a page by linking a CSS file that applies the rules set out within it.

4.1.2 CRUD
Delete-DELETE query

```
protected void DeleteButton_Click(object sender, System.Web.UI.ImageClickEventArgs e)
{
    ImageButton deleteButton = (ImageButton)sender;
    string courseId = deleteButton.CommandArgument;

    try
    {
        SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["RegisterString"].ConnectionString);
        con.Open();

        string deleteQuery = @"
                            DELETE e FROM [Excerpt] AS e
                            JOIN [Content] AS ct ON ct.ContentID = e.ContentID
                            JOIN [Course] AS c ON c.CourseID = ct.CourseID
                            WHERE c.CourseID = @CourseID

                            DELETE q FROM [Question] AS q
                            JOIN [Assessment] AS a ON a.AssessmentID = q.AssessmentID
                            JOIN [Course] AS c ON c.CourseID = a.CourseID
                            WHERE c.CourseID = @CourseID

                            DELETE ct FROM [Content] AS ct
                            JOIN [Course] AS c ON c.CourseID = ct.CourseID
                            WHERE c.CourseID = @CourseID

                            DELETE a FROM [Assessment] AS a
                            JOIN [Course] AS c ON c.CourseID = a.CourseID
                            WHERE c.CourseID = @CourseID

                            DELETE FROM [Course] WHERE CourseID = @CourseID
                            ";
        SqlCommand cmdDelete = new SqlCommand(deleteQuery, con);
        cmdDelete.Parameters.AddWithValue("@CourseID", courseId);

        cmdDelete.ExecuteNonQuery();
```

Figure 63: Source code for delete query

Based on the diagram above, the delete query is implemented. *DELETE FROM [Excerpt]* query removes passages related to the selected course and performs a *JOIN* procedure in order to match relevant passages with their related content and ultimately with the course itself. Only content and extracts associated with a specific course are deleted when *WHERE c.CourseID = @CourseID* is applied. The rest of the *DELETE FROM [Question]* query, *DELETE FROM [Content]* query, *DELETE FROM [Assessment]*, are all similar while the *DELETE FROM [Course]* query concludes by deleting the course using its *CourseID*.

4.2 Testing

Several testing have been conducted on the registration page, login page, and edit profile page.

4.2.1 Form validation (registration, etc)

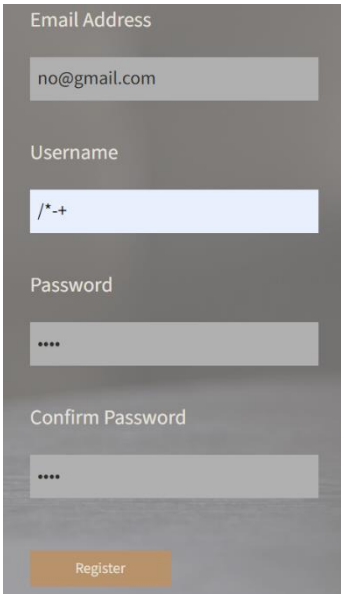


Figure 64: form validation for registration

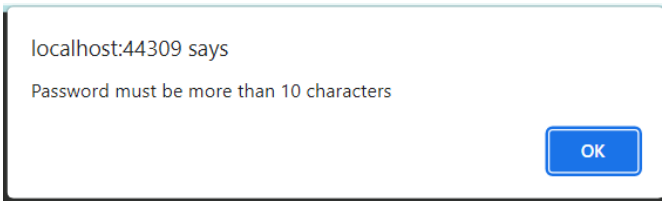


Figure 65: validation prompt

The diagram above shows the registration page the email address and username been entered, while the password is entered as "/*-+" which is the same value as the username. After clicking on the register button, a pop out shows that to create an account the password must be more than 10 characters.

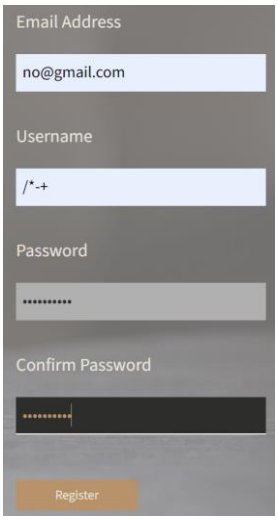


Figure 66: form of validation for registration

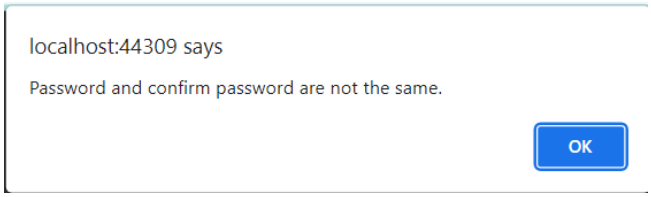


Figure 67: validation prompt

This time a password with 10 characters has been entered, however the account is still not been created due to different value is entered in the password and confirm password textbox.

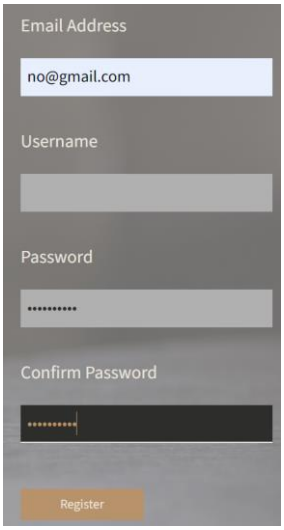


Figure 68: form of validation for registration

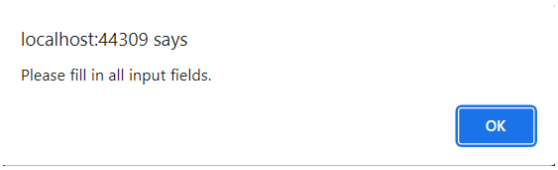


Figure 69: validation prompt

This time the username textbox is left blank while the email address, password and confirm password has fulfilled the requirement. A pop out shows that guest user must fill in all input fields in order to register an account after clicking on the register button.

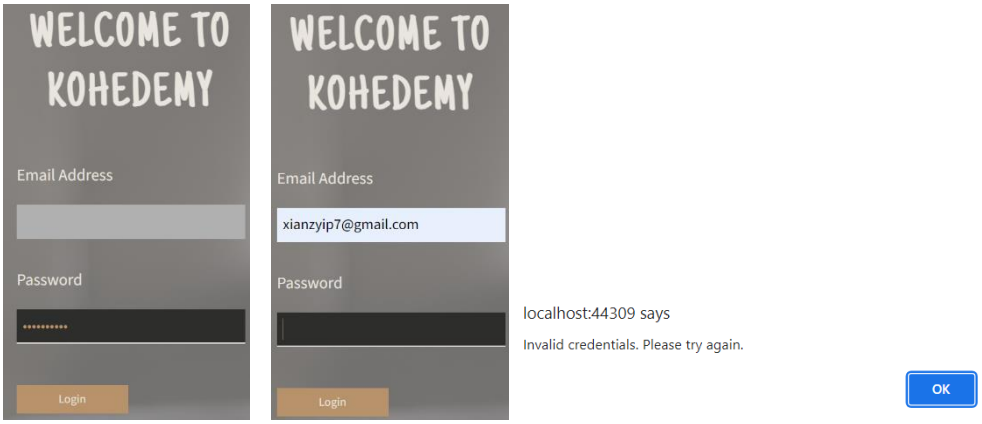


Figure 70: form of registration

If any of the textbox in the login page is empty as well, there will be a pop out indicate that the user has entered invalid credentials to log in.

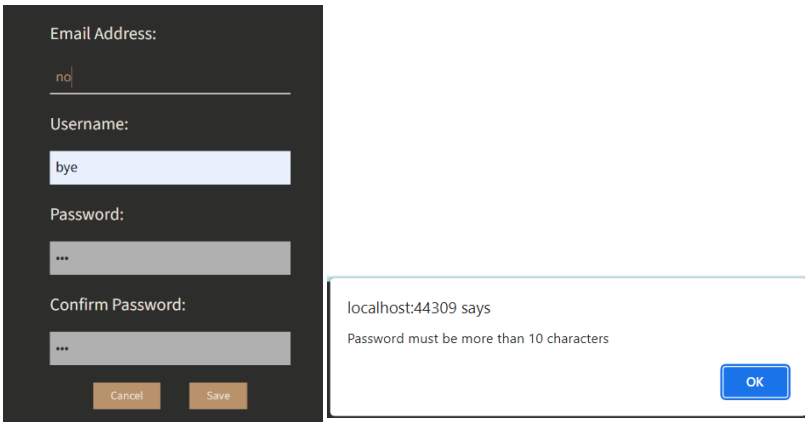


Figure 71: Edit user profile with validation

Typing a password which is less than 10 characters when editing the user profile will get the same notification as the one when registering for an account.

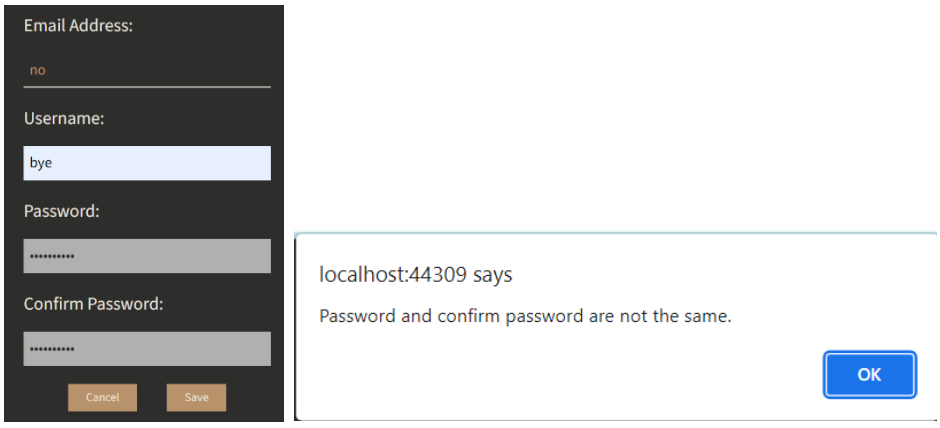


Figure 72: Edit user profile with different validation

Entering different password in the password textbox and confirm password textbox will be getting the same message when registering for an account as well.

