ISCWA: An Inter-School Competion Management Web Application for Schools.

Team Name: SoftwareDevelopers

Jay Malhotra, E23BCAU0050

Academic Session 2024-2025

Table of Contents

Introduction

The Inter-School Competition Management Web Application (ISCWA) is a revolutionary way to coordinate and oversee educational competitions between schools. In a time when digital methods are more common, ISCWA plans to use advanced technologies to make inter-school competitions run smoother, improving efficiency and participant satisfaction. This app deals with important problems linked to old-fashioned event planning methods, such as complicated sign-up procedures, manual scoring systems, and ineffective feedback mechanisms. ISCWA is establishing a new benchmark for educational event management by incorporating real-time data processing, user-friendly interfaces, and comprehensive management tools. This introduction discusses the difficulties of existing methods, presents the suggested advantages of ISCWA, and talks about the techniques utilized to realize this vision.

Background

Inter-school competitions are very important in the educational environment, providing students with chances to succeed in a variety of fields such as academics, arts, and sports. These activities promote both friendly competition and support interactions in culture and intellect among organizations. Normally, these activities are handled using traditional methods that require a lot of paperwork, coordination from school personnel, and participants to be present in person for registering and getting updates. The complexity of efficiently managing these events has increased as their scale and frequency have expanded.

Problem Statement

The management of inter-school competitions currently faces several significant challenges:

- **Registration Logistics**: On-site registrations are cumbersome, prone to errors, and create bottlenecks at the start of events, leading to delays and frustrations among participants and organizers.
- **Score Management**: Scores are often calculated manually or with basic spreadsheet tools, which are not only prone to errors but also fail to provide real-time updates to participants and spectators.
- **Feedback Collection**: There is no streamlined process for collecting and analyzing feedback from participating schools, which is crucial for improving event management and participant satisfaction in future iterations.
- **Information Dissemination**: Communicating updates, results, schedules, and other critical information is often delayed and inconsistent, impacting the overall experience of the participants.

Objectives/Features

The web-based application (Inter-School Competition Web Application - ISCWA) would allow participating schools to register beforehand and provide feedback, event coordinators to input and update scores for events, admin and school to view the scores real time and the automatic calculation of overall winners.

The reason for choosing a web-based application is that it can be accessed by different users through multiple browsers. I decided to use ASP.Net framework, C# as the programming language, JavaScript for client-side validations and SQL for the backend database.

ASP.Net framework is language independent and provides access to a rich set of UI elements such as GridViews for populating data in tabular form, Calendar Control for date selection, Validation controls etc. It also allows to retain data across postbacks through ViewState.

C# supports language interoperability and is object oriented which allows for modularity and code re-usability. Also, ASP.Net was designed as a framework to develop solutions using C# and offers libraries

for exception handling, string operations and mail integration that will allow me to develop a robust application. JavaScript has been used to incorporate client-side validations.

SQL is used because of its faster query processing and standardized language. SQL and .NET framework integrate seamlessly with ADO.Net that provides a set of classes and methods that executes commands for storing and retrieving data from database.

The application is built using the IDE Visual Studio because of its timesaving and productivity features such as IntelliSense.

Success Criteria

1.	Sign up by schools and event coordinators	The participating schools and event coordinators should have access to a registration page where they can sign up and create an account for themselves in the Inter-School Event Web Application.
2.	Login	The user should be able to login using their registered Email ID and Password.
3.	Change/Reset Password Functionality	All users should be able to change their password or reset it in case the password is lost.
4.	Role Based Access	The types of users that the web application will accommodate is the admin, the participating schools and the event coordinators. Each user will have access to different features and pages in the web application depending on their role.
5.	Fest and Event Creation along with Criteria	The admin should be allowed to create multiple fests and events for an inter-school competition. The admin should also be able to specify the criteria for each event such as eligible grades, maximum number of participants etc.
6.	Registering for the Competition	Individual schools should be able to register participants for different Fests and Events as per the criteria.
7.	Inputting Scores for individual events	The event coordinators should be able to input the scores for each school for the event they are coordinating. They should not be able to view or edit other events' scores.
8.	Updating Scores for individual events	The event coordinators should be able to update the given scores for their assigned event in case an incorrect entry was made or a change needs to occur.

9.	View Scores of the events	The participating schools should have access to view the scores for the various events they have participated in but not edit them. The admin should have access to view all scores for all schools and all events.
10.	Database Operations	Users of the system should be able to perform add/update/delete operations on the database for registration details, the event details, the scores of an event and more.
11.	Calculation of scores in real time for leader board	The system should be able to calculate the overall scores of each school for each event and show it in the form of a leader board in real time. This leader board will not be editable by any user and can be viewed by Admin, participating schools and event coordinators.
12.	Top 3 winners	The application should show the top 3 winners for the inter-school fest depending on criteria such as the overall score for the school and the number of events the school has participated in.
13.	User Friendly Error messages	If incorrect or incompatible information has been provided, there should be user friendly error messages that should inform the user why they can't proceed further.
14.	Intuitive and Professional User Interface	The User Interface of the web application should be simple, friendly and professional at the same time.
15.	Search based on Filters	All users would be able to filter records/data according to the search criteria chosen in the dropdowns.
16.	Validation	Validation should be incorporated such as presence or format checks. Other validations, such as the event must be created 1 to 3 months in advance, Schools will not be allowed to enroll/register more than once for a given event, Event Coordinators should not be able to input scores for the same event more than once etc.
17.	Feedback Page	Participating schools should be able to give feedback for the registered events.
18.	Viewing School Feedback	The admin should be able to view the feedback as given by the participating schools.

Development Model

Given the dynamic nature of the project requirements and the need for continuous adaptation, the Agile development model is selected. This model promotes a collaborative and flexible approach to development that involves:

- **Iterative Development**: Breaking down the project into manageable increments that allow for frequent reassessment of the development direction based on stakeholder feedback.

- **Continuous Collaboration**: Keeping communication channels open between developers, designers, and stakeholders to ensure that the project aligns with user needs and expectations.
- **Adaptive Planning**: Embracing changes in requirements as the project evolves, thereby increasing the relevance and effectiveness of the ISCWA.

Testing Strategy

Since this project involves multiple interconnected systems such as registration, score management, feedback collection, and real-time updates, both manual (system) testing and integration testing would be carried out. Here's why:

Manual Testing

- **Purpose**: Manual testing involves human testers interacting with the application to ensure it behaves as expected under varied usage scenarios. This approach allows for comprehensive verification of user experiences and UI details that automated testing might miss.
- **Benefits for ISCWA**:
    - **Visual and Usability Checks**: Manual testing is crucial for identifying visual issues and assessing the user experience, ensuring the application is intuitive and user-friendly.
    - **Exploratory Testing**: It enables testers to explore the application without predefined scripts, discovering issues that automated tests may not catch.
    - **Immediate Feedback**: Provides quick, actionable insights during development, crucial for agile processes.
    - **Flexibility**: Testers can adapt their testing strategy on the fly, which is especially useful for complex or changing requirements.

Integration Testing

- **Purpose**: Integration testing assesses the interaction between integrated units or components to ensure they work together correctly. This is crucial for a system like ISCWA, where different modules need to interact seamlessly.
- **Benefits for ISCWA**:
    - **System Cohesion**: Ensures that components such as the database, server, and client-side interfaces work together without issues.
    - **API Functionality**: Validates that API interactions, especially those involving third-party integrations or complex data exchanges within the application, are functioning correctly.

ISCWA involves complex interactions between its frontend and backend, necessitating thorough integration testing. Additionally, the use of technologies like ASP.NET, C#, and SQL Server underlines the need for rigorous unit testing to ensure the reliability of individual methods and database interactions.

In conclusion, both manual and integration testing are carried out for ISCWA to guarantee a high-quality, reliable application. Starting with unit testing during the initial development phases followed by integration testing once modules are combined will help in achieving a robust build.

Challenges with Unit Testing

Unit testing is not feasible in this project due to the extensive use of SQL commands embedded directly within both C# classes and ASPX pages. This setup complicates the isolation of the code necessary for effective unit testing. Tools like Moq, which are typically used to mock database interactions, are not suitable as they cannot intercept and mock inline SQL queries scattered throughout the codebase. This limitation underscores the importance of manual testing to ensure comprehensive coverage, particularly where automated testing methods fall short.

Target Audience

The target audience for the Inter-School Competition Management Web Application (ISCWA) primarily comprises educational institutions that host or participate in inter-school events. This includes schools at various educational levels—primary, secondary, and higher secondary—who seek to streamline the management and execution of academic, sports, and cultural competitions. Additionally, the platform is designed for event coordinators and school administrators who are directly responsible for organizing and overseeing these events. By providing tools for simplified registration, real-time score updates, and comprehensive event management, ISCWA aims to enhance the experience for both organizers and participants. Furthermore, the application offers features tailored to the needs of teachers and school staff involved in the preparations and day-to-day operations of such events, ensuring that everyone from the planning committee to the on-ground staff benefits from improved efficiency and communication. This broad yet focused targeting helps ISCWA address specific challenges faced by educational communities, making it an invaluable tool for enhancing inter-school collaboration and competition.

Project Timeline

Task Number	Planned Action	Planned Outcome	Time Estimated
1.	Identifying the problem	Requiring a digital system for hosting inter-school competitions.	3 days
2.	Discussion with teammates regarding the problem and proposed solution	Detailed Interaction amongst teammates about the current system and identifying a possible digital solution. Discuss and finalize making a web-based application on Visual Studio, using the ASP. NET framework, through the C# language with an SQL type database.	3 days
3.	Forming a success criterion	Develop a success criterion which incorporates and highlights key features of the solution such as inputting, updating and calculating scores, viewing school feedback, creating fest/events etc.	3 days
4.	Constructing the layout and the UI for each screen of the product	Using Mockflow Wireframe, we will create UI designs for each screen, such as the feedback page, the event details page, viewing scores page, creating an event page etc to understand how each screen will look like in the actual product .	6 days
5.	Constructing User Stories for the different users present in the product	Using Visual Paradigm, we will create user stories for the 3 users – Admin, Event Coordinator and School.	2 days
6.	Creating Use Case Diagram, Data Flow Diagram, Activity Diargams and Sequence Diagrams based on the key features.	Using Visual Paradigm, we will create UML diagrams based on the user stories and any other diagrams made afterwards.	1 week
7.	Constructing an ER diagram	Designing an ER diagram using draw.io to showcase the columns and primary and foreign key relationships between the data-tables in the database. An example is	2 days

		the EventDetails data-table, which includes multiple foreign keys, such as Fest Name and Event Coordinator Email, from other data-tables titled FestNames and EventCoordinatorCredentials.	
8.	Constructing a Class diagram	Designing a Class diagram using draw.io to showcase the properties and the methods of each class. Some classes, such as the AdminClass, SchoolClass and EventCoordinatorClass are derived from the base class, titled UserClass, which has multiple virtual override methods such as LoginCheck, ChangePassword and CheckEmail. Other classes, such as the SchoolFeedbackClass, are public classes which have their own defined methods such as SubmitFeedback and FetchFeedbackDetails.	2 days
9.	Creating a Test Plan	Formulating a Test Plan basis the success criteria to assess the product based on abnormal and normal conditions. An example can be a Test Plan for Schools registering for an event where multiple abnormal conditions are present such as when the user tries to register for an event without selecting the event first or when the school tries to register for the same event more than once and only one or two normal conditions such as the user selects an event for which they haven't registered for first, then clicks on the register button.	3 days
10.	Understanding the environment of Visual Studio and the development of an ASP.NET framework web-application with C# and MySQL type database, along with OOPS concept	Setting up Visual Studio and an ASP.NET web-application project. Understanding how an ASP.NET application works and how to code using C# by referring to websites and tutorials online.	1 week
11.	Creating a MySQL database with multiple tables basis the ER diagram	Creating the data-tables in the database, including setting up the primary- foreign key relationships, by referring to the ER diagram.	1 day
12.	Creating the classes basis the Class diagram	Creating a class library and populating all classes with their properties, get/set methods and method definitions.	2 days
13.	Creating the UI for all the web pages using the mock screens made earlier	Creating all the screens based on the mock screens made earlier using web- forms and the HTML tag and CssClass properties with a consistent background colour.	1 week
14.	Writing the code for basic and common pages across the 3 users (Registration,	Developing the code for the Registration, Login, Change and Forgot Password functionalities. Implementing OOPS concepts, stored procedures, making methods	1 week

	Login, Change and Forgot Password)	and connecting the backend with the database.	
15.	Testing the basic and common pages and solving any error(s) encountered	Finalizing the functionalities by testing the code by conducting multiple runs of the code until any errors found are solved.	2 days
16.	Writing the code for some of the Admin functionalities (Creating Fests and Events, Update Fest/Event Details)	Developing the code for creating fests and events, and the update fest and event details functionalities. Implementing OOPS concepts, making methods and connecting the backend with the database.	10 days
17.	Testing the creating fests/events and updating fests/events functionalities and solving any error(s) encountered	Finalizing the functionalities by testing the code by conducting multiple runs of the code until any errors found are solved.	2 days
18.	Writing the code for some of the School functionalities (Viewing Event Details and Registering for the Event)	Developing the code for viewing the event details and registering participants for the event functionalities. Implementing OOPS concepts, making methods and connecting the backend with the database.	10 days
19.	Testing the registering for upcoming events functionality and solving any error(s) encountered	Finalizing the functionalities by testing the code by conducting multiple runs of the code until any errors found are solved.	2 days
20.	Writing the code for all the Event Coordinator's functionalities (Choosing an Event to input or update scores for, inputting and updating scores)	Developing the code for choosing an event to input or update the scores of, and inputting and updating the scores for the chosen event functionalities. Implementing OOPS concepts, making methods and connecting the backend with the database.	10 days
21.	Testing the inputting and updating of scores functionalities and solving any error(s) encountered	Finalizing the functionalities by testing the code by conducting multiple runs of the code until any errors found are solved.	3 days
22.	Writing the code for the remaining School's functionalities (Viewing Fest and Event Scores, Giving Feedback on the Fest)	Developing the code for viewing the Fest and Event scores and giving feedback functionalities. Implementing OOPS concepts, making methods and connecting the backend with the database.	1 week
23.	Testing the viewing scores and giving feedback functionalities and solving any error(s) encountered	Finalizing the functionalities by testing the code by conducting multiple runs of the code until any errors found are solved.	2 days
24.	Writing the code for the remaining functionalities of the Admin (Viewing Fest and Event Specific Scores,	Developing the code for viewing Fest and Event- Specific Scores, Calculating Top 3 winners and viewing School feedback functionalities. Implementing OOPS concepts, making methods and connecting the	2 weeks

	Calculating Top 3 winners, and viewing School's feedback)	backend with the database.	
25.	Testing the viewing scores, calculating top 3 winners and viewing school feedback functionalities and solving any error(s) encountered	Finalizing the functionalities by testing the code by conducting multiple runs of the code until any errors found are solved.	5 days
26.	Writing the code for server and client- side validations to avoid abnormalities in the database	Adding validations where required, to solve errors such as submitting empty forms, entering duplicate data and more.	10 days
27.	Testing the validation checks and solving any error(s) encountered	Finalizing the validations by testing them by running multiple runs of the code and making the validation messages user-friendly.	2 days
28.	Testing and comparing each functionality of the product against the success criteria and by referring to the Test Plan.	The product should fulfil the requirements success criteria.	1 week
29.	Conducting an alpha test with teammates	Our group runs the complete product multiple times to eliminate any other errors and to ensure it is working as expected.	5 days
30.	Conducting a beta test by having students go through our product	The students will take 1 week to explore all the features and functionalities of the product. They will be given access to all the 3 users and will also be told how the pages are linked with each other to ensure smooth functionality. The students will give feedback on the product basis their experience.	1 week
31.	Making the final round of changes basis the feedback given by the students after beta testing the product	Reworking on the product basis the feedback received from the students, such as some validation messages stay on the screen after the process is finished, the font size is too small and more.	1 week
32.	Getting the final round of feedback from our teammates and students on the final product	Discussion with teammates and students regarding the final product in terms of its functionality, areas of improvements and more.	3 days
33.	Brainstorming on any other pages, features or functionalities that can be implemented in the future	Brainstorming improvements for the product in regard to the UI, validations, functionalities and more which can be implemented in the future.	3 days

Software Requirement Specification

Functional Requirements

1. **Sign Up by Schools and Event Coordinators**: Schools and event coordinators must be able to register and create accounts in the system.
2. **Login**: Users should be able to log in using their registered email ID and password.
3. **Change/Reset Password Functionality**: All users must have the ability to change their password or reset it if lost.
4. **Role-Based Access**: Different features and pages within the web application should be accessible based on user roles: admin, schools, and event coordinators.
5. **Fest and Event Creation along with Criteria**: Admins can create multiple fests and events, specifying criteria like eligible grades and maximum number of participants.
6. **Registering for Competitions**: Schools should be able to register participants for different fests and events as per the established criteria.
7. **Inputting Scores for Individual Events**: Event coordinators can input scores for each school for the events they manage. They should not have access to view or edit scores for events they are not managing.
8. **Updating Scores for Individual Events**: Event coordinators should be able to update scores for their assigned events in case of incorrect entries or required changes.
9. **Viewing Scores of Events**: Schools can view scores for the events they participated in but cannot edit them. Admins should have access to view all scores for all schools and events.
10. **Database Operations**: Users must be able to perform add/update/delete operations on the database for registration details, event details, and scores.
11. **Real-time Calculation of Scores for Leaderboard**: The system should automatically calculate and display the overall scores of each school for each event in real-time on a leaderboard.
12. **Display Top 3 Winners**: The application should display the top three winners for the inter-school fest based on criteria such as overall score and number of events participated in.
13. **User-Friendly Error Messages**: If incorrect or incompatible information is provided, user-friendly error messages should inform the user of the issue.
14. **Intuitive and Professional User Interface**: The user interface should be simple, user-friendly, and professional.
15. **Search Based on Filters**: Users should be able to filter records/data according to selected search criteria.
16. **Validation**: The application should incorporate validations such as presence or format checks. Additional validations might include checks like ensuring events are created 1 to 3 months in advance, schools cannot enroll more than once for a given event, and event coordinators cannot input scores more than once for the same event.
17. **Feedback Page**: Participating schools should be able to provide feedback on the events they register for.
18. **Viewing School Feedback**: Admins should be able to view feedback provided by the participating schools.

Non-Functional Requirements

1. **Performance**
    - Fast response times for user interactions.
    - Scalability to handle peak usage periods.
2. **Security**
    - Data encryption for sensitive information.
    - Secure authentication and authorization practices.
3. **Usability**

- Intuitive UI/UX design.
- Error Handling and Guidance.

4. **Reliability**
    - High availability and minimal downtime.
    - Robust data backup and recovery mechanisms.
    - Quality Assurance and Testing
5. **Maintainability**
    - Modular design for easy updates and maintenance.
    - Documentation for system architecture and codebase.

Hardware Requirements

1. **Developer Machines:**
    - **Processor:** Minimum Intel i5 or equivalent
    - **RAM:** Minimum 8GB, 16GB recommended for better performance
    - **Storage:** At least 20GB of free space for the development tools, source code, and databases
    - **Network:** Reliable internet connection for accessing online resources, updates, and collaborating with team members via version control systems
2. **Server Requirements:**
    - **Processor:** Multi-core processor (4 cores or more)
    - **RAM:** Minimum 16GB
    - **Storage:** SSD with at least 50GB of free space for the application, logs, and database
    - **Network:** High-speed internet connection with a static IP for hosting and external access

Software Requirements:

1. **For Windows Users:**
    - **Operating System:** Windows 10 or later
    - **Integrated Development Environment (IDE):** Microsoft Visual Studio 2019 or later with ASP.NET and web development packages installed
    - **Database Tools:**
        - Microsoft SQL Server Management Studio 2022 for managing SQL Server databases
        - MySQL Workbench as an alternative for managing MySQL databases
    - **Frameworks and Libraries:**
        - .NET Framework 4.8 or later
        - NuGet packages for SQL query implementation and other dependencies as required

2. **For Mac Users:**
    - **Operating System:** macOS Catalina or later
    - **Development Tools:**
        - Use a virtual machine software like Parallels Desktop to run a Windows environment
        - Install Visual Studio on the virtual Windows machine for ASP.NET development
        - Microsoft SQL Server Management Studio 2022 installed within the virtual machine for database management
    - **Alternative IDE Options:** If not using a virtual machine, any macOS-compatible IDE that supports ASP.NET web applications (e.g., Visual Studio Code with appropriate extensions)
    - **Additional Software:**
        - Docker for Mac to manage and connect to SQL databases through containers, ensuring compatibility and isolation

3. **Common Software Requirements:**

- **Source Control:** Git for version control, with repository hosting on platforms such as GitHub, Bitbucket, or GitLab
- **Web Browsers:** Latest versions of Google Chrome, Mozilla Firefox, Safari, and Edge for testing and debugging
- **API Testing Tools:** Postman for testing web API interactions and ensuring correct behavior before deployment

Technical Requirements

- **Front-End Development**: Technologies like HTML and CSS, along with the ASP.Net framework for building a dynamic and responsive user interface. JavaScript for client side validation.
- **Back-End Development**: C# for server-side logic and database interactions.
- **Database**: MySQL for its scalability and flexibility in handling large volumes of unstructured data.

User Stories

In the development of the Inter-School Competition Management Web Application (ISCWA), we employ user stories to ensure that our product is closely aligned with the needs and expectations of its end-users. These user stories articulate the desired functionalities from the perspective of different user roles, such as administrators, event coordinators, and participating schools. They serve as a vital tool for guiding the design and development process, ensuring that each feature developed is directly linked to user requirements. By focusing on these user-centered narratives, we can prioritize functionalities, enhance communication across the development team and stakeholders, and facilitate a flexible, iterative development process. The following is a list of user stories that detail the specific needs and goals identified for ISCWA, guiding our Agile development approach –

ID	Epic	User Story	Acceptance Criteria
A1	**Admin**	As an Admin, I want to create festivals and events with specific criteria like grade eligibility and participant limits so that events can be tailored to the intended audience and managed effectively.	Admin can create events with criteria such as grade eligibility and participant limits. Events are tailored and managed effectively.
A2	**Admin**	As an Admin, I want to manage event criteria and settings anytime before the event starts so that I can ensure all details are correct and up-to-date.	Admin can update event criteria and settings anytime before the event starts, ensuring accurate and current details.
A3	**Admin**	As an Admin, I want to view real-time updates of scores and the system calculates top winners automatically so that I can have current information and ensure a fair competition.	Admin can view real-time score updates. The system calculates and displays top winners automatically.
A4	**Admin**	As an Admin, I want to assign event coordinators to specific events to manage operations so that each event runs smoothly with a responsible point of contact.	Admin can assign event coordinators to specific events to manage operations.
A5	**Admin**	As an Admin, I want to adjust the scoring weightage for various events to determine final scores so that the importance of each event is accurately reflected in the overall competition.	Admin can adjust scoring weightage for events, affecting the determination of final scores.

A6	**Admin**	As an Admin, I want to view feedback provided by schools to gauge satisfaction and areas for improvement so that I can make informed decisions about future events.	Admin can view feedback from schools and use it to gauge satisfaction and identify areas for improvement.
A7	**Admin**	As an Admin, I want to modify existing fest/event details including dates, criteria, and participating schools so that the information remains relevant and accurate.	Admin can update fest/event details such as dates, criteria, and participating schools to ensure relevance and accuracy.
A8	**Admin**	As an Admin, I want to change my password through my profile settings to maintain account security so that unauthorized access is prevented.	Admin can change their password through their profile settings, enhancing account security.
A9	**Admin**	As an Admin, I want to reset my password through a secure process that verifies my email address and sends a reset link so that I can regain access to my account if I forget my password.	Admin can reset their password through a process that verifies their email address and sends a reset link.
A10	**Admin**	As an Admin, I want to log into the system to access administrative functionalities so that I can manage the events and participants effectively.	Admin can register to access the system and secure access that verifies their email address.
A11	**Admin**	As an Admin, I want to moderate listings and reviews to maintain platform integrity.	Admin can log into the system to access administrative functionalities and manage events and participants.
EC1	**Event Coordinators**	As an Event Coordinator, I want to enter and modify scores during the event without accessing other events' scores so I can ensure accurate scoring for my event.	Coordinators can enter and modify scores for their managed events independently of other events.
EC2	**Event Coordinators**	As an Event Coordinator, I want to view complete details of the events I am managing so I can effectively oversee the event operations.	Coordinators can view all details of the events they manage, including participant information and scoring criteria.
EC3	**Event Coordinators**	As an Event Coordinator, I want to see a real-time dashboard displaying ongoing scores and participant updates so I can monitor the event's progress accurately.	A real-time dashboard is available for coordinators to monitor ongoing event scores and updates.
EC4	**Event Coordinators**	As an Event Coordinator, I want to change my password to ensure my account remains secure.	Coordinators can change their passwords to maintain account security.
EC5	**Event Coordinators**	As an Event Coordinator, I want to reset my password if forgotten so that I can securely regain access to the system.	Coordinators can reset their password securely to regain access if forgotten.

EC6	**Event Coordinators**	As an Event Coordinator, I want to register for system access so I can manage my assigned events.	Coordinators can register to manage specific events within the system.
EC7	**Event Coordinators**	As an Event Coordinator, I want to log into the system so I can manage my assigned events.	Coordinators can login to manage specific events within the system.
S1	**School**	As a School, I want to register for events so that I can ensure participation in competitions that are relevant to our students.	Schools can register participants based on the criteria set for each event.
S2	**School**	As a School, I want to view scores and feedback in real-time so that I can monitor our participants' performance and receive immediate information.	Schools can view their own scores and feedback from past events in real-time.
S3	**School**	As a School, I want to submit feedback after the event so that we can provide constructive input on the event organization and execution.	Schools can submit feedback that is accessible to the admin for review and action.
S4	**School**	As a School, I want to access detailed information and rules for each event prior to participating so that we can prepare appropriately for the competitions.	Schools can access detailed event information and rules to ensure proper preparation.
S5	**School**	As a School, I want to change our password to maintain the security of our account and protect our data.	Schools can change their password through their profile settings for security reasons.
S6	**School**	As a School, I want to reset our password securely if it is forgotten so that we can regain access to our account and maintain continuity.	Schools can reset their password through a secure process when necessary.
S7	**School**	As a School, I want to register to access the system so that we can participate in the inter-school competitions and manage our entries.	Schools can register to access the system for event participation.
S8	**School**	As a School, I want to log into the system so that we can engage in the available functionalities to manage our event participation effectively.	Schools can log into the system to participate in events and access related information.

Design (Diagrams)

Gantt Chart

A Gantt Chart is a project management tool that visually represents tasks over a timeline, showing their start and end dates, durations, and dependencies in a bar-chart format. For the ISEWA project, it is particularly useful to plan and track key phases like event registration, participant registration, sending reminders, conducting events, and finalizing scores. It helps in visualizing the event timeline, ensuring deadlines are met, managing task dependencies (e.g., participant registration must complete before conducting events), and coordinating responsibilities among admins, schools, and event coordinators.

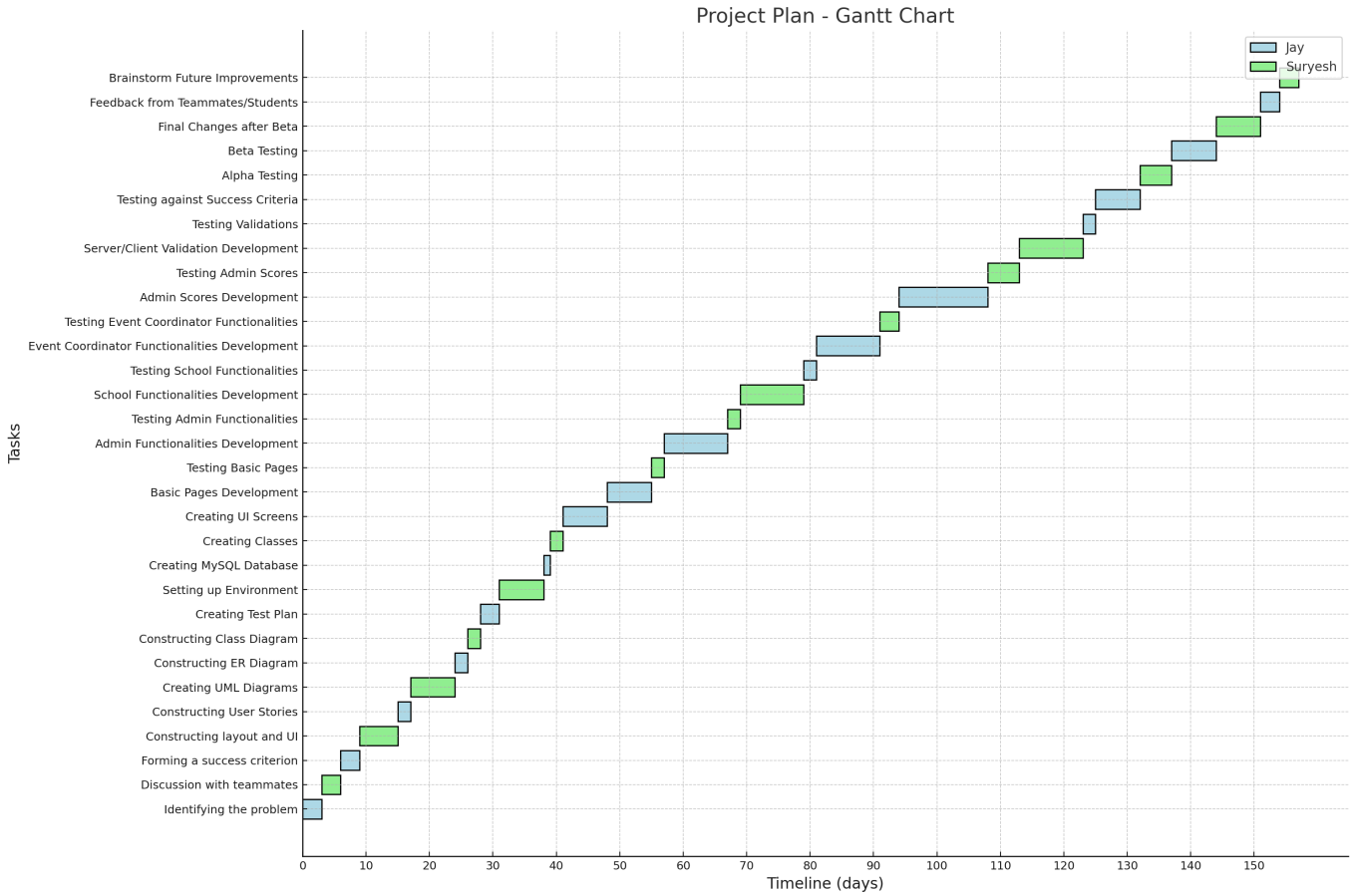


Fig. 1: Gantt Chart

Sprints

A sprint is a short, fixed-length period, typically ranging from 1 to 4 weeks, during which a team works on completing specific tasks from the backlog to deliver a usable product increment. In tools like Confluence and Jira, sprints are managed using a backlog containing epics (large features) and issues (individual tasks), with a board that tracks progress across "To Do," "In Progress," and "Done" stages. This approach helps in maintaining focus, delivering incremental progress, and improving visibility into the team's work, ensuring efficient planning and execution.

