SCRUM1-76 Sprint 4 Documentation

**Project:** ScavengeRUs

**Sprint:** 4

**SCRUM Task ID:** SCRUM1-76

**Author:** Jean Bilong

**Date:** 12/6/2023

## Overview

This documentation covers the implementation and functionality added in Sprint 4 for the ScavengeRUs project, focusing on the enhancements made to the user management system, particularly in the **UserController**.

### 1. UserController.cs Enhancements

**Functionality:** The **UserController** has been updated to handle batch user creation from a CSV file. This feature allows administrators to efficiently create multiple users at once, which is particularly useful for setting up large groups or teams.

**Key Methods:**

- **CreateUsers(IFormFile uploadedFile)**: This **[HttpPost]** action handles the uploaded CSV file, parses it, and creates users in the system. It includes error handling and returns appropriate success or failure messages.

### 2. ApplicationUser Model Updates

**Functionality:** The **ApplicationUser** class has been updated to support additional fields necessary for the new batch creation process.

**Key Additions:**

- Additional properties to store user-specific data parsed from the CSV file.

**3. CreateUsers.cshtml**

**Functionality:** A new view has been created to facilitate the batch user creation process. This view includes a form to upload the CSV file and displays success or error messages based on the outcome of the batch creation process.

**4. Frontend JavaScript Enhancements**

**Functionality:** JavaScript functions have been added to handle the CSV file upload and to dynamically update the frontend based on the success or failure of the user creation process.

**Key Functions:**

- **openFile(event)**: Handles the CSV file selection.
- **processData(csv)**: Parses the CSV file and prepares the data for submission.
- **submitData()**: Submits the processed data to the server.

**5. Backend Processing Logic**

**Functionality:** The backend logic for parsing the CSV file and creating users has been implemented in the **UserController**. This includes validation of the CSV format, parsing each user's data, and creating user accounts in the system.

**6. Testing and Validation**

**Functionality:** Testing was conducted to ensure the accuracy and reliability of the batch user creation process. This included testing with various CSV file formats and sizes to ensure robust error handling and user feedback.