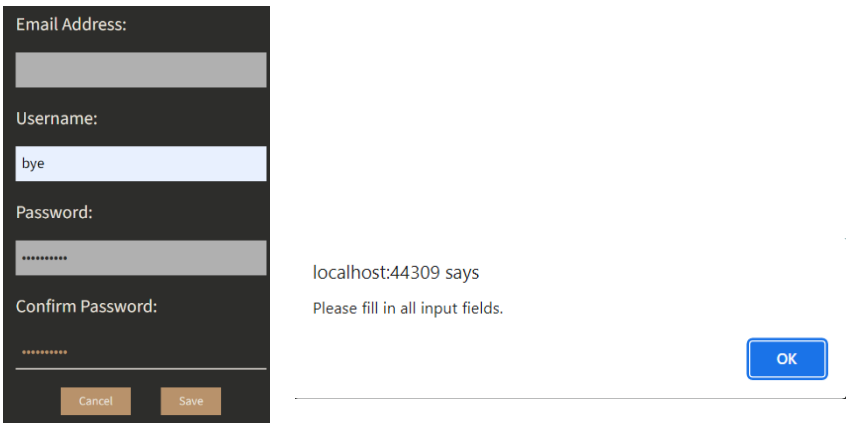


Figure 73: Edit user profile with validation

If the user blank one of the textbox fields when editing their profile, they will be getting a same alert when registering for an account as well.

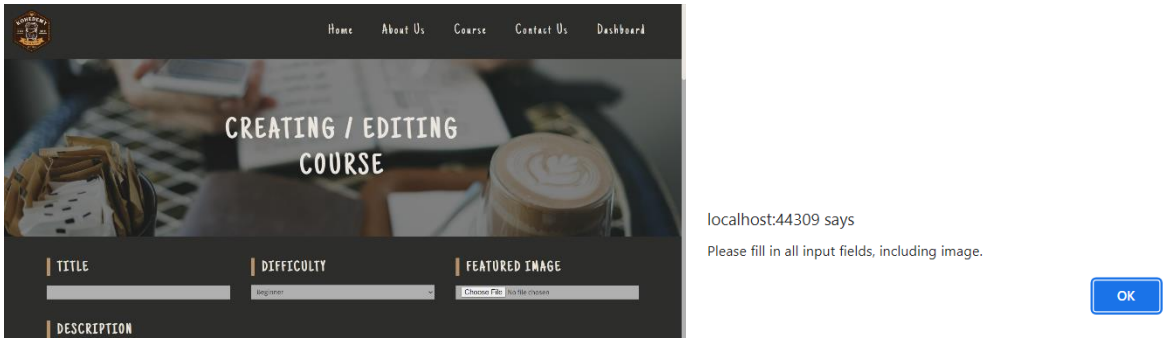


Figure 74:Empty fields

On the other hand, if admin leave any of the fields empty while creating or editing the course, the website will also prompt a message to remind admin to fill in all the required information which include image as well.

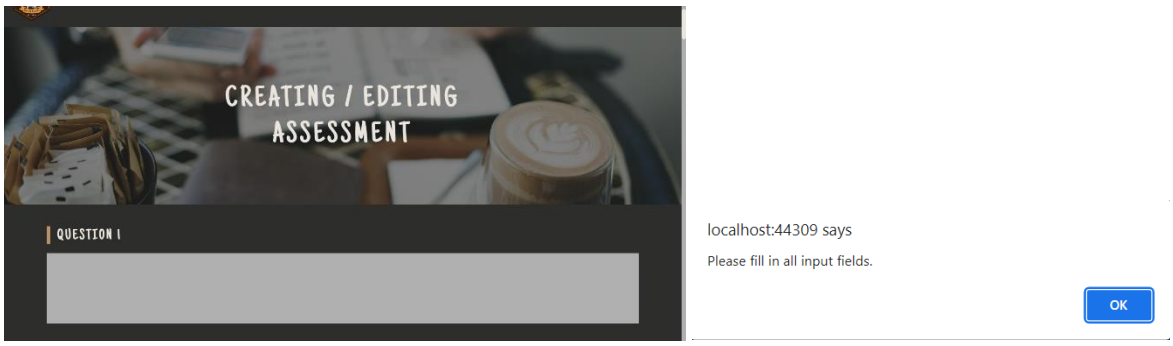


Figure 75:Empty fields

In case one of the fields is also leaven blank while admin is creating or editing assessment, admin will receive the same message which is to fill up all textbox fields.

4.2.2 Login verification (Admin and Registered User)

A registered account for both admin and user has been created which have the email address admin@kohedemy.com and password 1234 for the admin account; xianzyip7@gmail.com, and password 1234567890 for the user account.

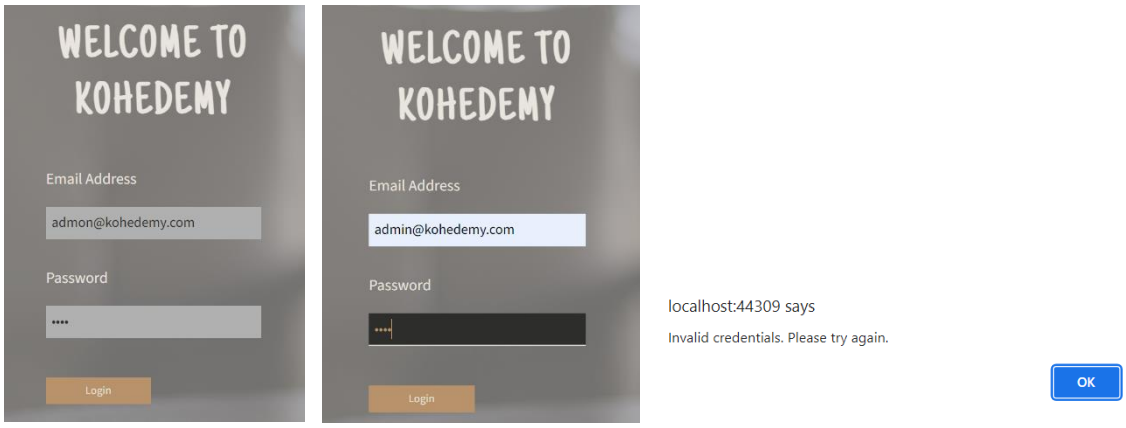


Figure 76: Login verification (admin)

By clicking the login button after entering the wrong email (admon@kohedemy.com) and correct password or correct admin email address and a wrong password (/*-+), a pop out is shown to alert the admin to try logging in again by typing the right credentials.

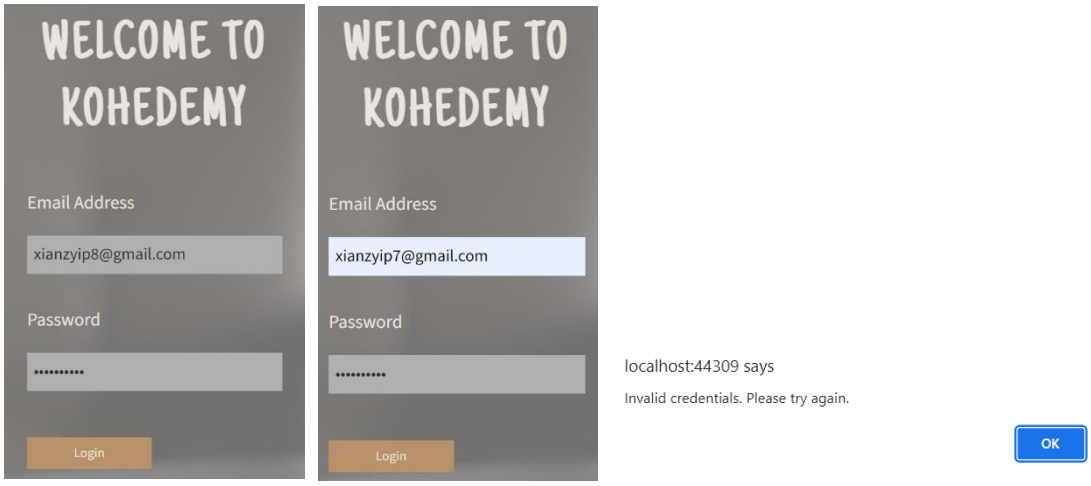


Figure 77: Login verification (user)

Similarly, the “Invalid credentials. Please try again.” pop out is shown if the user has entered the wrong email (xianzyip8@gmail.com) and correct password or correct admin email address and a wrong password (0123456789).

