الجامعـة الافتراضيّـة السوريّـة
SYRIAN VIRTUAL UNIVERSITY

SVU Student System

WEB PROGRAMMING II ITE_BWP501_ HW_S24

Tutor: Bassel Mustaffa alkhatib

Contributors by :

Mouhammad Nezar Alnajjar	**mouhammad_232243**	**C4**
Talal Awad	**Talal_196890**	**C2**

Hosting Service

The project is hosted on InfinityFree

Description

The SVU Student System is a dynamic and responsive web application designed to manage student records efficiently. Built using HTML, CSS, JavaScript, Bootstrap, PHP, and MySQL, it offers core CRUD operations along with advanced features like user authentication, search functionality, pagination, and AJAX form validation.

Features

- File Uploads: Upload and store student profile pictures with database references.
- Search Functionality: Filter students by name, email, or phone number.
- Pagination: Display student records in pages (e.g., 10 per page).
- AJAX Form Validation: Validate forms in real-time without page reloads.
- Responsive Design: Optimized for both desktop and mobile devices using Bootstrap.

Tech Stack

- Backend: PHP 8.0+, MySQL 5.7+
- Frontend: HTML, CSS, Bootstrap 5
- Interactivity: jQuery 3.6.0, AJAX

Installation

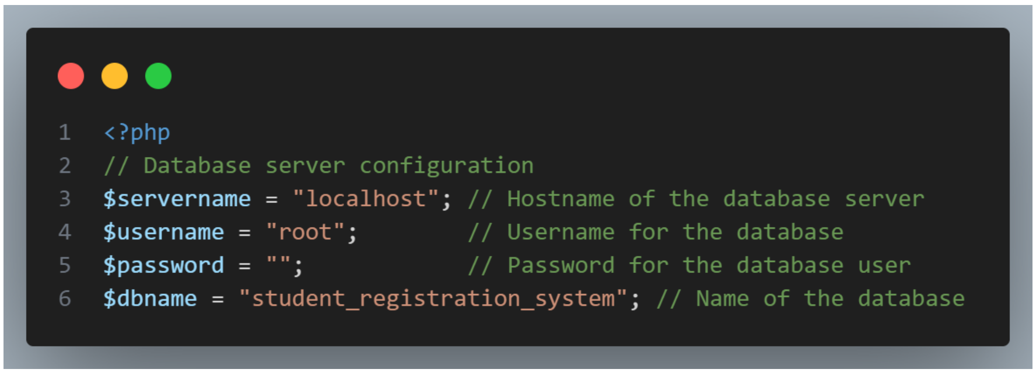
- Set Up XAMPP
    - Install XAMPP on your system.
    - Launch XAMPP and activate the Apache and MySQL modules.
- Project Setup
    - Extract the project files into a new folder, for example, "svu_std_sys".

- Place the folder in the following directory :

  C:\xampp\htdocs\svu_std_sys.

- Run the Application
  - Open your web browser and navigate to :

    http://localhost/svu_std_sys.
  - The system will automatically check if the required database exists:
  - If not, it will create the database and tables automatically.
  - Once completed, you will be redirected to the login/register page.
  - Note: If the database server hostname, username, or password differs from the default settings, update the config.php file with your own data:



```php
<?php
// Database server configuration
$servername = "localhost"; // Hostname of the database server
$username = "root";        // Username for the database
$password = "";            // Password for the database user
$dbname = "student_registration_system"; // Name of the database
```

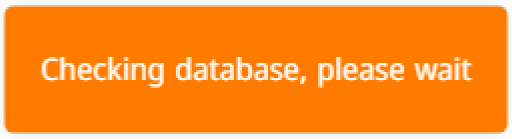
- **Dependencies**
  - Google Fonts:
    - Fonts:
      "Encode Sans Semi Condensed",
      "Noto Kufi Arabic"
    - Preconnect:

      https://fonts.googleapis.com
      https://fonts.gstatic.com
  - Bootstrap
    - CSS Framework: "assets/css/bootstrap.min.css"`
    - JavaScript Library: "assets/js/bootstrap.min.js"
    - Download from: https://getbootstrap.com/

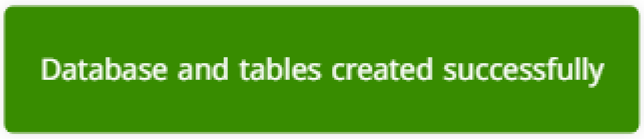
- jQuery
  - JavaScript Library: "assets/js/jquery-3.6.0.min.js"

Screenshots:

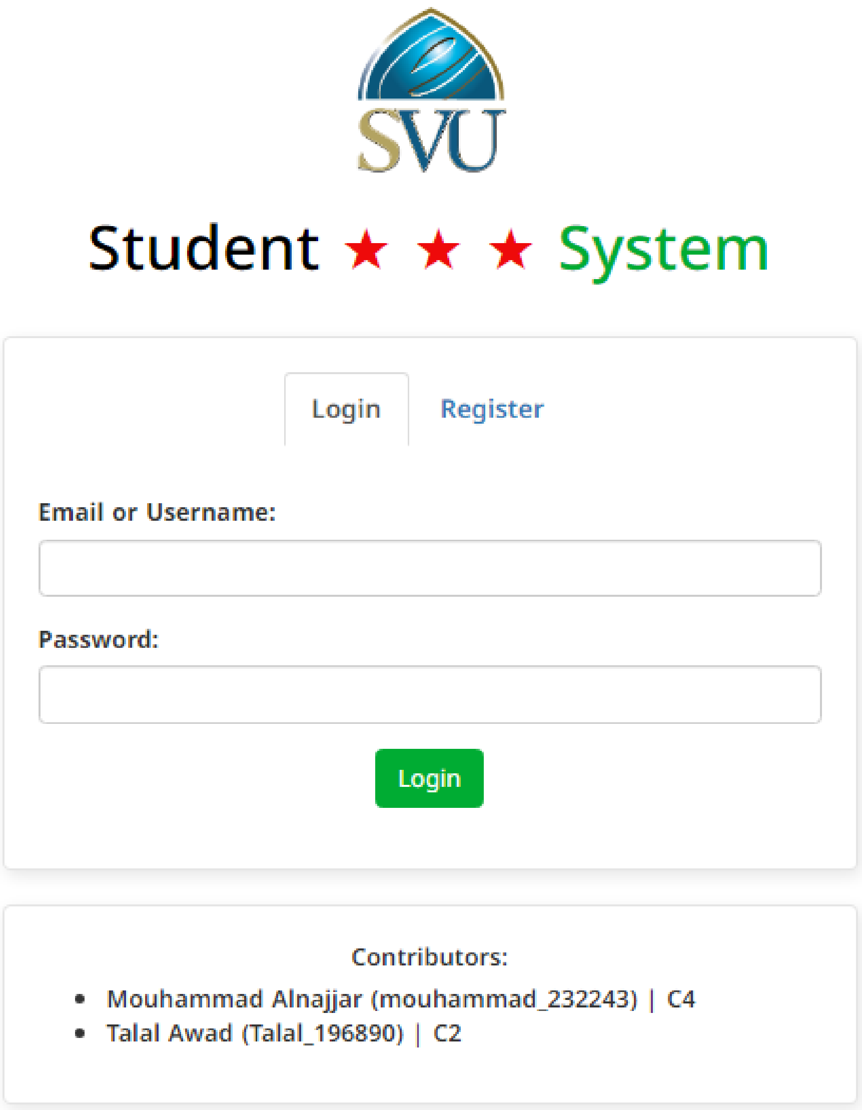
1. Checking Database:


Checking database, please wait

2. Database Created Message:


Database and tables created successfully

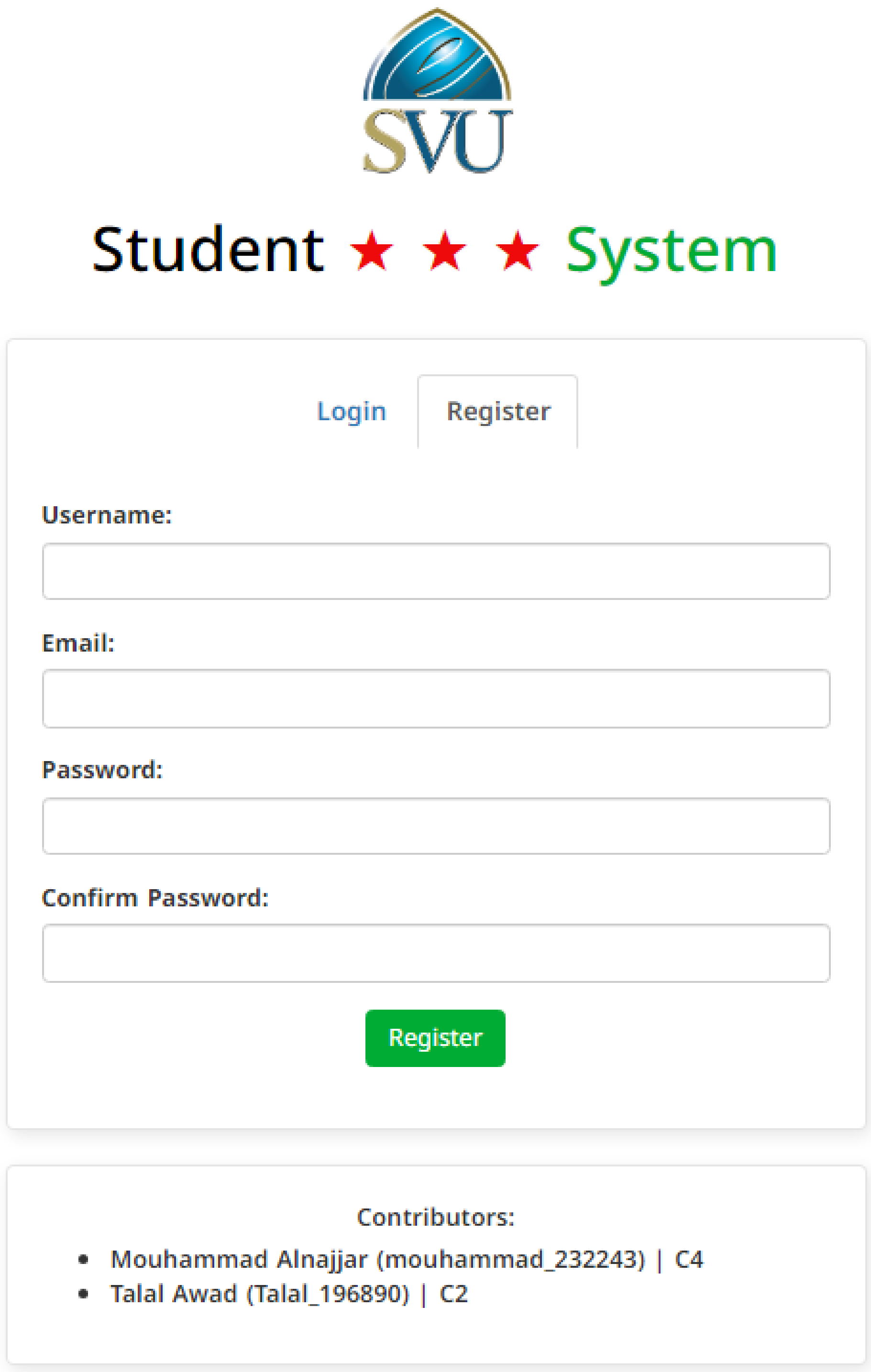
3. Login Section:



Contributors:

- Mouhammad Alnajjar (mouhammad_232243) | C4
- Talal Awad (Talal_196890) | C2

4. Register Section:

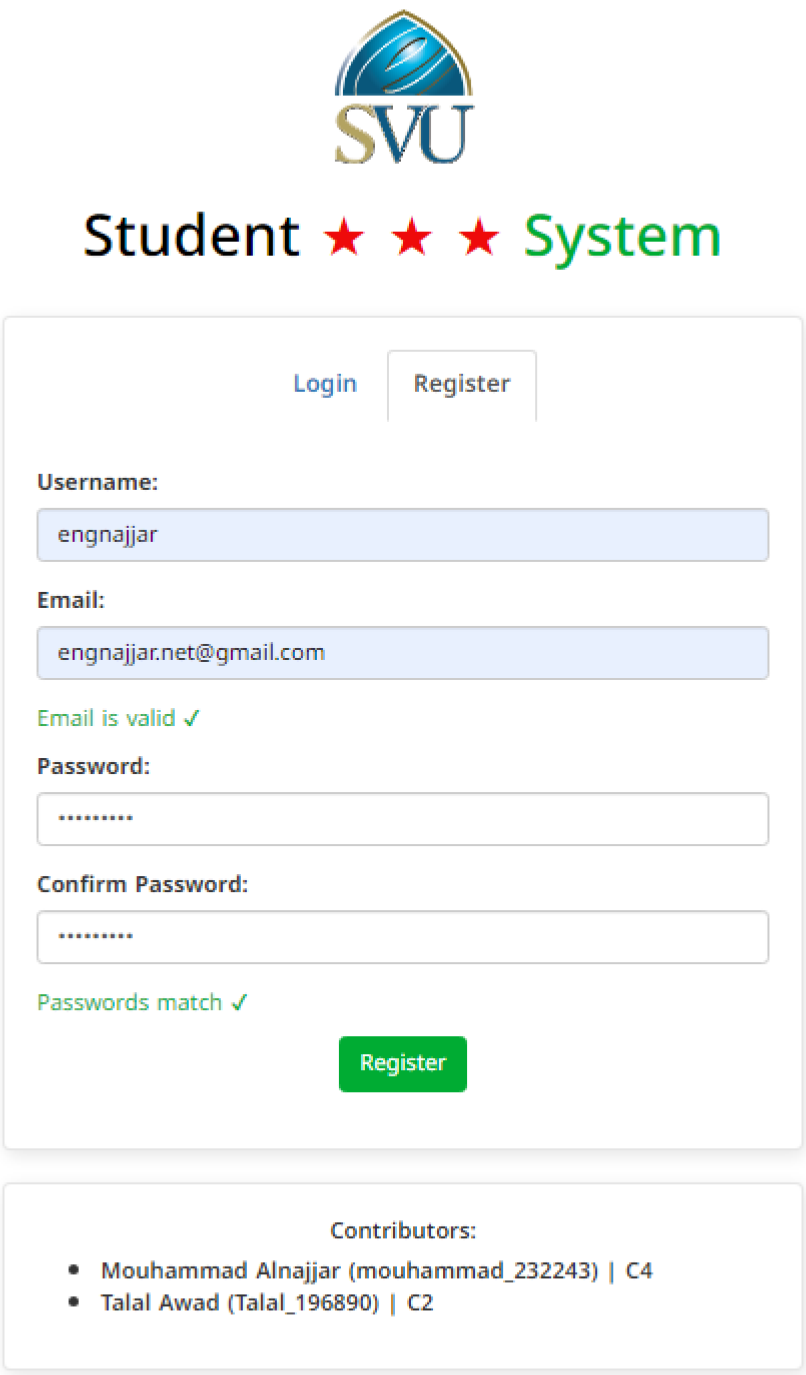


Student ★ ★ ★ System

Contributors:

- Mouhammad Alnajjar (mouhammad_232243) | C4
- Talal Awad (Talal_196890) | C2

5. Email Check & Password Match:



Username:

engnajjar

Email:

engnajjar.net@gmail.com

Email is valid ✓

Password:

•••••••••

Confirm Password:

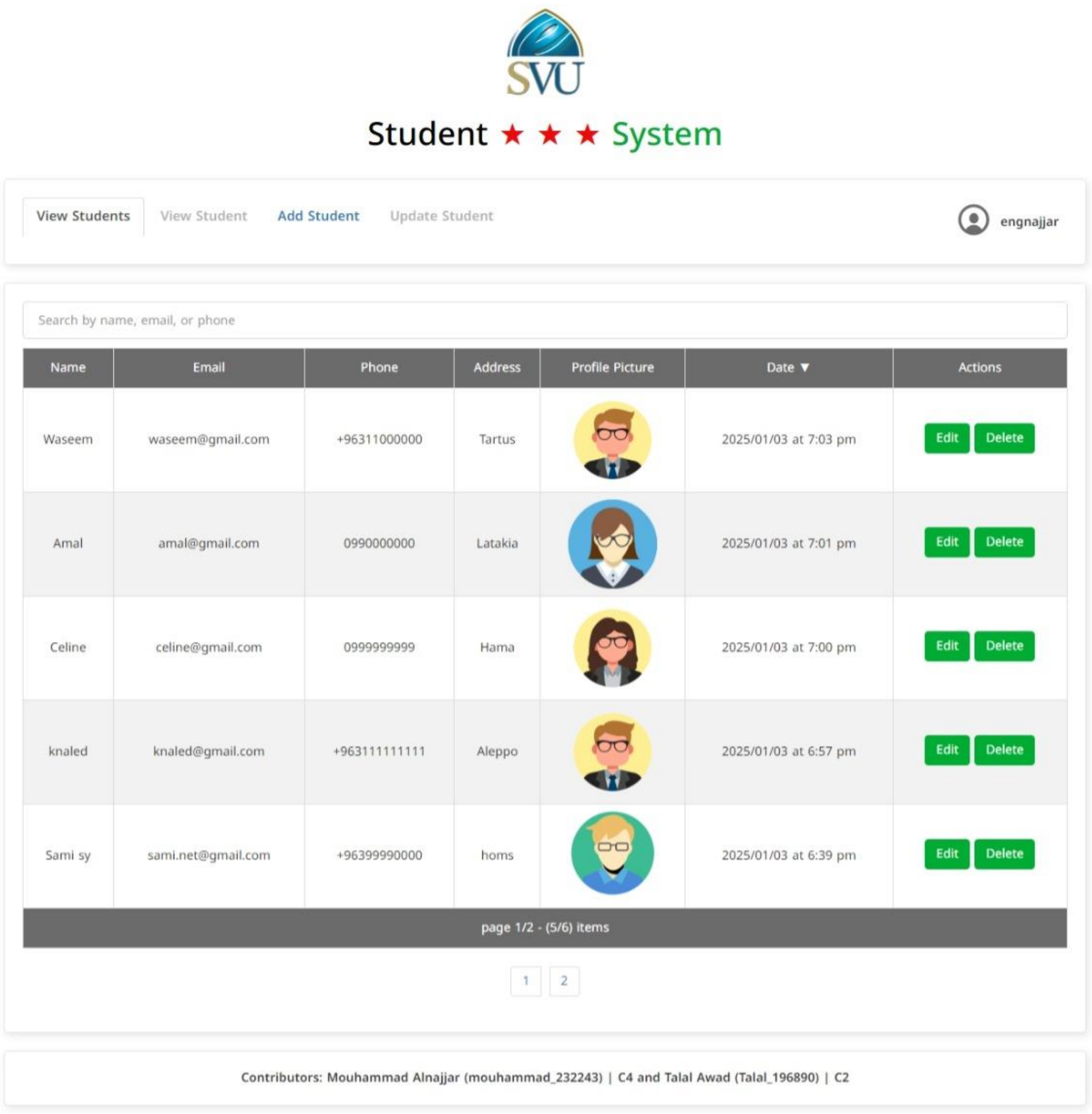
•••••••••

Passwords match ✓

Contributors:

- Mouhammad Alnajjar (mouhammad_232243) | C4
- Talal Awad (Talal_196890) | C2

6. Home Page - View Students Table:



Student ★ ★ ★ System

View Students | View Student | Add Student | Update Student | engnajjar

Search by name, email, or phone

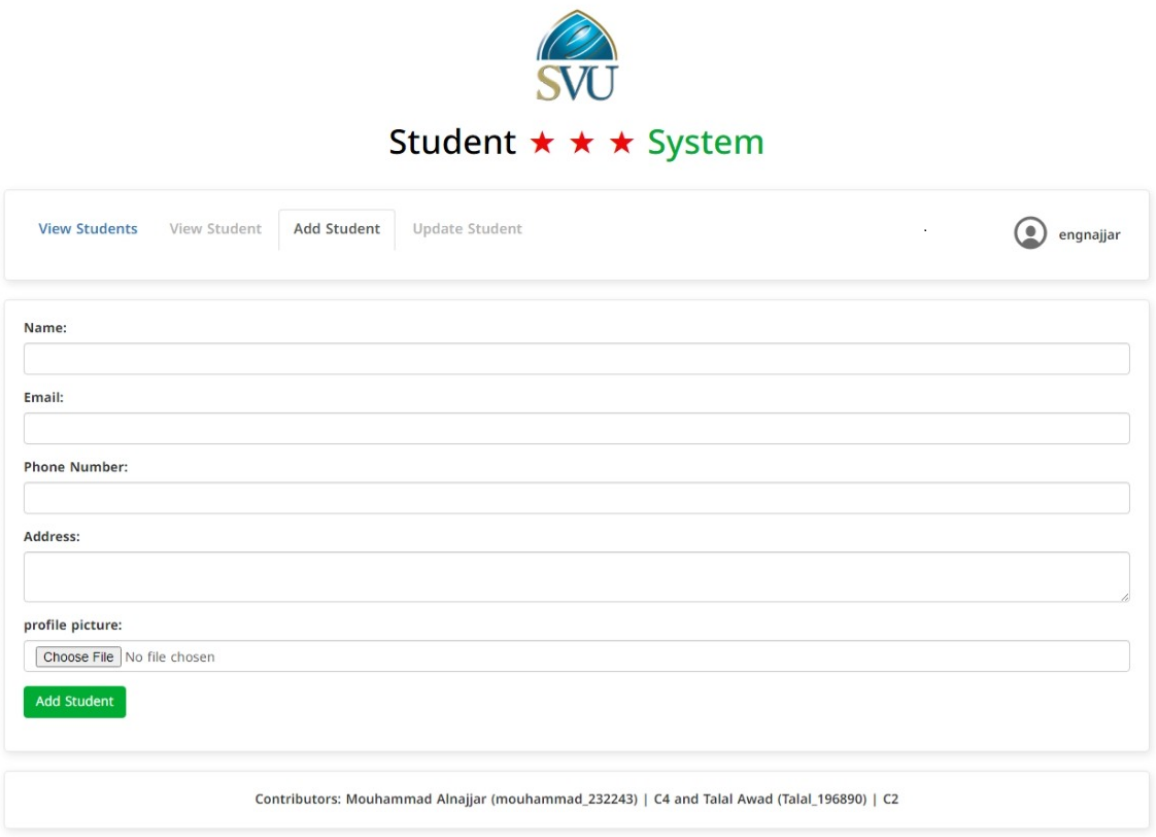
Name	Email	Phone	Address	Profile Picture	Date ▼	Actions
Waseem	waseem@gmail.com	+96311000000	Tartus		2025/01/03 at 7:03 pm	Edit Delete
Amal	amal@gmail.com	0990000000	Latakia		2025/01/03 at 7:01 pm	Edit Delete
Celine	celine@gmail.com	0999999999	Hama		2025/01/03 at 7:00 pm	Edit Delete
knaled	knaled@gmail.com	+963111111111	Aleppo		2025/01/03 at 6:57 pm	Edit Delete
Sami sy	sami.net@gmail.com	+96399990000	homs		2025/01/03 at 6:39 pm	Edit Delete

page 1/2 - (5/6) items

1 2

Contributors: Mouhammad Alnajjar (mouhammad_232243) | C4 and Talal Awad (Talal_196890) | C2

7. Add Student Section:



Student ★ ★ ★ System

View Students | View Student | Add Student | Update Student | engnajjar

Name:

Email:

Phone Number:

Address:

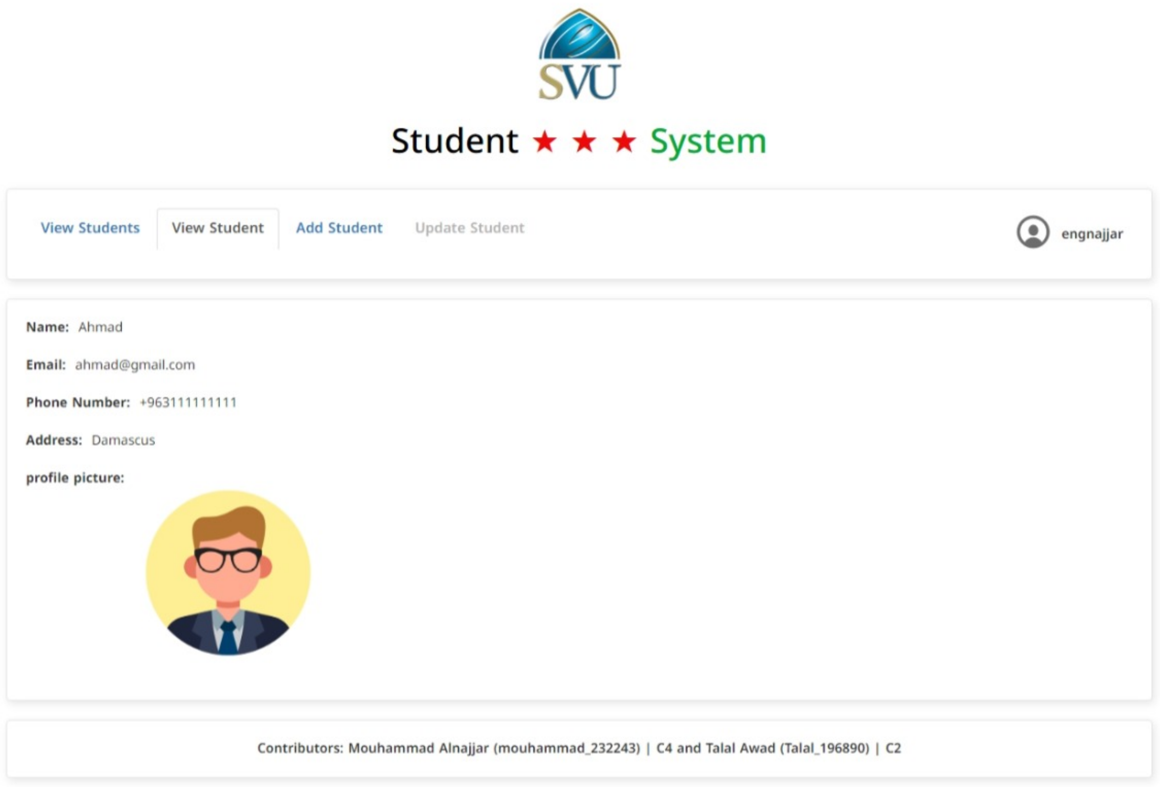
profile picture:

Choose File No file chosen

Add Student

Contributors: Mouhammad Alnajjar (mouhammad_232243) | C4 and Talal Awad (Talal_196890) | C2

8. View Student Info Section:



Name: Ahmad

Email: ahmad@gmail.com

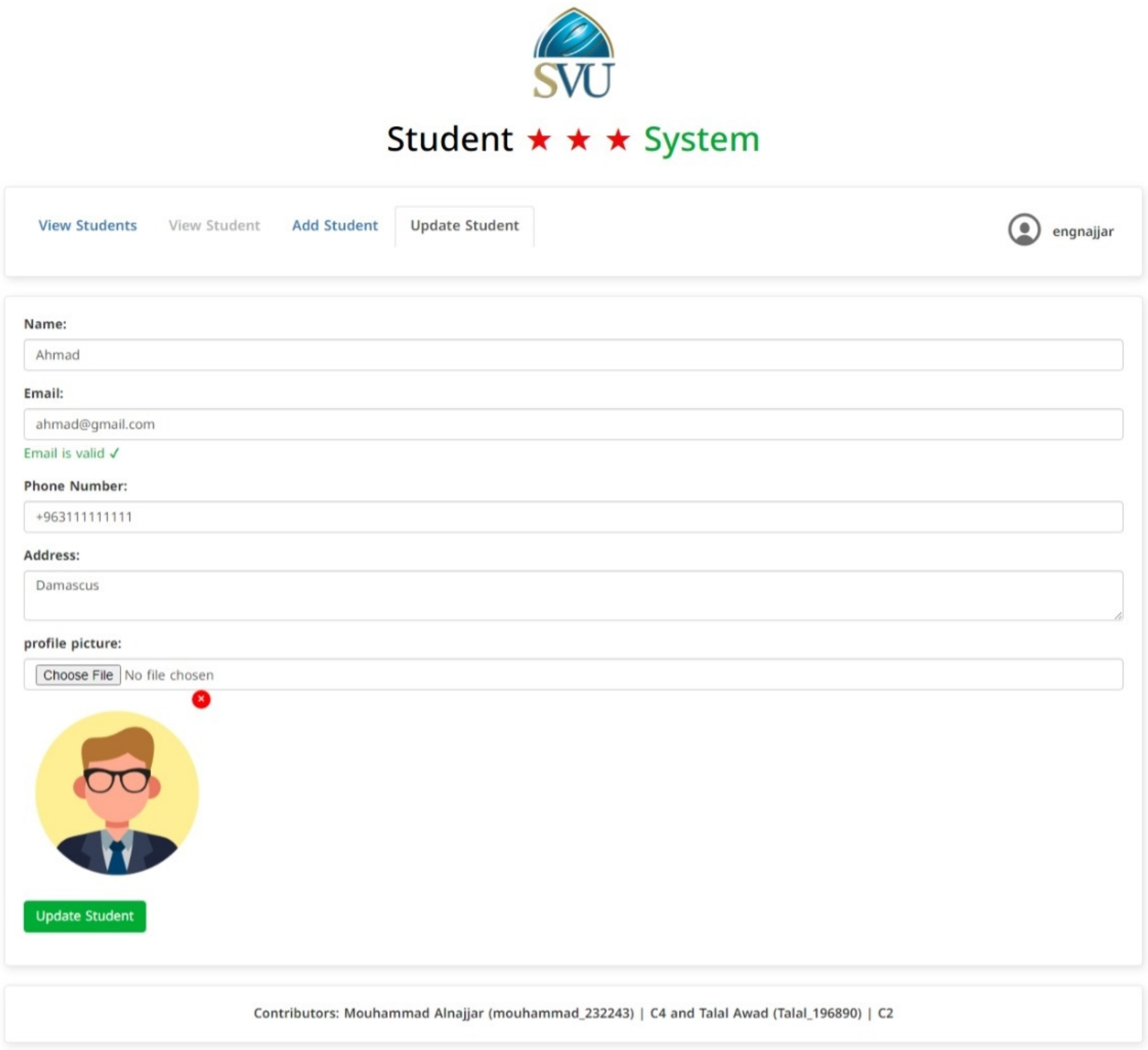
Phone Number: +963111111111

Address: Damascus

profile picture:

Contributors: Mouhammad Alnajjar (mouhammad_232243) | C4 and Talal Awad (Talal_196890) | C2