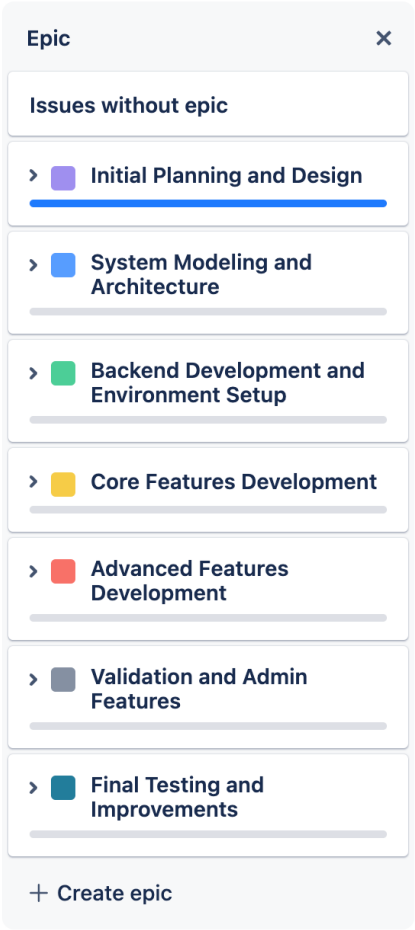


Fig. 2.1: Creating Epics

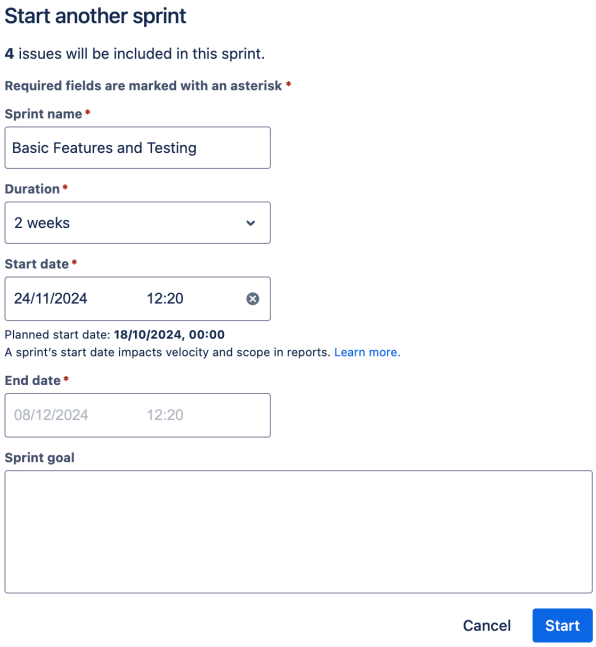


Fig. 2.2: Creating Sprints

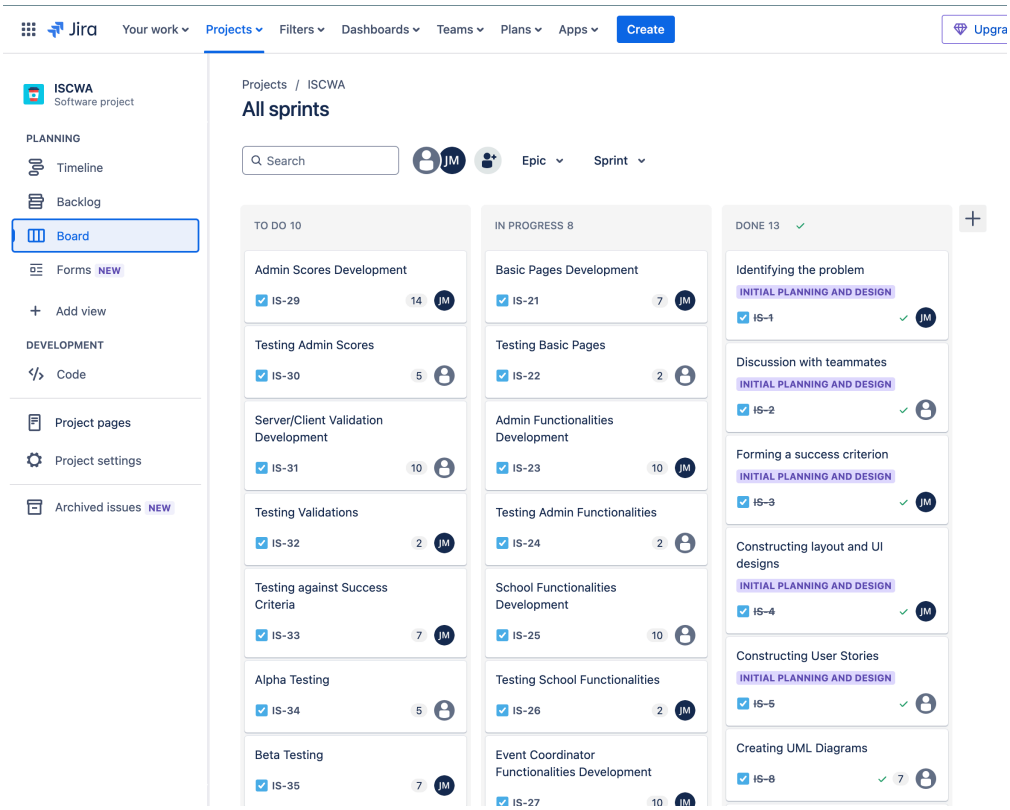


Fig. 2.3: Tracking Progress of Sprints

Use Case Diagram

In the architectural planning and design of the Inter-School Competition Management Web Application (ISCWA), use case diagrams play an integral role in visualizing the interactions between users and the system. A use case diagram is a graphic depiction of the requirements of a system and its interaction with external users, systems, or processes. These diagrams are essential because they outline the system's functionality from the user's perspective, providing a clear and concise map of all user interactions and the expected system behavior in various scenarios. By illustrating the relationships and dependencies between the system and its actors, use case diagrams help ensure that the development team fully understands the user's needs and system requirements. They facilitate communication among project stakeholders, making it easier to identify necessary functionalities and potential issues early in the development process. Below is the use case diagram for ISCWA, which captures the comprehensive functionalities addressed by the application –

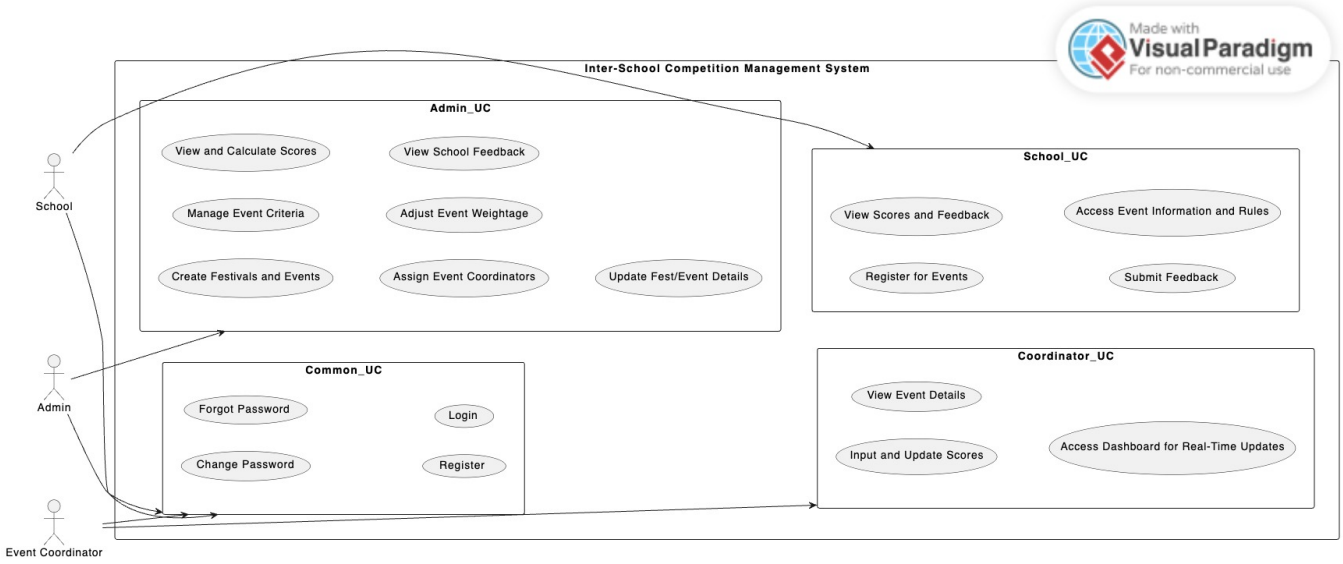


Fig. 1: Use Case Diagram for ISCWA.

Data Dictionary Diagram

A data dictionary (DD) diagram serves as a crucial tool for designing and understanding the database structure. A DD diagram visually represents the various entities within a system and the relationships between these entities. This diagram is fundamental for ensuring the database is well-organized and supports the application's needs efficiently.

Utilizing a DD diagram helps to clarify the data requirements and the relational structure, which in turn assists in maintaining data accuracy and integrity throughout the application. By defining entities (such as schools, events, and participants) and their relationships (like registrations, scores, and feedback), the DD diagram provides a blueprint that guides the database creation and modification. It ensures that all data interactions are logical and that the database supports all required queries and reports effectively.

This systematic approach to database design is vital for optimizing performance and scalability, especially for applications like ISCWA that handle complex data and have multiple user roles. Below is the DD diagram for ISCWA, illustrating the structured data approach and relationships critical for supporting the functional and non-functional requirements of the system –

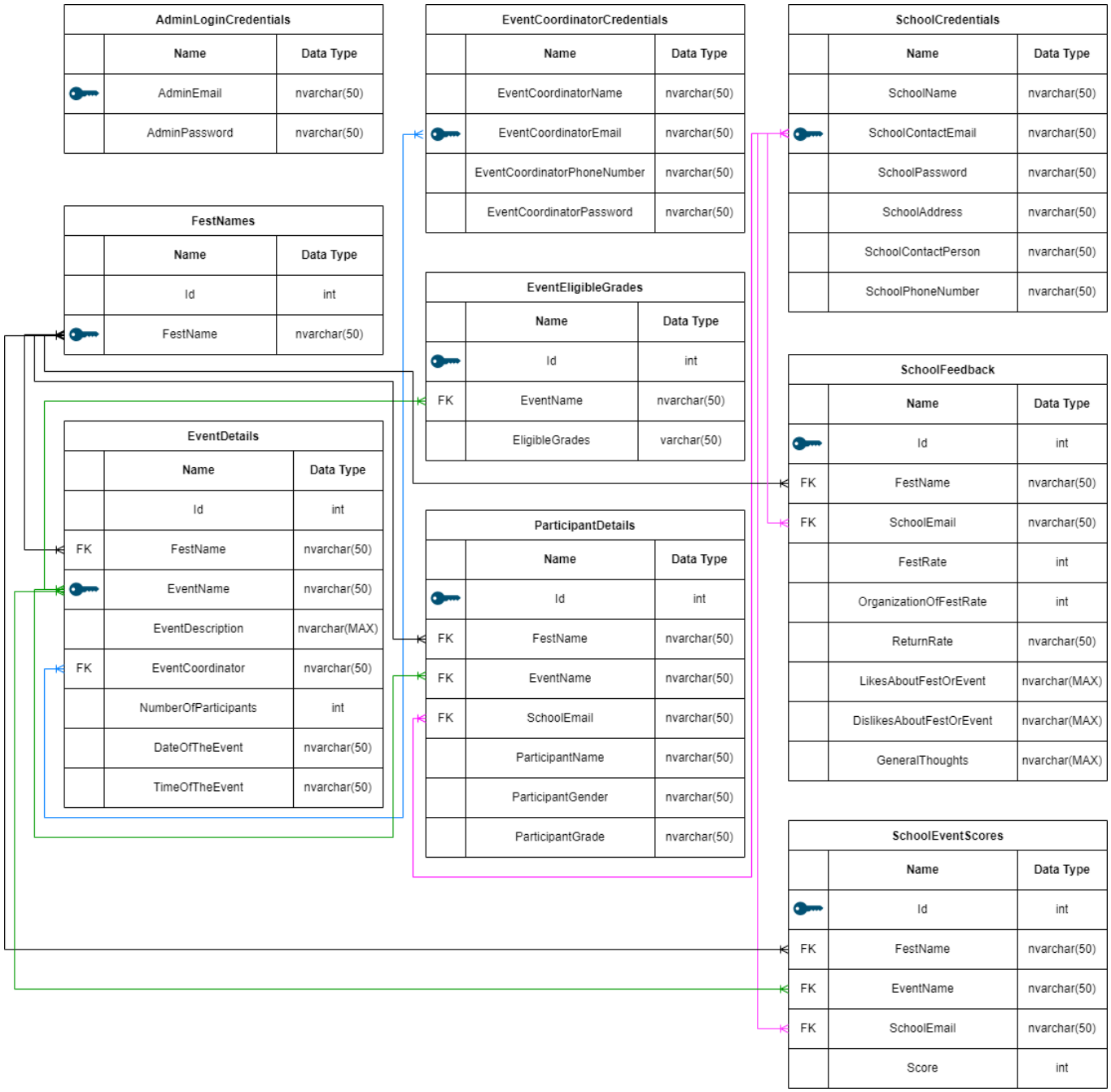


Fig. 2: Data Dictionary Diagram for ISCWA.

Class Diagram

The class diagram is instrumental in modeling the structure of the system at the code level. A class diagram provides a visual representation of classes, their attributes, methods, and the relationships among them, such as inheritance, associations, and dependencies. It serves as a key component of the system's architectural documentation, aiding developers in understanding how different parts of the application interact and are structured.

The class diagram helps ensure consistent and efficient implementation of the system's object-oriented design, promoting reusability and maintainability of code. By clearly outlining the software's building blocks, it facilitates a more streamlined development process and aids in the detection of potential issues

early on. Below is the class diagram for ISCWA, which delineates the relationships and interactions within the application's architecture –

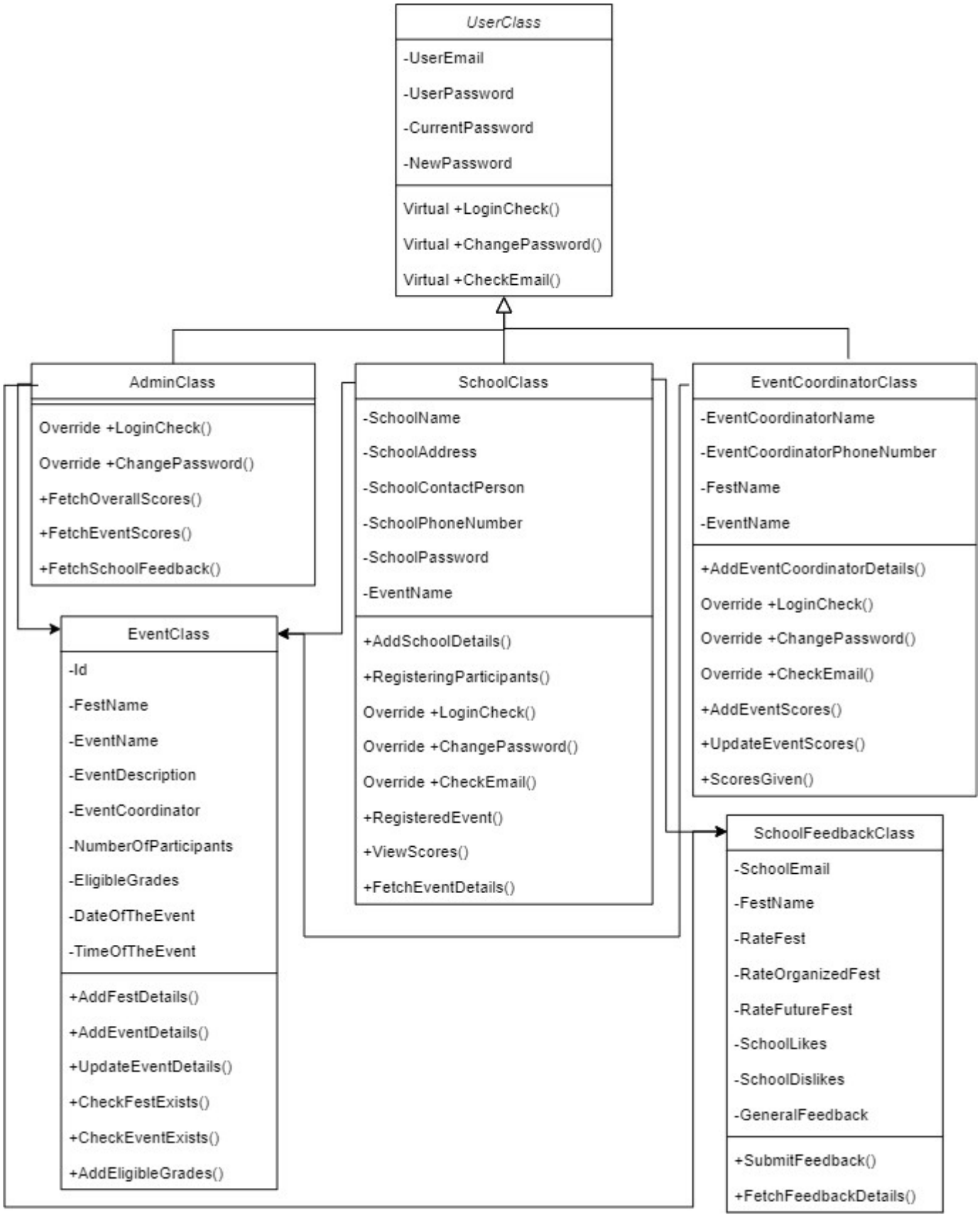


Fig. 3: Class Diagram for ISCWA.

Activity Diagrams

Activity diagrams are utilized to illustrate the dynamic aspects of the system. These diagrams provide a detailed graphical representation of workflows showing the sequence of activities involved in various processes within the application, such as registration, event management, and score updating. Activity diagrams help in visualizing the flow of control from one activity to another, including decisions, loops, and concurrent tasks.

The clarity provided by activity diagrams is invaluable for both developers and stakeholders to understand the procedural steps and business logic of different functionalities. They serve as a blueprint for implementing workflow management and enhance the understanding of complex operations, ensuring the system's behavior aligns with user expectations. Below are the activity diagrams for ISCWA, which details the operational sequences and decision-making pathways critical to the application's processes -

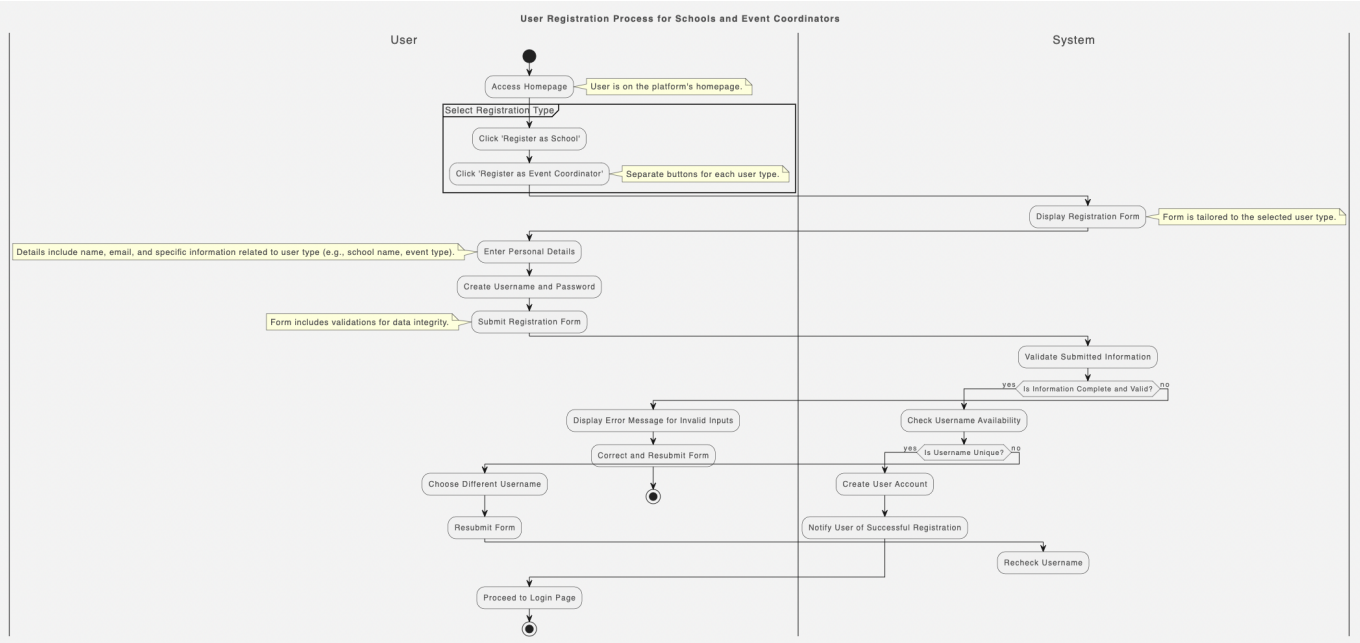


Fig. 4.1: Activity Diagram 1 - User Registration.

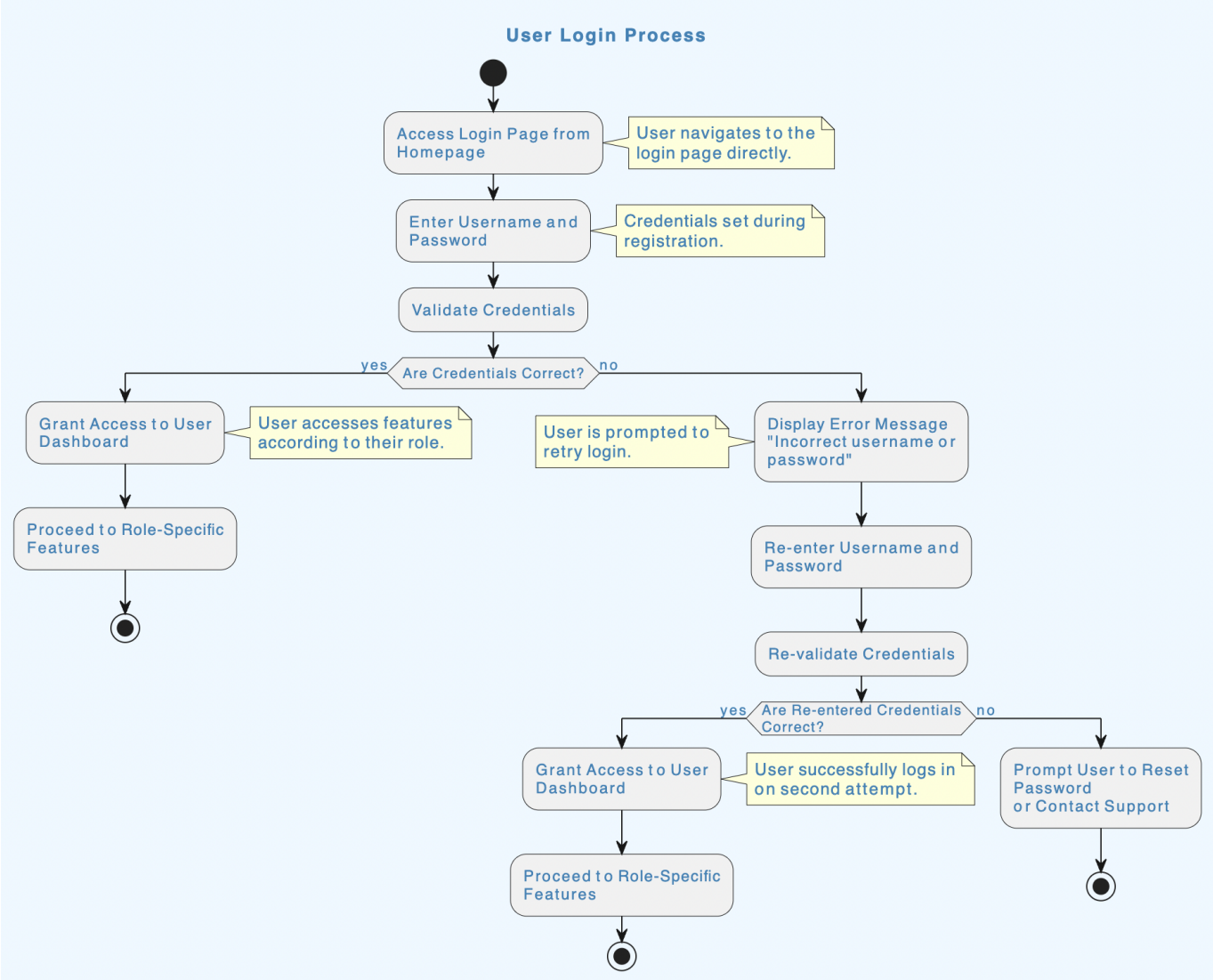


Fig. 4.2: Activity Diagram 2 - User Login.

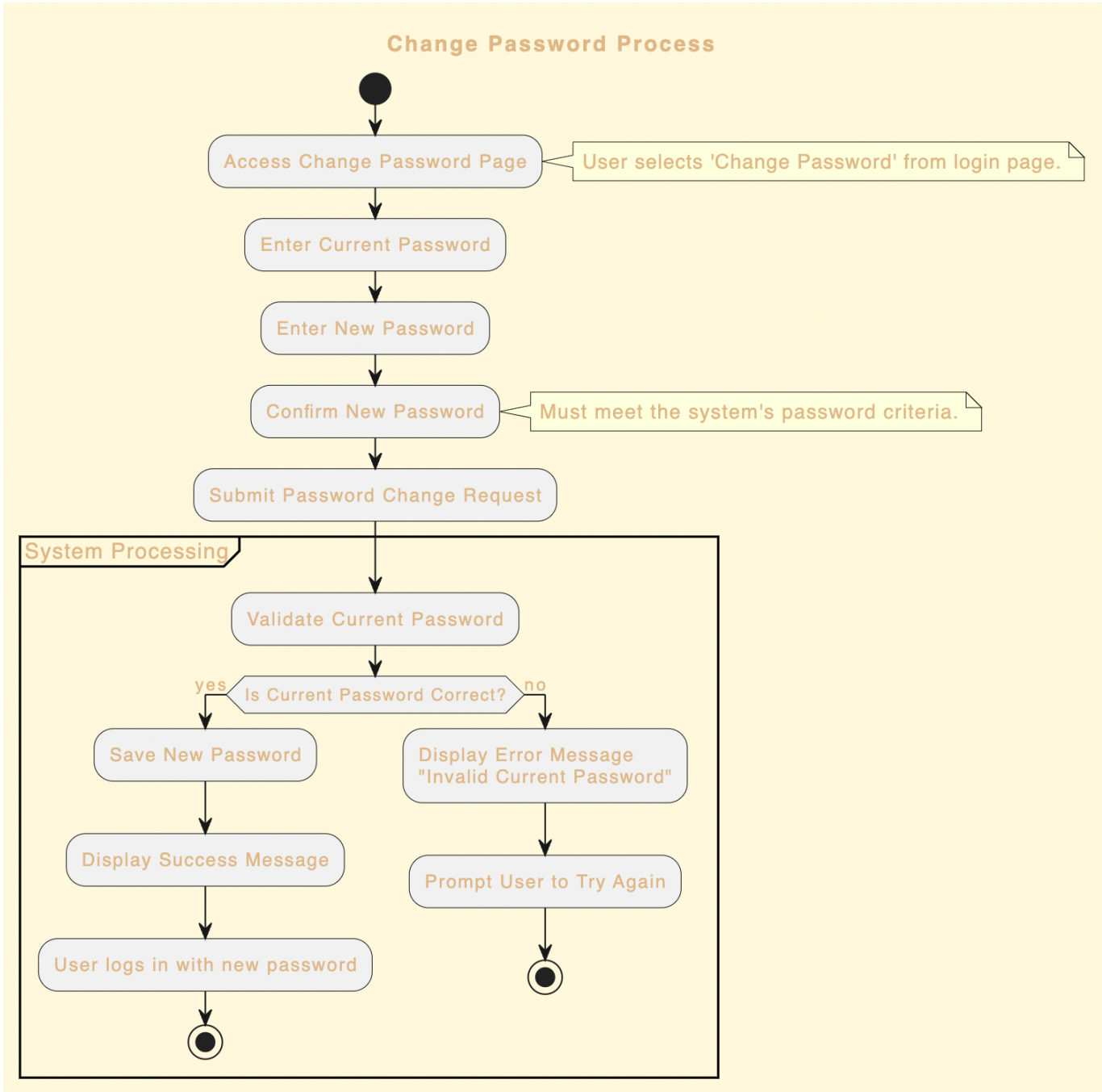


Fig. 4.3: Activity Diagram 3 – Change Password.

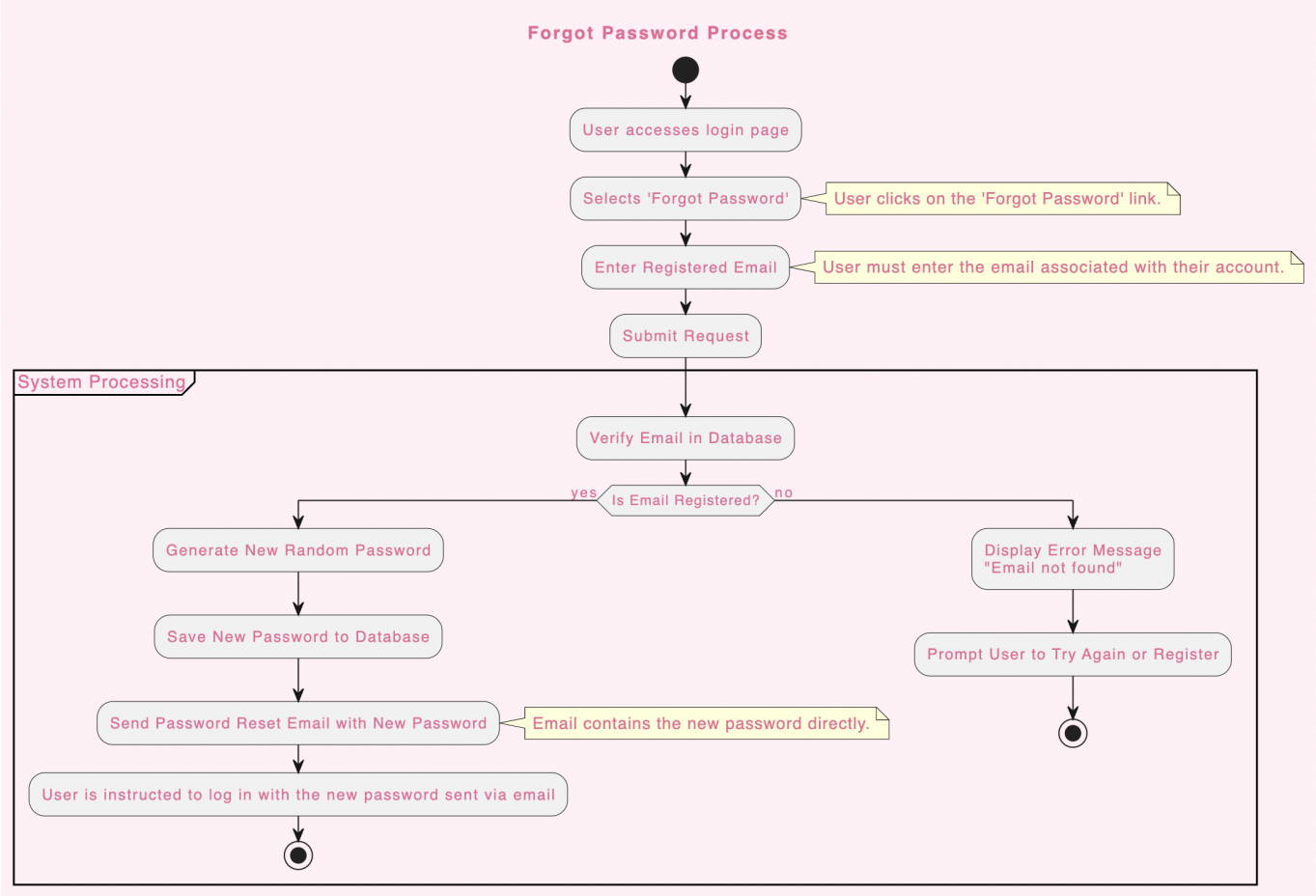


Fig. 4.4: Activity Diagram 4 – Forgot Password.

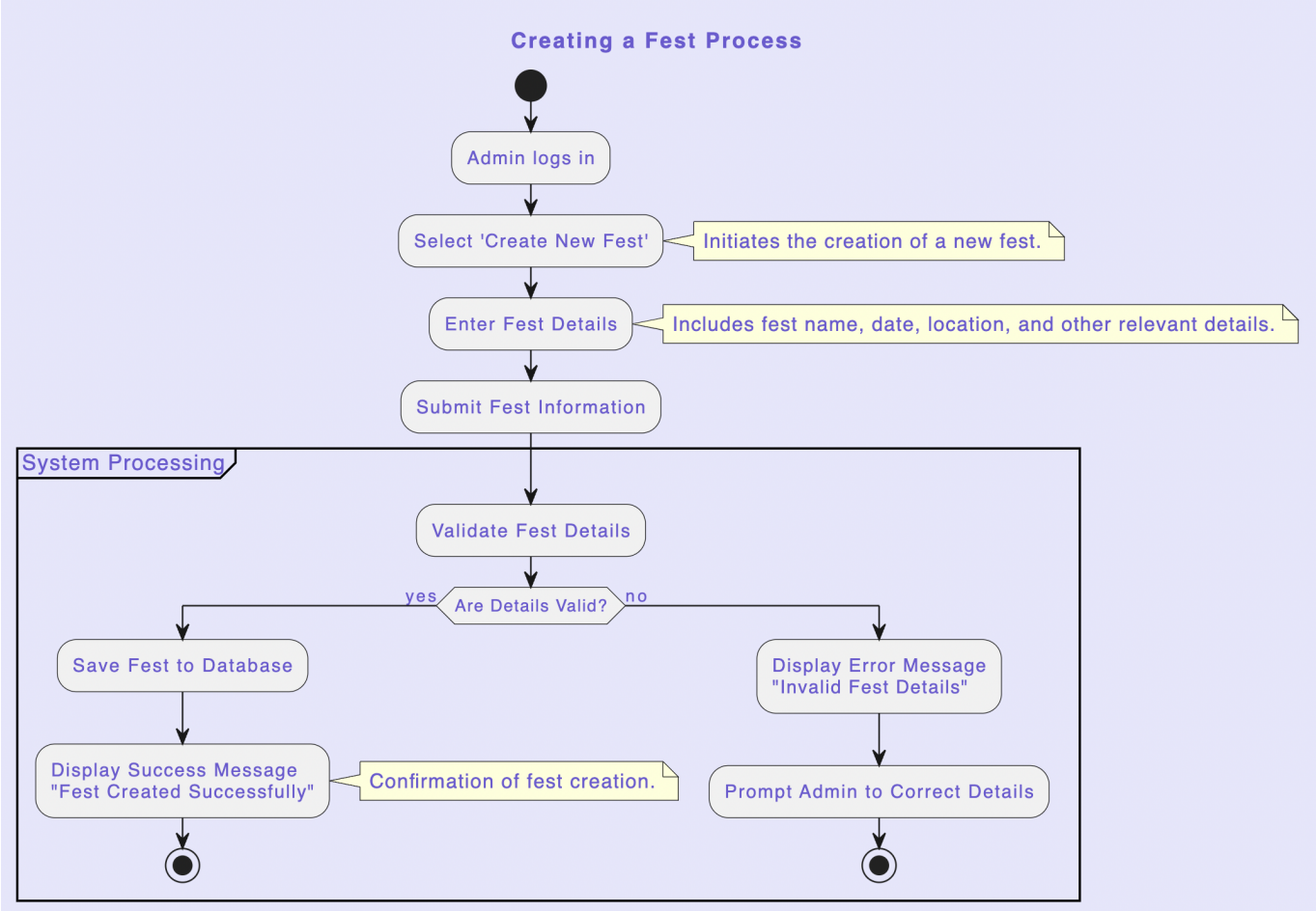


Fig. 4.5: Activity Diagram 5 – Creating a Fest.