4.2.3 Rendering on different browsers

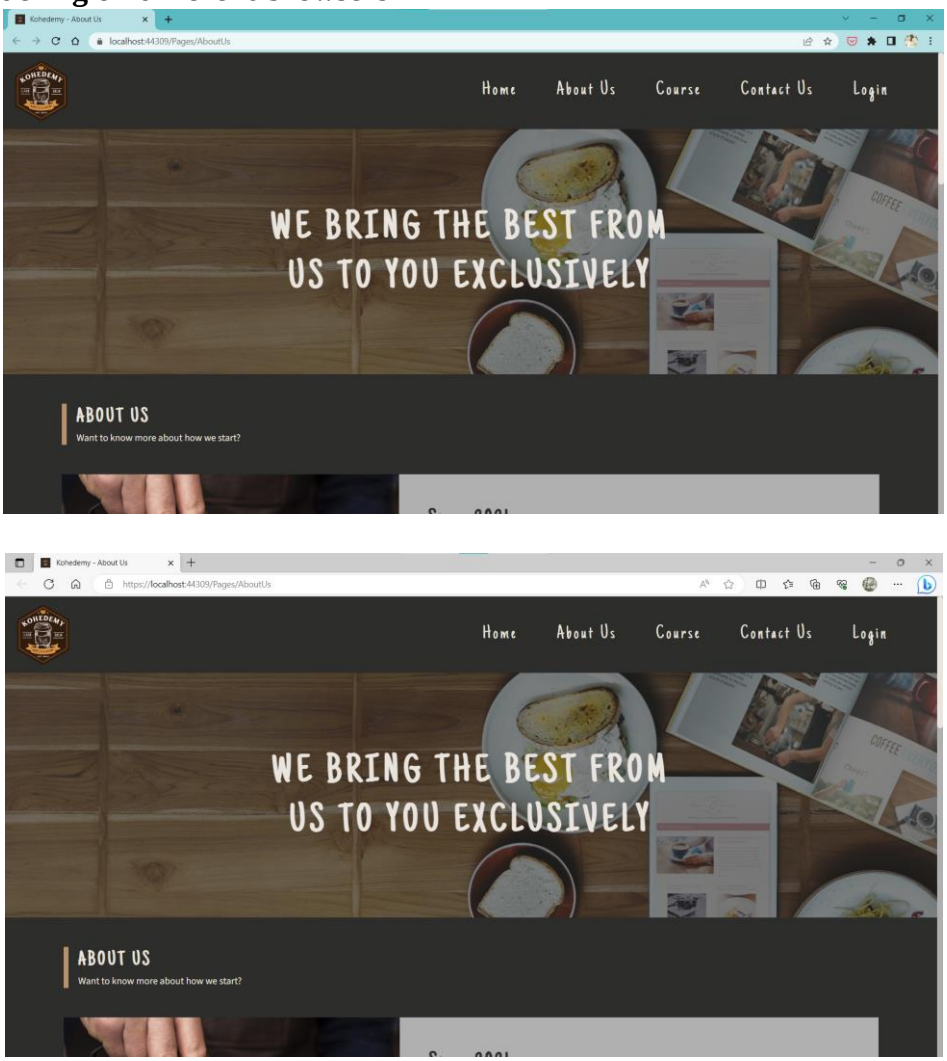


Figure 78: Rendering on different browsers

Rendering the website on different browsers both Google Chrome and Microsoft Edge, the appearance of the website looks the same and is able to run.

5.0 User Guidance

Home Page

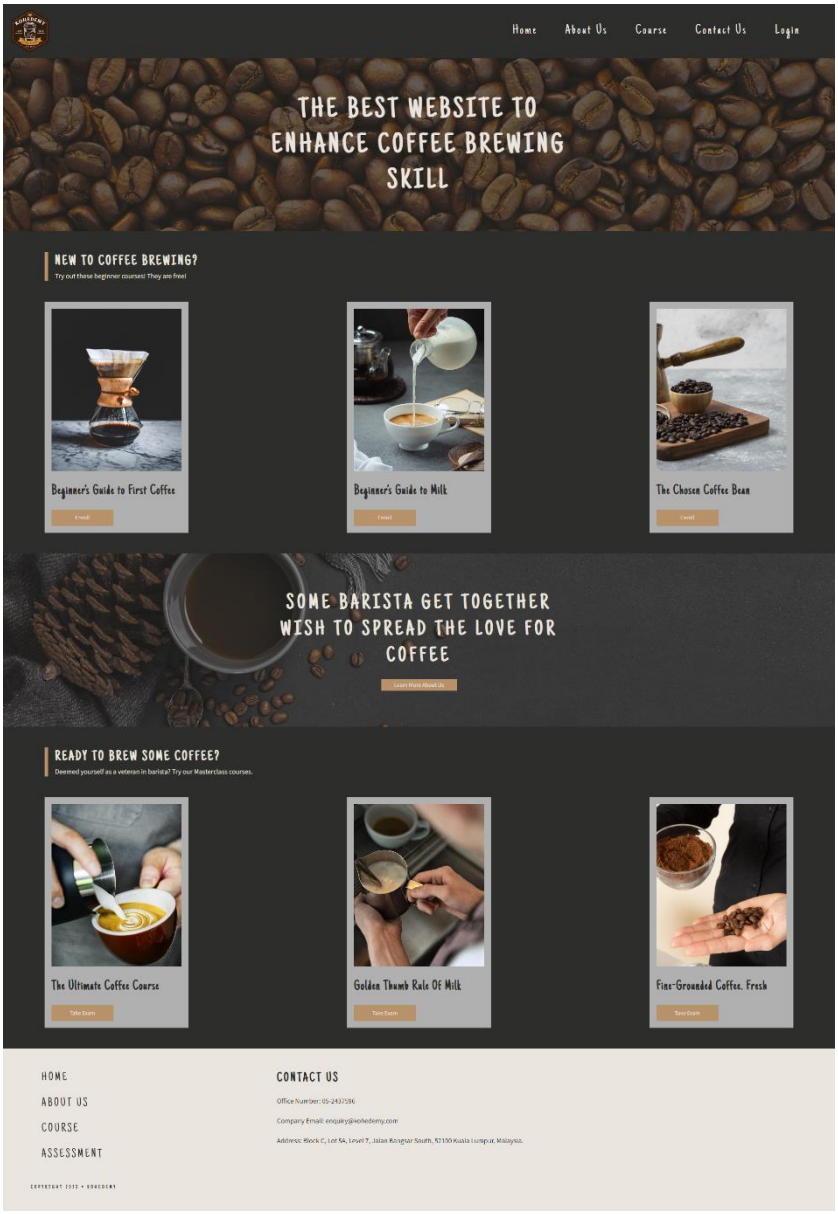


Figure 79: Kohedemy's Home Page

This is Kohedemy's home page a user views when the website is visited. An introduction to the website is given with navigations bar on the above of the website.

Login Page

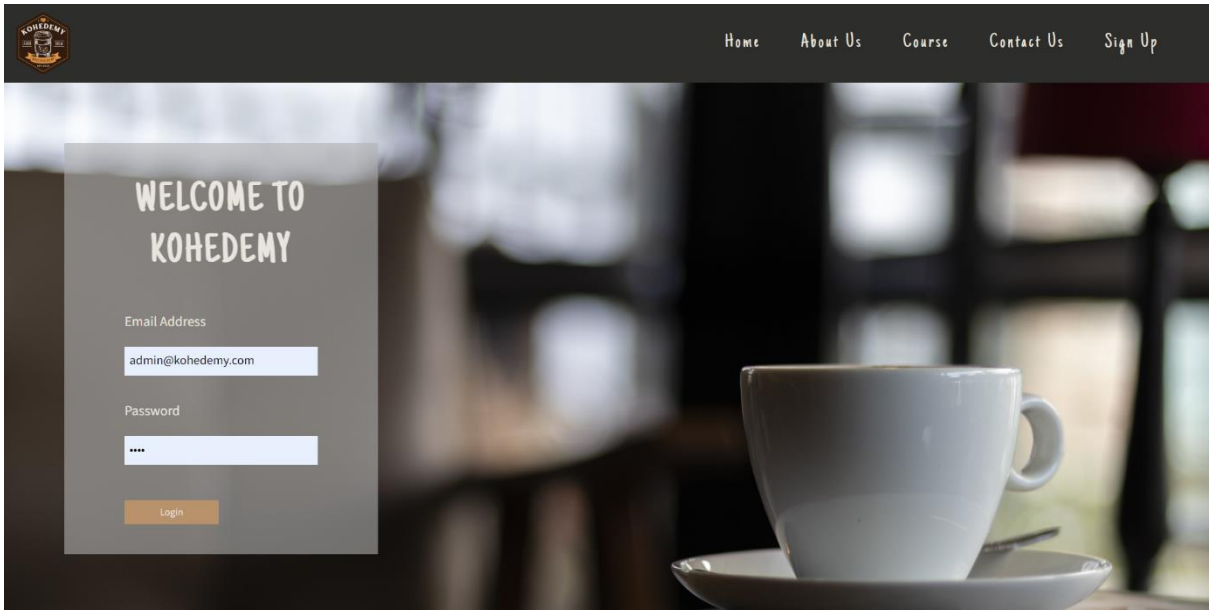


Figure 80: Login Page

This is the Kohedemy's login web page. When a user clicks the login in button on the navigation bar, the webpage will direct the user to the login with existing credential. The page is divided into two sections which is the email address field and the password field. Users enter their email address and password in the respective fields and click the "Login" button. If the login credentials are correct, the user will be logged into the website.