**Conclusion**

The implementation of batch user creation in Sprint 4 significantly enhances the administrative capabilities of the ScavengeRUs system, allowing for more efficient setup and management of user accounts. This feature is expected to save time and improve the overall user management experience.

```
/// <summary>
        /// HTTP GET action to display the CreateUsers view.
        /// </summary>
        /// <returns>The CreateUsers view.</returns>
        [HttpGet]
        public IActionResult CreateUsers()
        {
            return View();
        }


        /// <summary>
        /// HTTP POST action to create users from an uploaded CSV file.
        /// </summary>
        /// <param name="uploadedFile">The CSV file containing user data.</param>
        /// <returns>A JSON result indicating the success or failure of user
creation.</returns>
        [HttpPost]
        public async Task<IActionResult> CreateUsers(IFormFile uploadedFile)
        {
            if (uploadedFile == null || uploadedFile.Length == 0)
            {
                return Json(new { success = false, message = "Please upload a CSV
file." });
            }
```

```
            var errorList = new List<string>();

            try
            {
                using (var stream = new
StreamReader(uploadedFile.OpenReadStream()))
                {
                    string content = await stream.ReadToEndAsync();
                    var users = ParseCSVToUsers(content);

                    foreach (var user in users)
                    {
                        var result = await _userManager.CreateAsync(user,
defaultPassword);
                        if (!result.Succeeded)
                        {
                            errorList.Add($"Error creating user {user.Email}:
{string.Join(", ", result.Errors.Select(e => e.Description))}");
                        }
                    }
                }

                if (errorList.Any())
                {
                    return Json(new { success = false, message = "Some users
could not be created.", errors = errorList });
                }
                else
                {
                    return Json(new { success = true, message = "All users
created successfully." });
                }
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error processing uploaded file.");
                return Json(new { success = false, message = "An error occurred
while processing the file." });
            }
        }

        /// <summary>
        /// Parses a CSV string to create a list of ApplicationUser objects.
        /// </summary>
```

```
        /// <param name="csvContent">The CSV content as a string.</param>
        /// <returns>A list of ApplicationUser objects.</returns>

        private List<ApplicationUser> ParseCSVToUsers(string csvContent)
        {
            var users = new List<ApplicationUser>();
            var lines = csvContent.Split(new[] { "\r\n", "\n" },
StringSplitOptions.RemoveEmptyEntries);

            foreach (var line in lines)
            {
                var fields = line.Split(',');

                if (fields.Length == 5 && fields.All(field =>
!string.IsNullOrWhiteSpace(field)))
                {
                    var sanitizedEmail = WebUtility.HtmlEncode(fields[3].Trim());
                    var sanitizedPhone = WebUtility.HtmlEncode(fields[2].Trim());

                    if (!IsValidEmail(sanitizedEmail) ||
!IsValidPhoneNumber(sanitizedPhone))
                    {
                        continue; // Skip invalid entries
                    }

                    if (!Enum.TryParse<Carriers>(fields[4].Trim(), true, out var
carrier))
                    {
                        continue; // Skip if carrier parsing fails
                    }

                    var user = new ApplicationUser
                    {
                        FirstName = fields[0].Trim(),
                        LastName = fields[1].Trim(),
                        PhoneNumber = sanitizedPhone,
                        Email = sanitizedEmail,
                        UserName = sanitizedEmail, // Assuming email as username
                        Carrier = carrier
                    };

                    users.Add(user);
                }
            }
```

```
            return users;
        }

        /// <summary>
        /// Validates an email address format.
        /// </summary>
        /// <param name="email">The email address to validate.</param>
        /// <returns>True if the email is valid, false otherwise.</returns>

        private bool IsValidEmail(string email)
        {
            try
            {
                var addr = new System.Net.Mail.MailAddress(email);
                return addr.Address == email;
            }
            catch
            {
                return false;
            }
        }


        /// <summary>
        /// Validates a US phone number format.
        /// </summary>
        /// <param name="number">The phone number to validate.</param>
        /// <returns>True if the phone number is valid, false
otherwise.</returns>
        private bool IsValidPhoneNumber(string number)
        {
            return Regex.IsMatch(number, @"^\d{10}$"); // US phone number format
        }
```