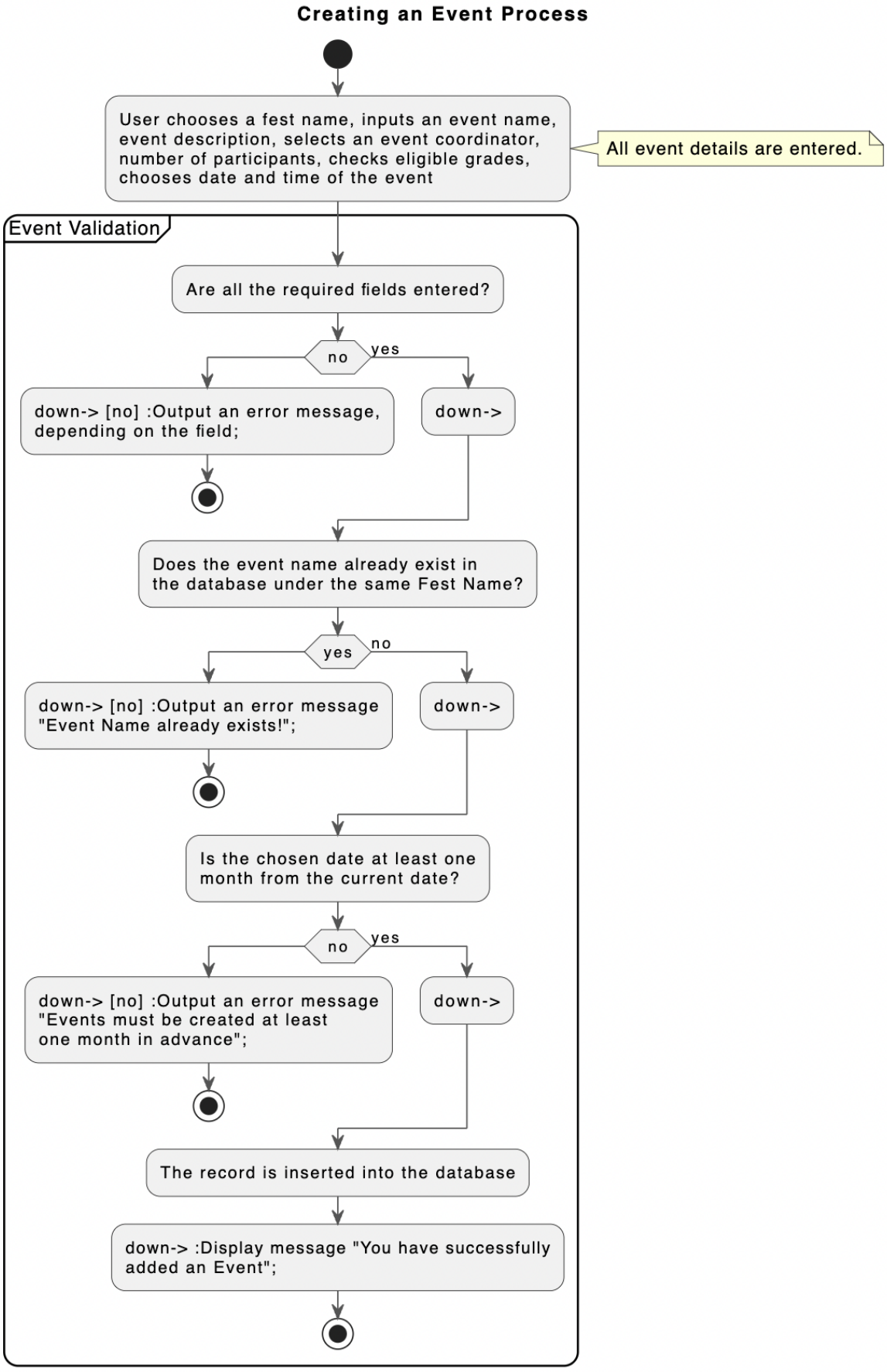


Fig. 4.6: Activity Diagram 6 – Creating an Event.

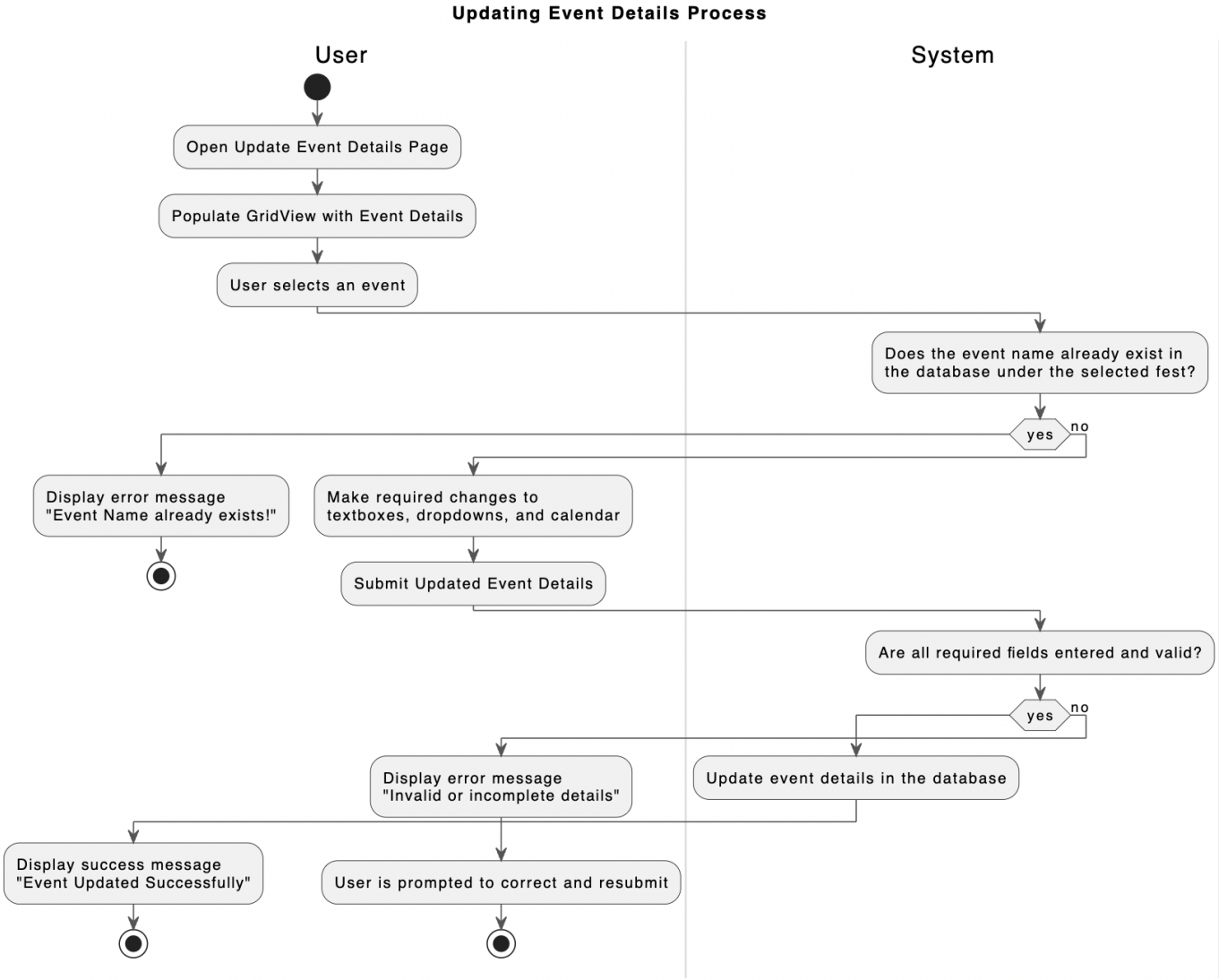


Fig. 4.7: Activity Diagram 7 – Updating Event Details.

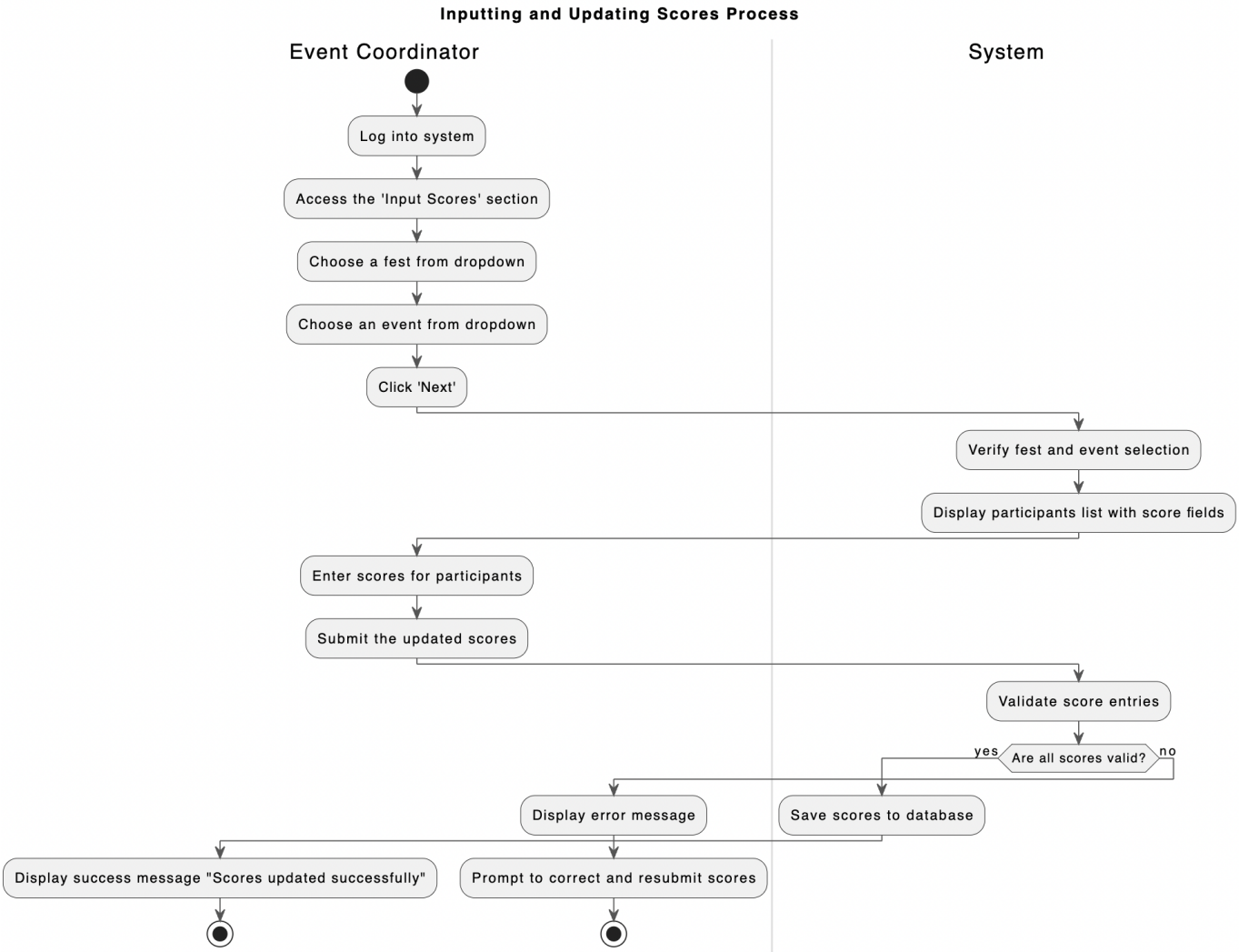


Fig. 4.8: Activity Diagram 8 – Inputting and Updating Scores.

Sequence Diagrams

Sequence diagrams are essential for detailing the interaction between objects over time. These diagrams depict how objects in the system interact with each other and in what order these interactions occur, capturing the dynamic behavior of different parts of the system during runtime.

Sequence diagrams are particularly useful for visualizing the flow of messages, events, and actions between various components and entities, such as users, system interfaces, and databases. They help clarify complex functionality, ensuring that all system interactions are aligned with the intended design and business logic.

By providing a clear sequence of operations, these diagrams facilitate the understanding and troubleshooting of the functional flows within the application, aiding developers in building and integrating components effectively. Below are the sequence diagrams for ISCWA, illustrating the critical interactions within the application that drive its core processes –

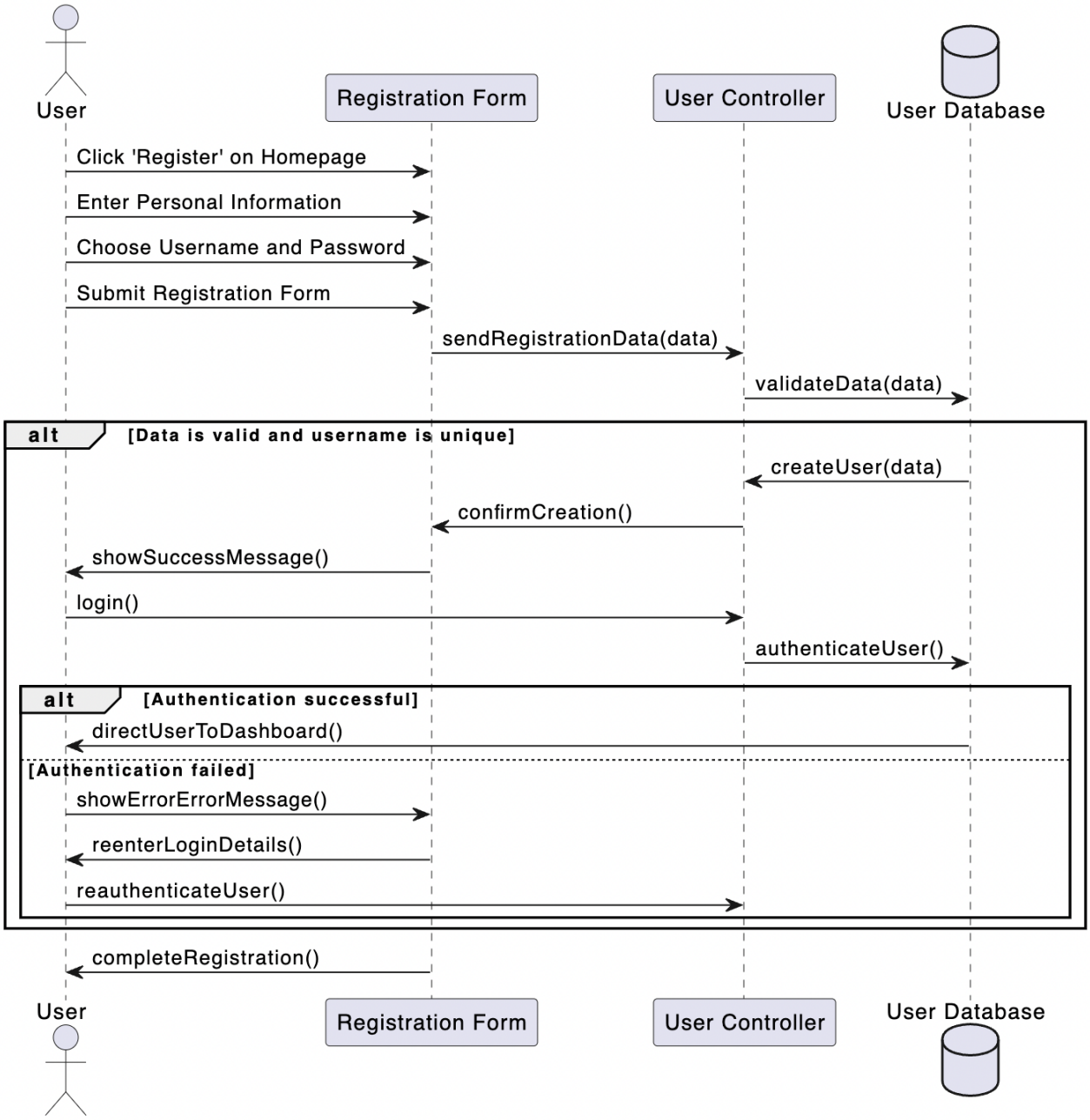


Fig. 5.1: Sequence Diagram 1 - User Registration.

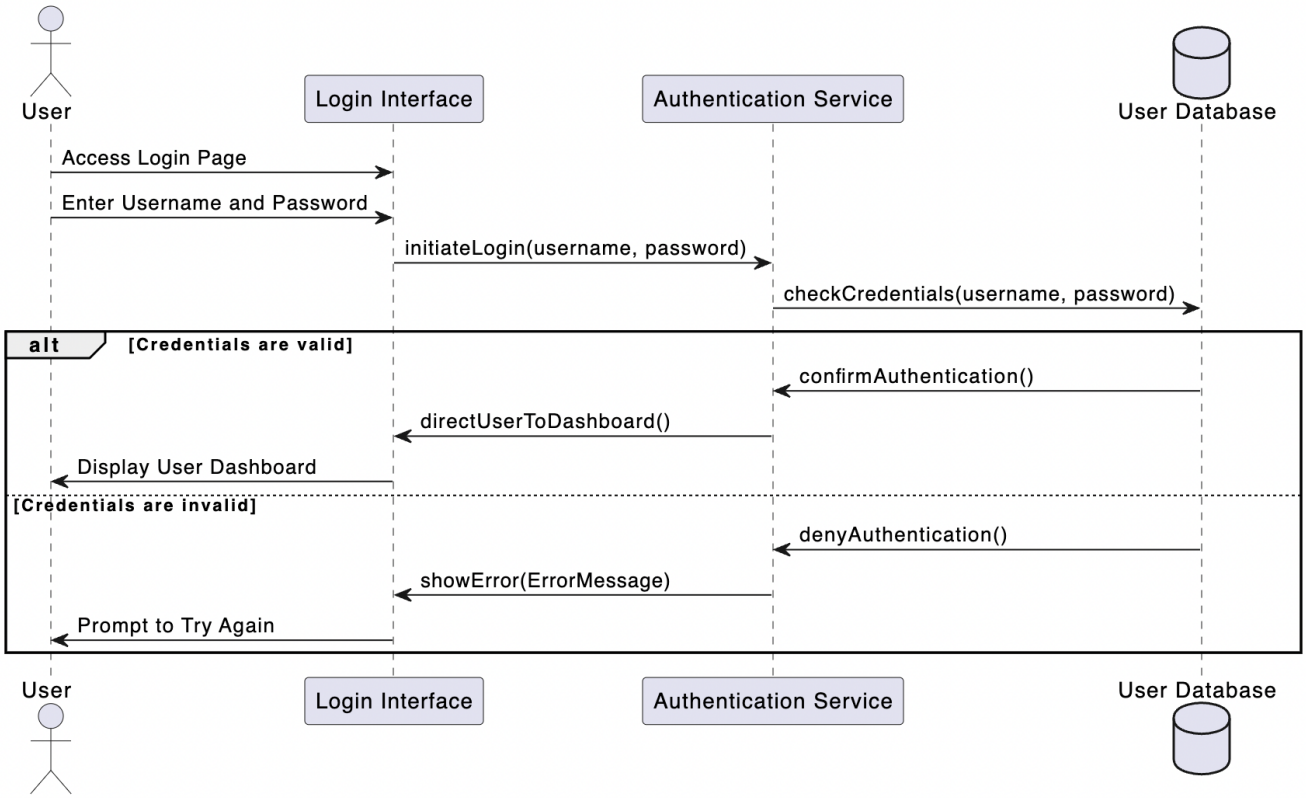


Fig. 5.2: Sequence Diagram 2 - User Login.

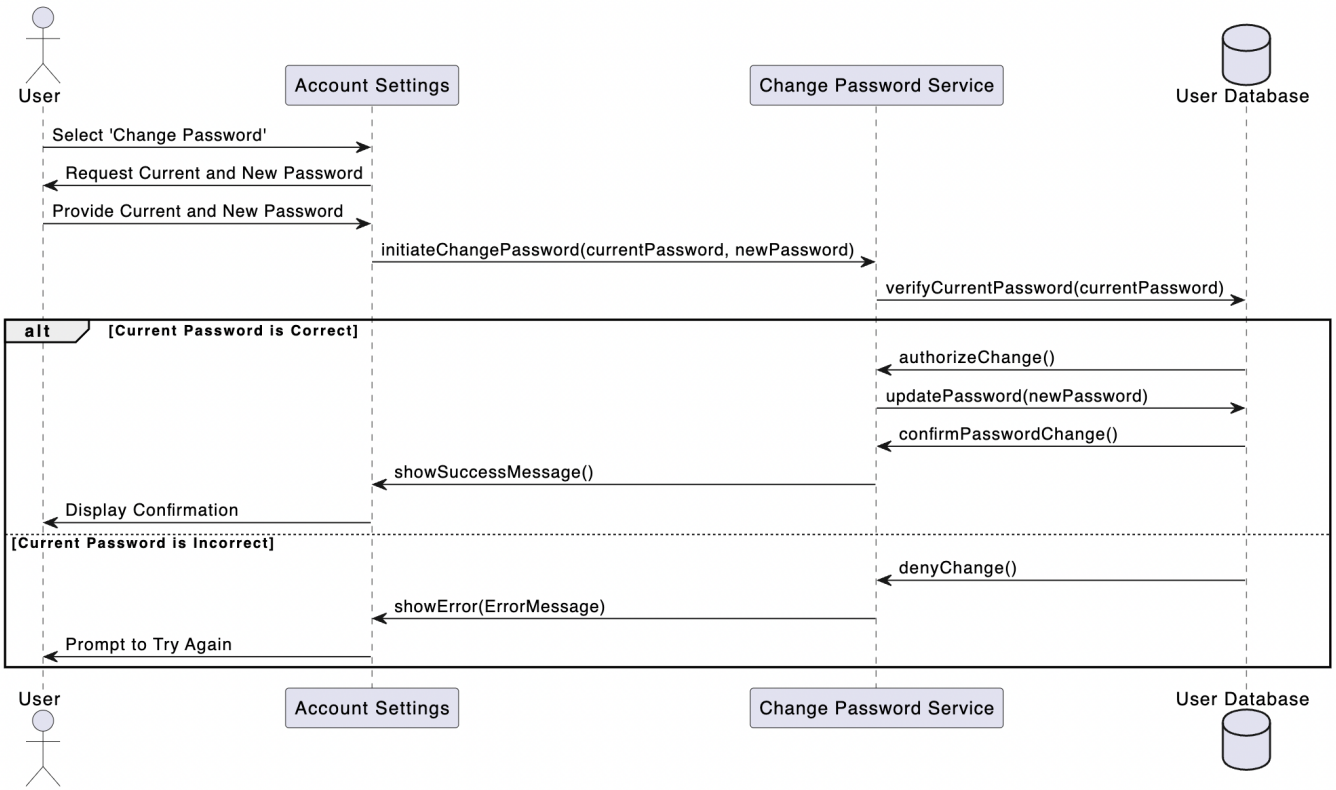


Fig. 5.3: Sequence Diagram 3 – Change Password.

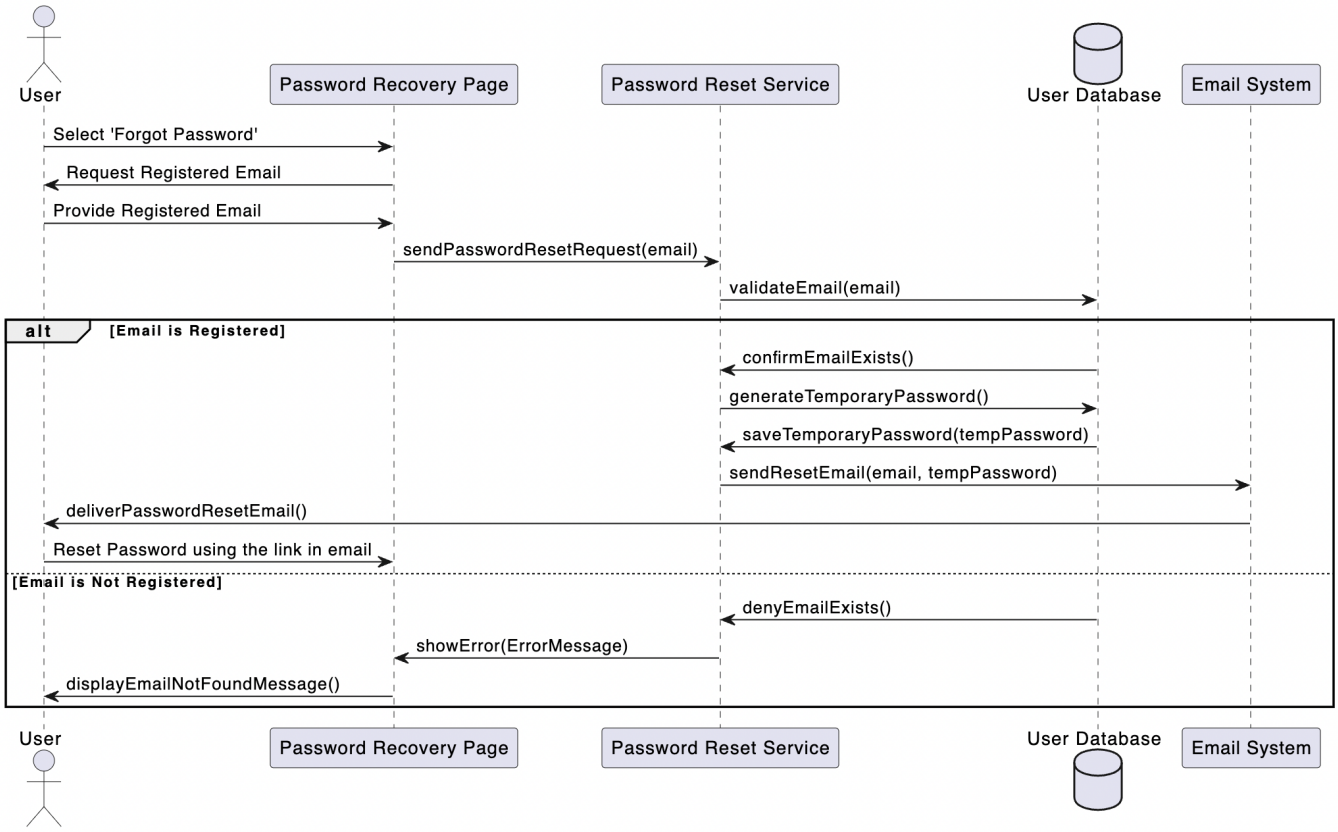


Fig. 5.4: Sequence Diagram 4 – Forgot Password.

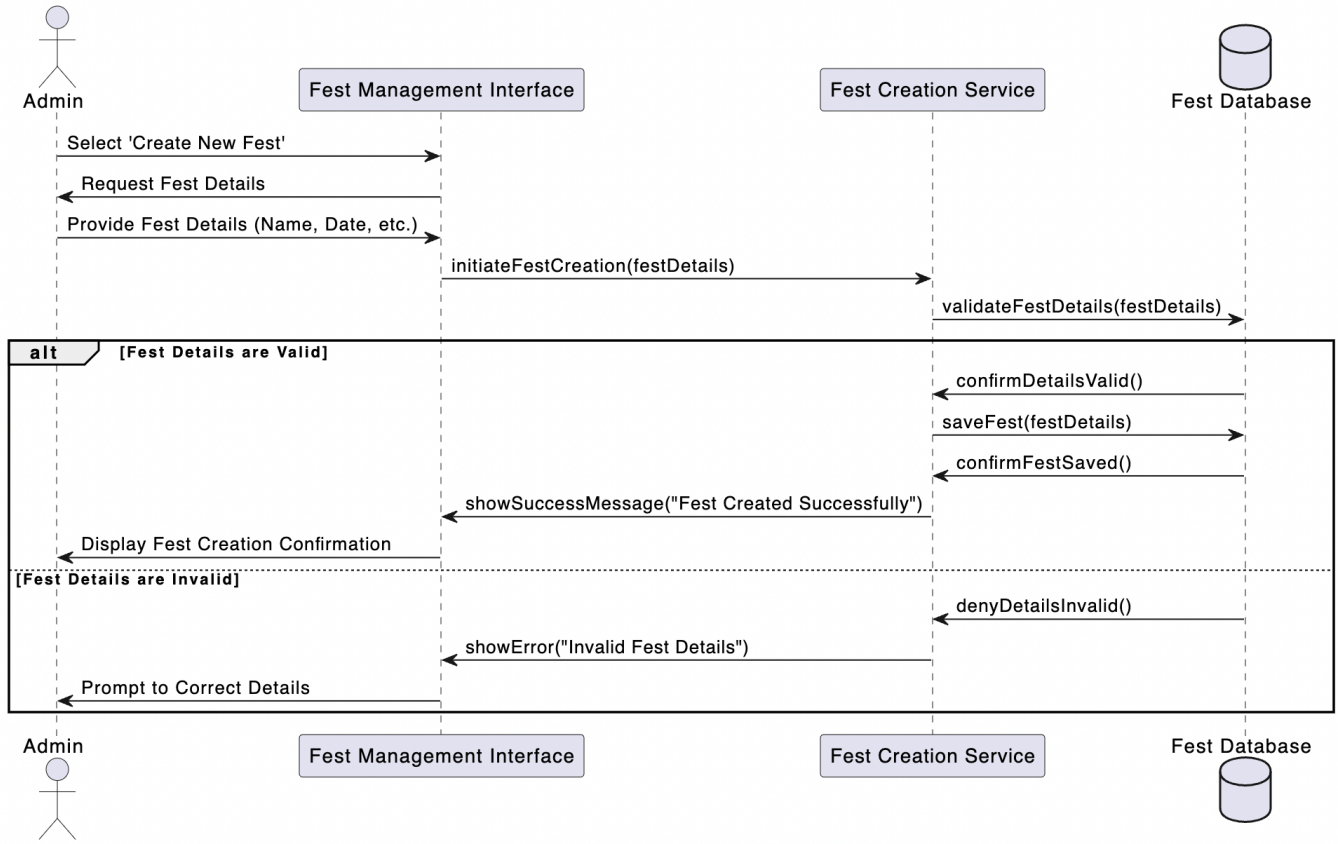


Fig. 5.5: Sequence Diagram 5 – Creating a Fest.

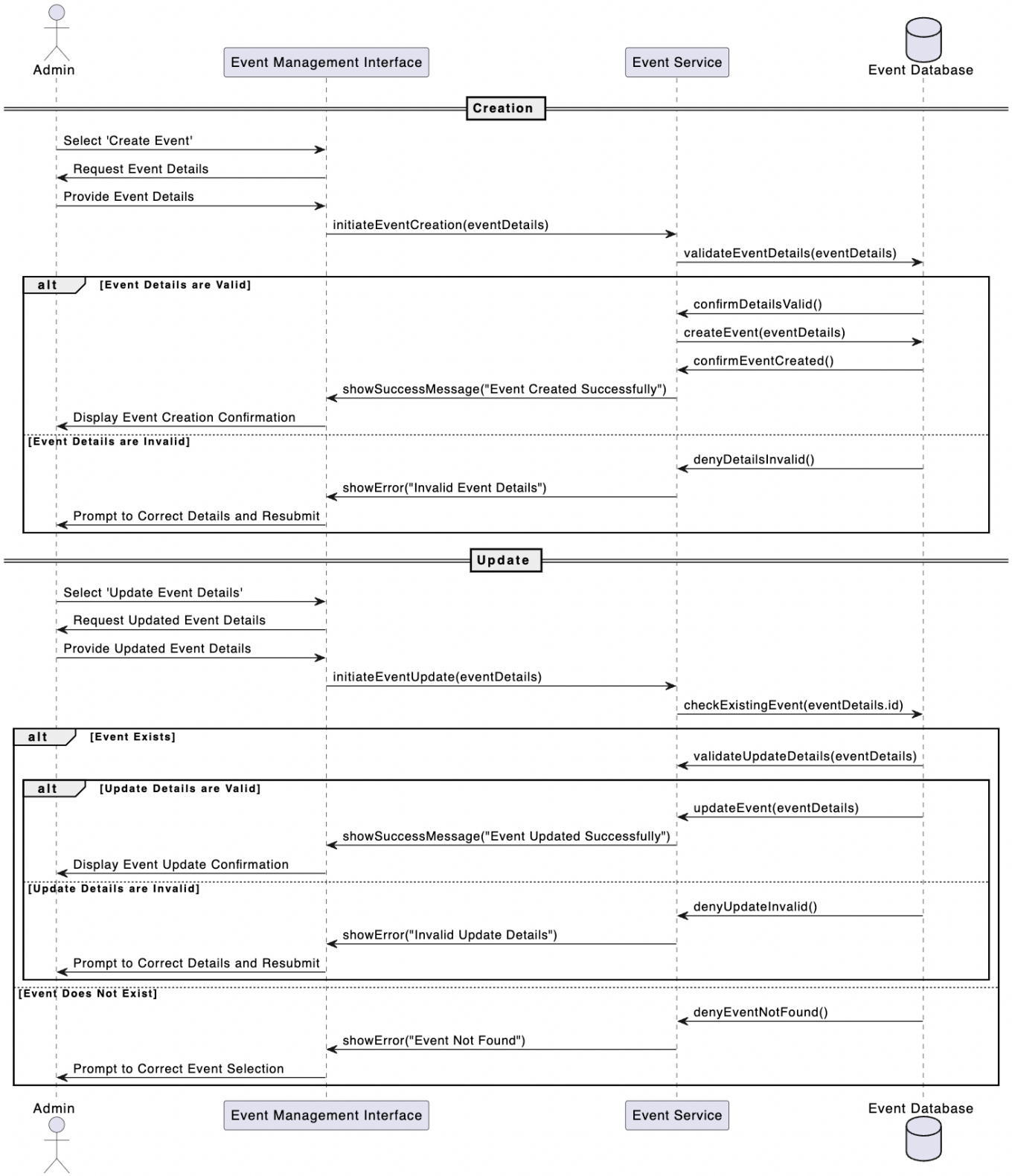


Fig. 5.6: Sequence Diagram 6 – Creating and Updating Event Details.

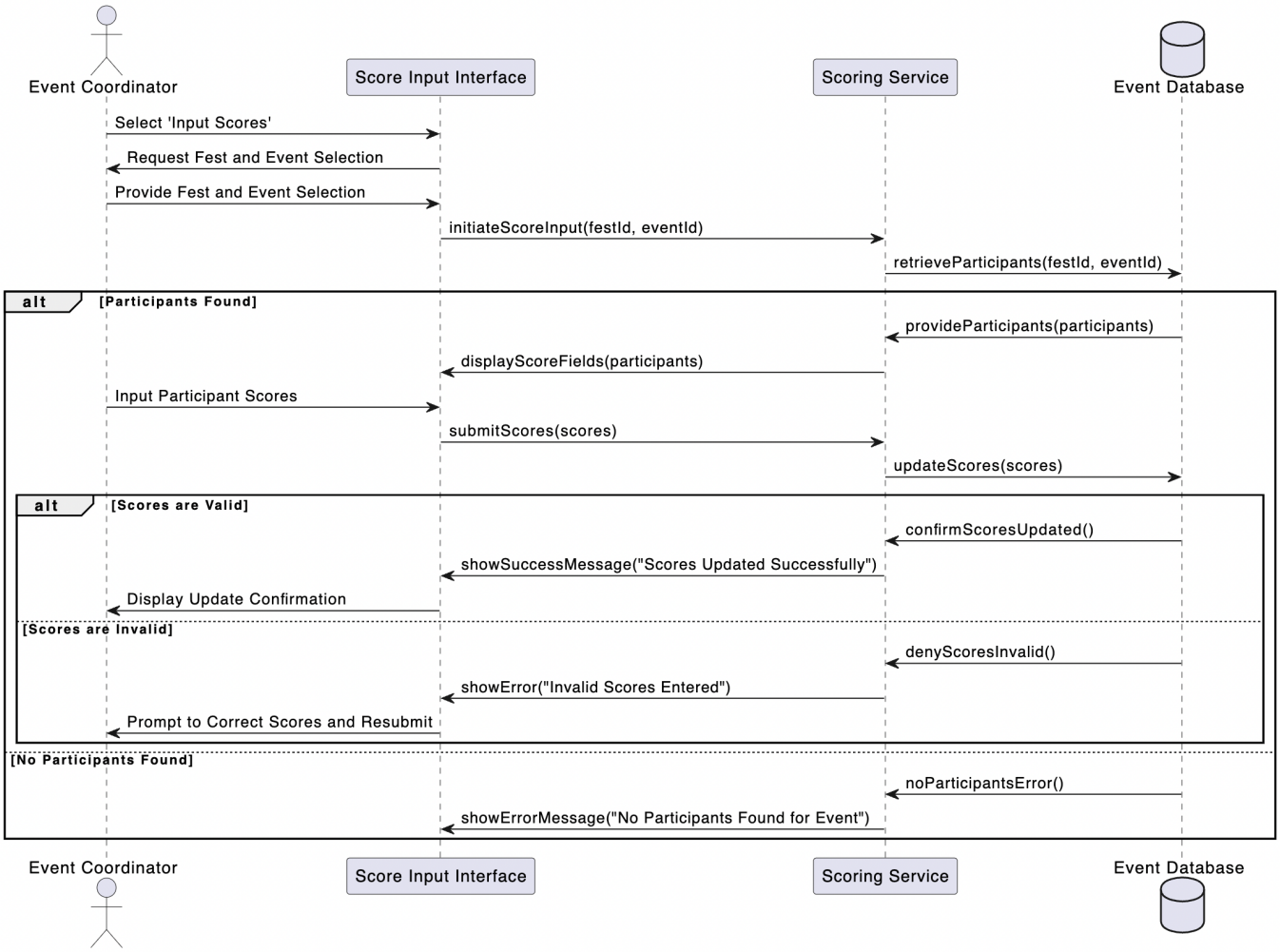


Fig. 5.7: Sequence Diagram 7 - Inputting and Updating Scores.