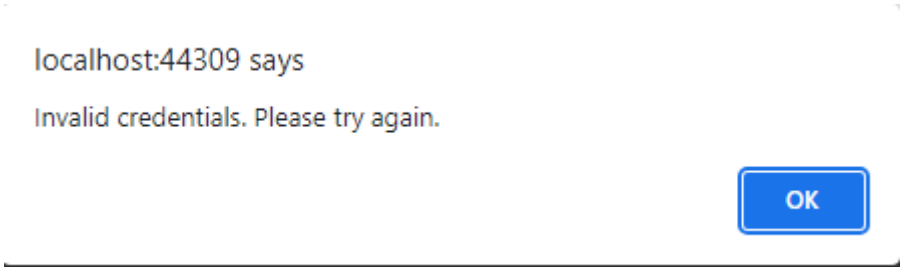


Figure 81: Validation prompt

If the login credentials incorrect, the user will see an error message. Both users and admins share the same login web page. When the users and admins input their password, it will be hidden as privacy.

Sign in Page

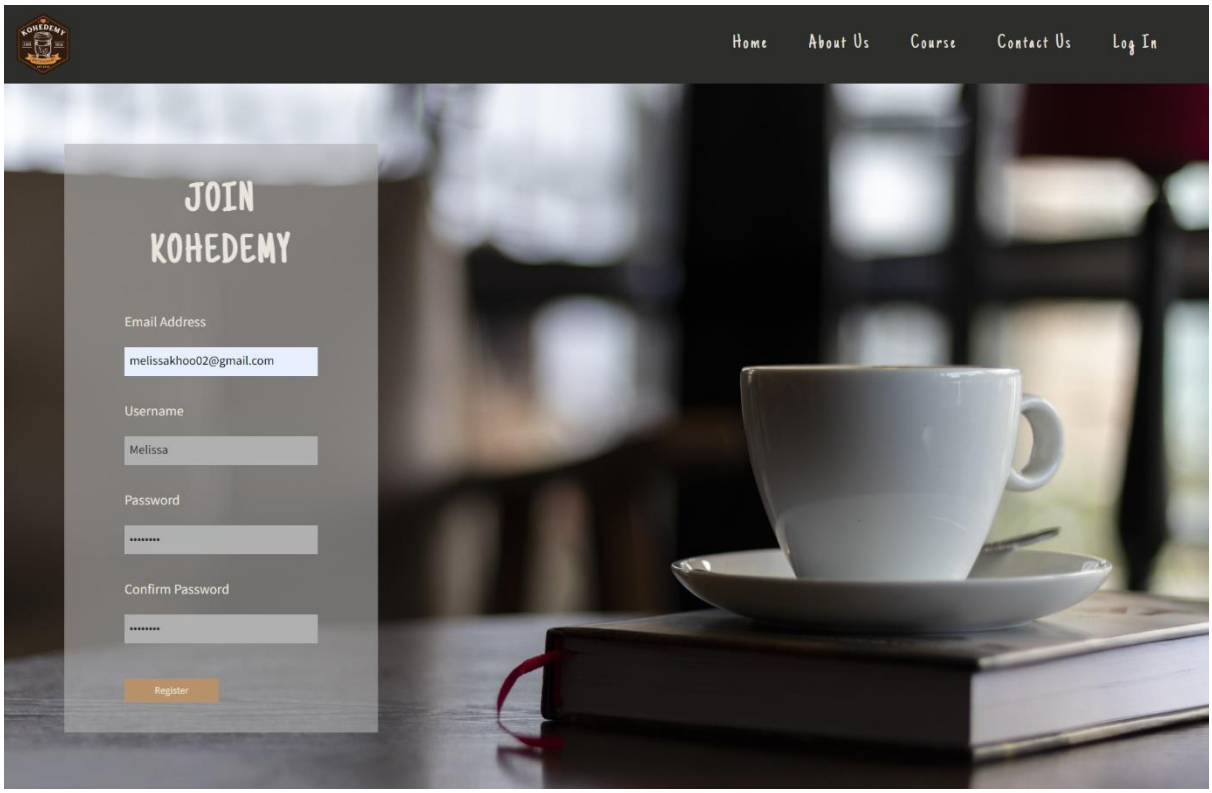


Figure 82: Sign in Page

If a user is unregistered, the user can click the "Sign Up" button on the Login screen, and it will direct to the sign-up page, where it will display the text fields which require user to input their details such as email address, username, password and confirm password.

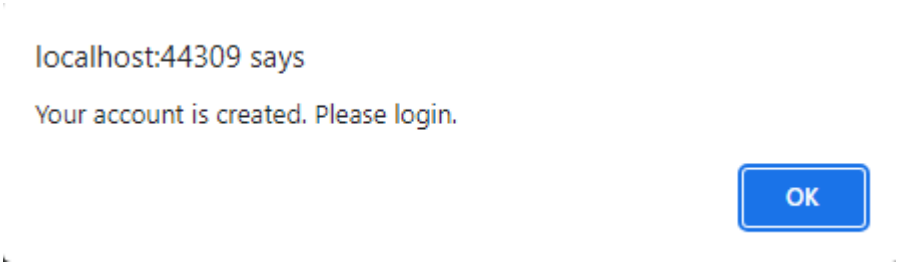


Figure 83: Prompt

Once the user has successfully registered a new account, the web page will prompt the user that his/her account has been created.

User Profile

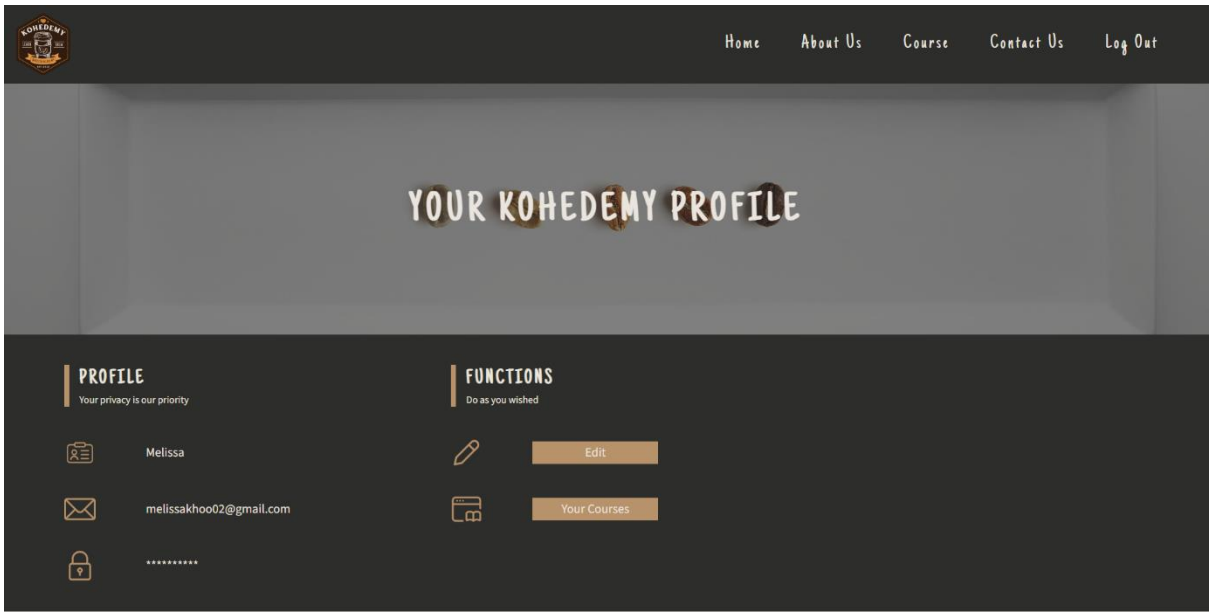


Figure 84: User Profile

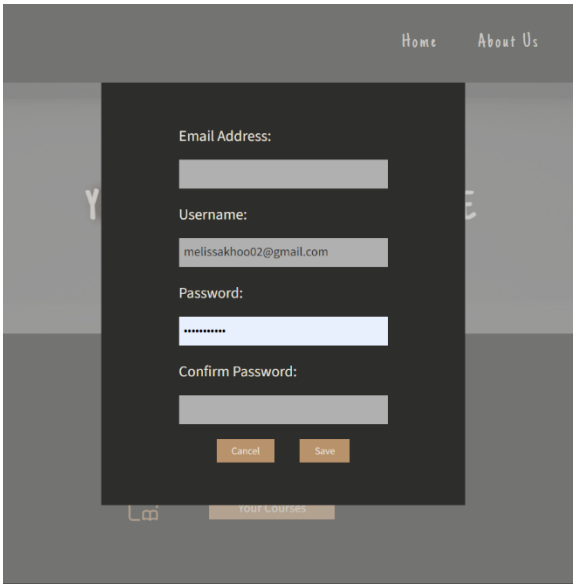


Figure 85: User profile (pop-up)

Above shows user interface of user profile and functions that user can be done in this web page, user can view and change email address and password. However, the user is not allowed to change the username as this is unique. After user changed the details in account, user have to click the “Save” button to store the new information.