9. Update Student Info Section:



Name:
Ahmad

Email:
ahmad@gmail.com
Email is valid ✓

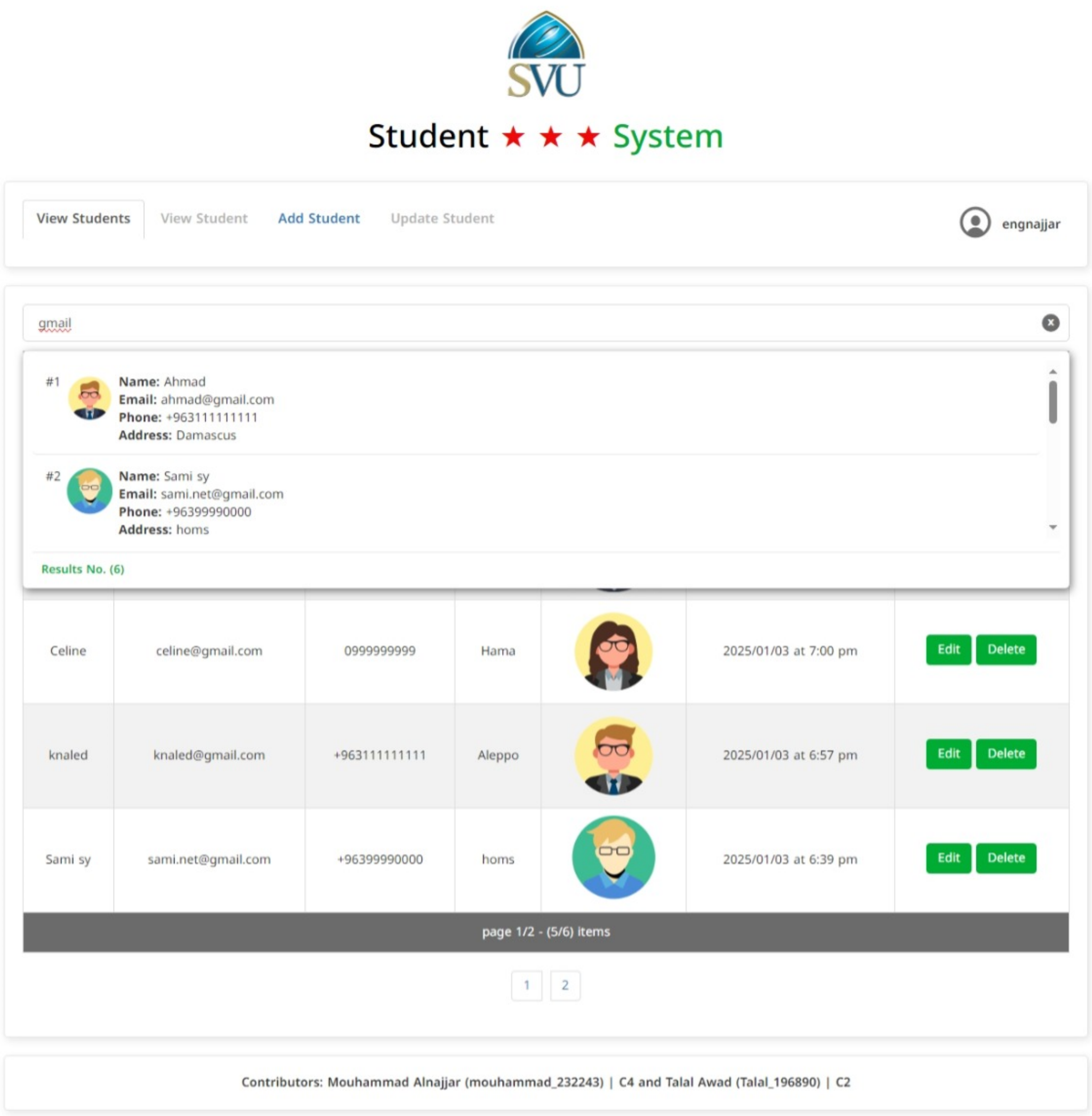
Phone Number:
+963111111111

Address:
Damascus

profile picture:
No file chosen

Contributors: Mouhammad Alnajjar (mouhammad_232243) | C4 and Talal Awad (Talal_196890) | C2

10.Search Result:



Student ★ ★ ★ System

gmail

#1 **Name:** Ahmad
Email: ahmad@gmail.com
Phone: +963111111111
Address: Damascus

#2 **Name:** Sami sy
Email: sami.net@gmail.com
Phone: +96399990000
Address: homs

Results No. (6)

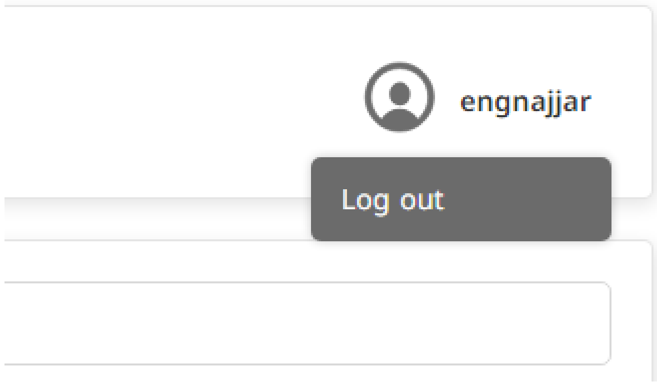
Celine	celine@gmail.com	0999999999	Hama		2025/01/03 at 7:00 pm	Edit Delete
knaled	knaled@gmail.com	+963111111111	Aleppo		2025/01/03 at 6:57 pm	Edit Delete
Sami sy	sami.net@gmail.com	+96399990000	homs		2025/01/03 at 6:39 pm	Edit Delete

page 1/2 - (5/6) items

1 2

Contributors: Mouhammad Alnajjar (mouhammad_232243) | C4 and Talal Awad (Talal_196890) | C2

11.User Avatar With Logout Option:

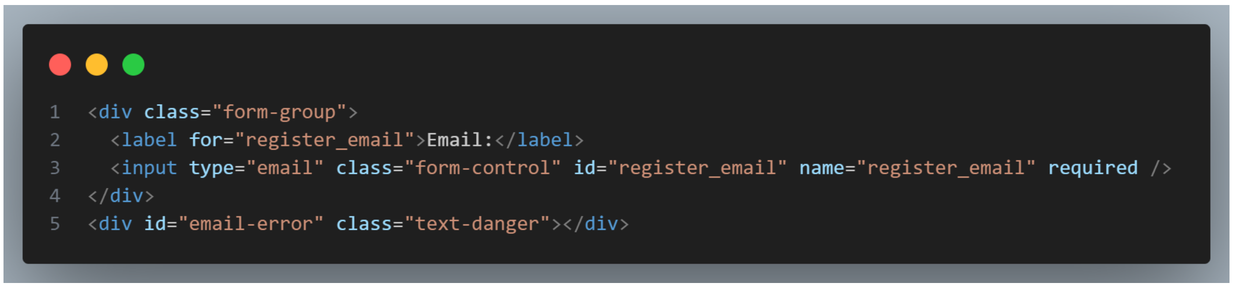


Code Explanation

How Email Validation is processed?

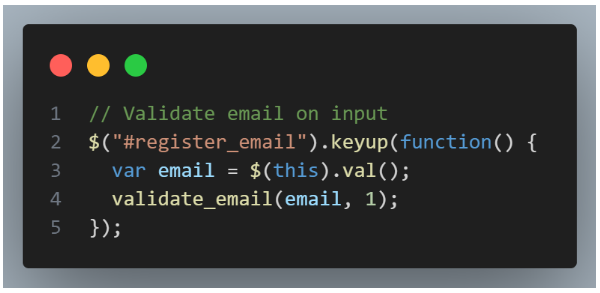
The provided code demonstrates how the email validation functionality works in your setup, both on the frontend using JavaScript and on the backend using PHP.

1. Using this input element with type="email" ensures that the browser performs basic email validation before submission. It requires the entered value to match the pattern of a valid email format:



```html
<div class="form-group">
  <label for="register_email">Email:</label>
  <input type="email" class="form-control" id="register_email" name="register_email" required />
</div>
<div id="email-error" class="text-danger"></div>
```

2. Real-time Validation by javascript:

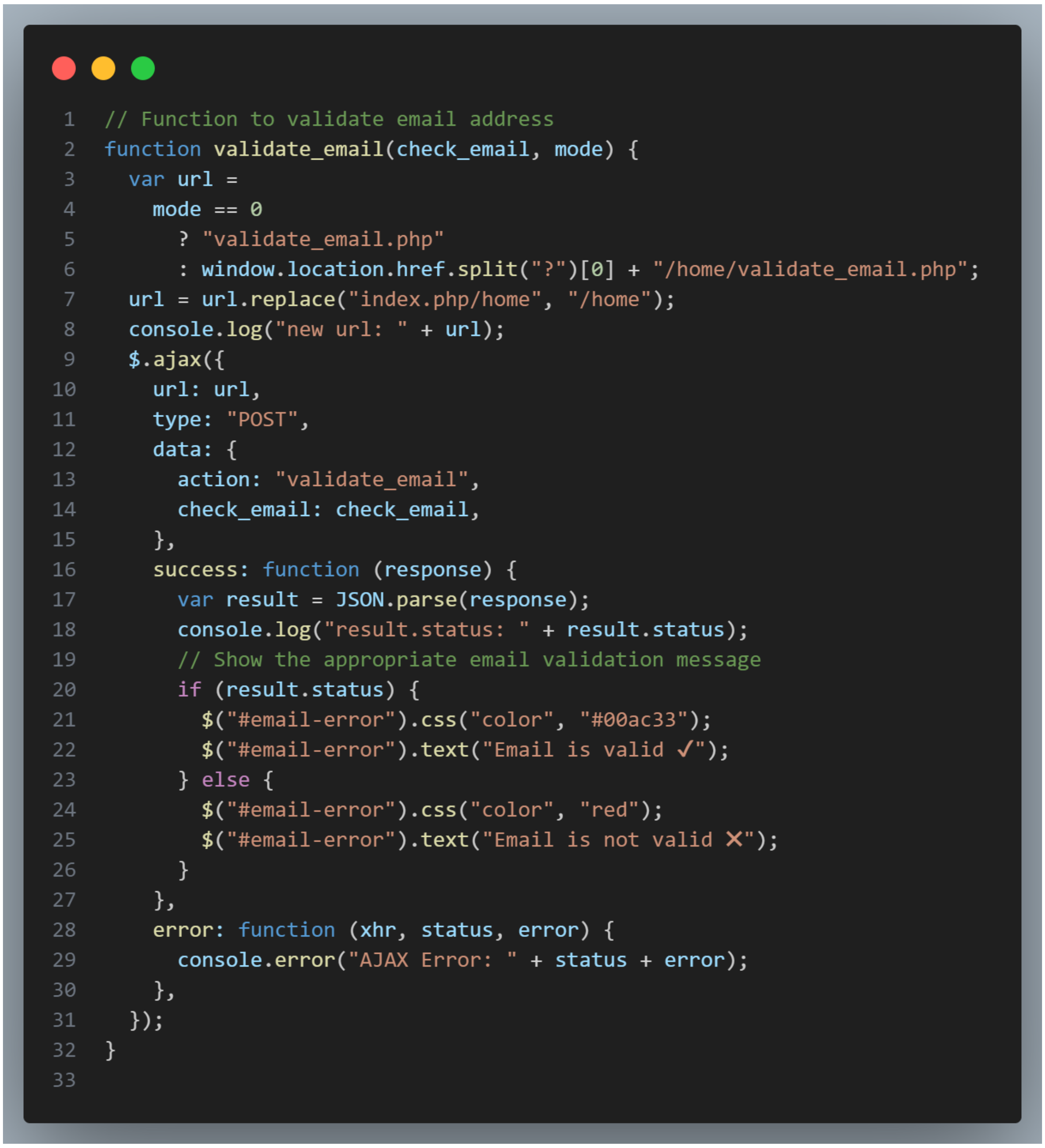


```javascript
// Validate email on input
$("#register_email").keyup(function() {
  var email = $(this).val();
  validate_email(email, 1);
});
```