Layered Architecture

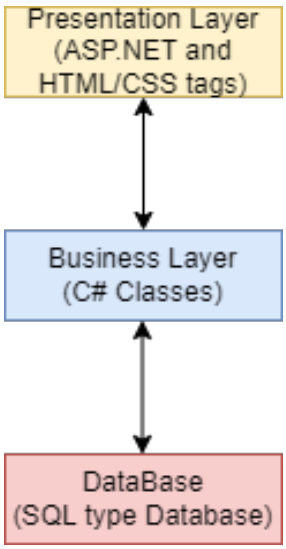
The decision to employ a 3-tier layered architecture in the development of the Inter-School Competition Management Web Application (ISCWA) was driven by several strategic and technical considerations. This architecture divides the application into three distinct layers—Presentation, Business, and Data—each with specific responsibilities, thereby promoting a clean separation of concerns. This separation not only enhances modularity but also improves maintainability, scalability, and security of the application.

1. Presentation Layer: The topmost level of the application, this layer serves as the interface between the user and the system. Developed using ASP.NET controls along with HTML and CSS, it is designed to provide a seamless and intuitive user experience while ensuring that the display logic is kept separate from the business logic. This layer handles all user interactions with the system, presenting data and accepting user inputs and commands. The use of ASP.NET allows for dynamic page generation based on the current state of the application, which is crucial for real-time functionalities such as score updates and feedback submissions.

2. Business Layer: At the heart of the ISCWA is the Business Layer, which encapsulates all the core business logic of the application. This layer, implemented in C#, processes information exchanged between the Presentation and Data layers, acting on user commands, making logical decisions, and executing business rules. By isolating the application's business logic in this middle layer, changes in business rules or processes can be made with minimal impact on the other layers. This modularity is vital for maintaining and upgrading the system as it evolves in response to user feedback or changing requirements.

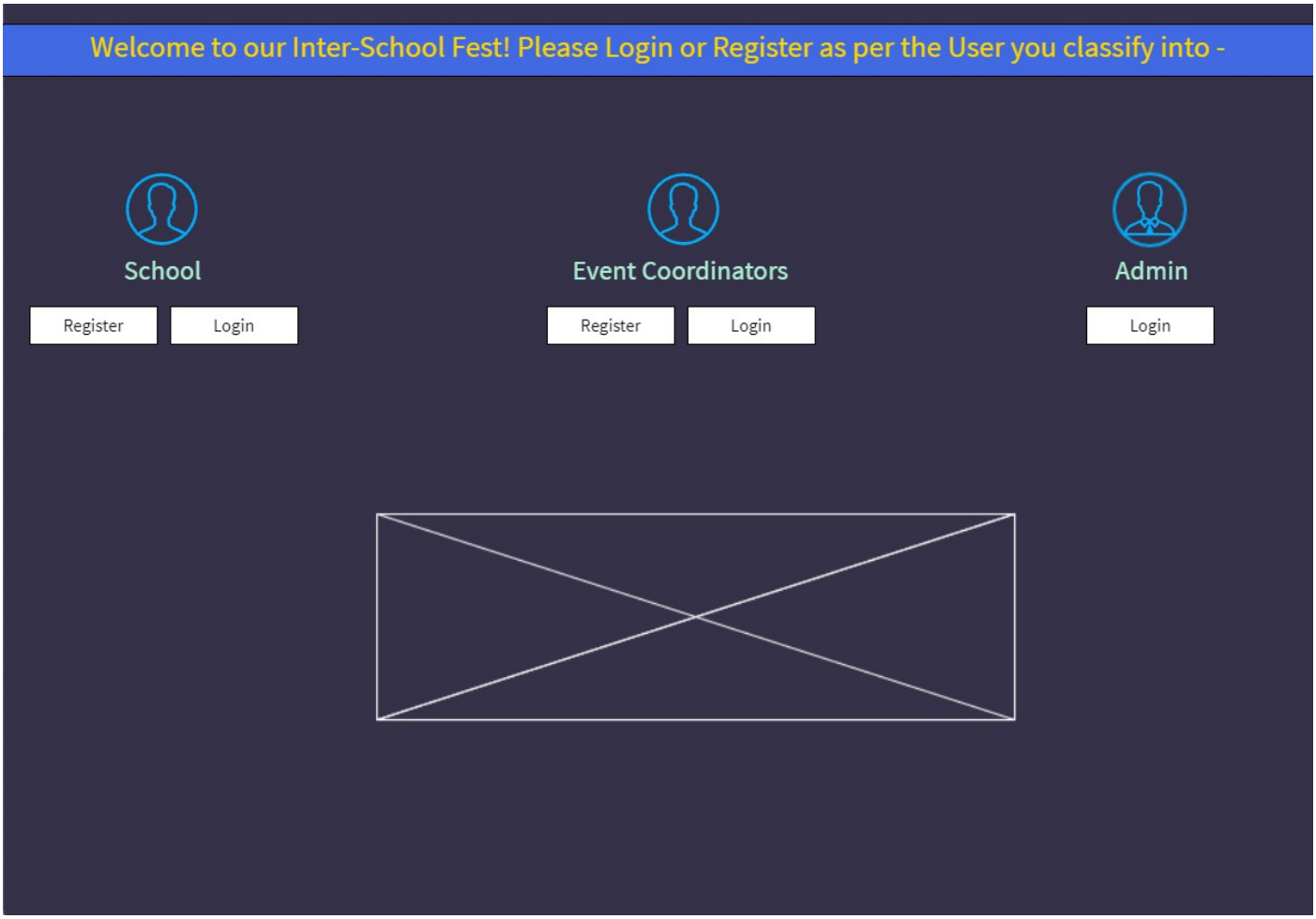
3. Data Tier Layer: The foundation of the ISCWA is its Data Tier Layer, which manages the storage, retrieval, and manipulation of data in a MySQL database. This layer ensures that all data interactions are performed securely and efficiently, providing persistent data storage and robust data access mechanisms. Structuring this layer to handle all interactions with the database consolidates data access functionalities and enhances data integrity and security. The use of SQL ensures that data queries are both powerful and flexible, supporting complex data retrieval required for the application's reporting and analytical features.

Choosing a 3-tier architecture aligns with best practices for building scalable, robust, and secure web applications. Each layer can be developed, tested, and maintained independently, reducing dependencies and potential conflicts during development cycles. This architecture also supports better distribution of responsibilities, which can be particularly beneficial for team development settings where different teams can focus on specific aspects of the application. Furthermore, this layered structure facilitates the scaling of each layer according to its load and complexity independently of the others, enhancing the overall performance and responsiveness of the application.



Screen Designs (Self-Made using MockFlow.com)

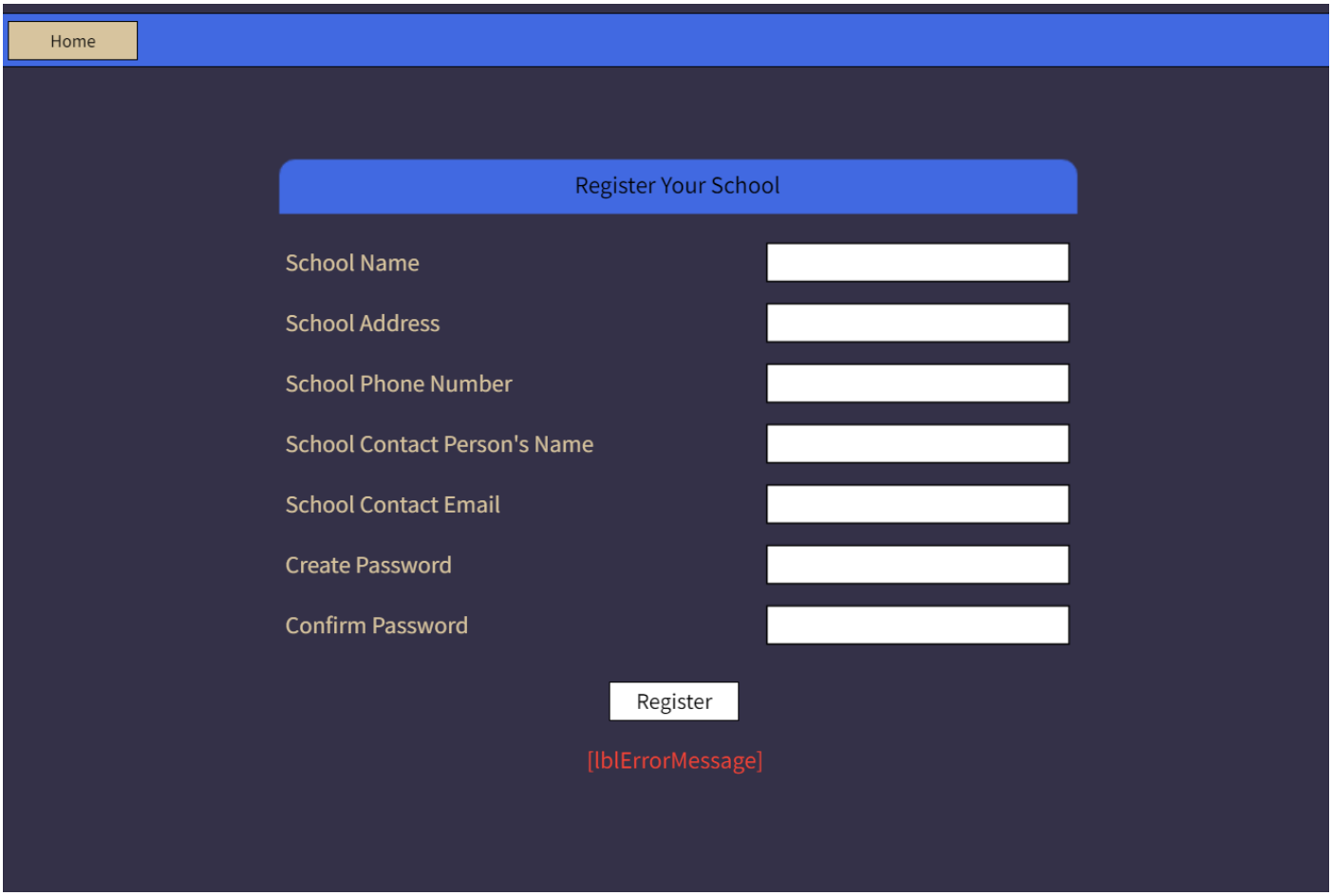
Homepage – This is the 1st screen that is displayed when the web application is run. It allows the School and the Event Coordinators to register. It also allows all the 3 users to login as well.



UI Component	Functionality
School Register Button	When clicked, it redirects the user to the School Registration Page where the user can register his or her school for the inter-school event.
School Login Button	When clicked, it redirects the user to the School Login Page where the user can login using their registered email ID and password.
Event Coordinator Registration Button	When clicked, it redirects the user to the Event Coordinator Registration Page where the user can register as an Event Coordinator.
Event Coordinator Login Button	When clicked, it redirects the user to the Event Coordinator Login Page where the user can login using their registered email ID and password.
Admin Login Button	When clicked, it redirects the user to the Admin Login Page where the user can login using pre-registered email and password created by the Host School.
Image Placeholder	When the page loads, a user-friendly image of a school

	will be displayed.

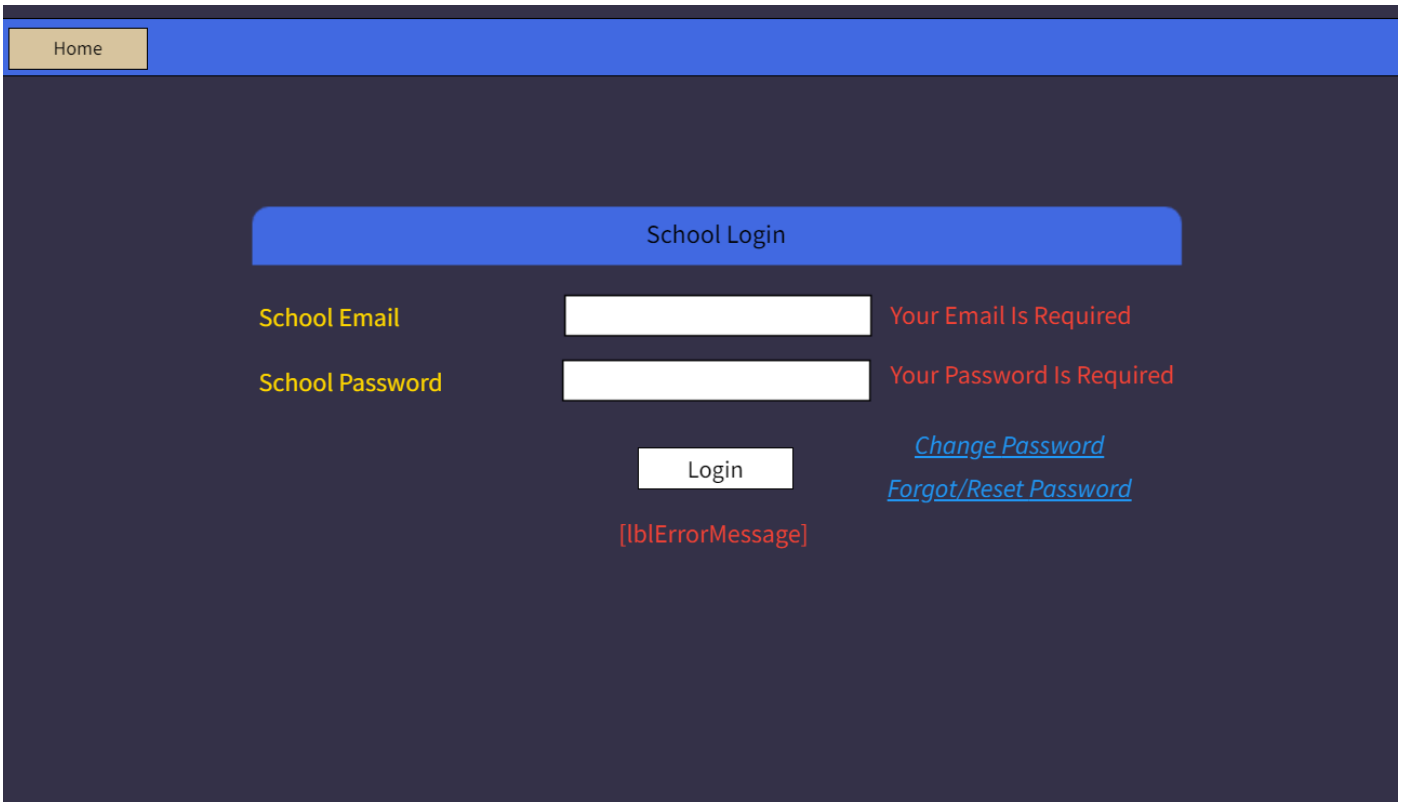
School Registration Page – This is the screen that the user is redirected to from the previous page (Homepage) when the user clicks on the "Register" button. It allows the Schools to register themselves on the web application.



UI Component	Functionality
Home Button	When clicked, it redirects the user back to the Homepage where the registration and login buttons are present.
School Name Textbox	Textbox in which the user enters the School Name.
School Address Textbox	Textbox in which the user enters the School's Address.
School Phone Number Textbox	Textbox in which the user inputs the School's Phone Number.
School Contact Person's Name Textbox	Textbox in which the user inputs the Name of the person which the Host School can contact.
School Contact Email Textbox	Textbox in which the user enters the Email of the person entered in the previous Textbox, "School Contact Person's Name".
Create Password Textbox	Textbox in which the user will create a Password for the School that can be used to Login.
Confirm Password Textbox	Textbox in which the user re-enters the password inputted in the previous Textbox, "Create Password" Textbox.

Register Button	When clicked, it saves the values inputted by the user in the designated SQL database table. It also redirects the user to the School Login Page once the registration is successful.
JavaScript Validation	When the Register Button is clicked, a validation takes place which ensures that no fields are left empty, the email is in the correct format, and the passwords match.
Label	Displays an error message if the registration failed, otherwise, displays a message that indicates that the user has registered successfully.

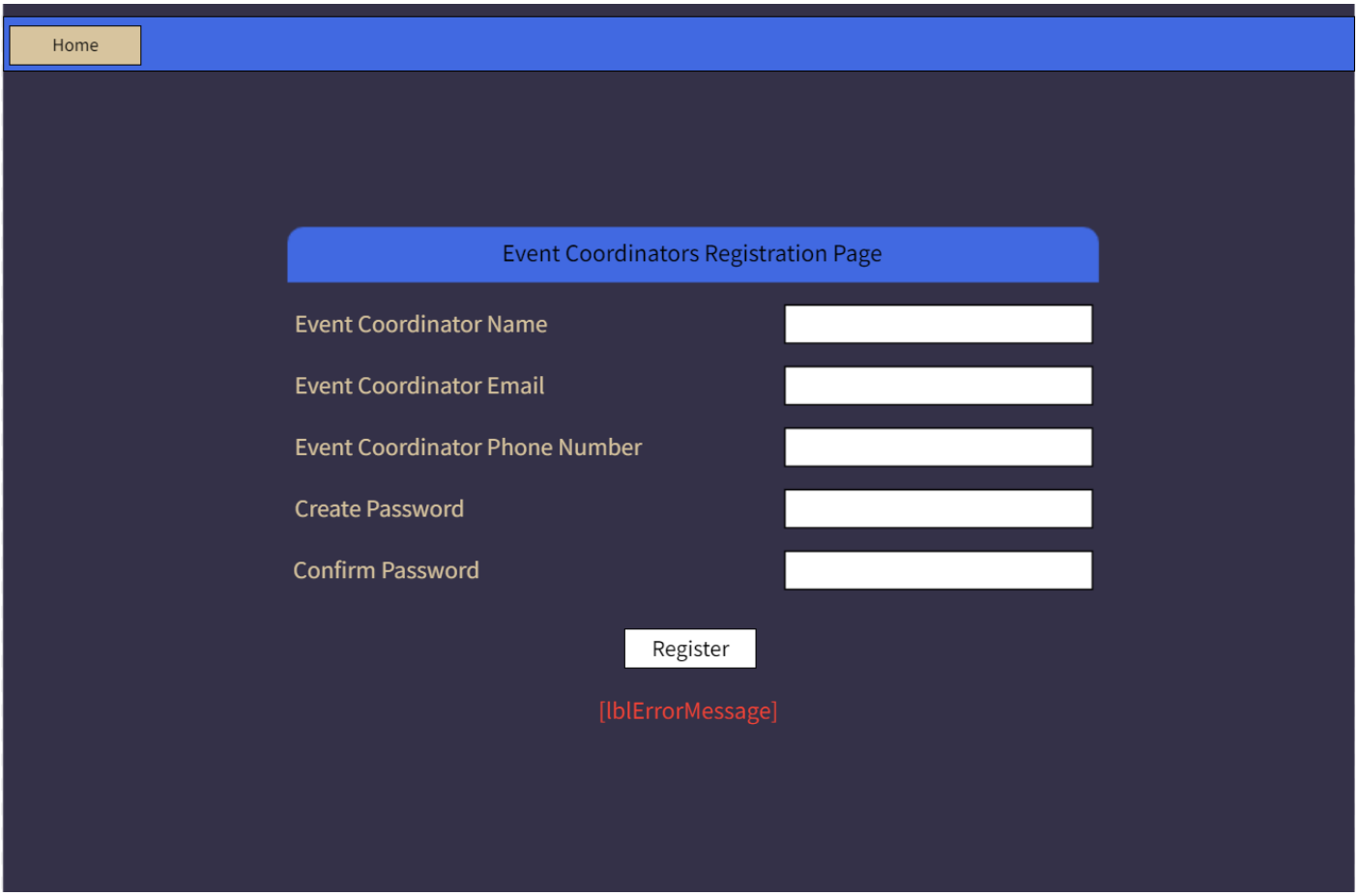
School Login Page **(All the 3 login screens are similar except the heading, email and password labels are user- specific)** – This is the screen that the user is automatically redirected to from the previous page (School Registration Page) if the registration was successful or when the user clicks on the "Login" button displayed under the "School" label. It allows the Schools to login into the web application.



UI Component	Functionality
Home Button	When clicked, it redirects the user back to the Homepage.
School Email Textbox	Textbox in which the user inputs the Registered School Email ID.
School Email Textbox Validation Error	Validation which checks whether the Textbox is empty.
School Password Textbox	Textbox in which the user inputs the Password they had created on the School Registration Page.

School Password Textbox Validation Error	Validation which checks whether the Textbox is empty.
Login Button	When clicked, the values inputted by the user are verified/crosschecked from the Database and redirects the user to the School Home Page if the login is successful.
Label	Displays an error message if the login fails.
Change Password Link Button	When clicked, it redirects the user to the School Change Password Page where the user can change the password.
Forgot/Reset Password Link Button	When clicked, it redirects the user to the School Forgot Password Page where the user can enter his or her registered email ID to get a new password on their mail.

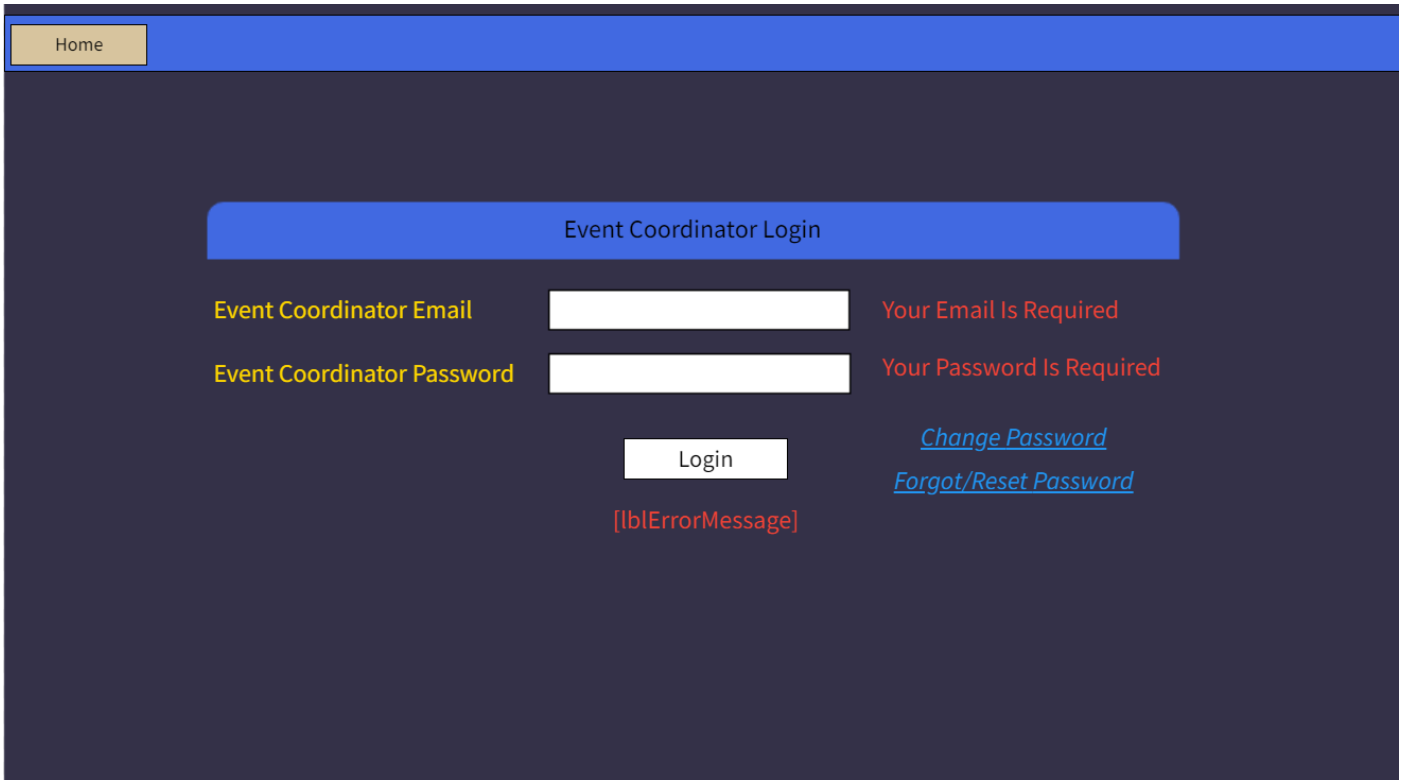
Event Coordinator Registration Page – This is the screen that the user is redirected to from the previous page (Homepage) when the user clicks on the "Register" button displayed under the "Event Coordinator" label. It allows the Event Coordinators to register themselves on the web application.



UI Component	Functionality
Home Button	When clicked, it redirects the user back to the Homepage.
Event Coordinator Name Textbox	Textbox in which the user enters his or her name.
Event Coordinator Email Textbox	Textbox in which the user enters his or her email.

Event Coordinator Phone Number Textbox	Textbox in which the user enters his or her phone number.
Create Password Textbox	Textbox in which the user will create a Password for the him or herself that can be used to Login.
Confirm Password Textbox	Textbox in which the user re-enters the password inputted in the previous Textbox, "Create Password" Textbox.
Register Button	When clicked, it saves the values inputted by the user in the designated SQL database table. It also redirects
	the user to the Event Coordinator Login Page once the registration is successful.
Label	Displays an error message if the registration failed.
JavaScript Validation	When the Register Button is clicked, a validation occurs which ensures that no fields are left empty, the email is in the correct format, and the passwords match.

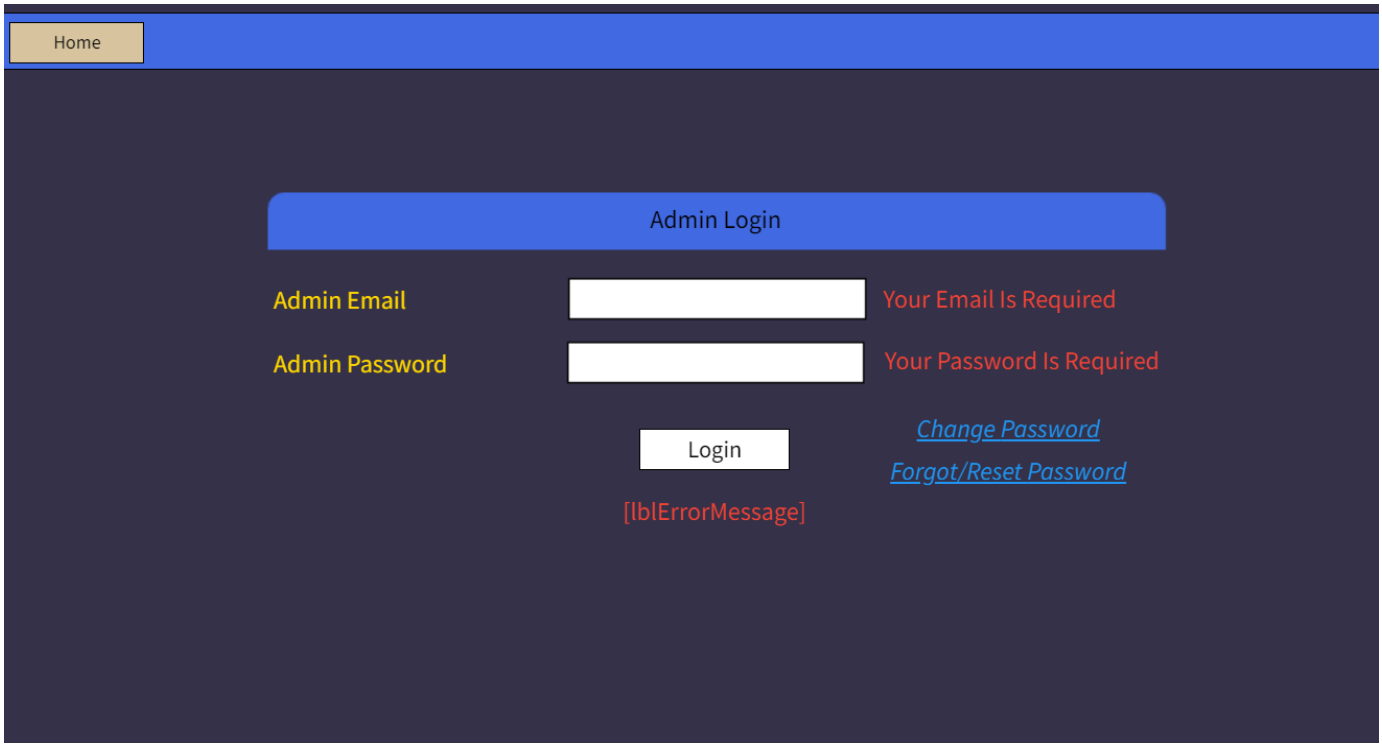
Event Coordinator Login Page – This is the screen that allows the Event Coordinators to login into the web application.



UI Component	Functionality
Home Button	When clicked, it redirects the user back to the Homepage.
Event Coordinator Email Textbox	Textbox in which the user inputs the Registered Email ID.
Event Coordinator Email Textbox Validation Error	Validation which checks whether the Textbox is empty.
Event Coordinator Password Textbox	Textbox in which the user inputs the Password.

Event Coordinator Password Textbox Validation Error	Validation which checks whether the Textbox is empty.
Login Button	When clicked, the values inputted by the user are verified/crosschecked from the Database and redirects the user to the Event Coordinator Home Page if the login is successful.
Label	Displays an error message if the login fails.
Change Password Link Button	When clicked, it redirects the user to the Event Coordinator Change Password Page where the user can change the password.
Forgot/Reset Password Link Button	When clicked, it redirects the user to the Event Coordinator Forgot Password Page where the user can enter his or her registered email ID to get a new password on their mail.

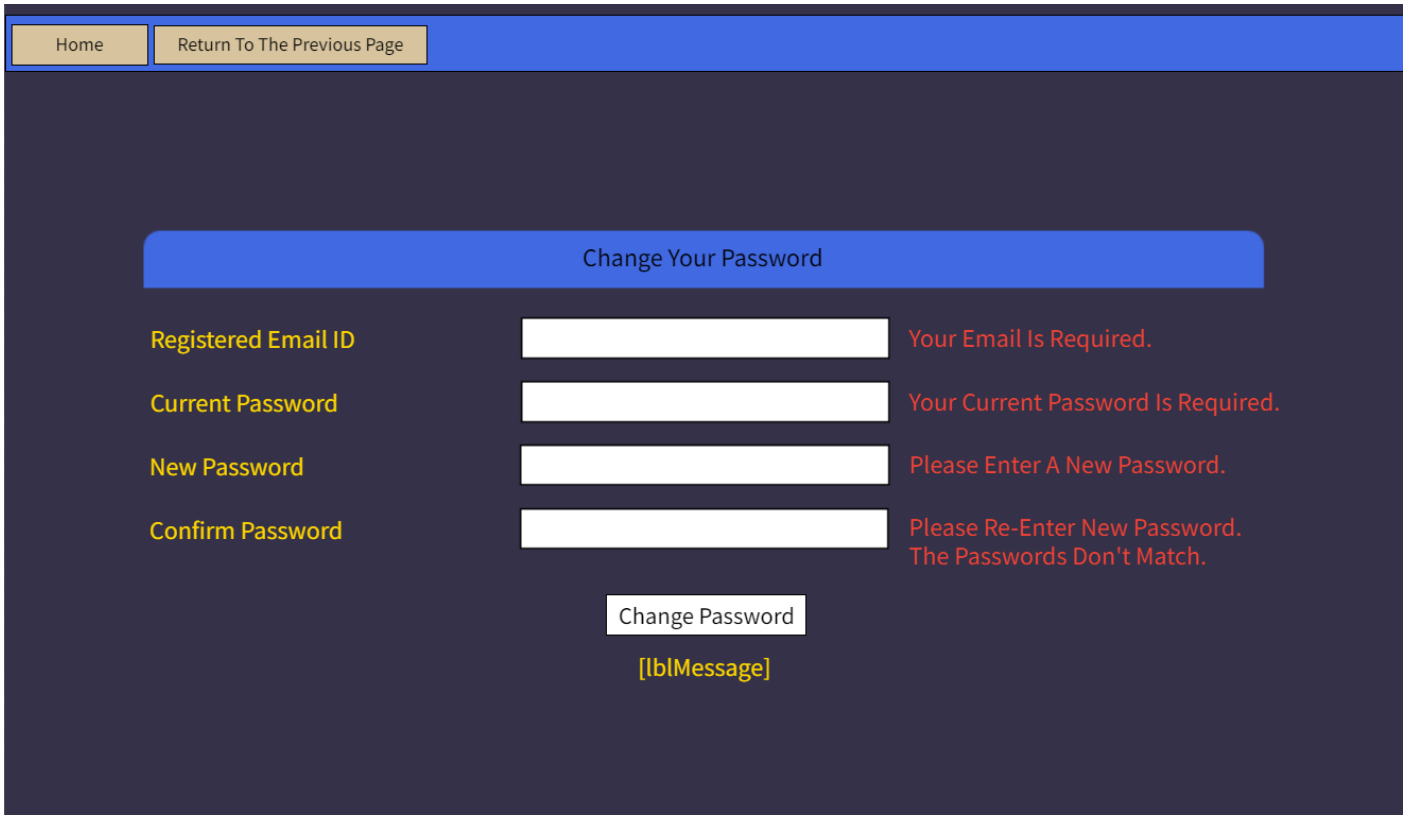
Admin Login Page – This is the screen that the user is redirected to when the user clicks on the "Login" button displayed under the "Admin" label on the Homepage. It allows the Admin(s) to login into the web application.