Course Selection

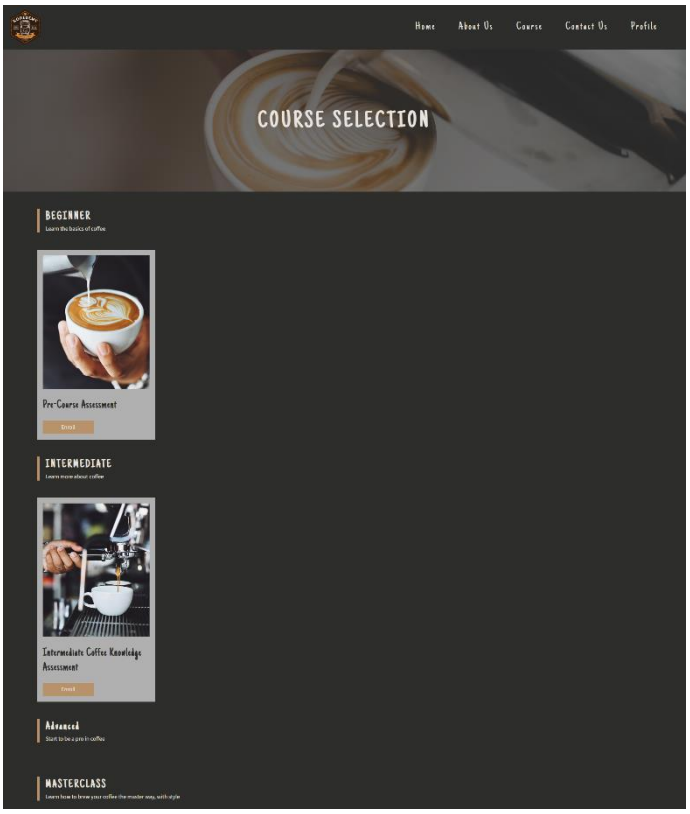


Figure 86: Course selection

Above depicted the level of courses provided in the course selection where there is beginner, intermediate, advanced and masterclass courses. Upon successful login, a user can enroll themselves in a course by clicking the "Enroll" button, which will then redirect them to the respective courses that the user intended to attend.

Enrolment

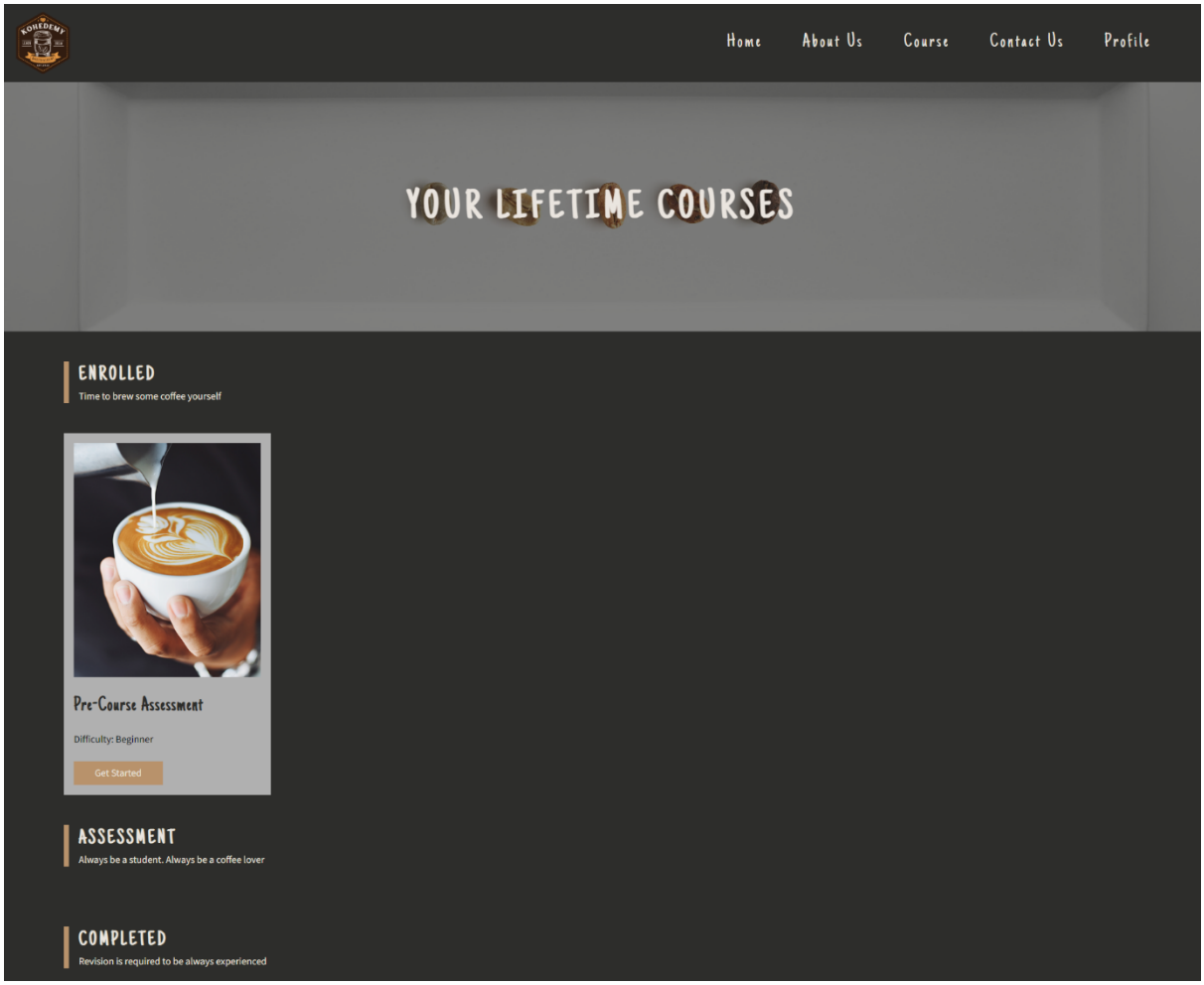


Figure 87: Enrolment page

After selecting a course level, the user will be prompted that it has successfully enrolled in the course and will be redirected to a new page that displays the details of their enrolment, including the course level, assessments, and completed course.

Learning material Page

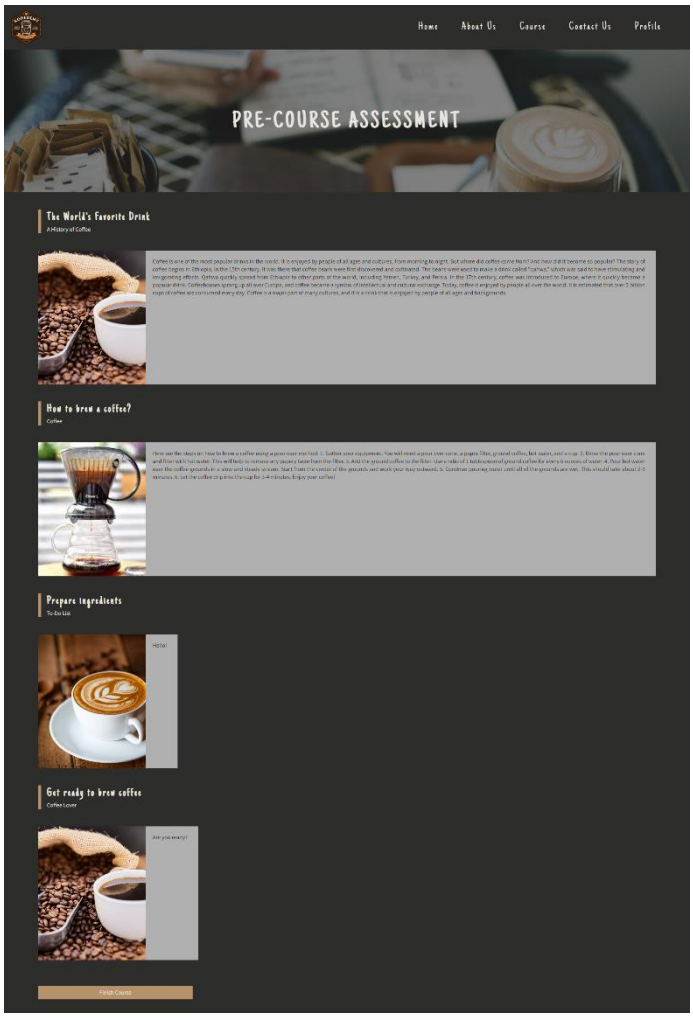


Figure 89: Learning material page

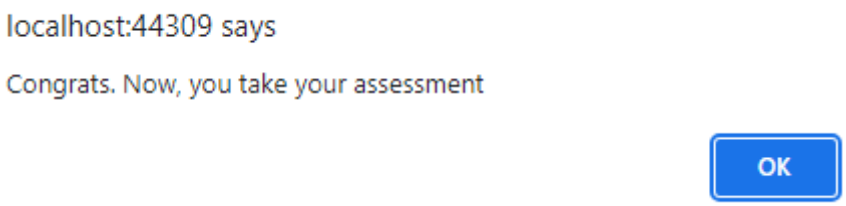


Figure 88: Upon completion of the course

The image above shows the learning materials page where user start their self-pace learning in the respective courses that the user enrolled. Upon completion of reading the materials, the user can proceed by clicking on the “Finish Course” and it will prompt a message to move on to that the assessment. =