On every keyup event in the email field, the validate_email function is called with the current input value and mode (1):

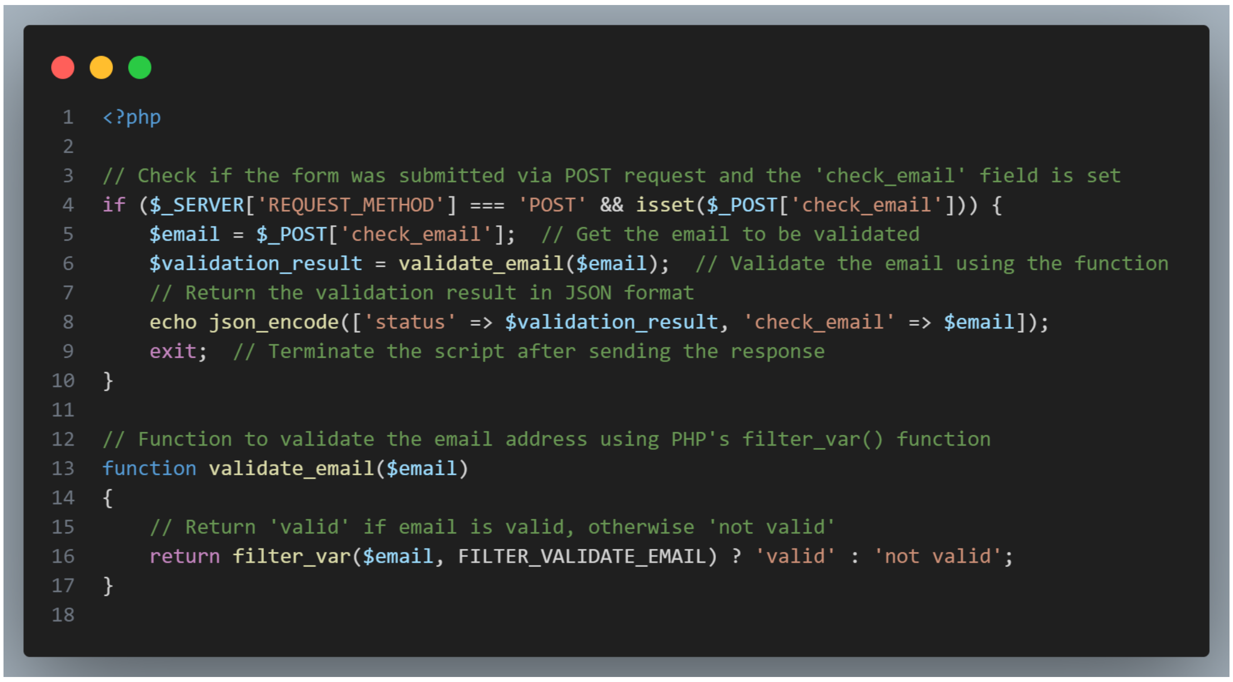
mode in validate_email function refer to the url of "validate_email.php" which is located in "/home/" folder so when we call validate_email function from a form inside "/home/" folder we use mode=0 such as validation email in (add_student_form.php, update_student_form.php), when calling the function out of "/home/" folder we use mode=1 such as validation email in register form.

3. The validate_email JavaScript Function sends the email to the server for validation via AJAX to "validate_email.php":



```javascript
// Function to validate email address
function validate_email(check_email, mode) {
  var url =
    mode == 0
      ? "validate_email.php"
      : window.location.href.split("?")[0] + "/home/validate_email.php";
  url = url.replace("index.php/home", "/home");
  console.log("new url: " + url);
  $.ajax({
    url: url,
    type: "POST",
    data: {
      action: "validate_email",
      check_email: check_email,
    },
    success: function (response) {
      var result = JSON.parse(response);
      console.log("result.status: " + result.status);
      // Show the appropriate email validation message
      if (result.status) {
        $("#email-error").css("color", "#00ac33");
        $("#email-error").text("Email is valid ✓");
      } else {
        $("#email-error").css("color", "red");
        $("#email-error").text("Email is not valid ✗");
      }
    },
    error: function (xhr, status, error) {
      console.error("AJAX Error: " + status + error);
    },
  });
}

```

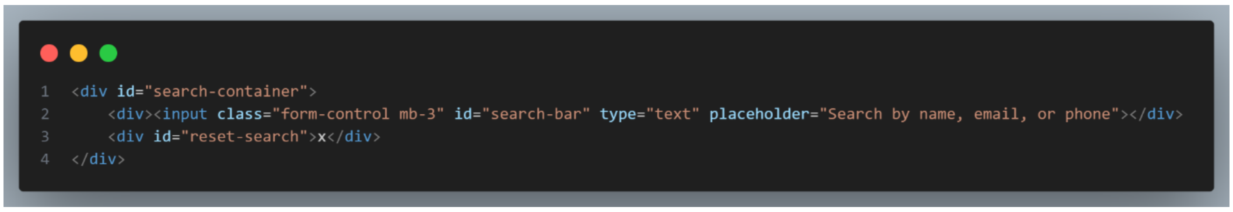
4. Backend Validation using "validate_email.php: The server validates the email using PHP's filter_var:



```php
<?php

// Check if the form was submitted via POST request and the 'check_email' field is set
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['check_email'])) {
    $email = $_POST['check_email'];  // Get the email to be validated
    $validation_result = validate_email($email);  // Validate the email using the function
    // Return the validation result in JSON format
    echo json_encode(['status' => $validation_result, 'check_email' => $email]);
    exit;  // Terminate the script after sending the response
}

// Function to validate the email address using PHP's filter_var() function
function validate_email($email)
{
    // Return 'valid' if email is valid, otherwise 'not valid'
    return filter_var($email, FILTER_VALIDATE_EMAIL) ? 'valid' : 'not valid';
}

```

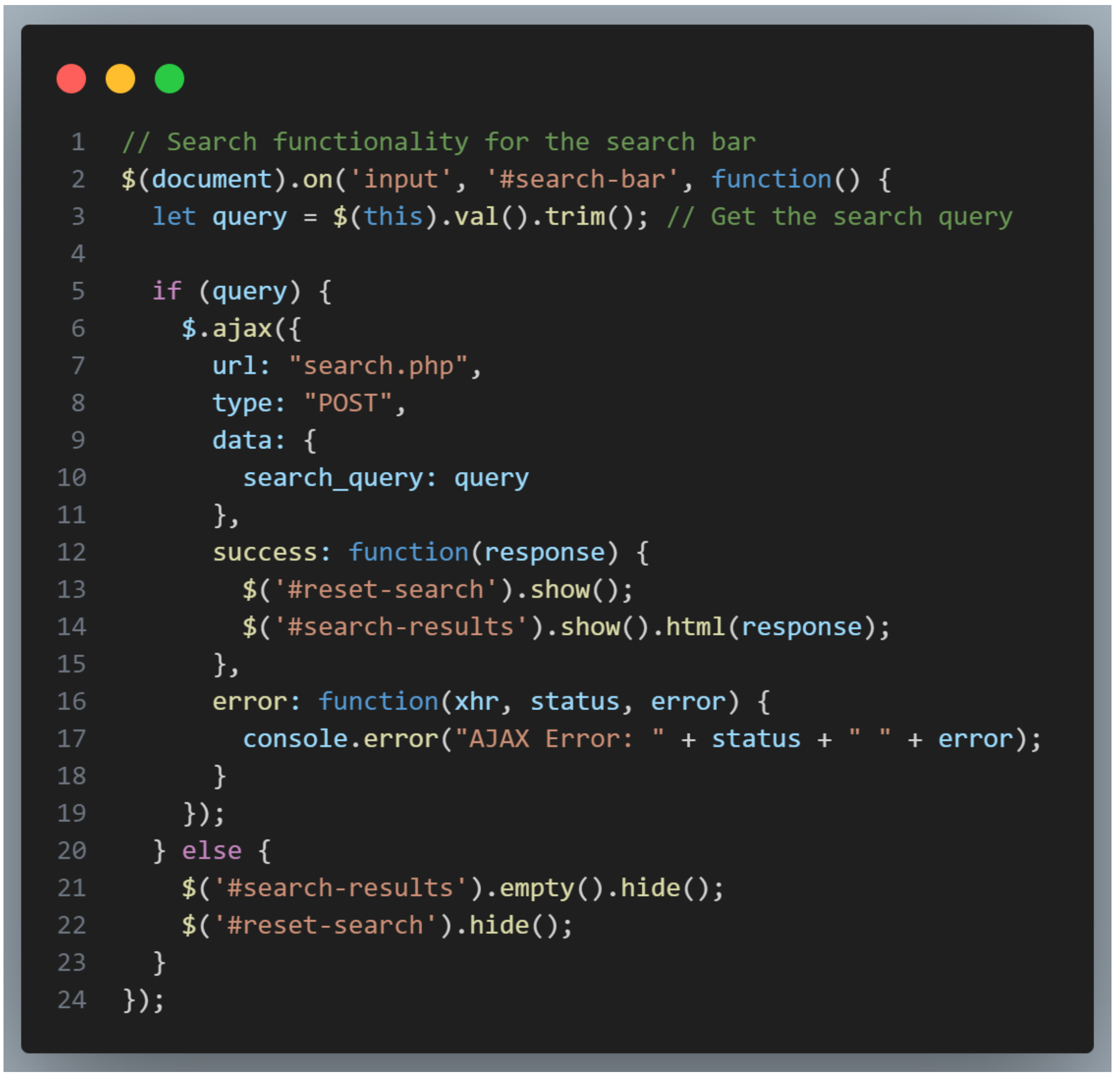
How Search works?