UI Component	Functionality
Home Button	When clicked, it redirects the user back to the Homepage.
Admin Email Textbox	Textbox in which the user inputs the Email ID.
Admin Email Textbox Validation Error	Validation which checks whether the Textbox is empty.
Admin Password Textbox	Textbox in which the user inputs the Password.
Admin Password Textbox Validation Error	Validation which checks whether the Textbox is empty.

Login Button	When clicked, the values inputted by the user are verified/crosschecked from the Database and redirects the user to the Admin Home Page if the login is successful.
Label	Displays an error message if the login fails.
Change Password Link Button	When clicked, it redirects the user to the Admin Coordinator Change Password Page where the user can change the password.
Forgot/Reset Password Link Button	When clicked, it redirects the user to the Admin Forgot Password Page where the user can enter his or her registered email ID to get a new password on their mail.

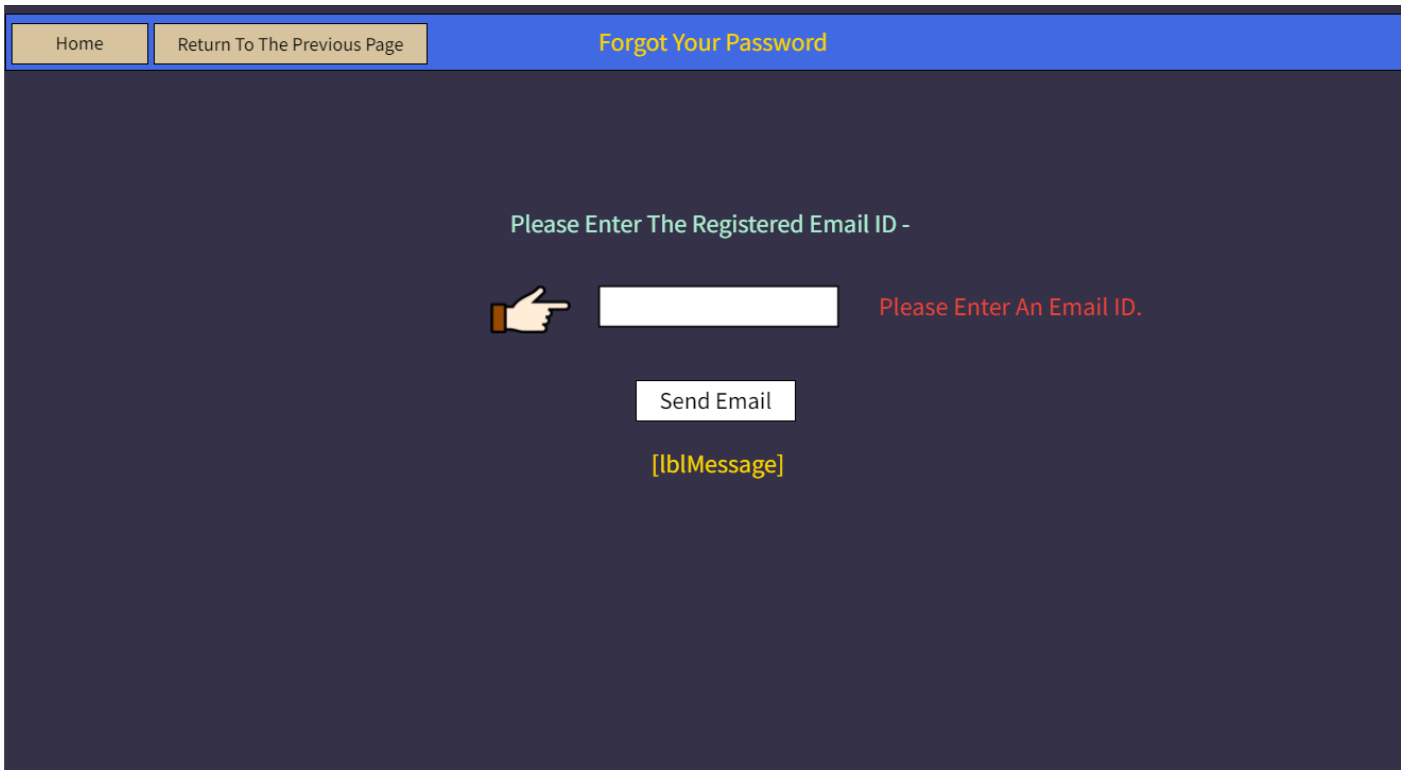
Admin Change Password Page **(All the 3 Change Password screens are similar)** - This is the screen that the user is redirected to when the user clicks on the "Change Password" button. This screen allows the user to change their current password to a new password.



UI Component	Functionality
Home Button	When clicked, it redirects the user back to the Homepage.
Return To The Previous Page Button	When clicked, it redirects the user to the previous page, their User Specific Home Page.
Registered Email ID Textbox	Textbox in which the user inputs their Registered Email ID.
Registered Email ID Textbox Validation Error	Validation which checks whether the Textbox is empty.
Current Password Textbox	Textbox in which the user inputs their current password.

Current Password Textbox Validation Error	Validation which checks whether the Textbox is empty.
New Password Textbox	Textbox in which the user inputs their new password.
New Password Textbox Validation Error	Validation which checks whether the Textbox is empty.
Confirm Password Textbox	Textbox in which the user re-enters their new password.
Confirm Password Textbox Validation Errors	Validation which checks whether the value entered in the confirm password textbox matches the value entered in the new password textbox.
Change Password Button	When clicked, the new password is saved in the database or the logged in user.
Label [lblMessage]	Displays a success/error message for password change.

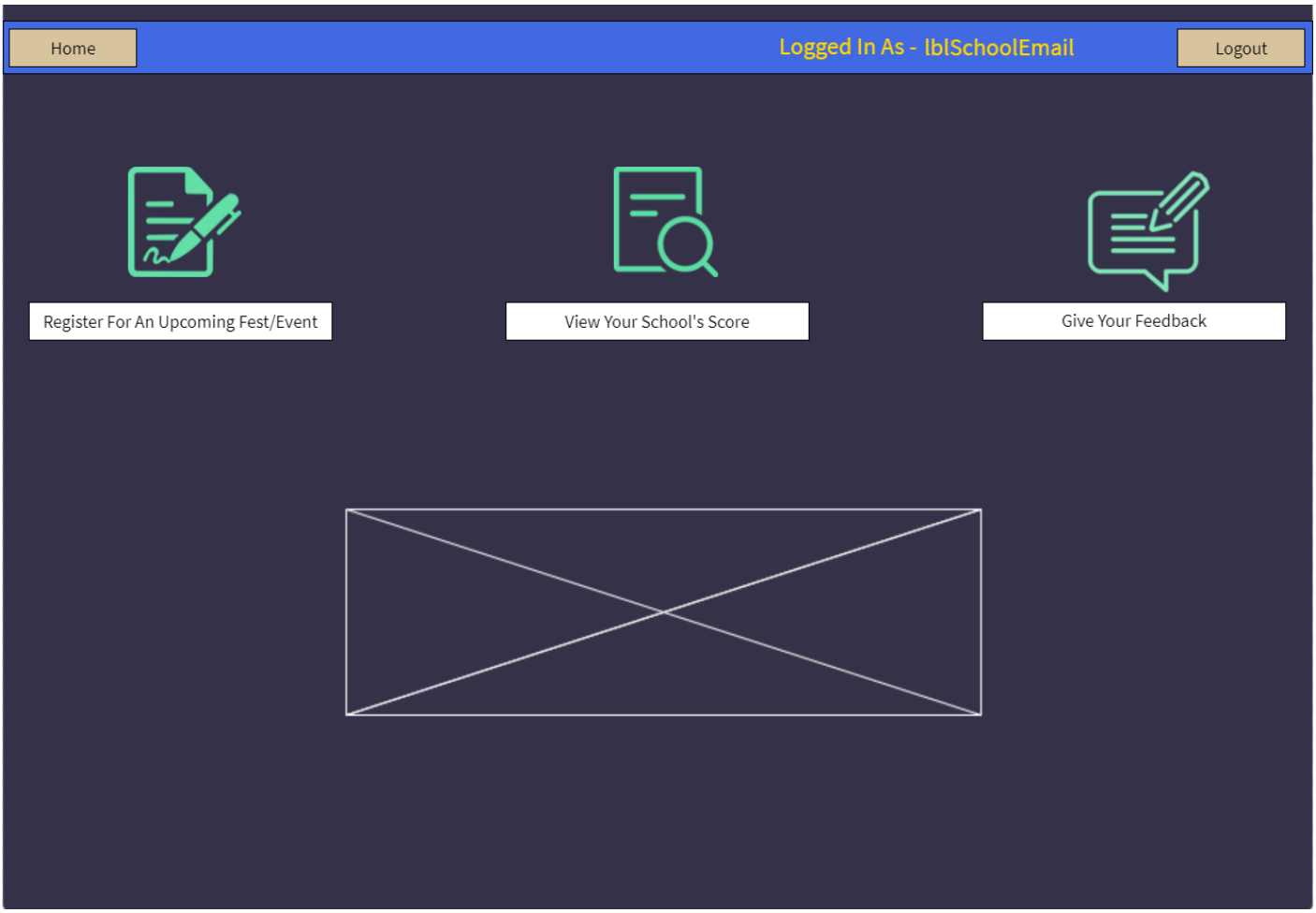
Admin Forgot Password Page **(All the 3 Forgot Password screens are similar)** - This is the screen where the user has to input their registered email ID to receive the new password.



UI Component	Functionality
Home Button	When clicked, it logs out the current user and redirects the user back to the Homepage.
Return To The Previous Page Button	When clicked, it redirects the user to the previous page, their User Specific Home Page.
Email ID Textbox	Textbox in which the user inputs their registered Email ID.
Email ID Textbox Validation	Validation which checks whether the Textbox is empty.
Send Email Button	When clicked, it sends an email containing the new password to the user's registered Email ID.

Label [lblMessage]	Displays a success/error message regarding the mail sent with the new password.

School Home Page – This is the screen that the user is redirected to on successful login. It displays 3 buttons for the school to click on, each serving a different purpose.



UI Component	Functionality
Home Button	When clicked, it logs out the current user and redirects the user back to the Homepage.
Logout Button	When clicked, it logs the user out and redirects the user back to the School Login Page.
School Email Label (lblSchoolEmail)	Displays the email ID from which the user logged in.
Register For An Upcoming Fest/Event Button	When clicked, it redirects the user to the School Event Details Page where the user can select a Fest or an Event to register for.
View Your School's Score Button	When clicked, it redirects the user to the School View Scores Page where the user can view all their past fest/event scores.
Give Your Feedback Button	When clicked, it redirects the user to the School Feedback Page where the user can give feedback about the fest/event.
Image Placeholder	When the page loads, a user-friendly image of a school will be displayed.

School Event Details Page - This screen allows the school to register for an upcoming fest/event. This screen displays all the Fests and their corresponding Events that the School can currently register for.

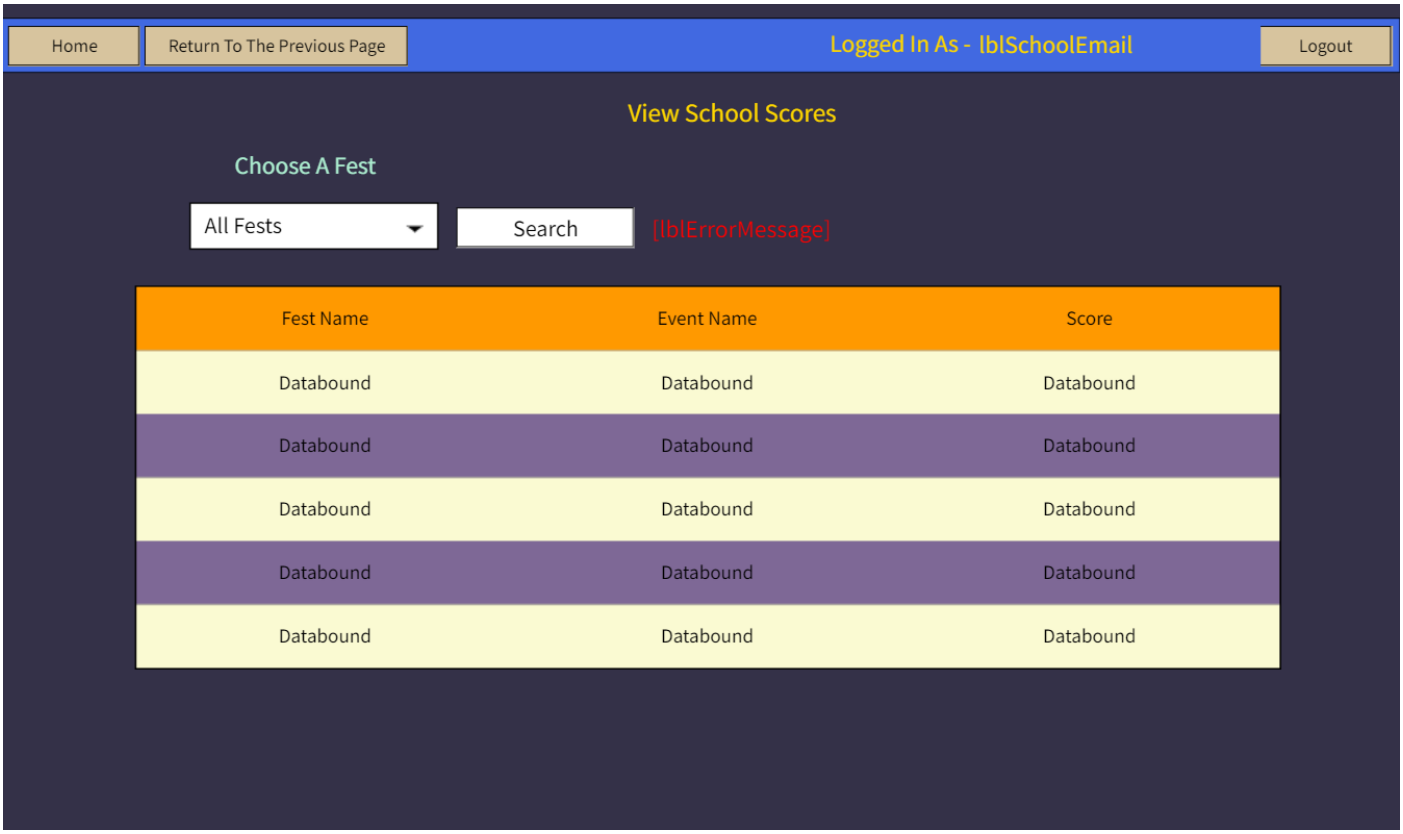
UI Component	Functionality
Home Button	When clicked, it logs out the current user and redirects the user back to the Homepage.
Logout Button	When clicked, it logs the user out and redirects the user back to the School Login Page.
School Email Label (lblSchoolEmail)	Displays the email ID which the user logged in from.
Return To The Previous Page Button	When clicked, it redirects the user to the previous page, School Home Page.
Select Fest Name Drop Down	When clicked, it would display all the Fests.
Search Button	When clicked, the Grid View would display the selected Fest and its corresponding events.
Error Message Label [lblErrorMessage]	If no data is found for the selected dropdown value, the label outputs an error message.
Grid View	On the page load event, the Grid View will get populated with all fest and event details from the database.
Select Button Inside the Grid View	This button will allow user to select a particular event from the table and register for it on the next page.
Register For This Event! Button	When clicked, the user gets redirected to the School Event Registration Page.

School Event Registration Page - This is the screen that displays a table where the School can register their participants for a particular fest and event.

UI Component	Functionality
Home Button	When clicked, it logs out the current user and redirects the user back to the Homepage.
Logout Button	When clicked, it logs the user out and redirects the user back to the School Login Page.
School Email Label (lblSchoolEmail)	Displays the email ID of the logged in user.
Return To The Previous Page Button	When clicked, it redirects the user to the previous page, School Fest/Event Registration Home Page.
Grid View	This will allow schools to register their participants for the selected fest and event name.
Participant Name Textbox Inside the Grid View	Textbox where the user will input the name of the Participant.
Required Field Validator for Participant Name Textbox	Validation that ensures a value is inputted in the textbox.
Participant Gender Drop Down List Inside the Grid View	Drop Down where the user will select the gender of the Participant.
Participant Grade Drop Down List Inside the Grid View	Drop Down where the user will select the grade of the Participant.
Delete Button Inside the Grid View	When clicked, the corresponding row will get deleted.
Add A New Participant Button	When clicked, a new row gets generated in the table.
Add A New Participant Button Validation	Validation which ensures that the number of rows in the Grid View do not exceed the Number of Participants allowed for that event.
Save Button	When clicked, it saves all values entered in the table to the database.
Error Message Label [lblErrorMessage]	Displays an error message if the user tries to add more participants than allowed.

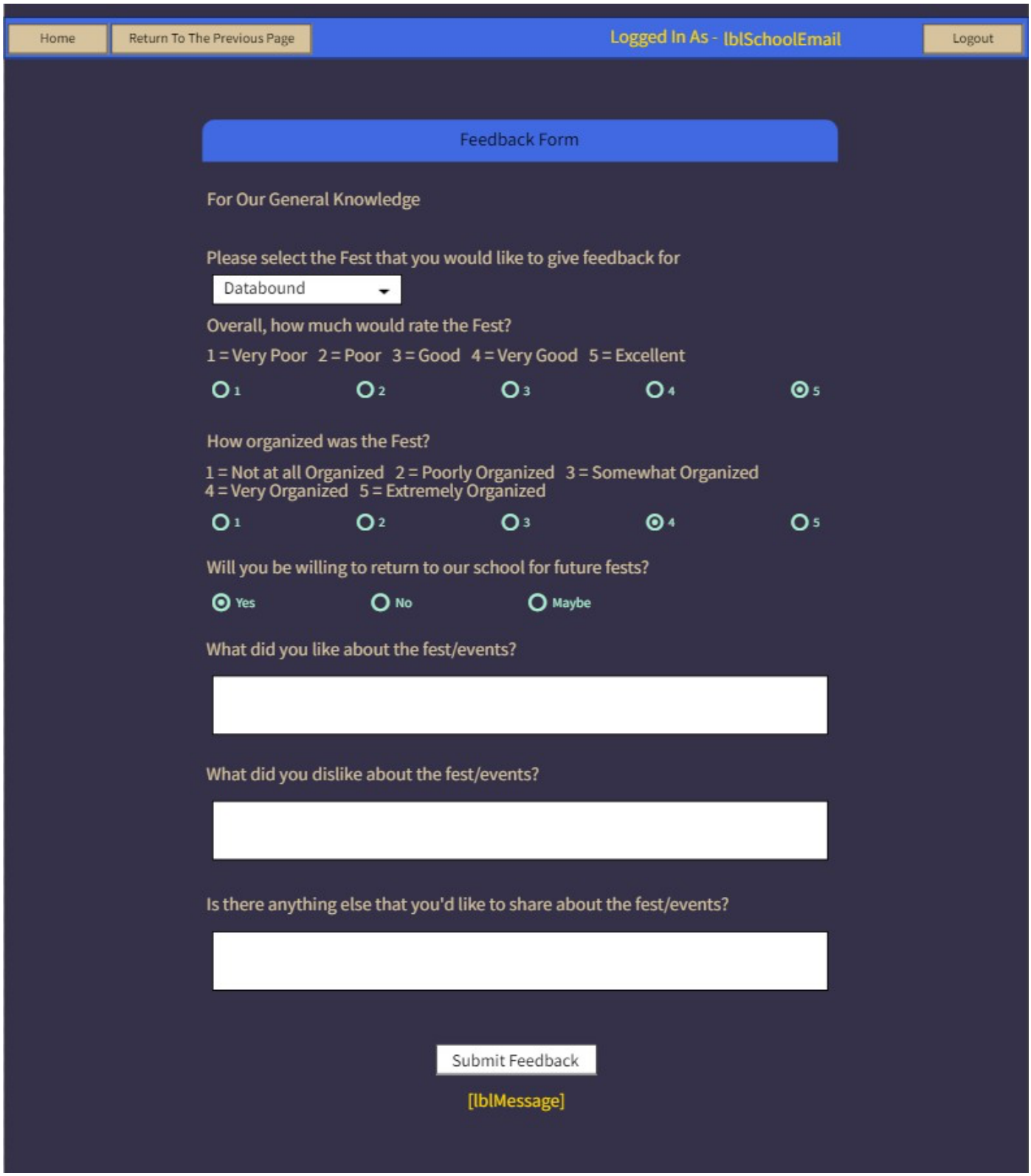
Label [lblMessage]	Displays a message that indicates that the School registered their participants successfully.

School View Scores Page - This is the screen where the user can view scores for all the current/past Fests, Events and the Scores that the school had participated in.



UI Component	Functionality
Home Button	When clicked, it logs out the current user and redirects the user back to the Homepage.
Logout Button	When clicked, it logs the user out and redirects the user back to the School Login Page.
School Email Label (lblSchoolEmail)	Displays the email ID from which the user logged in.
Return To The Previous Page Button	When clicked, it redirects the user to the previous page, School Home Page.
Choose a Fest Drop Down	When clicked, it would display all the Fests.
Search Button	When clicked, the Grid View would display the selected Fest and its corresponding event name and score.
Error Message Label [lblErrorMessage]	If no data is found for the selected dropdown value, the label outputs an error message.
Grid View	On the page load event, the Grid View will get populated with values fetched from datatable.

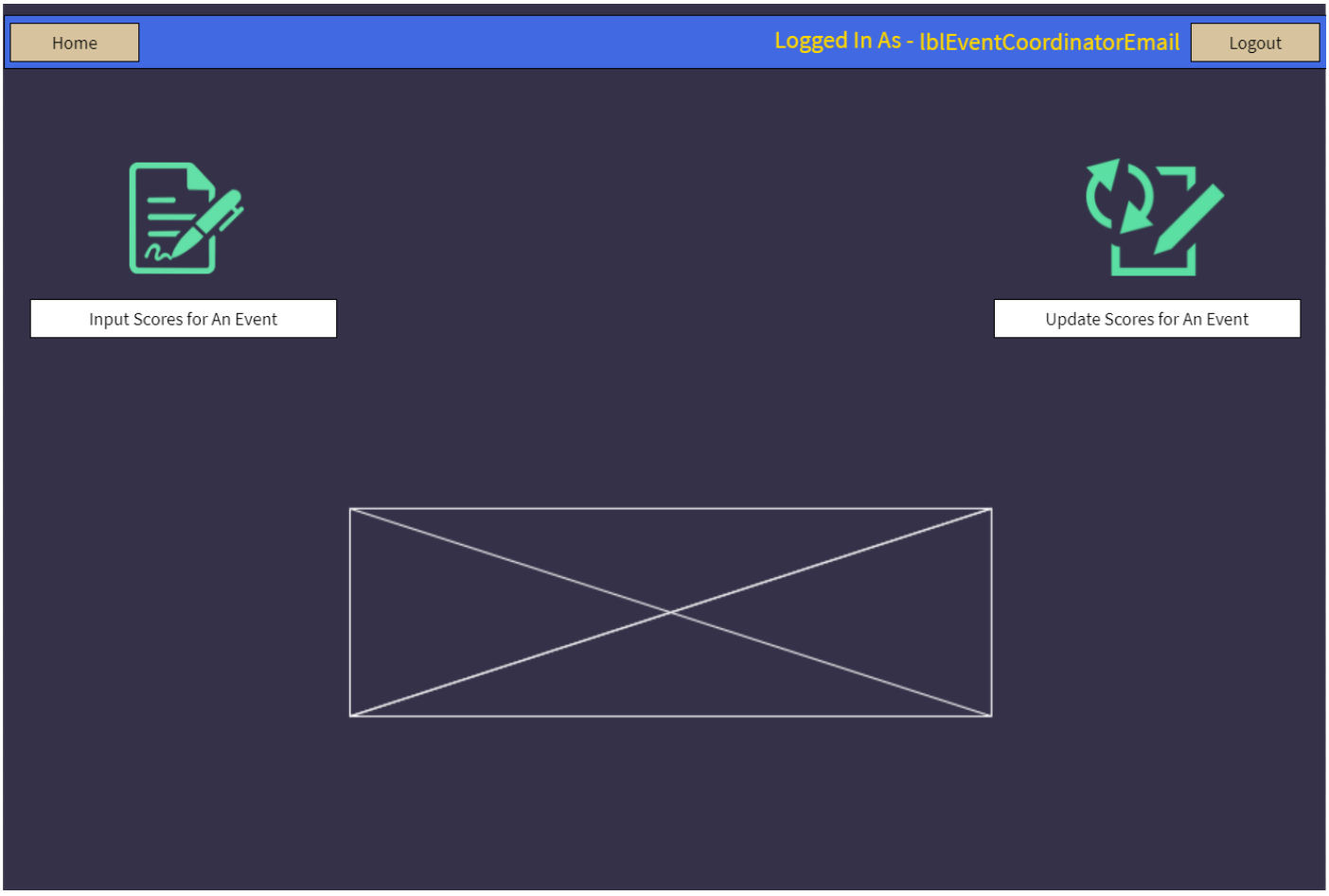
School Feedback Page - This is the screen that displays a feedback form where the School can leave their feedback.



UI Component	Functionality
Home Button	When clicked, it logs out the current user and redirects the user back to the Homepage.
Logout Button	When clicked, it logs the user out and redirects the user back to the School Login Page.
School Email Label (lblSchoolEmail)	Displays the email ID from which the user logged in.

Select a Fest DropDown List	When clicked, it would display all the Fests that the School have participated in currently.
Rate The Fest Radio Button Group	Radio Button Group that displays numbers from 1 to 5.
How Organized Was The Fest Radio Button Group	Radio Button Group that displays numbers from 1 to 5.
Will You Be Willing To Return Radio Button Group	Radio Button Group that displays the values Yes, No and Maybe.
What Did You Like Textbox	Textbox in which the user inputs their feedback for the question listed above this textbox.
What Did You Dislike Textbox	Textbox in which the user inputs their feedback for the question listed above this textbox.
Anything You Would Like To Share Textbox	Textbox in which the user inputs any other thoughts that they would like to share.
Submit Button	When clicked, it saves the values to a DataTable in the SQL database.
Label	Displays a success/error message indicating whether or not the form was submitted or not.

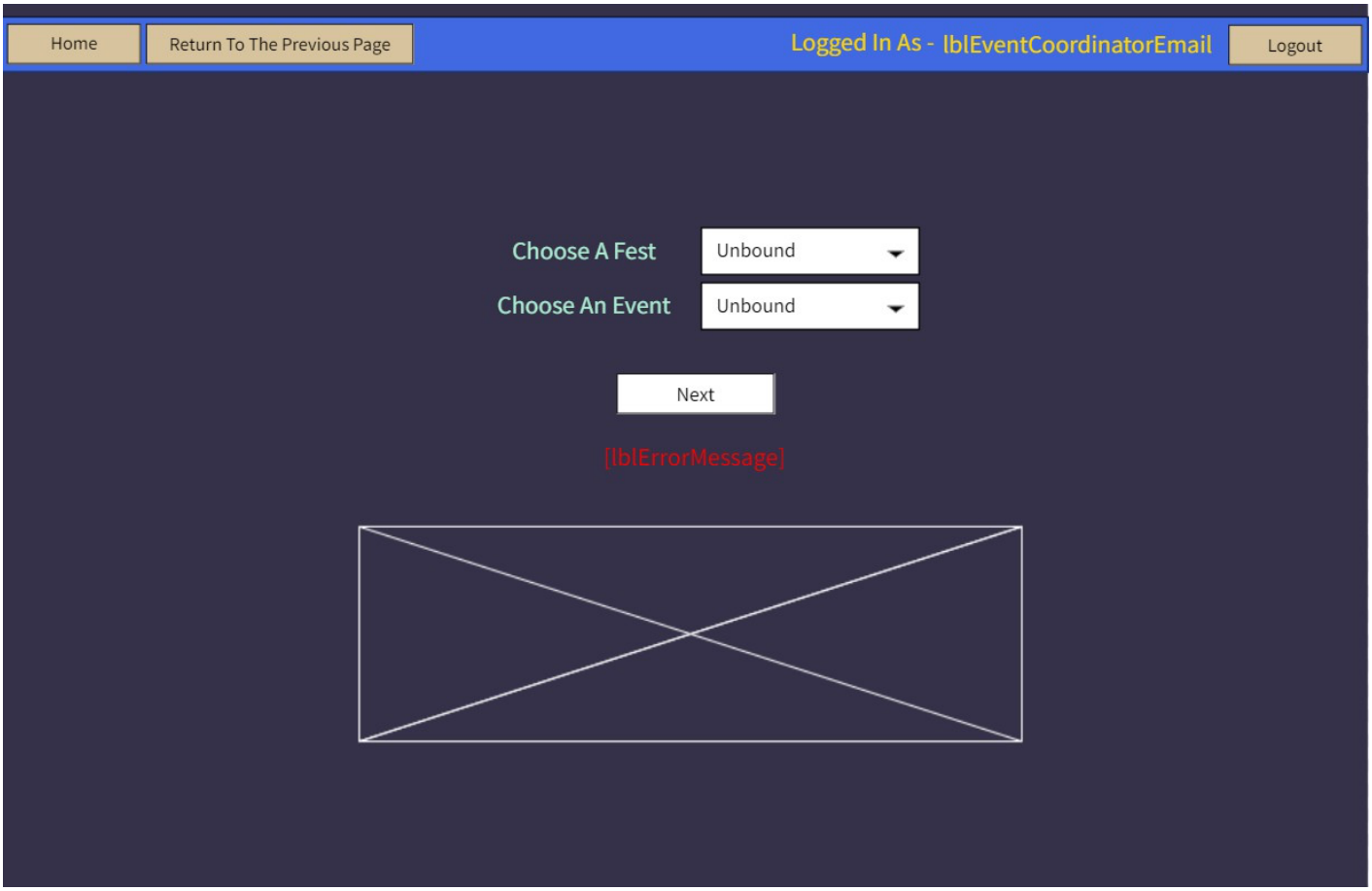
Event Coordinator Home Page – This is the screen that the user is redirected to from the Event Coordinator Login Page if the login was successful. It displays 2 buttons for the school to click on, each serving a different purpose.



UI Component	Functionality
Home Button	When clicked, it logs out the current user and redirects the user back to the Homepage.

Logout Button	When clicked, it logs the user out and redirects the user back to the Event Coordinator Login Page.
Event Coordinator Email Label (lblEventCoordinatorEmail)	Displays the email ID from which the user logged in.
Input Scores for An Event Button	When clicked, it redirects the user to the Event Coordinator Choose An Event where the user can select the Event to input the scores for.
Update Scores for An Event Button	When clicked, it redirects the user to the Event Coordinator Choose Event To Update Page where the user can select the Event to update the scores for.
Image Placeholder	When the page loads, a user-friendly image will be displayed.

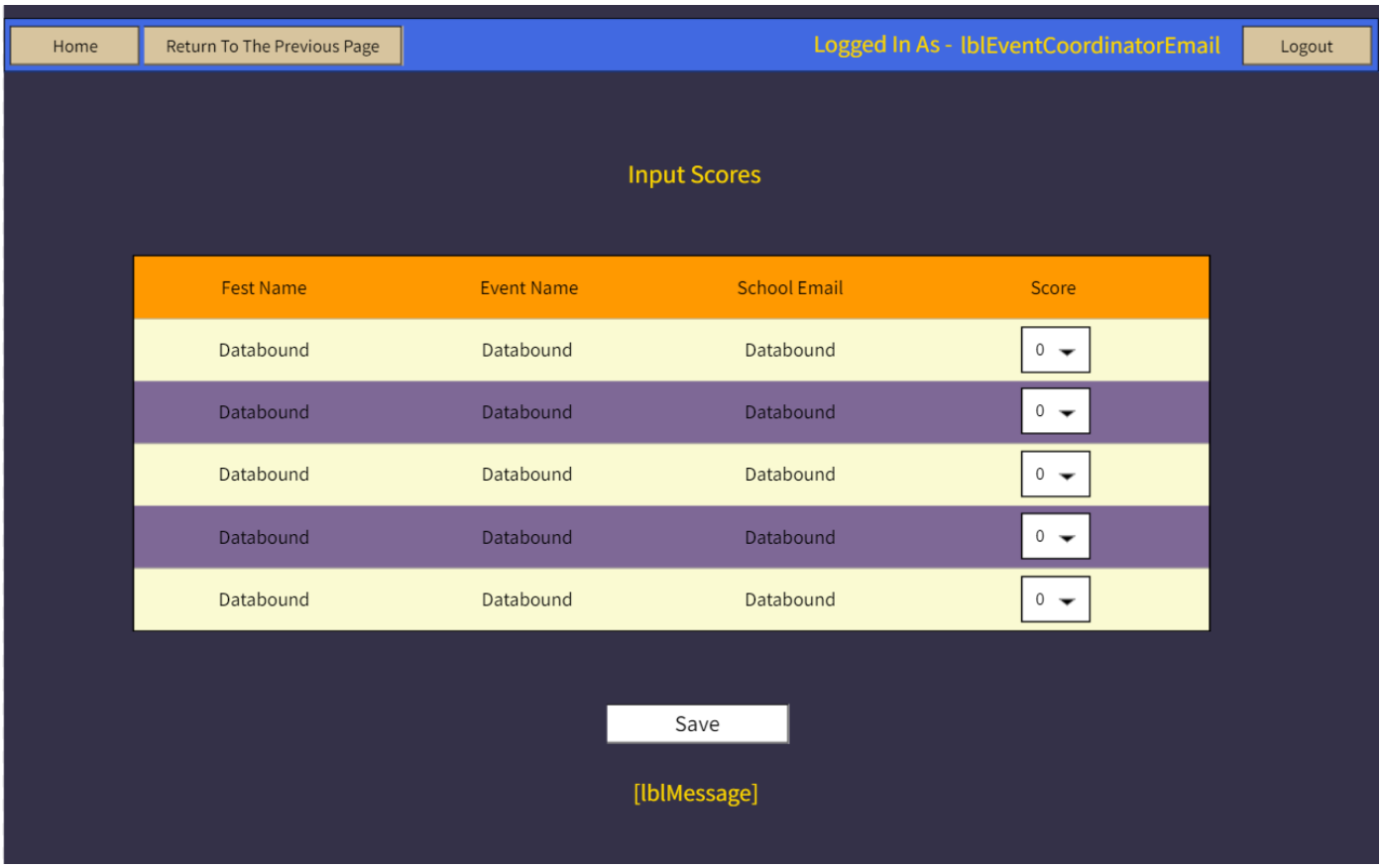
Event Coordinator Choose An Event Page – This screen displays 2 Drop Downs where the Event Coordinator can choose a Fest and the event to input the scores for from the list of events that they were assigned to by the Admin.