Assessment

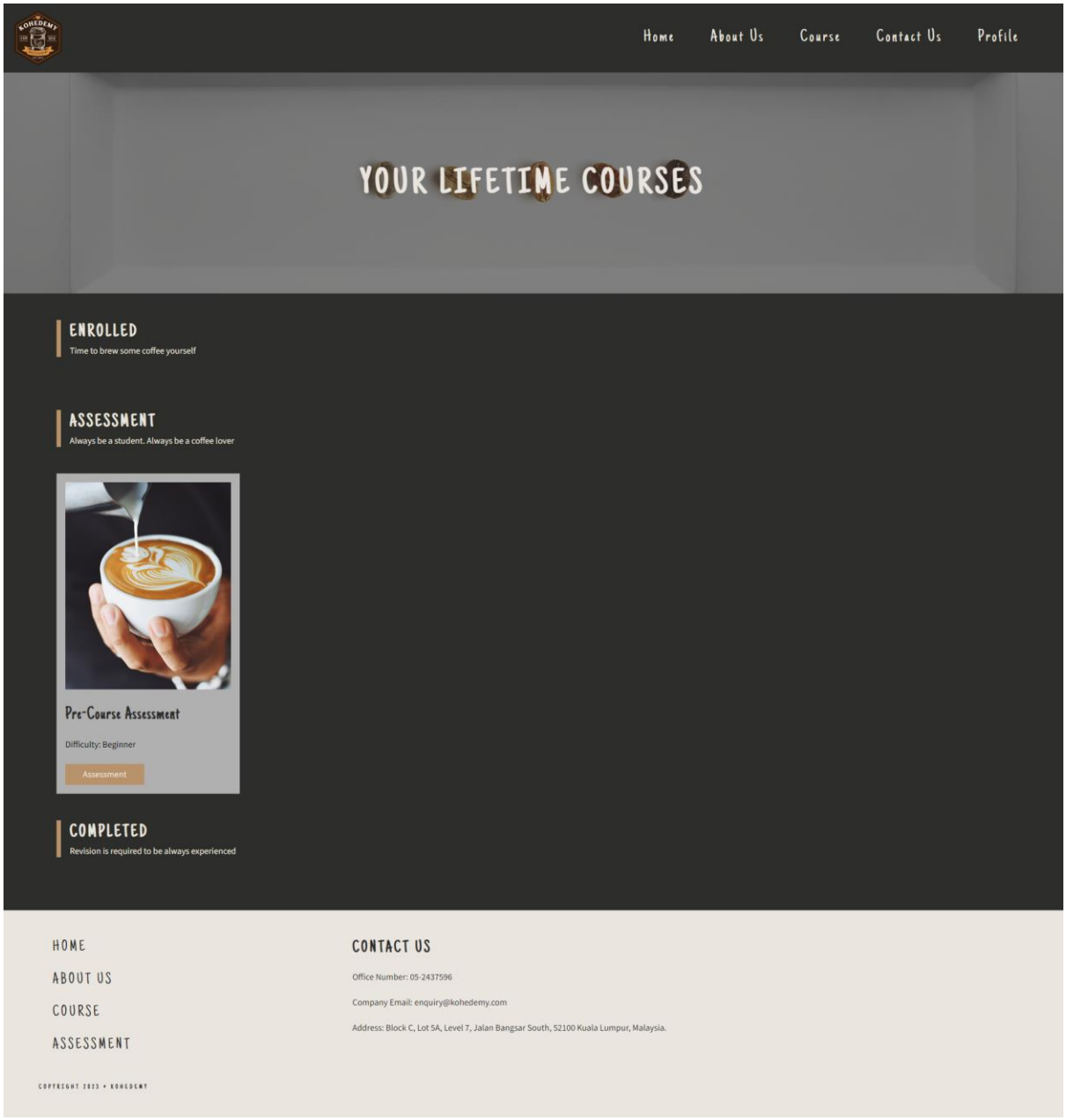


Figure 90: Assessment

The image shows that the user has completed the reading materials for the course and their course status has been updated to “Assessment”. The user is now required to take the assessment by clicking on the “Assessment” button.

In course assessment

Figure 91: In course assessment

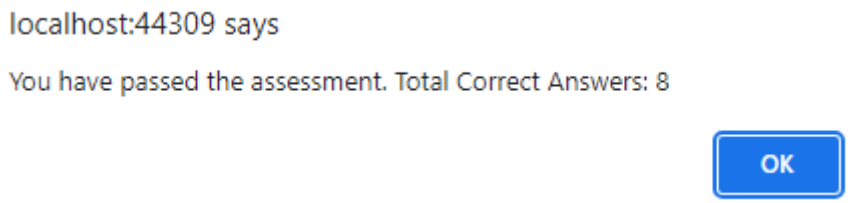


Figure 92: Prompt user the score

Upon completion of the self-paced learning course, the user is required to take a 10-question multiple choice assessment by selecting the radio button for the correct answer. This in course assessment is to test the user knowledge on brewing coffee. Once the user done the assessment, the user is required to click the “Submit Assessment” button in order to complete the whole course. After submitting the assessment, the user will be prompted with a message indicating the number of questions the user answered correctly.

Completed course

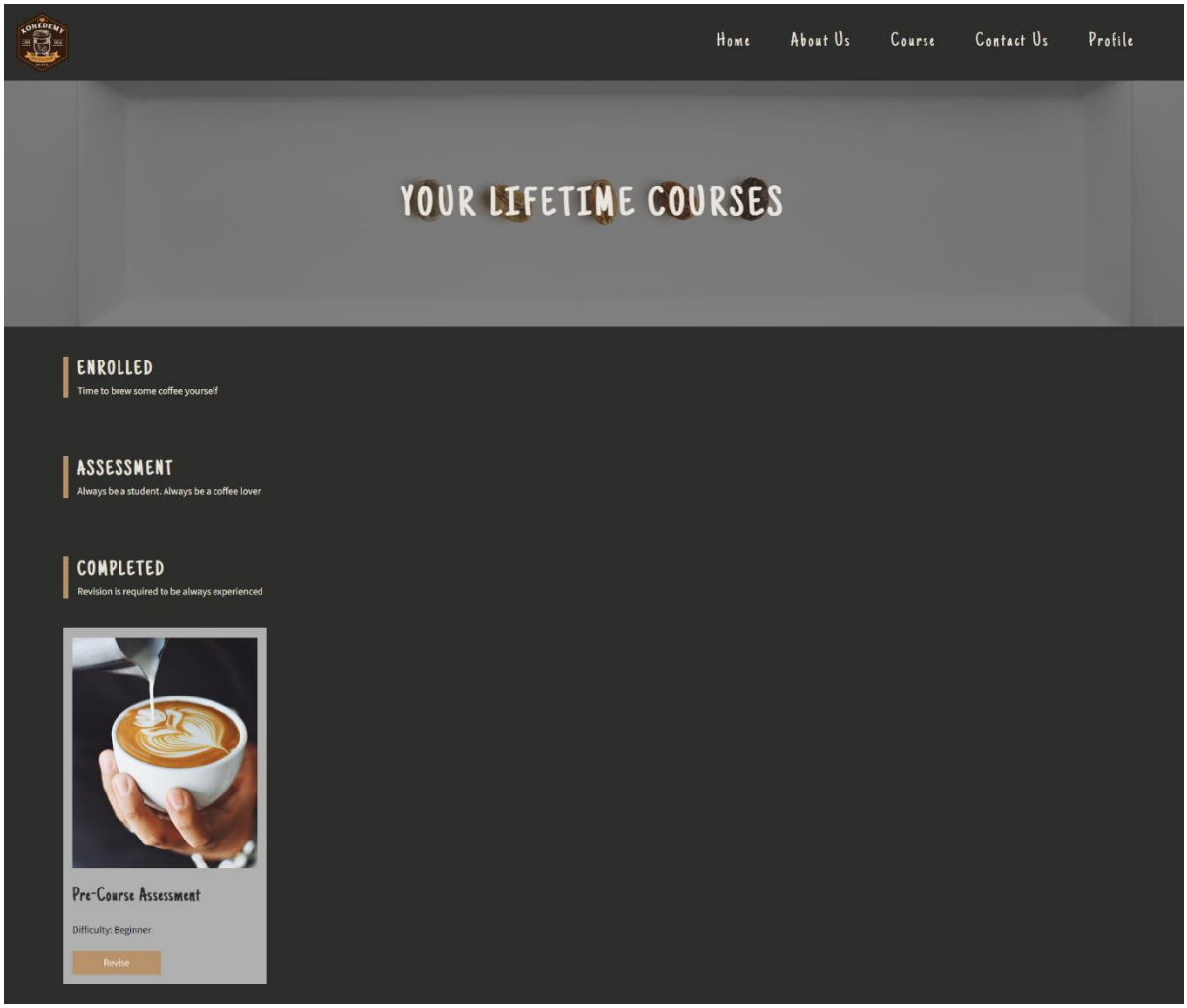


Figure 93: Completed course

The image shows that the user has completed the entire course and their course status has been updated to “Completed”. The user is no longer required to take the assessment, but the user is allow to revise back the reading materials provided to them.