1. Search input in the search bar triggers an input event. Here is the structure of input field:



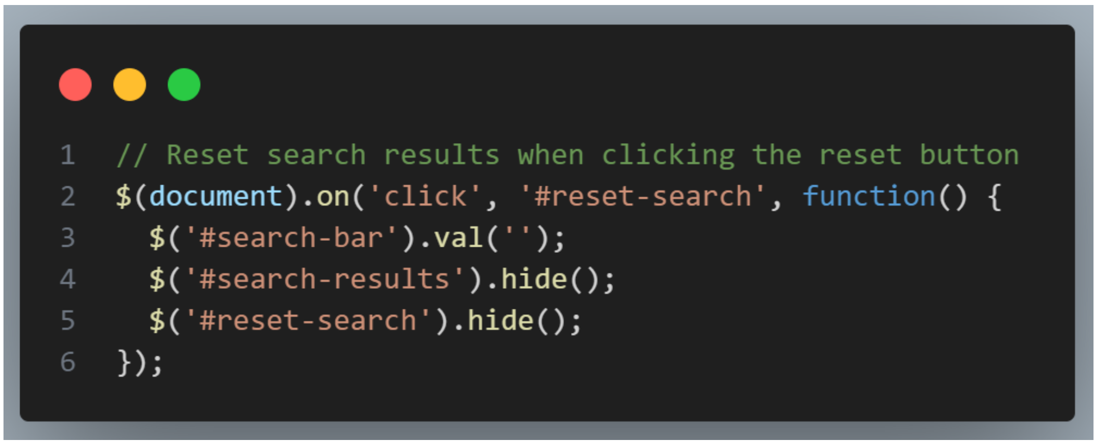
```html
<div id="search-container">
    <div><input class="form-control mb-3" id="search-bar" type="text" placeholder="Search by name, email, or phone"></div>
    <div id="reset-search">x</div>
</div>
```

2. **AJAX Request Handling For Search:** An AJAX request is made to search.php, sending the search query. Here's the relevant jQuery code:



```javascript
// Search functionality for the search bar
$(document).on('input', '#search-bar', function() {
  let query = $(this).val().trim(); // Get the search query

  if (query) {
    $.ajax({
      url: "search.php",
      type: "POST",
      data: {
        search_query: query
      },
      success: function(response) {
        $('#reset-search').show();
        $('#search-results').show().html(response);
      },
      error: function(xhr, status, error) {
        console.error("AJAX Error: " + status + " " + error);
      }
    });
  } else {
    $('#search-results').empty().hide();
    $('#reset-search').hide();
  }
});
```

3. PHP on the server processes the query, retrieves matching records from the database, and sends back the results as HTML.
4. The results are displayed in the #search-results container, and the reset button becomes visible.
5. User can reset the search by clicking the reset button, clearing the results and hiding the button. The reset button is handled with the following code:

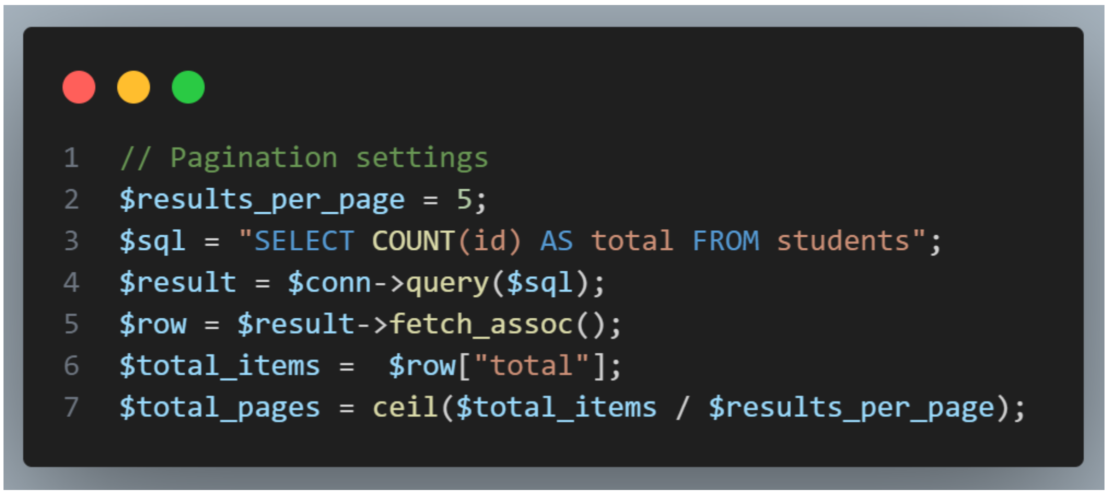


```javascript
// Reset search results when clicking the reset button
$(document).on('click', '#reset-search', function() {
  $('#search-bar').val('');
  $('#search-results').hide();
  $('#reset-search').hide();
});
```

How Pagination works?

Pagination splits the list of students into smaller, more manageable pages. This is essential for improving performance and usability when dealing with large datasets.

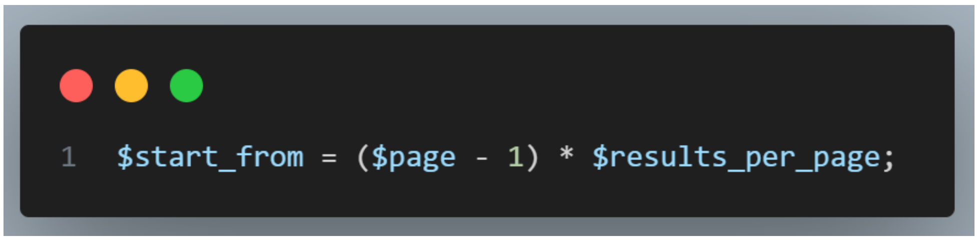
1. Determine Total Pages: In view_students.php, the total number of student records is fetched:



```php
// Pagination settings
$results_per_page = 5;
$sql = "SELECT COUNT(id) AS total FROM students";
$result = $conn->query($sql);
$row = $result->fetch_assoc();
$total_items =  $row["total"];
$total_pages = ceil($total_items / $results_per_page);
```

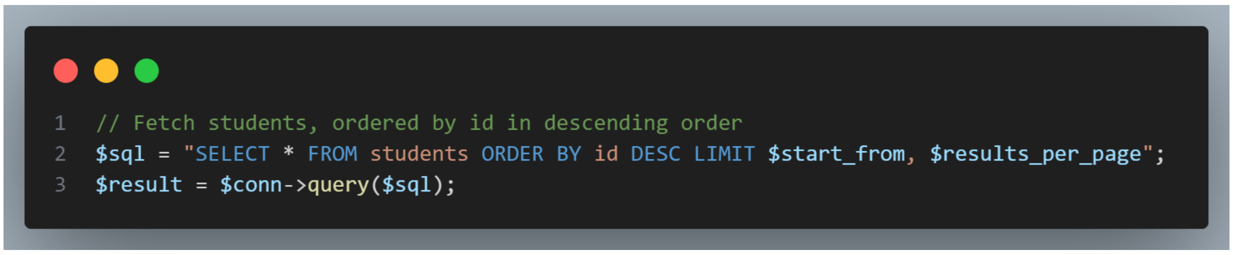
This calculation divides the total number of records by the number of results per page ($results_per_page = 5).

2. **Limit Results for the Current Page**: For the current page, the starting record index is calculated:



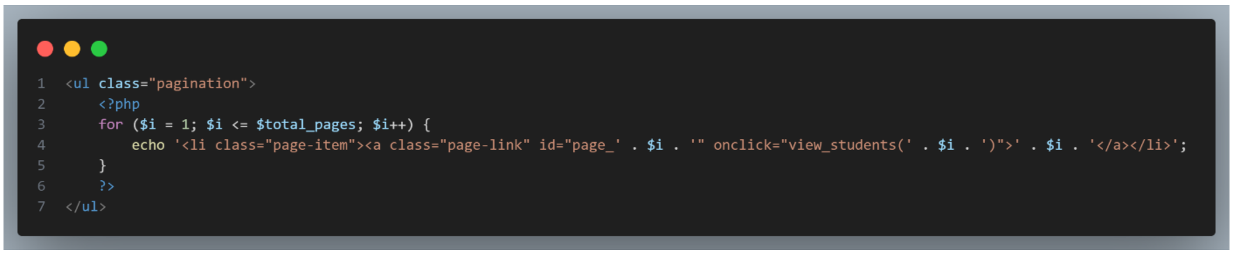
```php
$start_from = ($page - 1) * $results_per_page;
```

3. A query is then executed to fetch only the records for that page:



```php
// Fetch students, ordered by id in descending order
$sql = "SELECT * FROM students ORDER BY id DESC LIMIT $start_from, $results_per_page";
$result = $conn->query($sql);
```

4. **Rendering Pagination Controls**: Pagination links are generated dynamically:



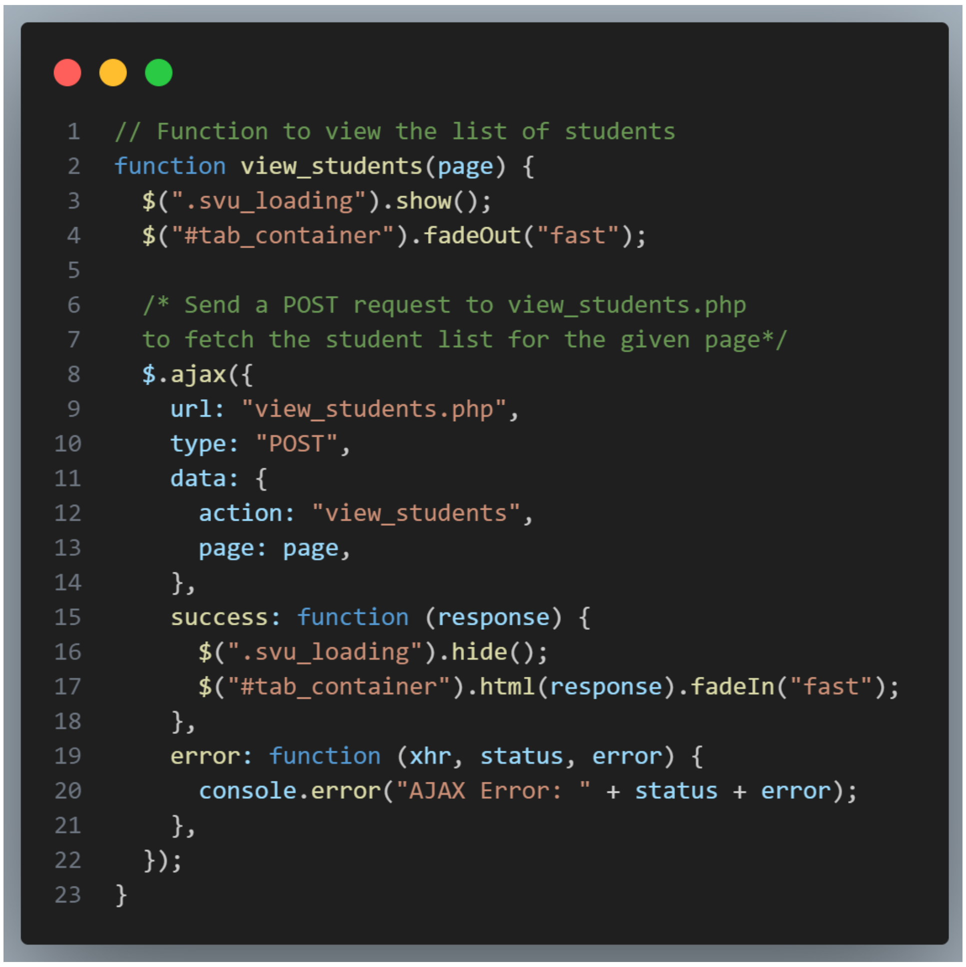
```
<ul class="pagination">
    <?php
    for ($i = 1; $i <= $total_pages; $i++) {
        echo '<li class="page-item"><a class="page-link" id="page_' . $i . '" onclick="view_students(' . $i . ')">' . $i . '</a></li>';
    }
    ?>
</ul>
```

Each link calls the (view_students) function with the corresponding page number.