UI Component	Functionality
Home Button	When clicked, it logs out the current user and redirects the user back to the Homepage.
Logout Button	When clicked, it logs the user out and redirects the user back to the Event Coordinator Login Page.
Event Coordinator Email Label (lblEventCoordinatorEmail)	Displays the email ID from which the user logged in.

Return To The Previous Page Button	When clicked, it redirects the user to the previous page, Event Coordinator Home Page.
Choose A Fest Drop Down	When clicked, it displays a list of Fests.
Choose An Event Drop Down	When clicked, it displays a list of events under the selected Fest.
Next Button	When clicked, it redirects the user to the Event Coordinator Input Scores Page.
Error Message Label [lblErrorMessage]	If scores have already been given for the event, error message is displayed.
Image Placeholder	When the page loads, a user-friendly image will be displayed.

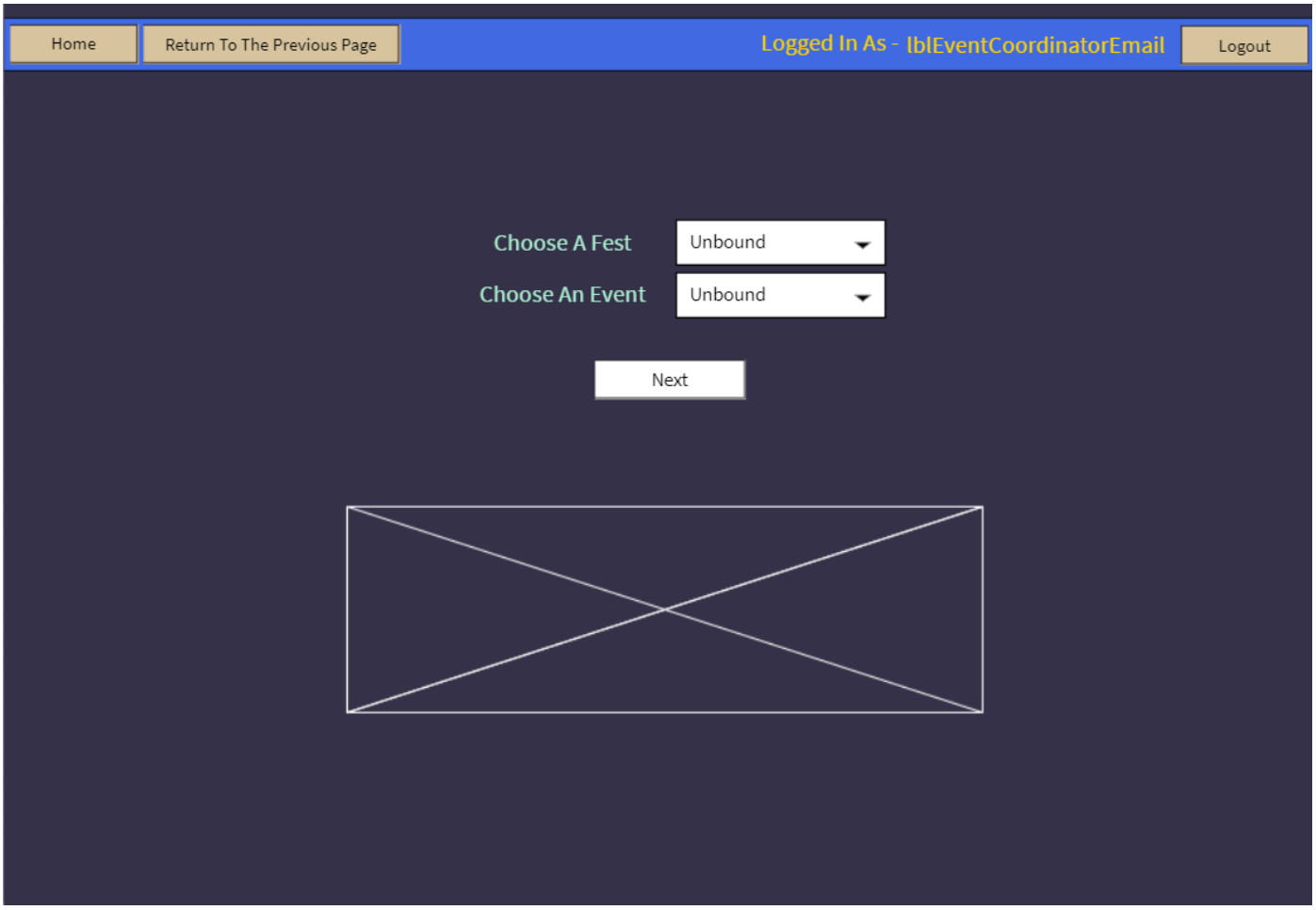
Event Coordinator Input Scores Page – This screen displays a GridView where the Event Coordinator can input the scores for of the event that was selected from the Drop Down in the previous page.



UI Component	Functionality
Home Button	When clicked, it logs out the current user and redirects the user back to the Homepage.
Logout Button	When clicked, it logs the user out and redirects the user back to the Event Coordinator Login Page.
Event Coordinator Email Label (lblEventCoordinatorEmail)	Displays the email ID from which the user logged in.
Return To The Previous Page Button	When clicked, it redirects the user to the previous page, Event Coordinator Choose An Event Page.
Grid View	Used so that the user can input scores in the dropdown.
Drop Down List Inside the Grid View	Drop Down where the user will select the score.

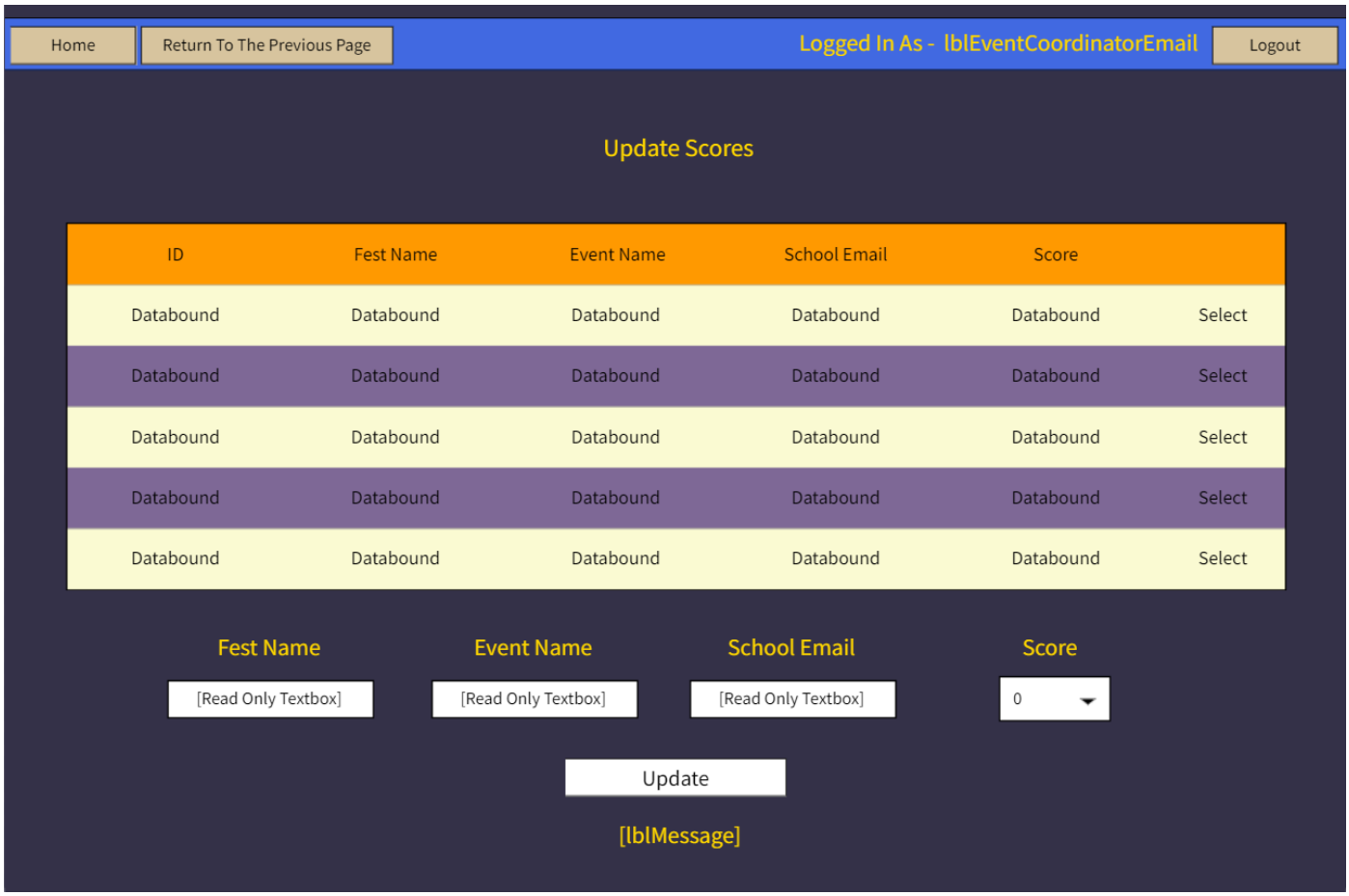
Save Button	When clicked, it saves the all the cell values in the Grid View to the SQL database.
Label [lblMessage]	Displays a success/error message depending on the status of insertion of record.

Event Coordinator Choose Event To Update Page – Similar to Event Coordinator Choose An Event Page.



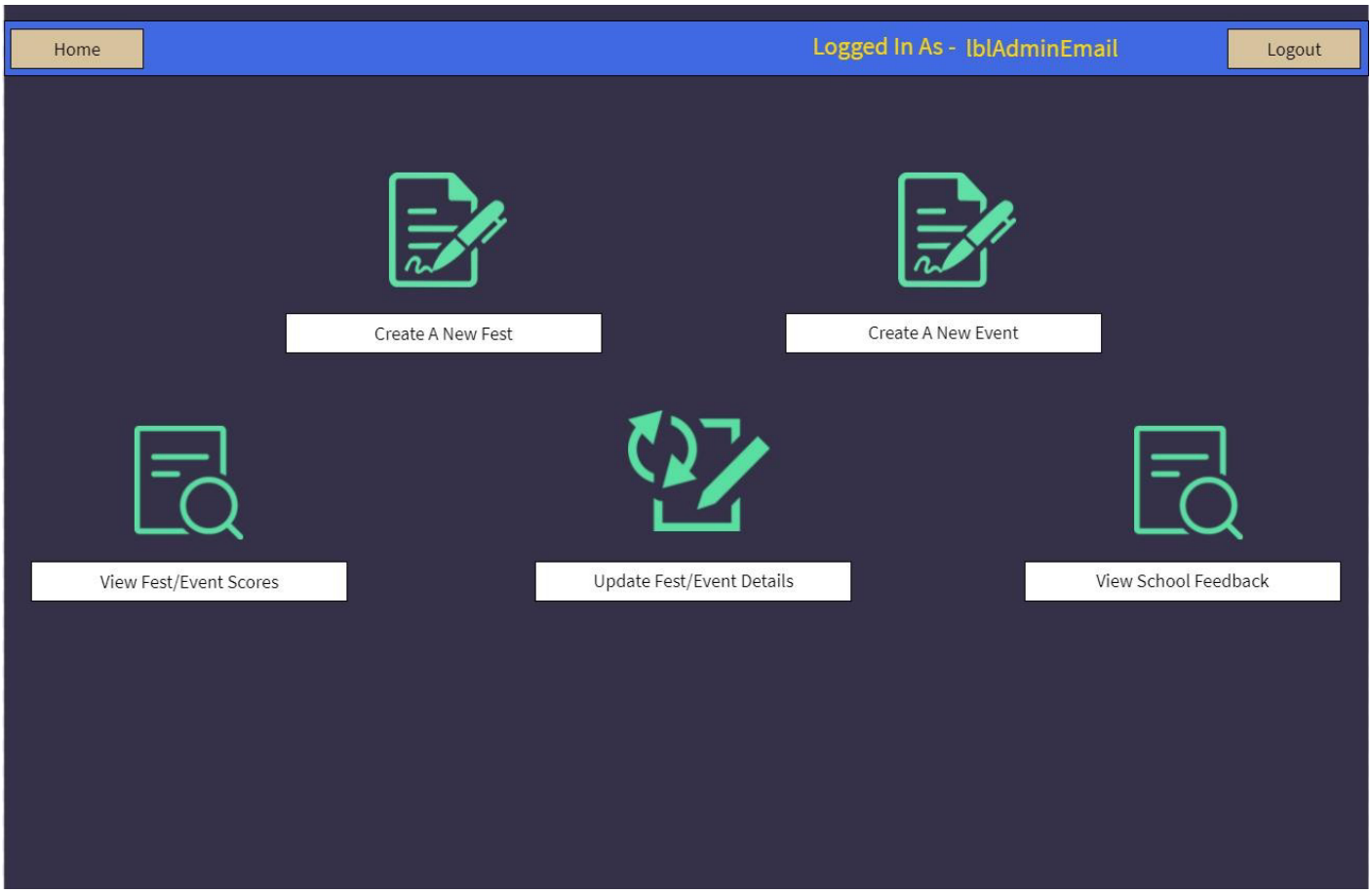
UI Component	Functionality
Home Button	When clicked, it logs out the current user and redirects the user back to the Homepage.
Logout Button	When clicked, it logs the user out and redirects the user back to the Event Coordinator Login Page.
Event Coordinator Email Label (lblEventCoordinatorEmail)	Displays the email ID from which the user logged in.
Return To The Previous Page Button	When clicked, it redirects the user to the previous page, Event Coordinator Home Page.
Choose A Fest Drop Down	When clicked, it displays a list of Fests.
Choose An Event Drop Down	When clicked, it displays a list of events under the selected Fest.
Next Button	When clicked, it redirects the user to the Event Coordinator Update Scores Page.
Image Placeholder	When the page loads, a user-friendly image will be displayed.

Event Coordinator Update Scores Page - This is the screen that displays a GridView where the user can update the scores for any row they choose.



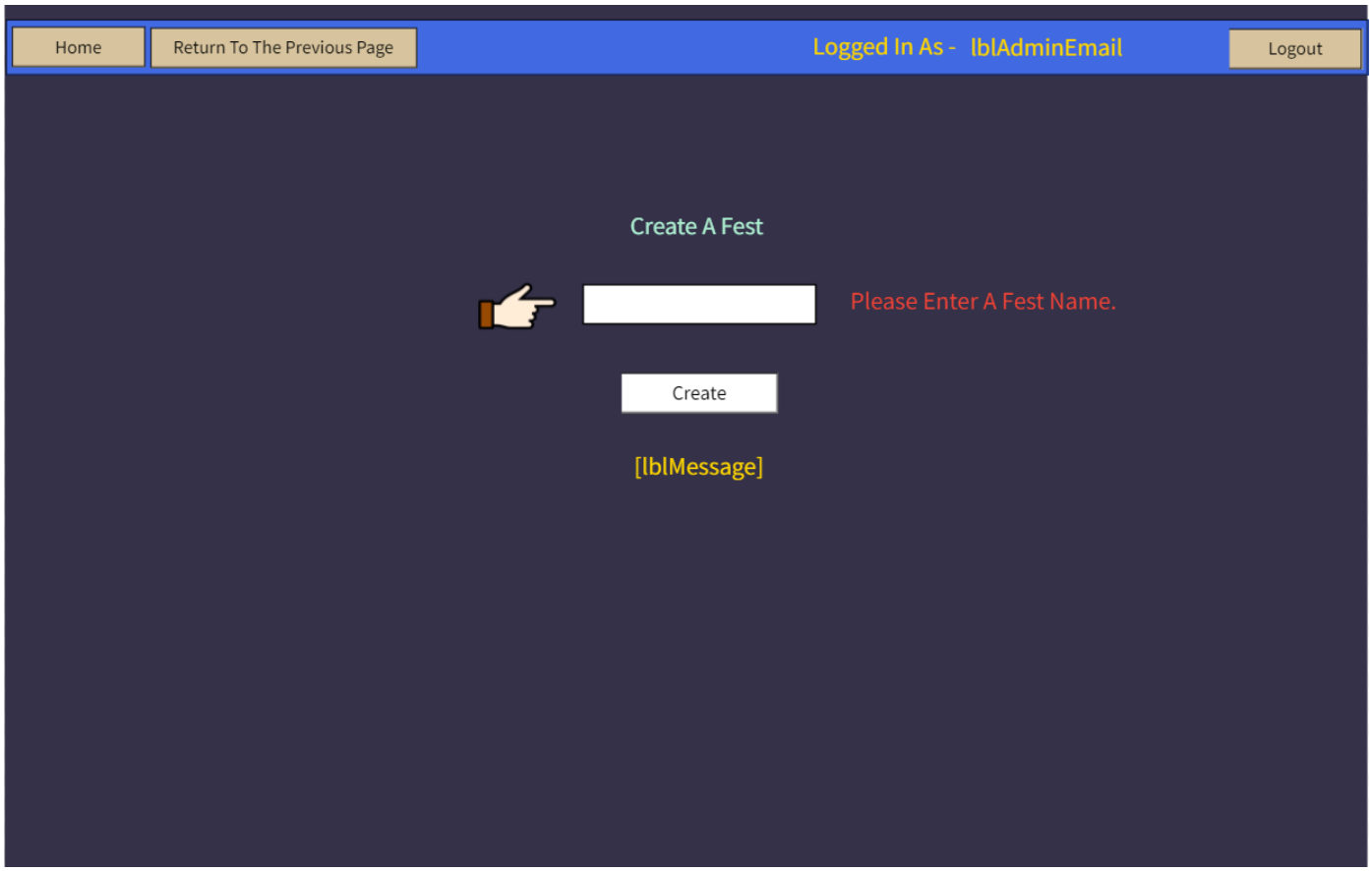
UI Component	Functionality
Home Button	When clicked, it logs out the current user and redirects the user back to the Homepage.
Logout Button	When clicked, it logs the user out and redirects the user back to the Event Coordinator Login Page.
Event Coordinator Email Label (lblEventCoordinatorEmail)	Displays the email ID from which the user logged in.
Return To The Previous Page Button	When clicked, it redirects the user to the previous page, Event Coordinator Choose Event To Update Page.
Grid View	Used to populate scores for all fests, events and schools.
Select Button Inside the Grid View	This button will allow user to select a school to update the scores for.
Update Button	When clicked, it saves all values entered in the table and updates the values in the database.
Label [lblMessage]	Displays a success/error message that indicates whether the scores have been updated.

Admin Home Page – This is the screen that the user is redirected to from the Admin Login Page if the login was successful. It displays 5 buttons for the Admin to choose from, each serving a different purpose.



UI Component	Functionality
Home Button	When clicked, it logs out the current user and redirects the user back to the Homepage.
Logout Button	When clicked, it logs the user out and redirects the user back to the Admin Login Page.
Admin Email Label (lblAdminEmail)	Displays the email ID from which the user logged in.
Create A New Fest Button	When clicked, it redirects the user to the Admin Fest Making Page where Admin can create a new Fest.
Create A New Event Button	When clicked, it redirects the user to the Admin Event Making Page where Admin can create a new Event.
View Fest/Event Scores Button	When clicked, it redirects the user to the Admin View Scores Page where Admin can view all the participating schools current scores.
Update Fest/Event Details	When clicked, it redirects the user to the Admin Update Details Page where the user can update the Fest(s) and/or the Event(s) details.
View School Feedback Button	When clicked, it redirects the user to the Admin View School Feedback Page where the user can view the School's feedback about the fest.

Admin Fest Making Page – This screen displays a singular textbox where the user can input a new Fest name.



UI Component	Functionality
Home Button	When clicked, it logs out the current user and redirects the user back to the Homepage.
Logout Button	When clicked, it logs the user out and redirects the user back to the Admin Login Page.
Admin Email Label (lblAdminEmail)	Displays the email ID from which the user logged in.
Return To The Previous Page Button	When clicked, it redirects the user to the previous page, Admin Home Page.
Create A Fest Textbox	Textbox in which the user inputs a new Fest Name.
Create A Fest Textbox Validation	Validation which checks whether the Textbox is empty.
Create Button	The Fest Name is inserted and saved in the database.
Label [lblMessage]	Displays a success/error message that indicates whether the Fest was created successfully.

Admin Event Making Page – This is the screen where the Admin can create a new Event.

UI Component	Functionality
Home Button	When clicked, it logs out the current user and redirects the user back to the Homepage.
Logout Button	When clicked, it logs the user out and redirects the user back to the School Login Page.
Admin Email Label (lblAdminEmail)	Displays the email ID from which the user logged in.
Return To The Previous Page Button	When clicked, it redirects the user to the previous page, Admin Home Page.
Fest Name Drop Down List	Drop Down in which the user selects a Fest Name.
Event Name Textbox	Textbox in which the user inputs an Event Name.
Event Description Textbox	Textbox in which the user enters the Event Description.

Event Coordinator Drop Down List	Drop Down List in which the user selects and assigns an Event Coordinator to the Event written in the Event Name Textbox.
Number Of Participants Drop Down List	Drop Down List in which the user selects the Maximum Number Of Participants eligible for that Event.
Eligible Grades Checkbox List	Checkbox List in which the user selects the eligible grade(s) for that Event.
Date Of The Event Calendar Control	Calendar Control where the user selects the Date Of The Event.
Date Of The Event Validation	Validation which ensures that the Event is registered 1 to 3 months in advance.
Time Of The Event Drop Down List	Drop Down List where the user selects the Time Of The Event.
Add Event Button	When clicked, it saves all values to a DataTable in the SQL database.
JavaScript Validation	When the Add Event Button is clicked, a validation takes place which ensures that no fields are left empty, the drop downs do not have the default values as the selected value and the date of the event is set within the stipulated range validator.
Label [lblMessage]	Displays a success/error message which indicates whether the Event registration was successful.

Admin View Scores Page – This is the screen that displays a Grid View which gets populated according to the Fest selected in the Drop Down. This also gives the user the ability to calculate the top 3 winners of each Fest.

UI Component	Functionality
Home Button	When clicked, it logs out the current user and redirects the user back to the Homepage.
Logout Button	When clicked, it logs the user out and redirects the user back to the Admin Login Page.
Admin Email Label (lblAdminEmail)	Displays the email ID from which the user logged in.
Return To The Previous Page Button	When clicked, it redirects the user to the previous page, Admin Home Page.
Choose a Fest Drop Down	When clicked, it would display all the Fests.
Search Button	When clicked, the Grid View would get populated according to the Fest selected in the Drop Down.
Error Message Label [lblErrorMessage]	If no data is found for the selected Fest, the label outputs an error message.
Grid View	Gets populated according to the selected Fest.
View Event Specific Scores Button	When clicked, it redirects the user to the Admin View Event Scores Page.
Calculate Top 3 Winners! Button	When clicked, the top 3 schools are calculated for the

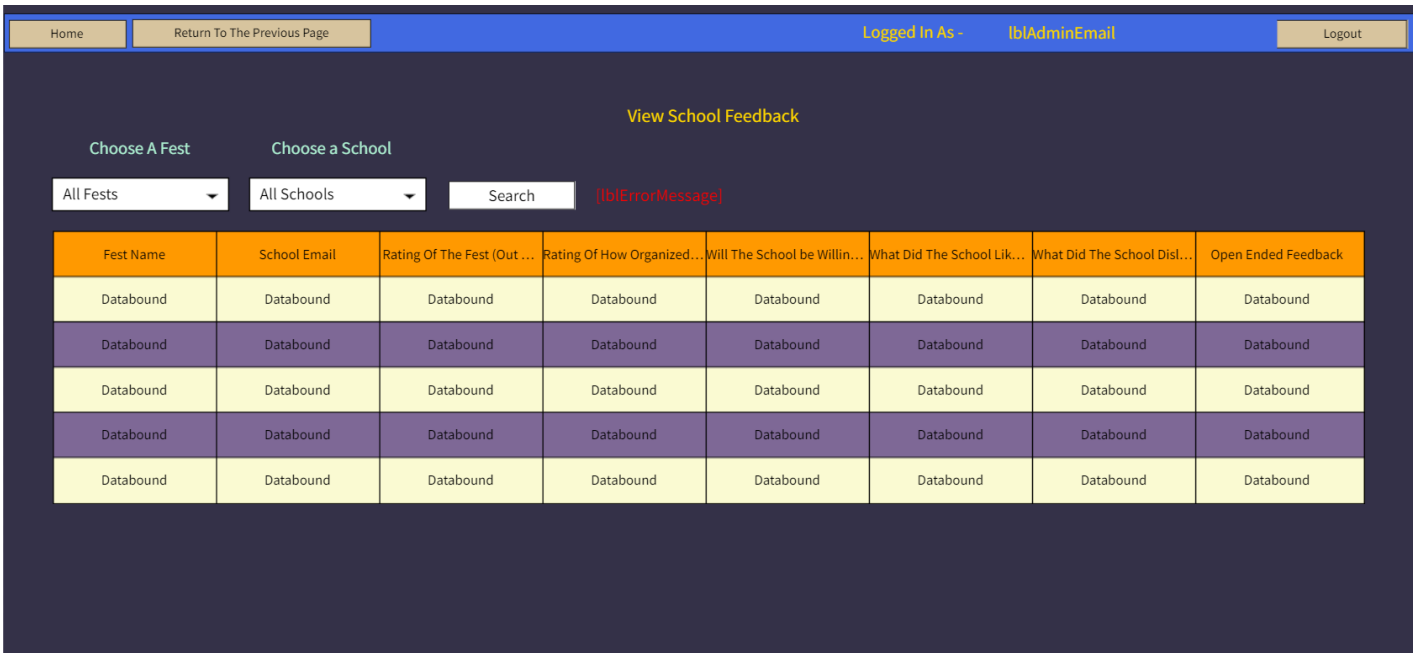
	selected Fest according to the number of events participated in and the total score column.
Label [lblSchoolName1]	Label in which the School that placed 1st is displayed.
Label [lblSchoolName2]	Label in which the School that placed 2nd is displayed.
Label [lblSchoolName3]	Label in which the School that placed 3rd is displayed.

Admin View Event Scores Page – This is the screen that the user is redirected to from the Admin View Scores Page when the user clicks on the "View Event Specific Scores" button. This screen displays a Grid View which gets populated with scores for all schools for all fests and events.

UI Component	Functionality
Home Button	When clicked, it logs out the current user and redirects the user back to the Homepage.
Logout Button	When clicked, it logs the user out and redirects the user back to the Admin Login Page.
Admin Email Label (lblAdminEmail)	Displays the email ID from which the user logged in.
Return To The Previous Page Button	When clicked, it redirects the user to the previous page, Admin View Scores Page.
Choose a Fest Drop Down	Drop Down which gets populated with the Fest Names.
Choose an Event Drop Down	Drop Down which gets populated with the Event Names.
Choose a School Drop Down	Drop Down which gets populated with the School Emails.
Search Button	When clicked, the Grid View would get populated according to the Fest selected in the Drop Down.

Error Message Label [lblErrorMessage]	If no data is found for the selected dropdown value(s), the label outputs an error message.
Grid View	Gets populated according to the search criteria selected in the DropDownLists.

Admin View School Feedback Page – This screen displays a Grid View which displays the feedback received from the schools according to the Fest and the School selected in the Drop Downs.



UI Component	Functionality
Home Button	When clicked, it logs out the current user and redirects the user back to the Homepage.
Logout Button	When clicked, it logs the user out and redirects the user back to the Admin Login Page.
Admin Email Label (lblAdminEmail)	Displays the email ID from which the user logged in.
Return To The Previous Page Button	When clicked, it redirects the user to the previous page, Admin Home Page.
Choose a Fest Drop Down	Drop Down which gets populated with the Fest Names.
Choose a School Drop Down	Drop Down which gets populated with the School Emails.
Search Button	When clicked, the Grid View would get populated according to the Fest and/or School selected in the Drop Down.
Error Message Label [lblErrorMessage]	If no data is found for the selected dropdown(s) value, the label outputs an error message.
Grid View	Gets populated according to the search criteria selected in the DropDownLists.

Admin Update Details Page – This screen that displays a Grid View which gets populated with event details where the user can either update or delete the event/event details.

UI Component	Functionality
Home Button	When clicked, it logs out the current user and redirects the user back to the Homepage.
Logout Button	When clicked, it logs the user out and redirects the user back to the Admin Login Page.
Admin Email Label (lblAdminEmail)	Displays the email ID from which the user logged in.
Return To The Previous Page Button	When clicked, it redirects the user to the previous page, Admin Home Page.
Grid View	Used to display all event details and give the option to select or delete a row.
Delete Button Inside Grid View	When clicked, the corresponding row will get deleted from both the Grid View and the database as well.
Select Button Inside the Grid View	When clicked, the update details table gets populated with the values of the selected row.
Update Button	When clicked, it updates the values in the corresponding DataTable in the SQL database.
Label [lblMessage]	Displays a success/error message that indicates whether the details have been updated.

Development

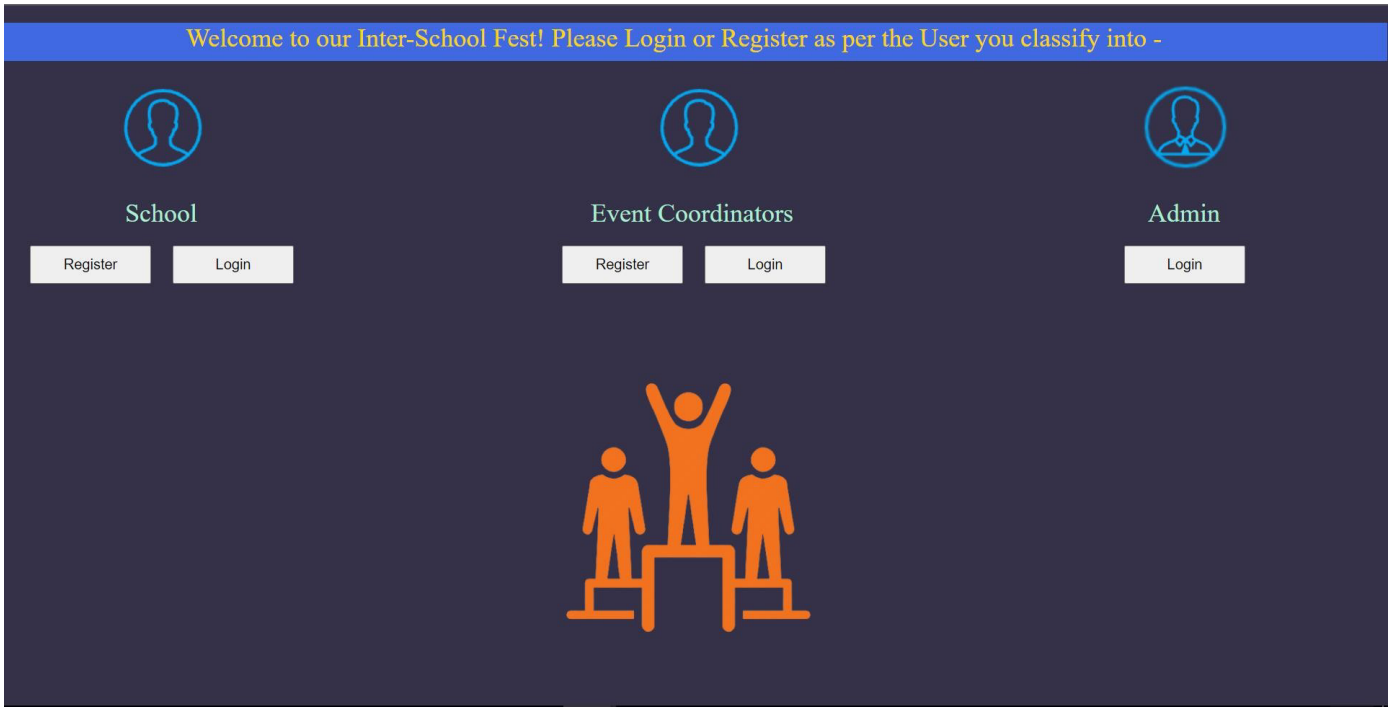
Serial Number	Technique Used	Purpose	Success Criteria
1	GUI using ASP.NET and HTML/CSS controls	To create a user-friendly, intuitive, and professional UI.	14
2	Layered Architecture	Each layer can be developed simultaneously and can be modified independent of other layers.	-
3	Database Design	To show primary-foreign key relationships.	10
4	Establishing Database connection	The database was required throughout the web-application.	10
5	Basic Database Operations (Add, Delete, Update)	Add operation for adding participants, scores, events and more. Delete operation for deleting events and participants. Update to update fest/event details and scores.	1, 5, 6, 7, 8, 10
6	Advanced Database Operations through SQL select queries	Populating 'Total Score' column in GridView by adding all event scores of the selected fest for each school.	11, 12
7	ViewState	For bulk inserts, for example, when school is registering participants for an event.	6
8	Client-Side Validations through JavaScript and ASP.NET validation controls	Provides a faster response time by validating the inputs on the browser itself without a round-trip to server.	13, 16
9	Server-Side Validations using C# programming language	To prevent invalid insertion of data in the database.	13, 16
10	Parsing and Mathematical Operations	To convert data types and calculate scores.	11, 12
11	Exception Handling (try-catch methods)	To catch unexpected/unhandled errors and outputting an error message.	13, 16
12	Data Structures (1D-Arrays, Lists, 2D-Arrays using Datasets and Datatables)	To fetch rows from database and displaying them on gridview(s). Array for storing eligible grades.	-
13	Sorting of Array	To display top 3 winners of fest in labels.	12
14	User-Defined Methods	To allow modular programming.	-
15	Iterations ('for' and 'for each' loops)	Looping through each row in GridView, looping through arrays.	9, 18
16	Nested and Complex If…Else statements.	For outcomes that require multiple dependent if statements and listing multiple search conditions.	15

17	OOPS concept (Polymorphism, Encapsulation,Inheritance)	To create objects and re-use them throughout the application.	-
18	Stored Procedure	Used for change password functionality.	3
19	StringBuilder, StmpClient and MailClient	To send a randomly generated password to the user's registered email in case they forgot their password.	3
20	ASP.NET GridView controls (DataBound rows, Item Template, DataKeyNames, OnSelectedIndexChanged, OnRowDeleting)	Used to run methods, display editable fields, fetch column from database butnot display in GridView and more.	-
21	Manipulation of GridView and Calendar Control.	When user selects a row in GridView, the row colour changes. When user selects date in calendar, selected date's background colour changes.	-
22	Role-Based Access	Redirects user to specific pages based on their email.	4
23	Session Variable	To display logged-in email on each page or to pass values from dropwdowns etc.	2, 4
24	Date-Time Functions	To compare dates using range validator.	5

GUI

All screens were created using HTML/CSS tags to create an intuitive and professional UI and ASP.NET controls provided interactive elements. Each screen has a consistent layout with the same background colour and similar colours for text, buttons and more.