Admin Profile Page

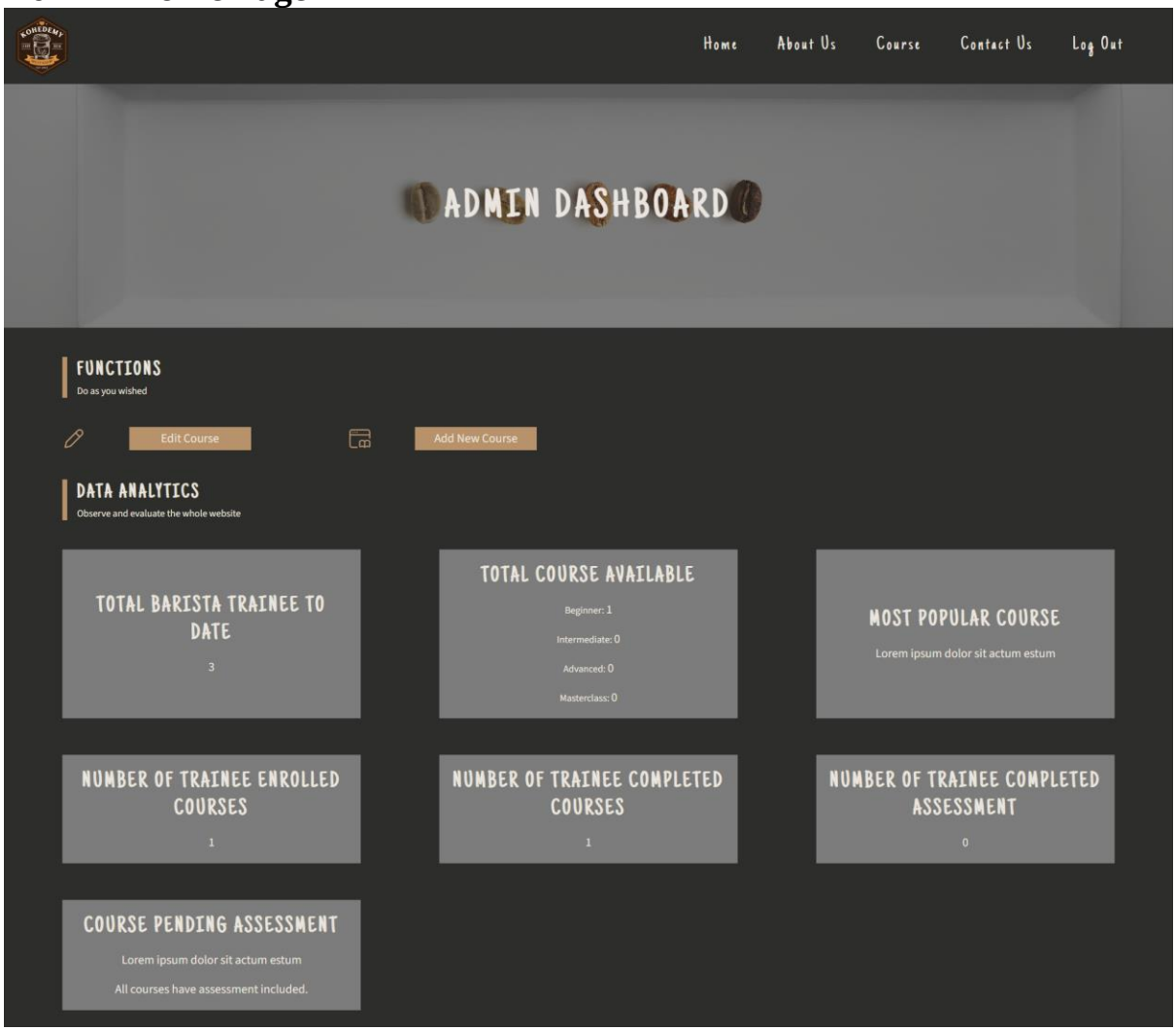


Figure 94: Admin Profile Page

Above shows the admin profile page where administrator manage all the user through this web page. Admin can create and edit course from this web page. Besides that, admin also can view the auto generated data from the data analytics section where the number of users sign up for the course and how many of the user attempted the assessment.

Create or Edit Course

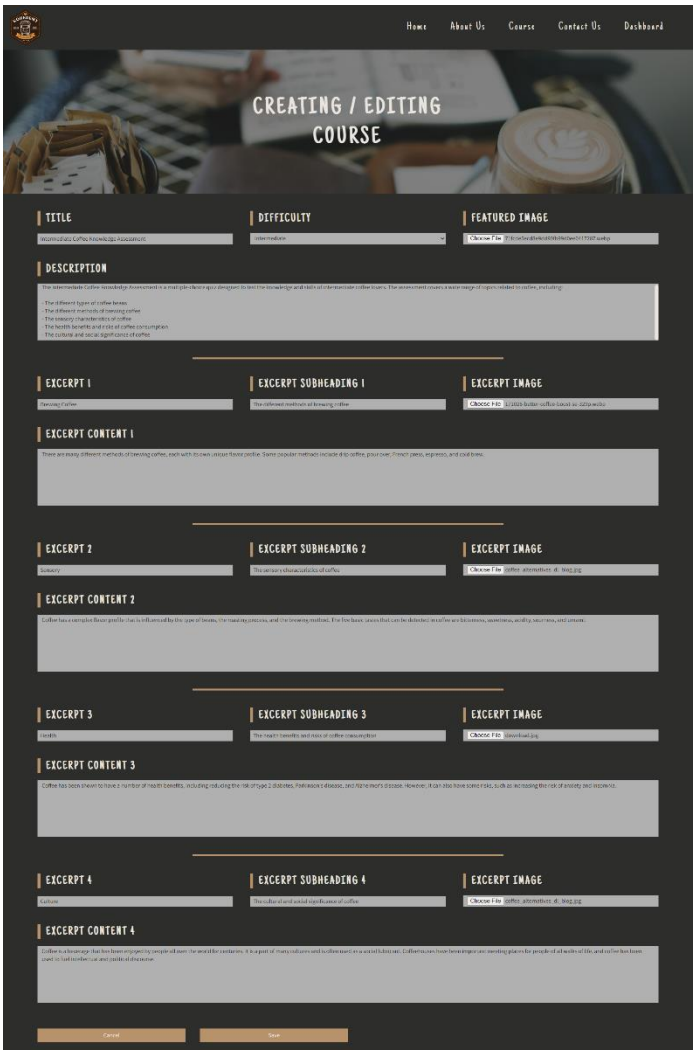


Figure 95: Create/Edit Course

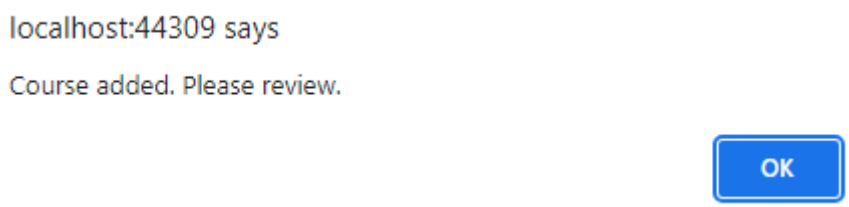


Figure 96: prompt user the course added

Once admin click onto the “Add New Course” the webpage will direct the admin to create a new course which require the admin to input the relevant information in each of the text fields provided. Once the admin input all the relevant information, then the admin can click on the “Save” button to store the information in the database and it will prompt the admin that the course added.

Course Management

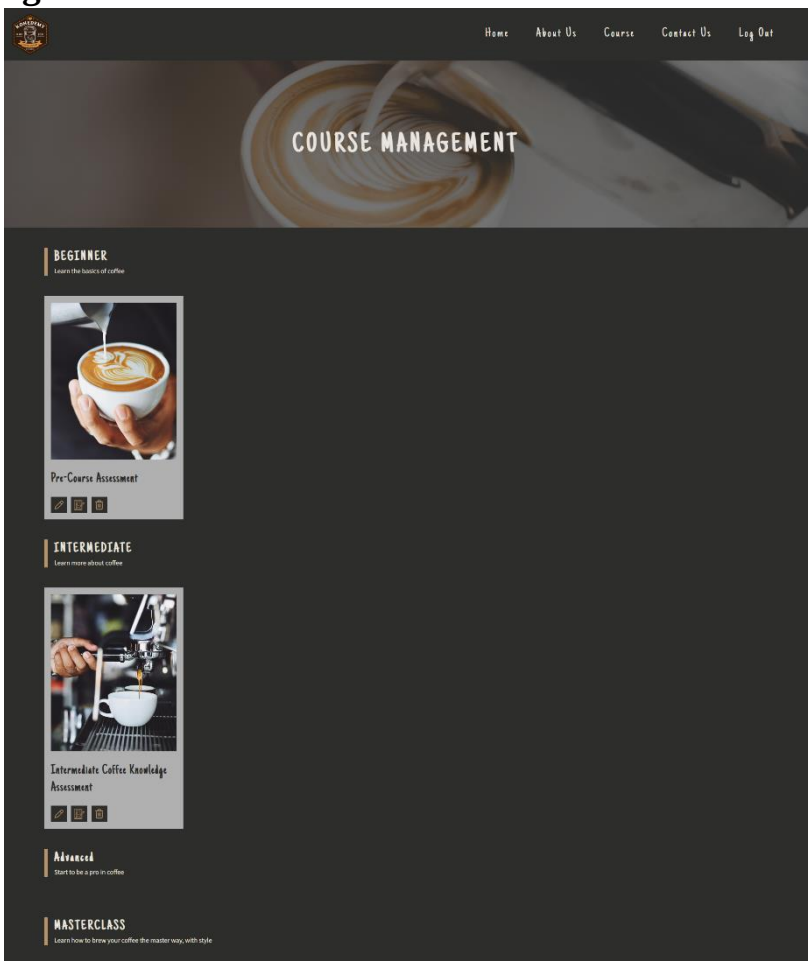


Figure 97: Course management

Once the course has been created, in the course management there will be display the courses that admin has added. Each card contains three functions which is edit course, create questions for assessment and delete course. It is applicable for advanced and masterclass courses too.