CreateUsers.cshtml:

```
@model ScavengeRUs.Models.Entities.ApplicationUser
@{
    ViewData["Title"] = "CreateUsers";
}
@if (ViewData["ErrorList"] is List<string> errorList && errorList.Any())
{
```

```
    <div class="alert alert-danger">
        <ul>
            @foreach (var error in errorList)
            {
                <li>@error</li>
            }
        </ul>
    </div>
}



<h1>Batch Create Users</h1>
<button onclick="openModal()">Upload CSV File</button> <!-- Button to open modal
-->

<!-- File Upload Modal -->
<div id="fileUploadModal" class="modal">
    <div class="modal-content">
        <span class="close">&times;</span> <!-- Close button for modal -->
        <h2>Upload File</h2>
        <input type="file" id="fileInput" accept=".csv"
onchange="openFile(event)">
        <button onclick="submitData()">Submit Data</button> <!-- Button to submit
data -->
    </div>
</div>
<div id="error-message" style="display: none; color: red;"></div> <!-- Error
message display -->

<div id="loadingSpinner" style="display: none;">
    <div class="spinner-border" role="status">
        <span class="sr-only">Loading...</span>
    </div>
</div>

<script>



    // Get the modal
    var modal = document.getElementById("fileUploadModal");

    // Get the <span> element that closes the modal
    var span = document.getElementsByClassName("close")[0];
```

```
    var outputArray = []; // Global array to store processed data

    // When the user clicks anywhere outside of the modal, close it
    window.onclick = function (event) {
        if (event.target == modal) {
            modal.style.display = "none";
        }
    };

    // Function to open the modal
    function openModal() {
        modal.style.display = "block";
    }

    // When the user clicks on <span> (x), close the modal
    span.onclick = function () {
        modal.style.display = "none";
    };

    var openFile = function (event) {
        var reader = new FileReader();
        reader.onload = (event) => {
            this.processData(event.target.result);
        };
        reader.onerror = function (event) {
            displayErrorMessage("File read error: " +
event.target.error.message);
        };
        //Read file into memory as UTF-8
        reader.readAsText(event.target.files[0]);
    };

    var processData = function (csv) {
        try {
            var allTextLines = csv.split(/\r\n|\n/);
            var lines = [];
            for (let i = 1; i < allTextLines.length; i++) {
                let line = allTextLines[i].trim();
                if (line === "") continue; // Skip empty lines

                let values = line.split(",");
                if (
                    values.length === 5 &&
                    validateEmail(values[3]) &&
```

```
                    validatePhoneNumber(values[2])
                ) {
                    lines.push(values);
                } else {
                    throw new Error("Invalid data format at line " + (i + 1) + ":
" + line);
                }
            }
            console.log(lines);
            this.arraysToObjects(lines);
            this.tableCreate(lines);
        } catch (error) {
            console.error("Error processing CSV data: ", error);
            displayErrorMessage("Error processing CSV file: " + error.message);
        }
    };

    //arraysToObjects converts the arrays to an array of objects
    var arraysToObjects = function (lines) {
        object = {};
        var i;
        var j;
        outputArray = [];
        for (j = 1; j < lines.length; j++) {
            obj = {};
            for (i = 0; i < lines[0].length; i++) {
                obj[lines[0][i]] = lines[j][i];
            }
            outputArray.push(obj);
        }
        console.log(outputArray);
    };

    // Email validation function
    function validateEmail(email) {
        var re = /^[^@@\s]+@@[^@@\s]+\.[^@@\s]+$/;
        return re.test(email);
    }

    // Phone number validation function (for US numbers)
    function validatePhoneNumber(number) {
        var re = /^\d{10}$/;
        return re.test(number);
    }
```

```
// Display error message
function displayErrorMessage(message) {
    var errorMessageElement = document.getElementById("error-message");
    if (errorMessageElement) {
        errorMessageElement.innerText = message;
        errorMessageElement.style.display = "block";
    } else {
        alert(message);
    }
}

function tableCreate(lines) {
    //body reference
    var body = document.getElementsByTagName("body")[0];

    //create elements <table> and a <tbody>
    var tbl = document.createElement("table");
    var tblBody = document.createElement("tbody");

    //cells creation
    for (var rows = 0; rows < lines.length; rows++) {
        //table row creation
        var row = document.createElement("tr");

        for (var col = 0; col < lines[0].length; col++) {
            //Create element <td> and text node
            //Make text node the contents of <td> element
            //Put <td> at end of the table row
            var cell = document.createElement("td");
            var cellText = document.createTextNode(lines[rows][col]);

            cell.appendChild(cellText);
            row.appendChild(cell);
        }
        //row added to end of table body
        tblBody.appendChild(row);
    }
    //append the <tbody> inside the <table>
    tbl.appendChild(tblBody);
    //put <table> in the <body>
    body.appendChild(tbl);
    //tbl border attribute
    tbl.setAttribute("border", "2");
}
```

```
    var submitData = function () {
        var fileInput = document.getElementById("fileInput");
        var file = fileInput.files[0];

        if (!file) {
            displayErrorMessage("Please select a file to upload.");
            return;
        }

        // Show progress indicator (example: a loading spinner)
        showProgressIndicator(true);

        var formData = new FormData();
        formData.append("uploadedFile", file);

        fetch("/User/CreateUsers", {
            method: "POST",
            body: formData,
        })
            .then((response) => {
                // Hide progress indicator
                showProgressIndicator(false);

                if (!response.ok) {
                    throw new Error("Network response was not ok: " +
response.statusText);
                }

                // Check if the response is JSON
                const contentType = response.headers.get("content-type");
                if (contentType && contentType.indexOf("application/json") !== -
1) {
                    return response.json().then((data) => {
                        console.log("Success:", data);
                        alert(data.message || "Users successfully created");
                    });
                } else {
                    return response.text().then((text) => {
                        // Handle non-JSON response (like HTML)
                        document.body.innerHTML = text;
                    });
                }
            })
            .catch((error) => {
                console.error("Error:", error);
```

```
                alert(error.message || "Error submitting data");
            });
    };

    // Show or hide progress indicator

    function showProgressIndicator(show) {
        var spinner = document.getElementById("loadingSpinner");
    }

</script>
```