5. **AJAX Request Handling for Pagination**: When a user clicks a pagination link, the (view_students) function sends an AJAX request with the selected page number:

```
view_students(page);
```

Here is (view_students) function:



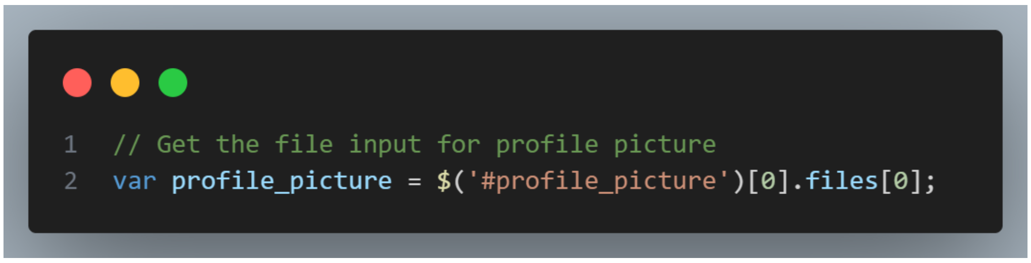
```
// Function to view the list of students
function view_students(page) {
  $(".svu_loading").show();
  $("#tab_container").fadeOut("fast");

  /* Send a POST request to view_students.php
  to fetch the student list for the given page*/
  $.ajax({
    url: "view_students.php",
    type: "POST",
    data: {
      action: "view_students",
      page: page,
    },
    success: function (response) {
      $(".svu_loading").hide();
      $("#tab_container").html(response).fadeIn("fast");
    },
    error: function (xhr, status, error) {
      console.error("AJAX Error: " + status + error);
    },
  });
}
```

The server responds with the data for the requested page, and the HTML is updated dynamically in the table container (#tab_container).

How File Upload is processed?

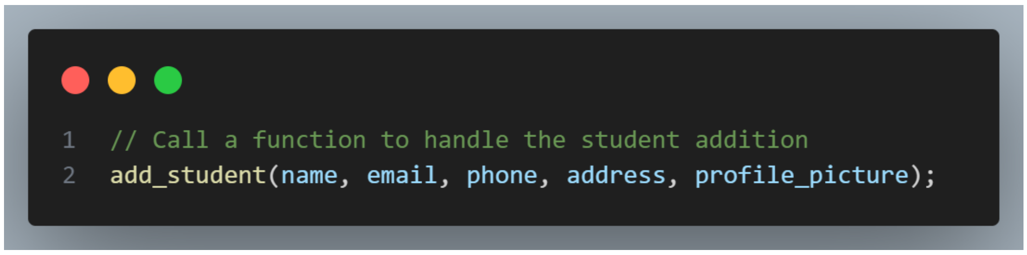
The provided code implements the file upload functionality in the student registration system. Here's a step-by-step explanation:

1. The Image of profile picture is read by this variable :



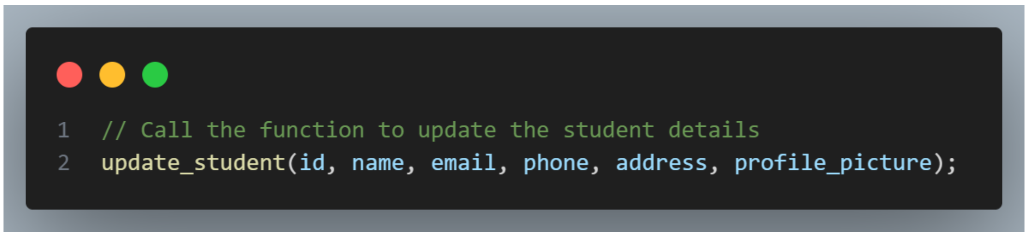
```javascript
// Get the file input for profile picture
var profile_picture = $('#profile_picture')[0].files[0];
```

2. Then Using the functions :



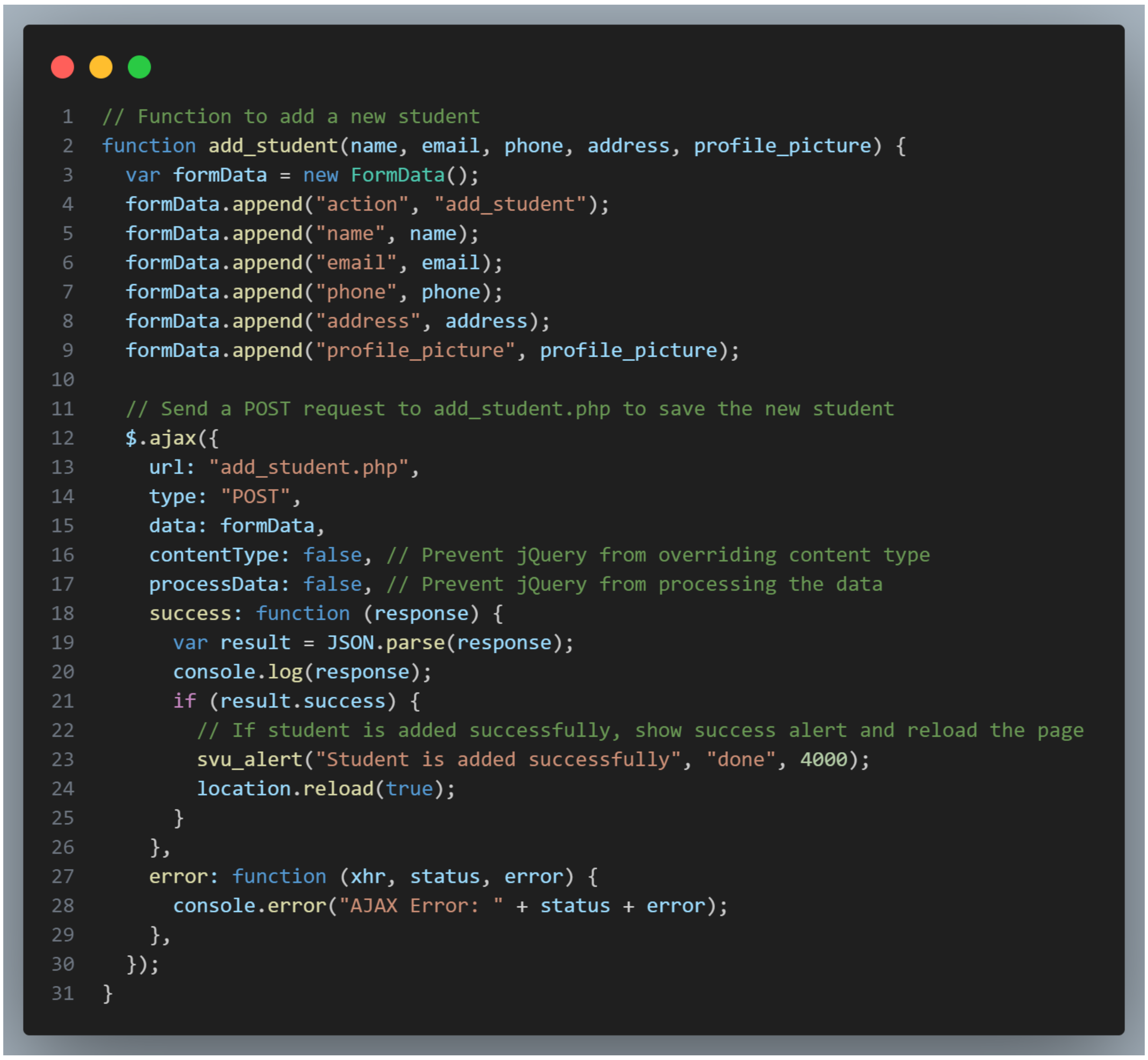
```javascript
// Call a function to handle the student addition
add_student(name, email, phone, address, profile_picture);
```

or



```javascript
// Call the function to update the student details
update_student(id, name, email, phone, address, profile_picture);
```

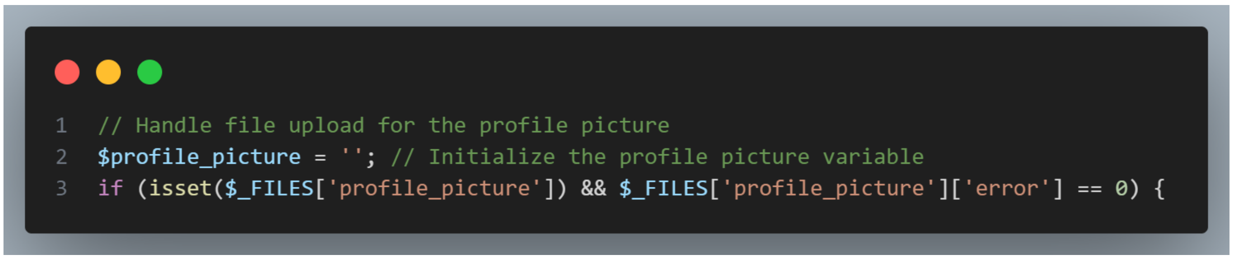
profile_picture is sent by js function to POST to php function by ajax that are found in the functions.js file:



```javascript
// Function to add a new student
function add_student(name, email, phone, address, profile_picture) {
  var formData = new FormData();
  formData.append("action", "add_student");
  formData.append("name", name);
  formData.append("email", email);
  formData.append("phone", phone);
  formData.append("address", address);
  formData.append("profile_picture", profile_picture);

  // Send a POST request to add_student.php to save the new student
  $.ajax({
    url: "add_student.php",
    type: "POST",
    data: formData,
    contentType: false, // Prevent jQuery from overriding content type
    processData: false, // Prevent jQuery from processing the data
    success: function (response) {
      var result = JSON.parse(response);
      console.log(response);
      if (result.success) {
        // If student is added successfully, show success alert and reload the page
        svu_alert("Student is added successfully", "done", 4000);
        location.reload(true);
      }
    },
    error: function (xhr, status, error) {
      console.error("AJAX Error: " + status + error);
    },
  });
}
```