6.0 Conclusion

In conclusion, the developed web application represents a comprehensive self-paced learning platform dedicated to coffee brewing, effectively balancing content richness for informative purposes while ensuring a user-friendly and universally accessible interface. The most challenging part of this project was implementing the data and linking it to the SQL database. Future enhancements aim to elevate the assessment system by introducing more stimulating and interactive elements, including user feedback mechanisms, alongside the addition of a discussion forum and community chat room to foster knowledge exchange and connectivity among users passionate about coffee brewing

Appendix

Proposal Report

Group 1 Members:

a. Yip Zi Xian
b. Goh Min Xuan
c. Melissa Khoo Yen Yin
d. Lam Wai Yan

Website:

Kohedemy

Target user:

Coffee Lover, Coffee Aficionado (Javaphile)

Purpose:

To promote the actual and accurate ways of coffee brewing to potential users

Subject:

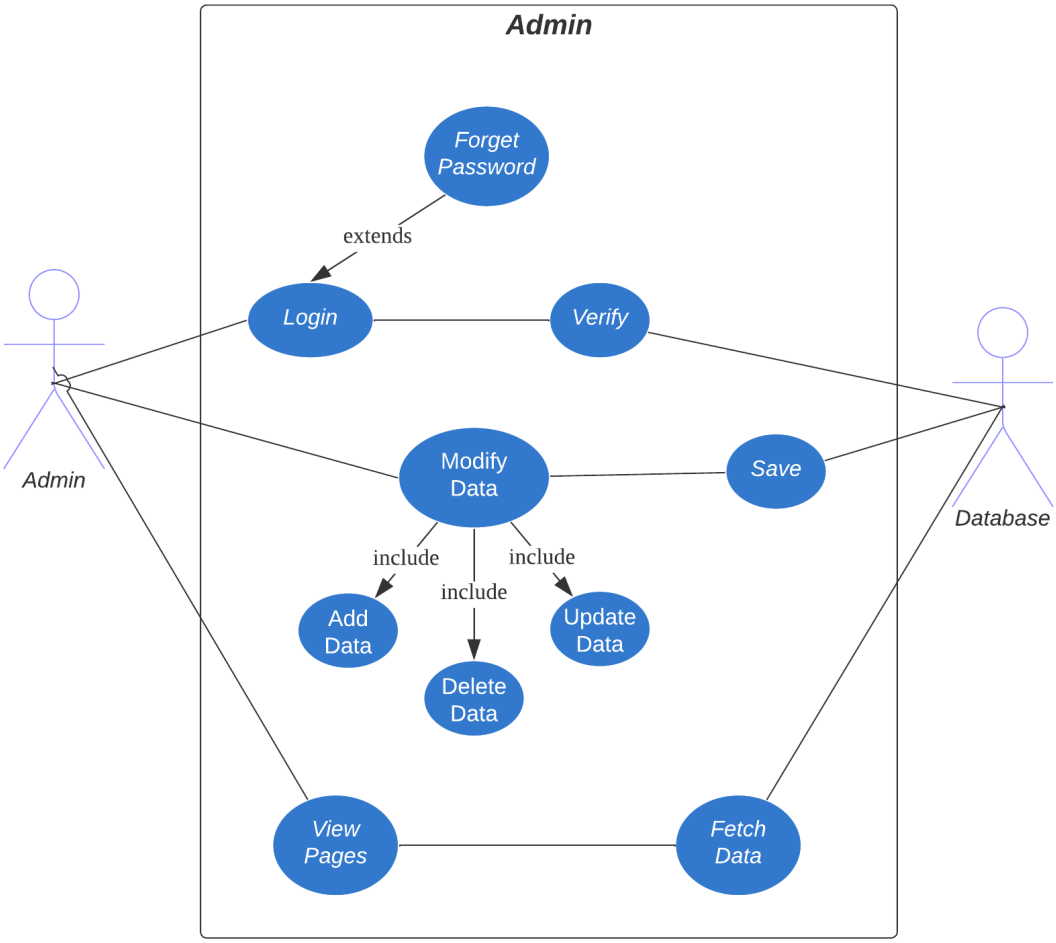
Coffee Brewing

Objectives:

To provide valuable information and techniques to enhance their coffee brewing skills

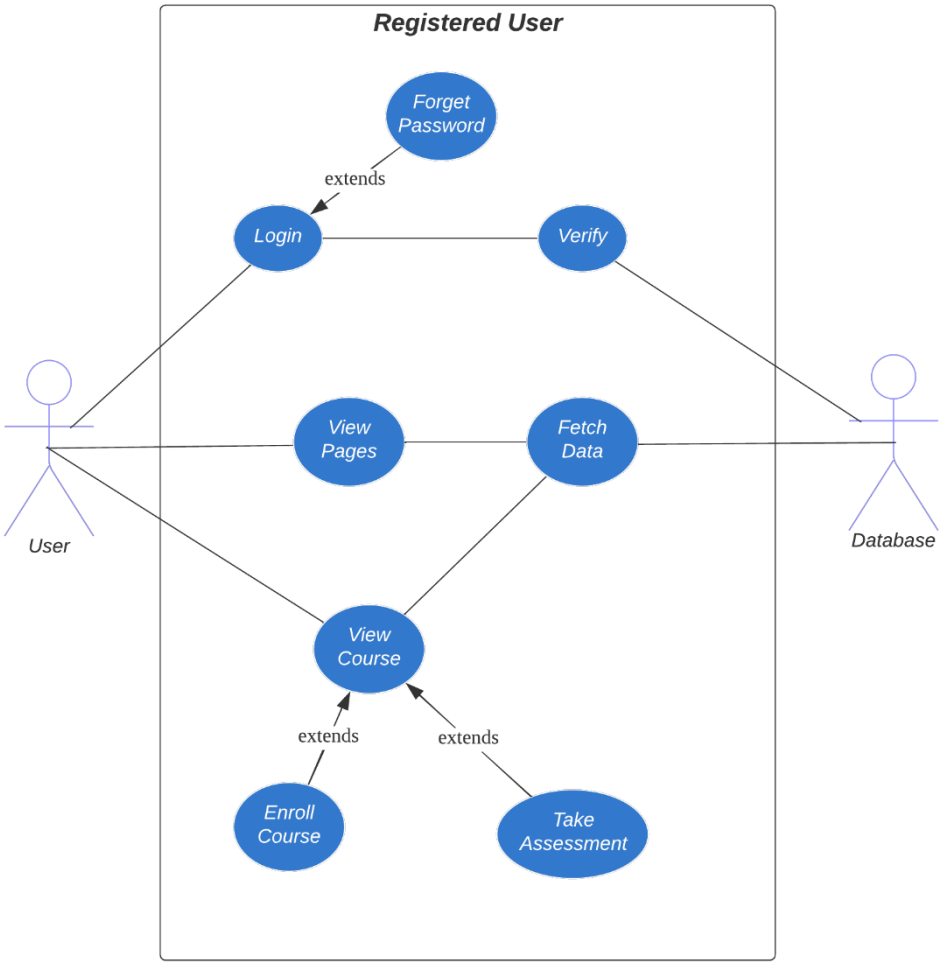
Mission:

To empower coffee lovers with proper methods to enhance their brewing skills.



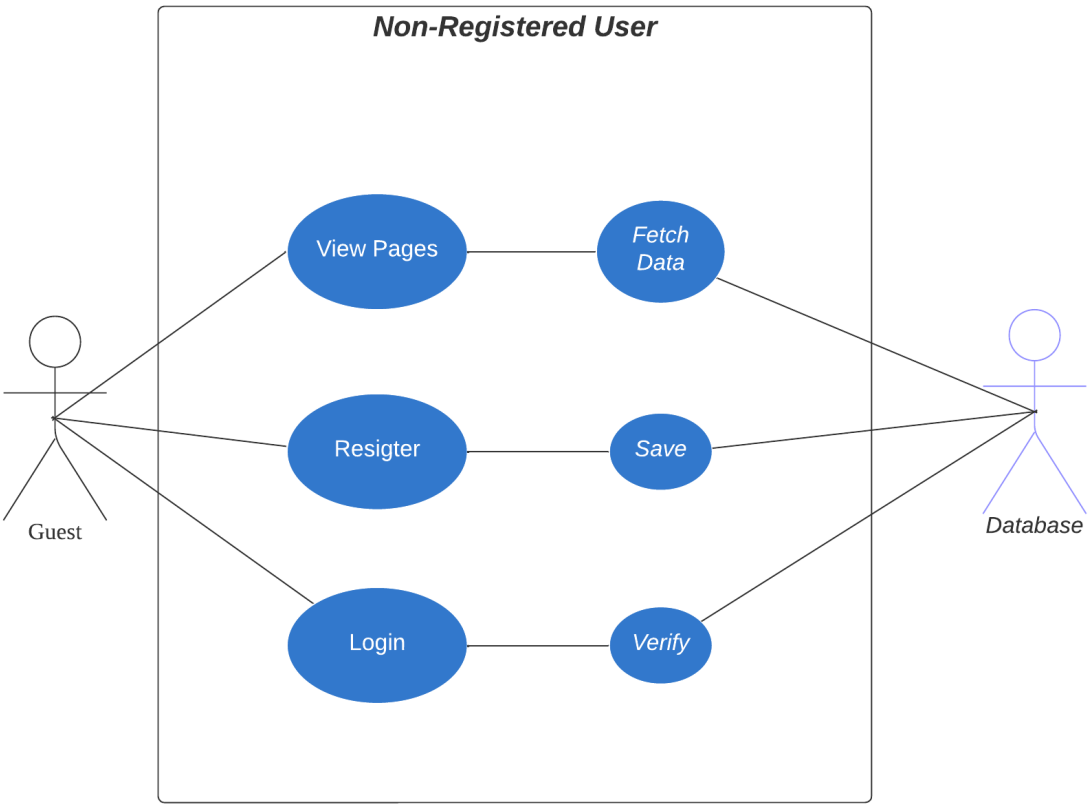
Admin

By default, admin will have only one account within the database. All admins will share the same account thus they will not need to register a new admin account. However, if admin forgets the password, it can be reset with admin verification. Admin will have different page view compared to other users because they have the permission to perform CRUD operations on the website.



Registered User

As a registered user, they could reset their password if they forgot when they are at the login page. Registered user can view all courses available on the website. Once registered user login to the page, user is enrolled to the course. Registered user also can take assessment once it completed the course.



Non-Registered User

For non-registered users or known as guest, they have limited access to the website. They can only view all pages, register as a new user or login as existing users. If they proceed to purchasing a course, they will be prompted to login and redirected to the login and register page.