Homepage –



Homepage Source Code –

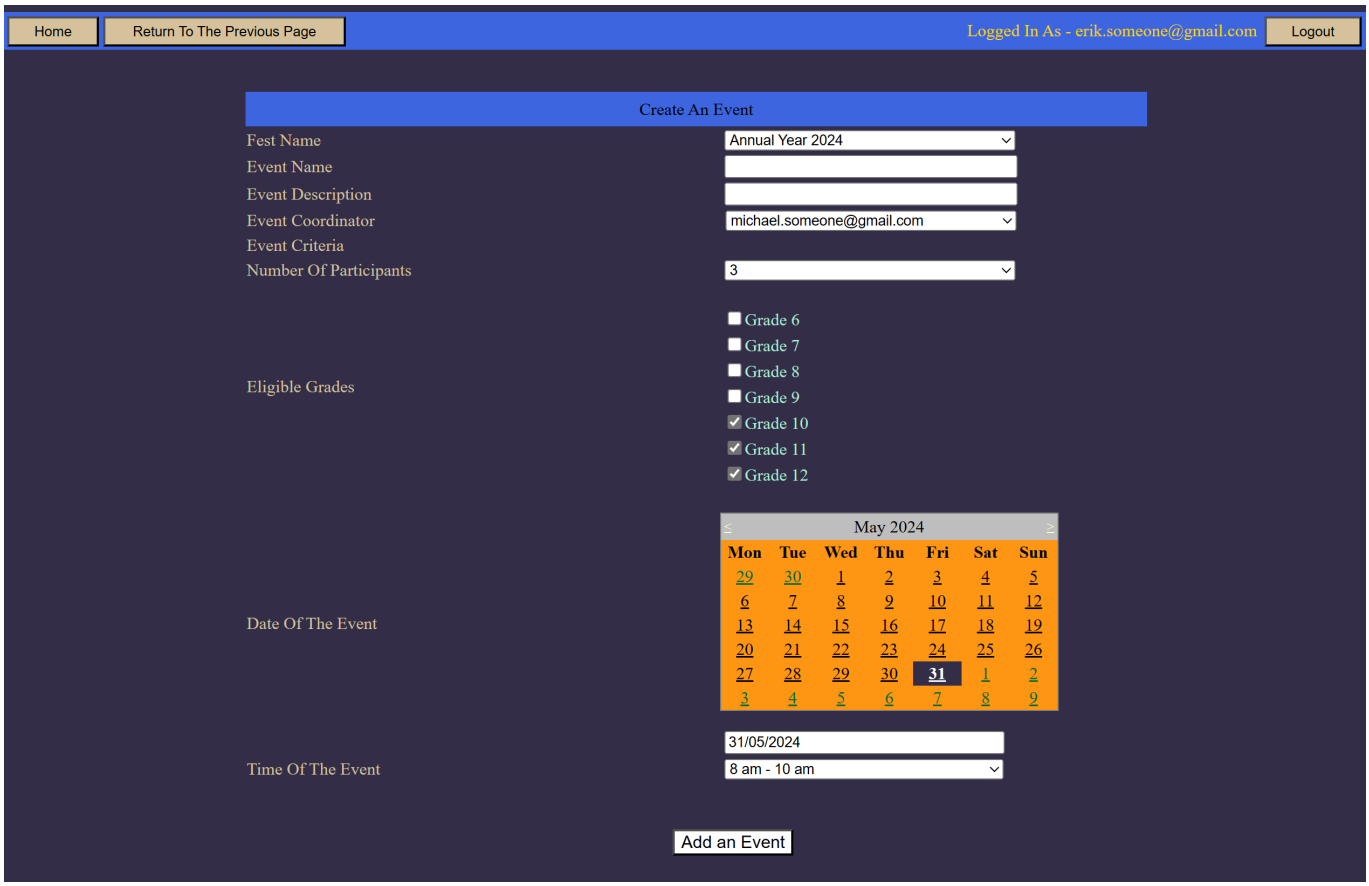
```
35  <body id="PageBody" runat="server" bgcolor ="#343148" style="width: 1279px; height: 589px; margin-left: 0px;">
36      <form id="form1" runat="server">
37          <div>
38              <asp:Panel ID="Panel1" runat="server" BackColor="RoyalBlue" Height="35px" Width="100%">
39                  <p class="auto-style3">
40                                              
                      <asp:Label ID="Label4" runat="server" Font-Size="X-Large" ForeColor="Gold" Text="Welcome to our Inter-School Fest!
                  </p>
              </asp:Panel>
          <p>
                                      
46            <img alt="" class="auto-style4" src="Icons/Male%20User.png" />   &
47            <img alt="" class="auto-style4" src="Icons/Male%20User.png" />   &
48            <img alt="" class="auto-style5" src="Icons/Admin.png" /></p>
49          <p>
50                                  
51              <asp:Label ID="Label2" runat="server" Text="School" ForeColor="#ADEFD1" Font-Size="X-Large"></asp:Label>
52                                      
53              <asp:Label ID="Label1" runat="server" Text="Event Coordinators" ForeColor="#ADEFD1" Font-Size="X-Large"></asp:Label>
54                           
55              <asp:Label ID="Label3" runat="server" Text="Admin" ForeColor="#ADEFD1" Font-Size="X-Large"></asp:Label>
56          </p>
            <p>
                <asp:Button ID="btnRegSchool" runat="server" Height="35px" Text="Register" Width="112px" OnClick="btnRegSchool_Click" />
                <asp:Button ID="btnLogSchool" runat="server" Height="35px" Text="Login" Width="112px" OnClick="btnLogSchool_Click" />
```

Class tags defining a group of HTML elements

Image HTML tags specifying the source of the images.

ASP.NET buttons with OnClick methods

Creating an Event –


Home
Return To The Previous Page
Logged In As - erik.someone@gmail.com
Logout
Create An Event
Fest Name
Annual Year 2024
Event Name
Event Description
Event Coordinator
michael.someone@gmail.com
Event Criteria
Number Of Participants
3
Eligible Grades
Grade 6
Grade 7
Grade 8
Grade 9
Grade 10
Grade 11
Grade 12
Date Of The Event
May 2024
Mon Tue Wed Thu Fri Sat Sun
29 30 1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31 1 2
3 4 5 6 7 8 9
31/05/2024
Time Of The Event
8 am - 10 am
Add an Event

Source Code –

```
<head runat="server">
    <title></title>
    <script src="AdminEventMakingPageValidations.js" type="text/javascript"></script>
    <style type="text/css">
        body {background-color: #343148;}
        .auto-style1 {
            width: 840px;
            height: 420px;
            margin-left: 222px;
            margin-right: 0px;
        }
        .auto-style7 {
            margin-left: 530px;
            height: 34px;
            margin-top: 0px;
            width: 707px;
        }
        .auto-style8 {
            margin-top: 0px;
        }
        .auto-style9 {
            margin-left: 620px;
            margin-top: 43px;
        }
        .auto-style11 {
            margin-top: 0px;
        }
        .auto-style10 {
            margin-left: 3px;
            margin-top: 4px;
        }
```

Script tag referring to JavaScript file that contains code for validation

Style tag to give background colour and to maintain positions and size of controls

```
<td class="auto-style15">
     <asp:DropDownList ID="ddlNumberOfParticipants" runat="server" CssClass="auto-style12" Width="270px">
        <asp:ListItem>Select</asp:ListItem>
        <asp:ListItem>1</asp:ListItem>
        <asp:ListItem>2</asp:ListItem>
        <asp:ListItem>3</asp:ListItem>
        <asp:ListItem>4</asp:ListItem>
        <asp:ListItem>5</asp:ListItem>
    </asp:DropDownList>
       </td>
  </tr>
   <tr>
       <td class="auto-style14">
           <asp:Label ID="Label3" runat="server" ForeColor="#D7C49E" Text="Eligible Grades"></asp:Label>
       </td>
<td class="auto-style15">
            
    <asp:CheckBoxList ID="cblEligibleGrades" runat="server" ForeColor="#ADEFD1" Width="202px" CssClass="auto-style12">
        <asp:ListItem>Grade 6</asp:ListItem>
        <asp:ListItem>Grade 7</asp:ListItem>
        <asp:ListItem>Grade 8</asp:ListItem>
        <asp:ListItem>Grade 9</asp:ListItem>
        <asp:ListItem>Grade 10</asp:ListItem>
        <asp:ListItem>Grade 11</asp:ListItem>
        <asp:ListItem>Grade 12</asp:ListItem>
    </asp:CheckBoxList>
   </td>
  </tr>
```

DropDown List control with pre-defined valuesto limit selection

CheckBox List control toselect multiple values from pre-defined values to limit selection

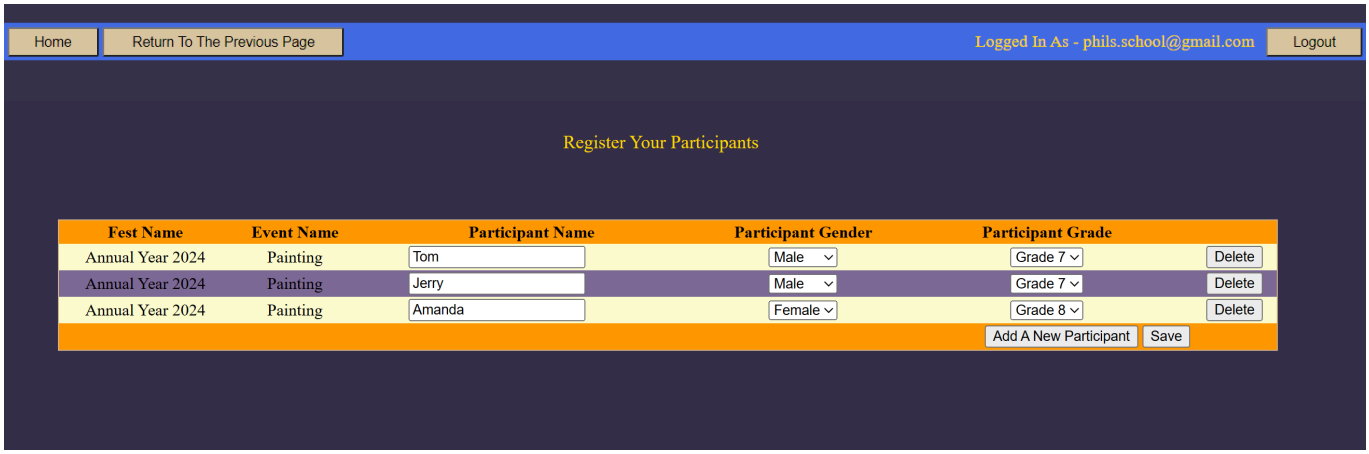
```
        <asp:Calendar ID="Calendar1" runat="server" BackColor="#FF9912" OnSelectionChanged="Calendar1_SelectionChanged" CssClass="auto-style12" Width="314px">
            <SelectedDayStyle BackColor="#343148" Font-Bold="True" />
    <SelectorStyle BackColor="#FFCC66" />
    <TodayDayStyle BackColor="#03A89E" ForeColor="Black" />
    <OtherMonthDayStyle ForeColor="#006B38" />
    <NextPrevStyle Font-Size="9pt" ForeColor="#FFFFCC" />
        </asp:Calendar>
                <br />
                 <asp:TextBox ID="txtDateOfTheEvent" runat="server" Width="252px" ReadOnly="True"></asp:TextBox>
                <asp:RangeValidator ID="Rvalid" runat="server" ControlToValidate="txtDateOfTheEvent"
                    Display="Dynamic" ErrorMessage="The Event has to be registered 1 to 3 months in advance." ForeColor="Red"></asp:RangeValidator>
            </td>
           </tr>
           <tr>
            <td class="auto-style14">
                <asp:Label ID="Label1" runat="server" ForeColor="#D7C49E" Text="Time Of The Event"></asp:Label>
                <br />
               </td>
            <td class="auto-style15">
                 <asp:DropDownList ID="ddlTimeOfTheEvent" runat="server" CssClass="auto-style12" Width="259px">
                    <asp:ListItem>8 am - 10 am</asp:ListItem>
                    <asp:ListItem>10 am - 12 pm</asp:ListItem>
                    <asp:ListItem>12 pm - 2 pm</asp:ListItem>
                    <asp:ListItem>2 pm - 4 pm</asp:ListItem>
                </asp:DropDownList>
            </td>
           </tr>
       </table>
<p class="auto-style9">
    <asp:Button ID="BtnAddEvent" runat="server" Text="Add an Event" OnClientClick="return AdminEventMakingPageValidation();" OnClick="BtnAddEvent_Click" CssCl
</p>
 <div class="auto-style7">
    <asp:Label ID="lblAddAnEvent" runat="server" ForeColor="Gold"></asp:Label>
</div>
```

Calendar Control to select a date for the event

Range Validator for Calendar Control to avoid unnecessary errors

Button that returns the validation function in the .js file

Registering for Events –



Source Code –

```
<asp:GridView ID="gvParticipant" AutoGenerateColumns="false" runat="server" ShowFooter="true" BackColor="#c7d3d4"
BorderColor="Tan" BorderWidth="1px" CellPadding="2" ForeColor="Black" GridLines="None" Width="1176px"
OnRowDeleting="OnRowDeleting" CssClass="auto-style14" >
<RowStyle HorizontalAlign="Center"></RowStyle>
<AlternatingRowStyle BackColor="#7e6896" />
<FooterStyle BackColor="#ff9900" />
<HeaderStyle BackColor="#ff9900" Font-Bold="True" />
<PagerStyle BackColor="#7e6896" ForeColor="DarkSlateBlue" HorizontalAlign="Center" />
<SelectedRowStyle BackColor="DarkSlateBlue" ForeColor="GhostWhite" />
<SortedAscendingCellStyle BackColor="#FAFAE7" />
<SortedAscendingHeaderStyle BackColor="#DAC09E" />
<SortedDescendingCellStyle BackColor="#E1DB9C" />
<SortedDescendingHeaderStyle BackColor="#C2A47B" />
<Columns>
<asp:BoundField DataField="FestName" HeaderText="Fest Name" />
<asp:BoundField DataField="EventName" HeaderText="Event Name" />
<asp:TemplateField HeaderText="Participant Name">
    <ItemTemplate>
        <asp:TextBox ID="txtParticipantName" runat="server"></asp:TextBox>
        <asp:RequiredFieldValidator ID="rfvParticipantName" ControlToValidate="txtParticipantName" runat="server"
            ErrorMessage="Participant Name Required." ForeColor="Red" ></asp:RequiredFieldValidator>
    </ItemTemplate>
</asp:TemplateField>
```

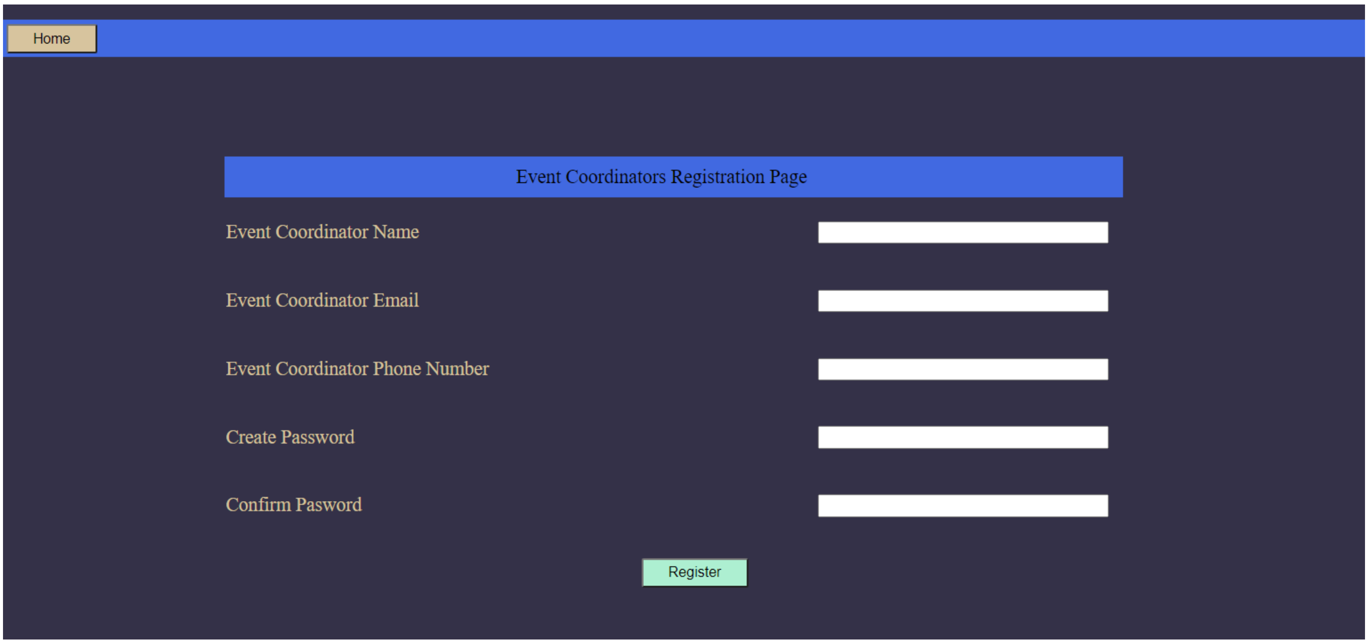
GridView with various interactive elements for adding participants

DataBound fields to display un-editable data

Required Field Validator to ensure control has value

ItemTemplate tags that contain editable controls

Event Coordinator Registration Page –



Source Code

Table element to give an organized layout

```
<table class="auto-style1">
 <tr>
     <td colspan="2" style="background-color: royalblue; height: 30px; width: 100% ; color: White;" class="auto-style7">
                               
         <asp:Label ID="Label8" runat="server" Text="Event Coordinators Registration Page" Width="417px" Font-Size="Large" ForeColor="Black"
     </td>
 </tr>
 <tr>
     <td class="auto-style13">
         <asp:Label ID="Label1" runat="server" ForeColor="#D7C49E" Text="Event Coordinator Name" Font-Size="Large"></asp:Label>
     </td>
     <td class="auto-style6">
               
<asp:TextBox ID="txtEventCoordinatorName" Width="264px" ValidationGroup="Validform" runat="server" CssClass="auto-style8"></asp:TextBox>
     </td>
 </tr>
```

```
    <tr>
        <td class="auto-style13">
            <asp:Label ID="Label6" runat="server" ForeColor="#D7C49E" Text="Create Password" Font-Size="Large"></asp:Label>
        </td>
 <td class="auto-style14">
                   
<asp:TextBox ID="txtCreatePasswordEventCoordinator" Width="264px" runat="server" TextMode="Password" CssClass="auto-style8" ></asp:TextBox>
        </td>
   </tr>
   <tr>
        <td class="auto-style13">
            <asp:Label ID="Label7" runat="server" ForeColor="#D7C49E" Text="Confirm Pasword" Font-Size="Large"></asp:Label>
        </td>
  <td class="auto-style14">
                       
        <asp:TextBox ID="txtConfirmPasswordEventCoordinator" Width="264px" runat="server" TextMode="Password" CssClass="auto-style8" ></asp:TextBox>
        </td>
             
   </tr>
   </table>
```

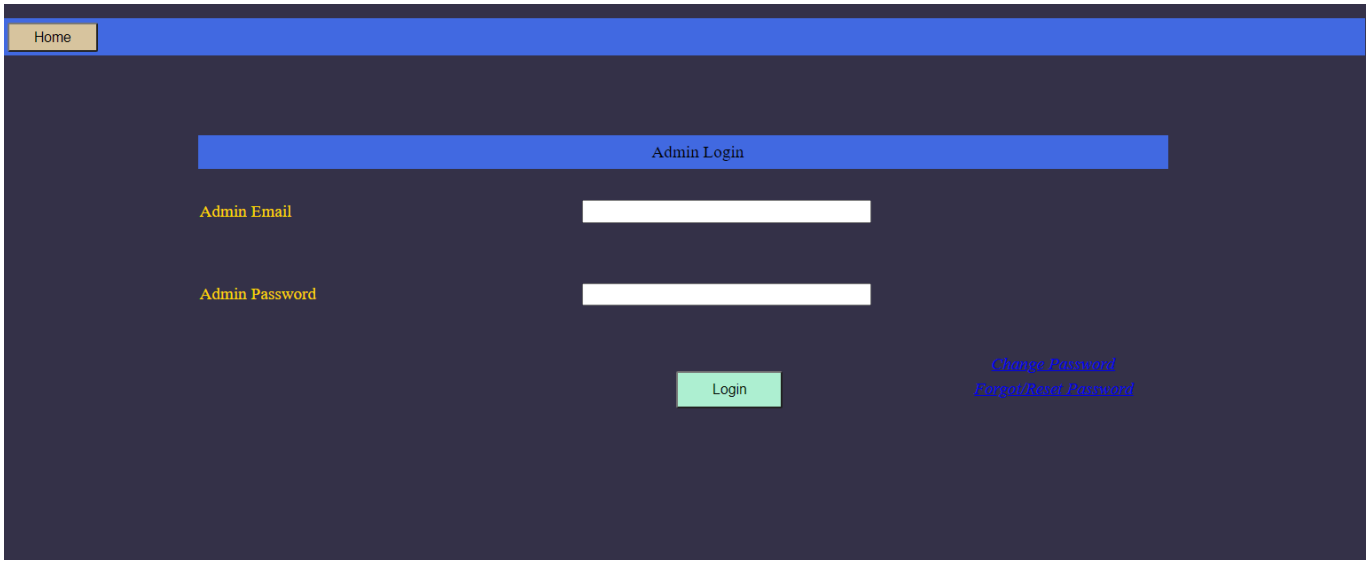
TextMode used to mask all characters entered in the textbox

runat=”server” used to ensure it is accessible at server side

Layered Architecture

The product consists of a presentation layer (UI), business layer (Classes), and a data layer (MySQL database) to allow modular approach.

Presentation Layer –



Source Code –

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <style type="text/css">
        body {background-color: #343148;}
        .auto-style1 {
            width: 915px;
            height: 143px;
            margin-bottom: 15px;
            margin-left: 181px;
            margin-right: 0px;
        }
        .auto-style5 {
            height: 74px;
            width: 356px;
        }
        .auto-style4 {
            height: 74px;
        }
        .auto-style2 {
            margin-left: 0px;
        }
        .auto-style10 {
            margin-left: 3px;
            margin-top: 4px;
        }
        </style>
</head>
<body style="margin-left: 0px; height: 589px; width: 1279px; margin-top: 0px;">
    <form id="form1" runat="server">
```

Code Behind Login Page –

```
public partial class AdminLogin : System.Web.UI.Page
{
    0 references
    protected void BtnLoginAdmin_Click(object sender, EventArgs e)
    {
        UserClass objAdminLogin = new AdminClass();
        objAdminLogin.UserEmail = txtAdminEmail.Text;
        objAdminLogin.UserPassword = txtAdminPassword.Text;

        int Result = objAdminLogin.LoginCheck();
        if (Result == 1)
        {
            lblAdminLogin.Visible = false;
            Session["AdminEmail"] = txtAdminEmail.Text;
            Response.Redirect("AdminHomePage.aspx");
        }
        else
        {
            lblAdminLogin.Visible = true;
            lblAdminLogin.Text = "Login Credentials Invalid. Please try again!";
        }
    }
    0 references
    protected void LinkbtnChangePasswordAdmin_Click(object sender, EventArgs e)
    {
        Response.Redirect("AdminChangePasswordPage.aspx");
    }

    0 references
    protected void LinkbtnForgotPasswordAdmin_Click(object sender, EventArgs e)
    {
        Response.Redirect("AdminForgotPasswordPage.aspx");
    }

    0 references
    protected void btnHomePage_Click(object sender, EventArgs e)
    {
        Response.Redirect("HomePage.aspx");
    }
}
```

Creating objects of base class and declaring it to a new AdminClass

Calling the result of the method from the AdminClass

Business Layer –

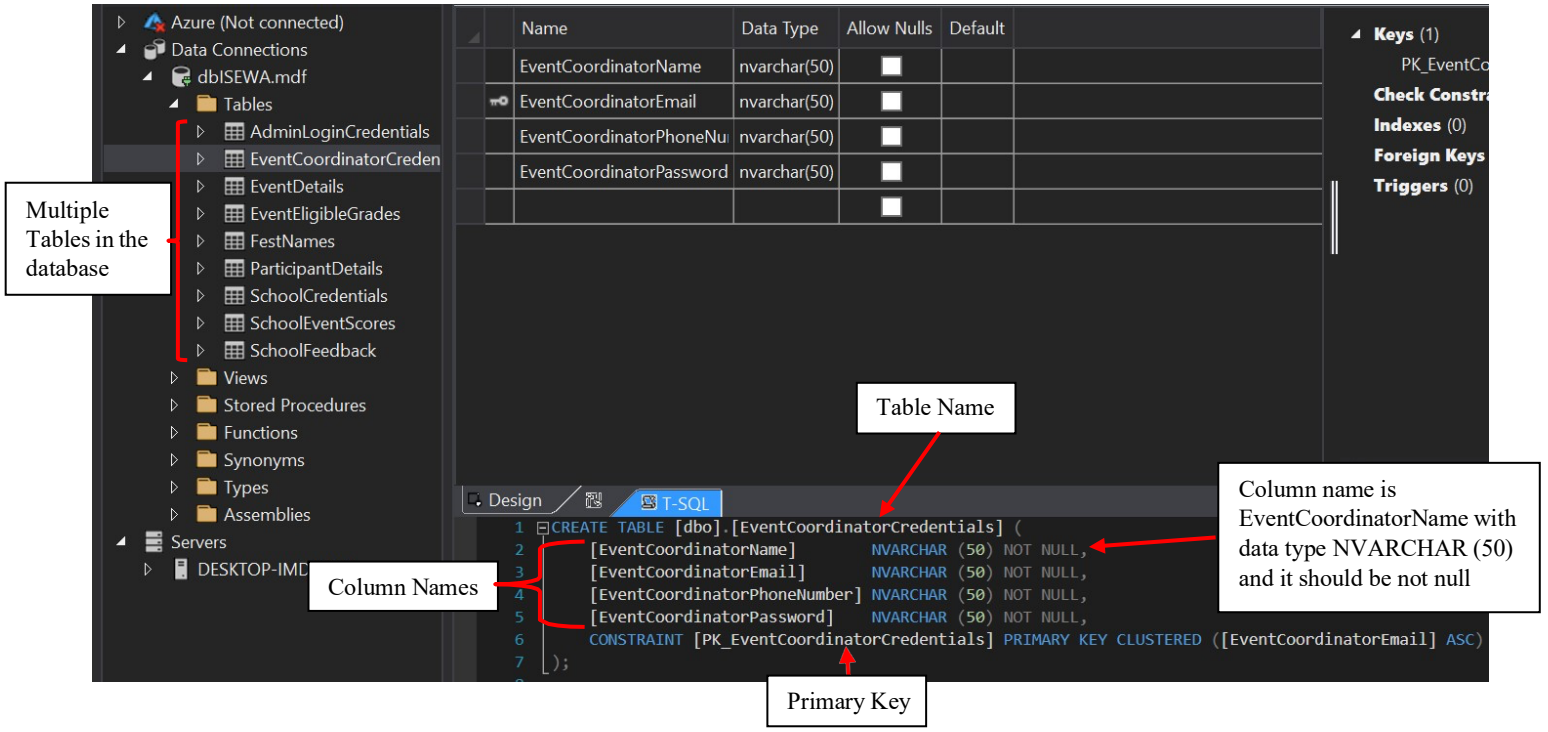
```
public override int LoginCheck()
{
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["mycon"].ToString());
    con.Open();
    SqlCommand cmd = new SqlCommand("SELECT * FROM AdminLoginCredentials WHERE" +
        " AdminEmail='" + this.UserEmail + "' and AdminPassword='" + this.UserPassword + "'", con);
    cmd.Parameters.AddWithValue("@AdminEmail", this.UserEmail);
    cmd.Parameters.AddWithValue("@AdminPassword", this.UserPassword);
    SqlDataAdapter da = new SqlDataAdapter(cmd);
    DataTable dt = new DataTable();
    da.Fill(dt);
    if (dt.Rows.Count > 0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

Data Layer –

	AdminEmail	AdminPassword
	erik.someone@gmail.com	dedsef

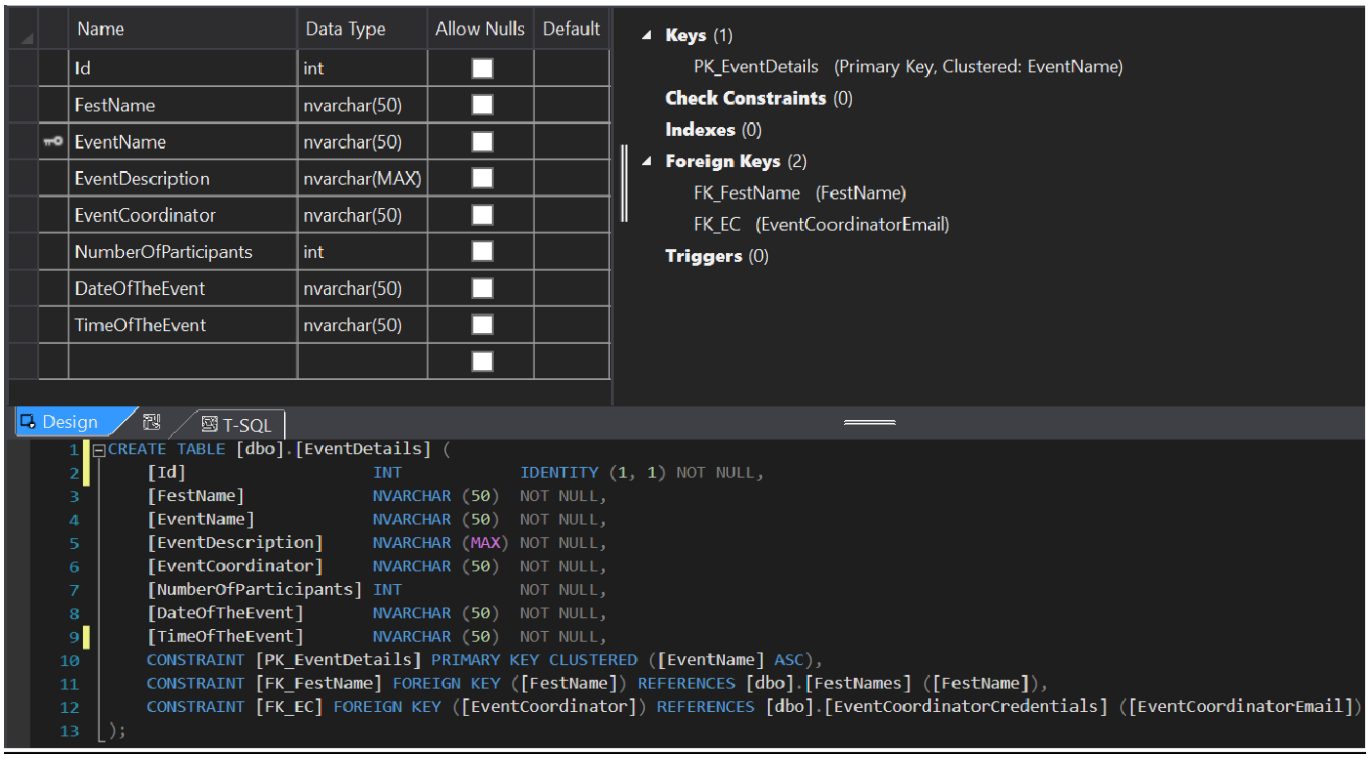
Database Design

Multiple data-tables have been created and used throughout the application. The database is normalized to 3NF, reducing redundancy and removing transitive dependencies.