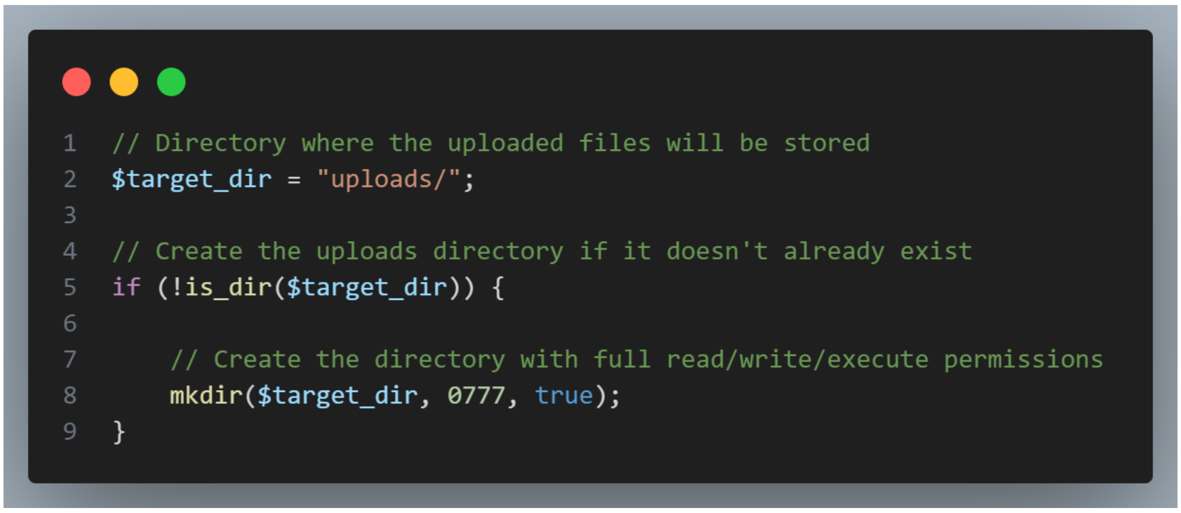
3. Then using "add_student" php function that is found in add_student.php or "update_student" function that is found in update_student.php, profile_picture will be uploaded.

4. The script checks if a file was uploaded using the $_FILES superglobal and if there were no errors during the upload:



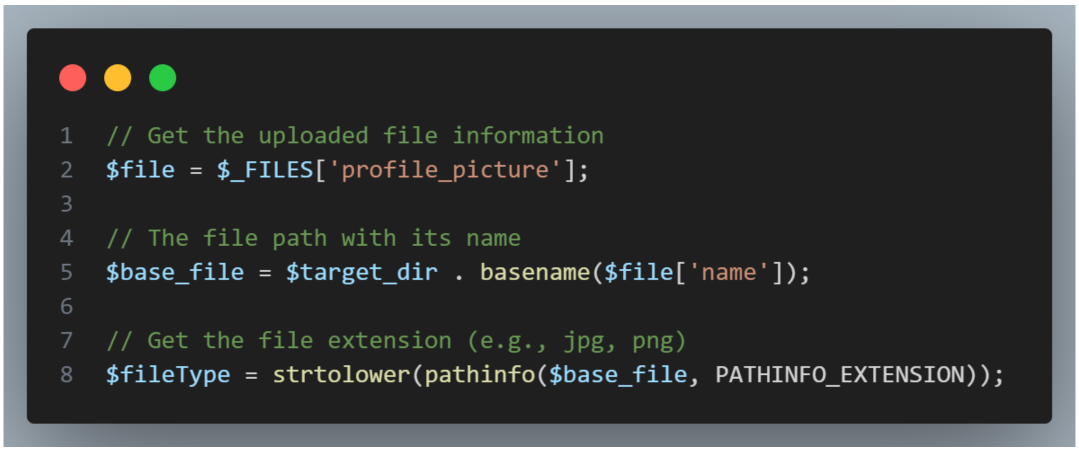
```php
// Handle file upload for the profile picture
$profile_picture = ''; // Initialize the profile picture variable
if (isset($_FILES['profile_picture']) && $_FILES['profile_picture']['error'] == 0) {
```

5. The directory where the file will be stored is set to uploads/. If the directory doesn’t exist, it’s created:



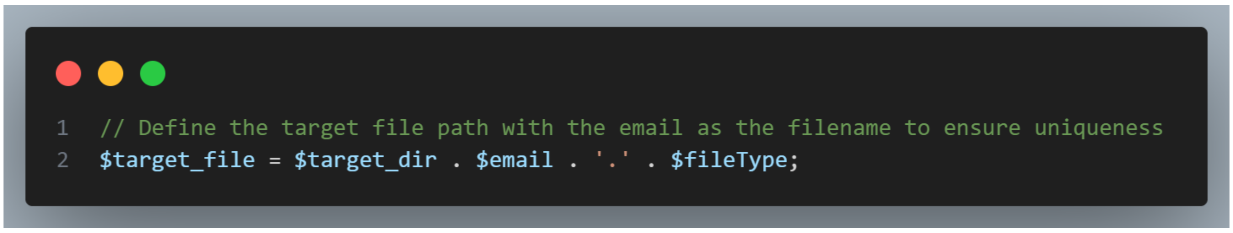
```php
// Directory where the uploaded files will be stored
$target_dir = "uploads/";

// Create the uploads directory if it doesn't already exist
if (!is_dir($target_dir)) {

    // Create the directory with full read/write/execute permissions
    mkdir($target_dir, 0777, true);
}
```

6. The uploaded file's details are fetched, and its path and extension are extracted:



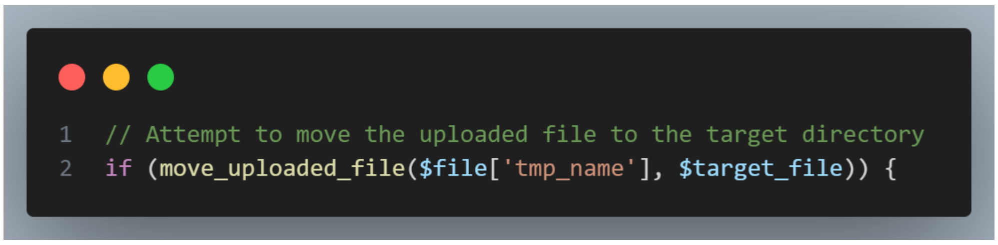
```php
// Get the uploaded file information
$file = $_FILES['profile_picture'];

// The file path with its name
$base_file = $target_dir . basename($file['name']);

// Get the file extension (e.g., jpg, png)
$fileType = strtolower(pathinfo($base_file, PATHINFO_EXTENSION));
```

7. To prevent overwriting existing files, the file is renamed using the user's email to ensure uniqueness:



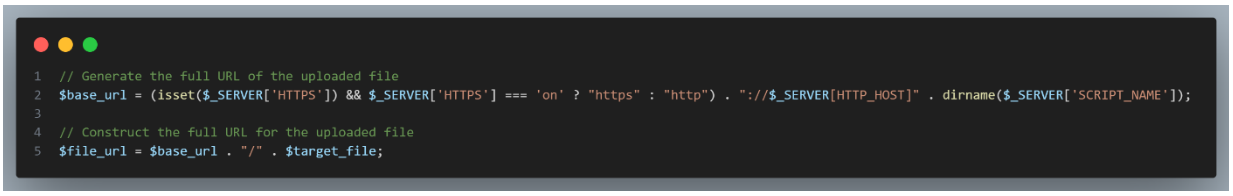
```php
// Define the target file path with the email as the filename to ensure uniqueness
$target_file = $target_dir . $email . '.' . $fileType;
```

8. The uploaded file is moved from its temporary location to the target directory:



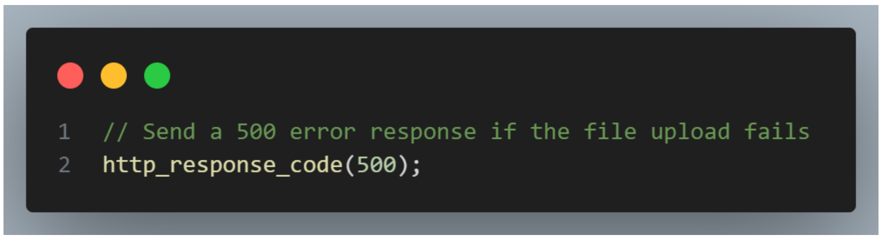
```php
// Attempt to move the uploaded file to the target directory
if (move_uploaded_file($file['tmp_name'], $target_file)) {
```

9. Once the file is successfully uploaded, its URL is generated:



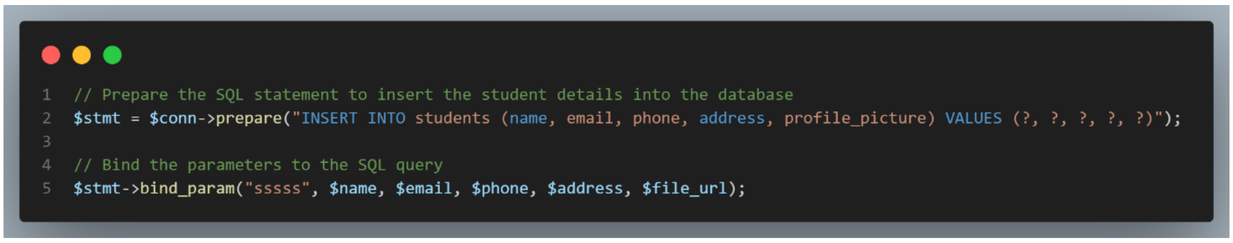
```php
// Generate the full URL of the uploaded file
$base_url = (isset($_SERVER['HTTPS']) && $_SERVER['HTTPS'] === 'on' ? "https" : "http") . "://$_SERVER[HTTP_HOST]" . dirname($_SERVER['SCRIPT_NAME']);

// Construct the full URL for the uploaded file
$file_url = $base_url . "/" . $target_file;
```

10. If the file upload fails, an HTTP 500 error is sent:



```php
// Send a 500 error response if the file upload fails
http_response_code(500);
```

11. Once the file URL ($file_url) is generated, It is saved in the database using a prepared SQL statement:



```php
// Prepare the SQL statement to insert the student details into the database
$stmt = $conn->prepare("INSERT INTO students (name, email, phone, address, profile_picture) VALUES (?, ?, ?, ?, ?)");

// Bind the parameters to the SQL query
$stmt->bind_param("sssss", $name, $email, $phone, $address, $file_url);
```