	EventCoordina...	EventCoordinatorE...	EventCoordina...	EventCoordina...
	Anne Gathion	annegathion140@g...	9648009263	dedsef
	Erik Someone	erik.someone@yah...	9908789045	dedsef
	Eri Singh	erisingh@gmail.com	9911181162	password
	James John	james.john@gmail.c...	9999000876	dedsef
	Jeffery	jeffery.malhotra@m...	9333374653	dedsef
	Michael	michael.someone@...	9327530947	dedsef
	Piper McLean	pipermclean21@bin...	9326490989	dedsef
	NULL	*NULL*	*NULL*	*NULL*

Event Details Table –

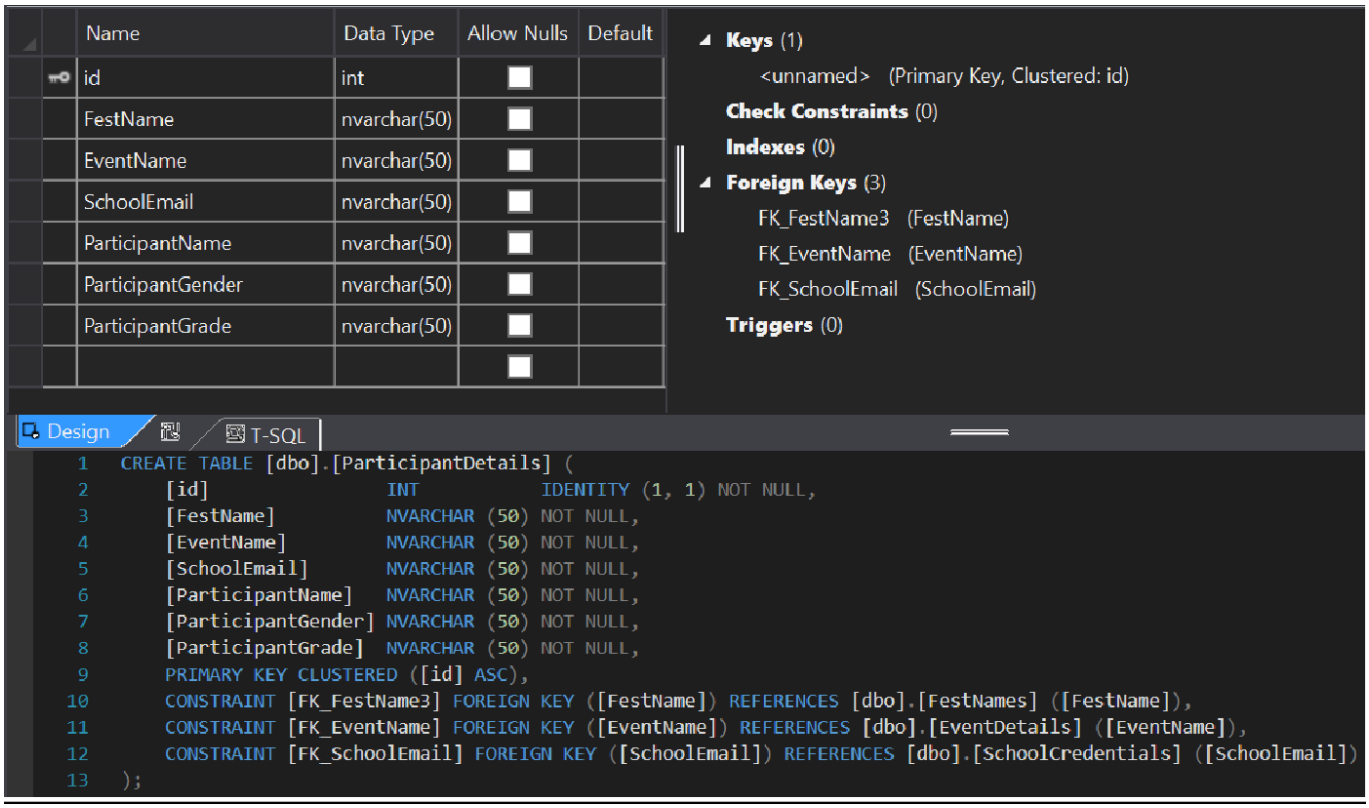


Name	Data Type	Allow Nulls	Default
Id	int	☐	
FestName	nvarchar(50)	☐	
EventName	nvarchar(50)	☐	
EventDescription	nvarchar(MAX)	☐	
EventCoordinator	nvarchar(50)	☐	
NumberOfParticipants	int	☐	
DateOfTheEvent	nvarchar(50)	☐	
TimeOfTheEvent	nvarchar(50)	☐	
		☐	


```
CREATE TABLE [dbo].[EventDetails] (
    [Id]                  INT            IDENTITY (1, 1) NOT NULL,
    [FestName]            NVARCHAR (50)  NOT NULL,
    [EventName]           NVARCHAR (50)  NOT NULL,
    [EventDescription]    NVARCHAR (MAX) NOT NULL,
    [EventCoordinator]    NVARCHAR (50)  NOT NULL,
    [NumberOfParticipants] INT           NOT NULL,
    [DateOfTheEvent]      NVARCHAR (50)  NOT NULL,
    [TimeOfTheEvent]      NVARCHAR (50)  NOT NULL,
    CONSTRAINT [PK_EventDetails] PRIMARY KEY CLUSTERED ([EventName] ASC),
    CONSTRAINT [FK_FestName] FOREIGN KEY ([FestName]) REFERENCES [dbo].[FestNames] ([FestName]),
    CONSTRAINT [FK_EC] FOREIGN KEY ([EventCoordinator]) REFERENCES [dbo].[EventCoordinatorCredentials] ([EventCoordinatorEmail])
);
```

	Id	FestName	EventName	EventDescripti...	EventCoordina...	NumberOfPa...	DateOfTheEvent	TimeOfTheEvent
	1	Annual Year 2020	All you can Eat!	Participants will ...	erik.someone@...	3	28-11-2021	12 pm - 2 pm
	2	Fun Fest!	Basketball Fever	Schools will co...	james.john@g...	5	24-11-2021	10 am - 12 pm
	3	Annual Year 2020	Doodling	Participants will ...	pipermclean21...	2	28-11-2021	8 am - 10 am
	4	Fun Fest!	Gaming	Participants will ...	james.john@g...	1	24-11-2021	8 am - 10 am
	5	Annual Year 2020	Painting	Participants will ...	james.john@g...	3	28-11-2021	12 pm - 2 pm
	6	Fun Fest!	Singing at the B...	Participants hav...	erik.someone@...	2	27-11-2021	2 pm - 4 pm
▷	*N...*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

Participants Details Table –



Name	Data Type	Allow Nulls	Default
id	int	☐	
FestName	nvarchar(50)	☐	
EventName	nvarchar(50)	☐	
SchoolEmail	nvarchar(50)	☐	
ParticipantName	nvarchar(50)	☐	
ParticipantGender	nvarchar(50)	☐	
ParticipantGrade	nvarchar(50)	☐	
		☐	

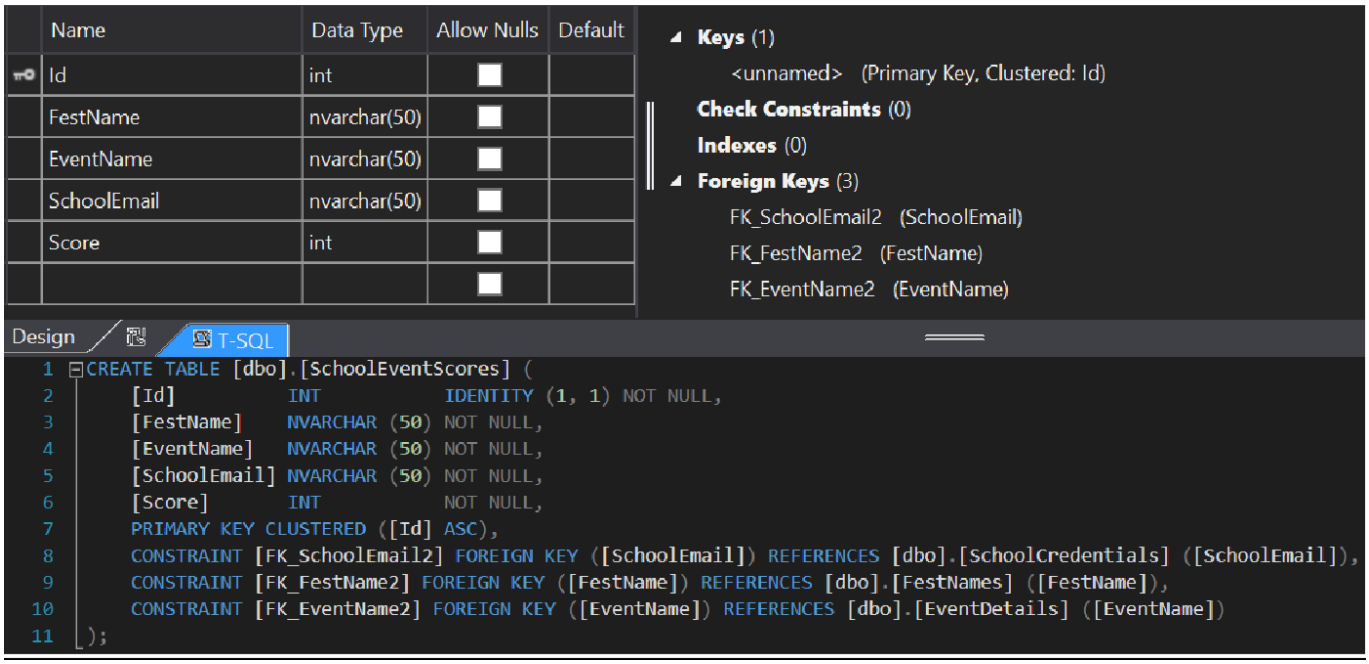
Keys (1)
<unnamed> (Primary Key, Clustered: id)
Check Constraints (0)
Indexes (0)
Foreign Keys (3)
FK_FestName3 (FestName)
FK_EventName (EventName)
FK_SchoolEmail (SchoolEmail)
Triggers (0)

Design | T-SQL

```
CREATE TABLE [dbo].[ParticipantDetails] (
    [id]                INT           IDENTITY (1, 1) NOT NULL,
    [FestName]          NVARCHAR (50) NOT NULL,
    [EventName]         NVARCHAR (50) NOT NULL,
    [SchoolEmail]       NVARCHAR (50) NOT NULL,
    [ParticipantName]   NVARCHAR (50) NOT NULL,
    [ParticipantGender] NVARCHAR (50) NOT NULL,
    [ParticipantGrade]  NVARCHAR (50) NOT NULL,
    PRIMARY KEY CLUSTERED ([id] ASC),
    CONSTRAINT [FK_FestName3] FOREIGN KEY ([FestName]) REFERENCES [dbo].[FestNames] ([FestName]),
    CONSTRAINT [FK_EventName] FOREIGN KEY ([EventName]) REFERENCES [dbo].[EventDetails] ([EventName]),
    CONSTRAINT [FK_SchoolEmail] FOREIGN KEY ([SchoolEmail]) REFERENCES [dbo].[SchoolCredentials] ([SchoolEmail])
);
```

	Id	FestName	EventName	SchoolEmail	ParticipantName	ParticipantGen...	ParticipantGrade
	1	Annual Year 2024	Painting	jermylynch@na...	Mishel	Female	Grade 6
	2	Annual Year 2024	Painting	jermylynch@na...	Zack	Male	Grade 8
	3	Annual Year 2024	Painting	lexi.bellweather...	Jason	Male	Grade 6
	4	Annual Year 2024	Painting	lexi.bellweather...	Piper Weasely	Female	Grade 7
	5	Annual Year 2024	Painting	lexi.bellweather...	Manny	Male	Grade 6
	6	Annual Year 2024	Painting	lexi.bellweather...	Piper Weasely	Female	Grade 7
	7	Annual Year 2024	Doodling	anantasschool...	Annabeth	Female	Grade 12
	8	Annual Year 2024	Doodling	anantasschool...	Hazel	Female	Grade 12
	9	Annual Year 2024	Doodling	bhumika@mou...	Reyna	Female	Grade 12
	10	Annual Year 2024	Doodling	bhumika@mou...	Nico	Male	Grade 12
	11	Annual Year 2024	Painting	bhumika@mou...	Megan Molly	Female	Grade 8
	12	Annual Year 2024	Painting	bhumika@mou...	Harry James	Male	Grade 8
	13	Annual Year 2024	Painting	bhumika@mou...	Percy	Male	Grade 7
	14	Annual Year 2024	Singing at the ...	philip.phineas...	Erik	Male	Grade 7
	15	Annual Year 2024	Singing at the ...	philip.phineas...	Peter Parker	Male	Grade 8
	16	Annual Year 2024	Singing at the ...	jermylynch@na...	Zack	Male	Grade 6
	17	Annual Year 2024	Singing at the ...	jermylynch@na...	Erik	Male	Grade 6
	18	Annual Year 2024	Gaming	jermylynch@na...	Percy Jackson	Male	Grade 10
	19	Annual Year 2024	Gaming	anantasschool...	Nick	Male	Grade 7
	20	Annual Year 2024	Gaming	bhumika@mou...	Jason	Male	Grade 7
	21	Annual Year 2024	Basketball Fever	philip.phineas...	Tyson	Male	Grade 6
	22	Annual Year 2024	Basketball Fever	philip.phineas...	Vikram	Male	Grade 6
	23	Annual Year 2024	Basketball Fever	philip.phineas...	Jay	Male	Grade 7
	24	Annual Year 2024	All you can Eat!	philip.phineas...	Slogo	Male	Grade 6

School Event Scores Table –



	Id	FestName	EventName	SchoolEmail	Score
	1	Annual Year 2024	Basketball Fever	philip.phineas...	7
	2	Annual Year 2024	Gaming	anantasschool...	8
	3	Annual Year 2024	Gaming	jermylynch@na...	9
	4	Annual Year 2024	Singing at the ...	jermylynch@na...	9
	5	Annual Year 2024	Singing at the ...	philip.phineas...	6
	6	Annual Year 2024	Doodling	anantasschool...	6
	7	Annual Year 2024	Doodling	bhumika@mou...	9

School Feedback Table –

Name	Data Type	Allow Nulls	Default
Id	int	☐	
FestName	nvarchar(50)	☐	
SchoolEmail	nvarchar(50)	☐	
FestRate	int	☐	
OrganizationOfFestRate	int	☐	
ReturnRate	nvarchar(50)	☐	
LikesAboutFestOrEvent	nvarchar(MAX)	☑	
DislikesAboutFestOrEvent	nvarchar(MAX)	☑	
GeneralThoughts	nvarchar(MAX)	☑	

Keys (1)
<unnamed> (Primary Key, Clustered: Id)
Check Constraints (0)
Indexes (0)
Foreign Keys (2)
FK_FestName4 (FestName)
FK_SchoolEmail3 (SchoolEmail)
Triggers (0)

Design | T-SQL

```
CREATE TABLE [dbo].[SchoolFeedback] (
    [Id]                    INT             IDENTITY (1, 1) NOT NULL,
    [FestName]              NVARCHAR (50)   NOT NULL,
    [SchoolEmail]           NVARCHAR (50)   NOT NULL,
    [FestRate]              INT             NOT NULL,
    [OrganizationOfFestRate] INT            NOT NULL,
    [ReturnRate]            NVARCHAR (50)   NOT NULL,
    [LikesAboutFestOrEvent] NVARCHAR (MAX)  NULL,
    [DislikesAboutFestOrEvent] NVARCHAR (MAX) NULL,
    [GeneralThoughts]       NVARCHAR (MAX)  NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC),
    CONSTRAINT [FK_FestName4] FOREIGN KEY ([FestName]) REFERENCES [dbo].[FestNames] ([FestName]),
    CONSTRAINT [FK_SchoolEmail3] FOREIGN KEY ([SchoolEmail]) REFERENCES [dbo].[SchoolCredentials] ([SchoolEmail])
);
```

Id	FestName	SchoolEmail	FestRate	OrganizationOf...	ReturnRate	LikesAboutFest...	DislikesAboutF...	GeneralThough...
1	Fun Fest!	louise.gathion...	5	5	Yes	Organized	Harsh Judges	We loved the fe...
2	Fun Fest!	jermylynch@na...	4	5	Yes	It was Fun	None	We will be back!
3	Annual Year 2024	bhumika@mou...	4	5	No	None	Unorganized	None.
4	Annual Year 2024	jermylynch@na...	4	5	Maybe	Variety of events	None	None.
NULL	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

Establishing Database Connection

Web.config file

```
  <add name="mycon" connectionString="Data Source=(LocalDB)\MSSQLLocalDB;
       AttachDbFilename=C:\Users\jayma\source\repos\ISEWA\App_Data\dbISEWA.mdf;
       Integrated Security=True" />
 </connectionStrings>
</configuration>
```

Creating the connection string in the web.config file to connect the application to database

Calling the connection string –

```
public DataSet FetchSchoolFeedback(string SchoolEmail)
{
    string constr = ConfigurationManager.ConnectionStrings["mycon"].ConnectionString;
    SqlConnection con = new SqlConnection(constr);
    SqlCommand cmd = new SqlCommand();
    string sql;

    if (SchoolEmail != "All Schools")
    {
        sql = "SELECT * FROM [dbo].[SchoolFeedback] WHERE ([SchoolEmail] = '" + SchoolEmail + "'))";
    }
    else
    {
        sql = "SELECT * FROM SchoolFeedback WHERE ([SchoolEmail] = '" + SchoolEmail + "'))";
    }

    SqlDataAdapter da = new SqlDataAdapter(sql, con);
    DataSet ds = new DataSet();
    con.Open();
    da.Fill(ds, "Schoolfeedback");
    con.Close();
    return ds;
}
```

Calling the connection string

Establishing a new connection within this method to the database

SQL command to execute SQL statements

SqlDataAdapter retrieves data from data source to a dataset

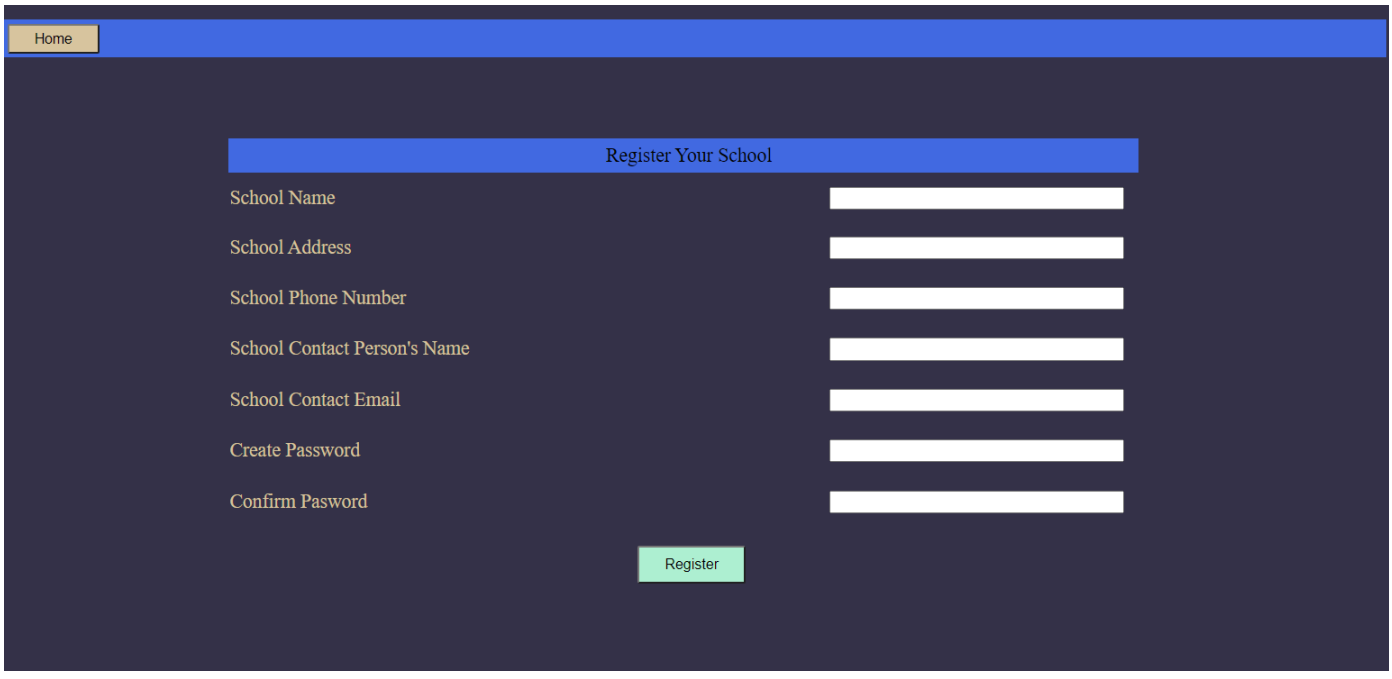
DataSet holds the result after executing the SqlCommand

Basic Database Operations

Multiple database operations were used such as adding scores or participants, deleting existing records from the database, updating event details and searching for records. One example of each operation is shown below –

Add/Insert

Used when user is registering on the application.



```
protected void BtnRegister_Click(object sender, EventArgs e)
{
    SchoolClass objSchoolRegister = new SchoolClass();
    objSchoolRegister.SchoolName = txtSchoolName.Text;
    objSchoolRegister.SchoolAddress = txtSchoolAddress.Text;
    objSchoolRegister.SchoolPhoneNumber = txtSchoolPhoneNumber.Text;
    objSchoolRegister.SchoolContactPerson = txtSchoolContactPersonName.Text;
    objSchoolRegister.UserEmail = txtSchoolContactEmail.Text;
    objSchoolRegister.UserPassword = txtCreatePassword.Text;

    int Result = objSchoolRegister.AddUserDetails();
    if (Result > 0)
    {
        lblRegisterSchoolMessage.Visible = true;
        lblRegisterSchoolMessage.Text = "You have successfully registered your school";
        Response.Redirect("SchoolLoginPage.aspx");
    }
    else
    {
        lblRegisterSchoolMessage.Visible = true;
        lblRegisterSchoolMessage.Text = "Registration Unsuccessful. Please try again!";
    }
}
```

When button is clicked, this method is executed

An object in the SchoolClassis created

The user-defined method returns a value and is fetched to an int variable

```
SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["mycon"].ToString());
1 reference
public int AddUserDetails()
{
    try
    {
        string qry = "insert into SchoolLoginCredentials values(@SchoolName, @SchoolEmail, @SchoolPassword, " +
            "@SchoolAddress, @SchoolContactPerson, @SchoolPhoneNumber)";
        SqlCommand cmd = new SqlCommand(qry, con);
        cmd.Parameters.AddWithValue("@SchoolName", this.SchoolName);
        cmd.Parameters.AddWithValue("@SchoolContactEmail", this.UserEmail);
        cmd.Parameters.AddWithValue("@SchoolPassword", this.UserPassword);
        cmd.Parameters.AddWithValue("@SchoolAddress", this.SchoolAddress);
        cmd.Parameters.AddWithValue("@SchoolContactPerson", this.SchoolContactPerson);
        cmd.Parameters.AddWithValue("@SchoolPhoneNumber", this.SchoolPhoneNumber);

        con.Open();
        int Result = cmd.ExecuteNonQuery();
        cmd.Dispose();
        return Result;
    }
    catch (Exception ex)
    {
        return 0;
    }
}
```

Connection to the database is established

SQL command to execute query in the database

An SQL insert query is defined that specifies the table and column names

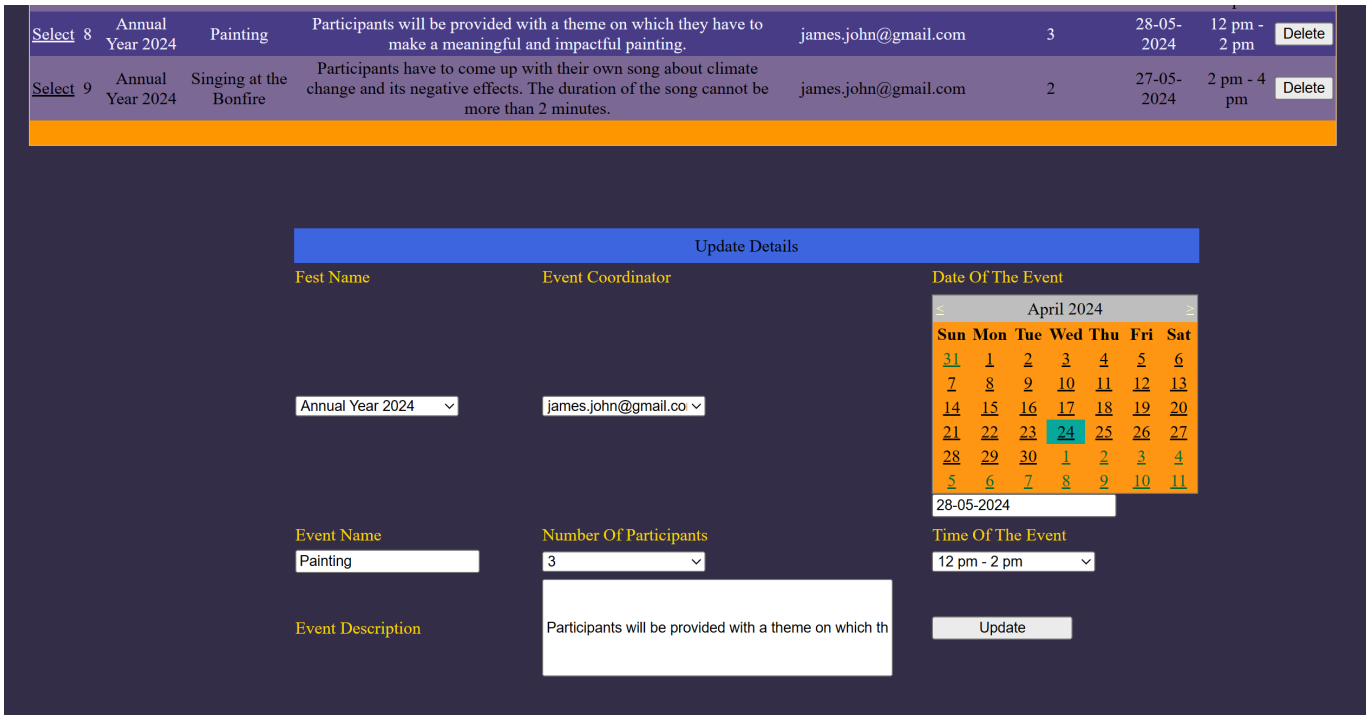
Declaring parameters and their corresponding values

Opening the connection, executing the query, closing the connection, then returning the value in an int variable

Delete and Update Operations

Both operations are used when the admin wants to update the event details.

When user selects a row from the GridView –



```
protected void btnUpdate_Click(object sender, EventArgs e)
{
    string constr = ConfigurationManager.ConnectionStrings["mycon"].ConnectionString;
    using (SqlConnection con = new SqlConnection(constr))
    {
        string query = "UPDATE EventDetails SET FestName=@FestName, EventName=@EventName, " +
            "EventDescription=@EventDescription, EventCoordinator=@EventCoordinator, " +
            "NumberOfParticipants=@NumberOfParticipants, DateOfTheEvent=@DateOfTheEvent, " +
            "TimeOfTheEvent=@TimeOfTheEvent WHERE Id = @Id";
        using (SqlCommand cmd = new SqlCommand(query, con))
        {
            cmd.Parameters.AddWithValue("@Id", lblId.Text);
            cmd.Parameters.AddWithValue("@FestName", ddlFestName.SelectedValue);
            cmd.Parameters.AddWithValue("@EventName", txtEventName.Text);
            cmd.Parameters.AddWithValue("@EventDescription", txtEventDescription.Text);
            cmd.Parameters.AddWithValue("@EventCoordinator", ddlEventCoordinator.Text);
            cmd.Parameters.AddWithValue("@NumberOfParticipants", ddlNumberOfParticipants.Text);
            cmd.Parameters.AddWithValue("@DateOfTheEvent", txtDateOfTheEvent.Text);
            cmd.Parameters.AddWithValue("@TimeOfTheEvent", ddlTimeOfTheEvent.SelectedValue);
            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
            tblUpdate.Visible = false;
        }
    }

    lblMessage.Visible = true;
    lblMessage.Text = "You have sucessfully updated the Event Details!";
}
```

SQL command to execute query in the database

An SQL update query with a WHERE clause is defined that specifies the table and column names

After the details are successfully updated, table's visible proper becomes false to avoid errors

User-Friendly message is displayed to the user in a label

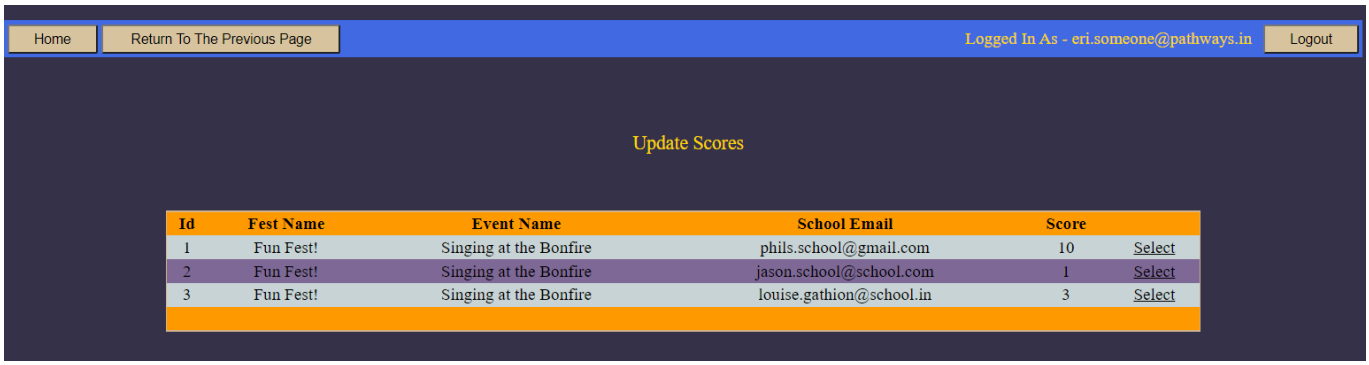
```
protected void OnRowDeleting(object sender, GridViewDeleteEventArgs e)
{
    //Get the value of column from the DataKeys using the RowIndex.
    int Id = Convert.ToInt32(gvEventDetailsUpdate.DataKeys[e.RowIndex].Values[0]);
    string constr = ConfigurationManager.ConnectionStrings["mycon"].ConnectionString;
    using (SqlConnection con = new SqlConnection(constr))
    {
        using (SqlCommand cmd = new SqlCommand("DELETE FROM EventDetails WHERE Id = @Id"))
        {
            cmd.Parameters.AddWithValue("@Id", Id);
            cmd.Connection = con;
            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
        }
    }
    this.BindGrid();
}
```

Delete button's CommandFielfd's OnRowDeleting event

An SQL delete query with a where clause which uses the 'Id' column as that cannot be changed

Method that re-binds the GridView using a data-table

Another example of the Update operation being used is when the user needs to update scores –



When the user selects a row from the GridView –


```
protected void btnUpdate_Click(object sender, EventArgs e)
{
    string constr = ConfigurationManager.ConnectionStrings["mycon"].ConnectionString;
    using (SqlConnection con = new SqlConnection(constr))
    {
        string query = "UPDATE SchoolEventScores SET Score = @Score WHERE Id = @Id";
        using (SqlCommand cmd = new SqlCommand(query, con))
        {
            cmd.Parameters.AddWithValue("@Id", lblId.Text);
            cmd.Parameters.AddWithValue("@Score", ddlScore.SelectedValue);
            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
            Response.Redirect(Request.Url.AbsoluteUri);
        }
    }
}
```

Advanced Database Operatons

Advanced SQL queries were used for calculating total scores or for populating GridView.

Sum, Count + Distinct, Group By, Order By –

```
protected void btnCalculateWinners_Click(object sender, EventArgs e)
{
    string constr = ConfigurationManager.ConnectionStrings["mycon"].ToString();
    // connection string
    SqlConnection con = new SqlConnection(constr);
    con.Open();
    string qry = "SELECT SchoolEmail, SUM(Score) As OverallScore, COUNT (DISTINCT SchoolEventScores.EventName)" +
        "As NumberOfEvents FROM SchoolEventScores WHERE FestName = '" + ddlFestName.SelectedValue + "' " +
        "GROUP BY SchoolEmail ORDER BY OverallScore DESC";
    SqlCommand com = new SqlCommand(qry, con);
    SqlDataAdapter da = new SqlDataAdapter(com);
    DataTable ds = new DataTable();
    da.Fill(ds);  //fill dataset
    gvOverallScore.DataSource = ds;    //assigning datasource to the gridview
    gvOverallScore.DataBind();

    RankSchools(ds);
}
```

Sum to add all scores from all events for a school

Count returns the total number of events

Group By to sort multiple rows into groups

Distinct to eliminate multiple records of same event as multiple schools can participate in same event

Order By to sort data in descending order dependent on their overall scores

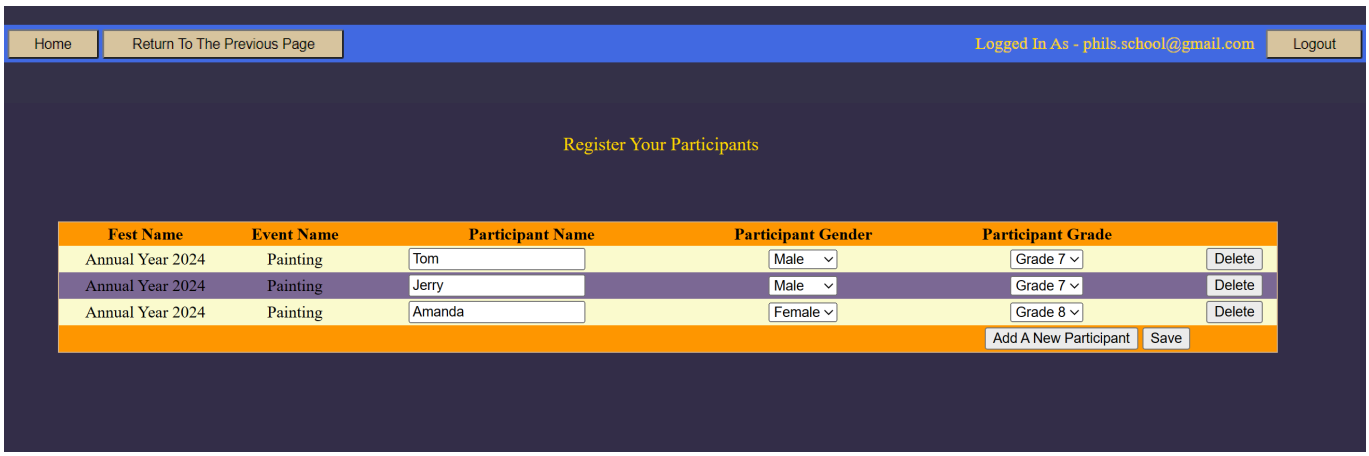
Collate query in Forgot Password Page –

```
try
{
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["mycon"].ConnectionString);
    con.Open();
    // Get records matching the Email ID. Make sure it is case sensitive and accent sensitive
    cmd = new SqlCommand("SELECT * FROM AdminLoginCredentials WHERE AdminEmail COLLATE Latin1_general_CS_AS=@AdminEmail", con);

    cmd.Parameters.AddWithValue("@AdminEmail", Convert.ToString(txtAdminEmail.Text.Trim()));
    dr = cmd.ExecuteReader();
    cmd.Dispose();
```

Collate is used to ensure the entered email is both case and accent sensitive

ViewState

ViewState was used when the application has to do bulk inserts in the database to retain the updates across postbacks, while inputting scores or registering participants.



```
private void SetInitialRow()
{
    DataTable dt = new DataTable();
    DataRow dr = null;
    dt.Columns.Add(new DataColumn("FestName", typeof(string)));
    dt.Columns.Add(new DataColumn("EventName", typeof(string)));
    dt.Columns.Add(new DataColumn("ParticipantName", typeof(string)));
    dt.Columns.Add(new DataColumn("ParticipantGender", typeof(string)));
    dt.Columns.Add(new DataColumn("ParticipantGrade", typeof(string)));
    dr = dt.NewRow();
    dr["FestName"] = Session["festname"].ToString();
    dr["EventName"] = Session["eventname"].ToString();
    dr["ParticipantName"] = string.Empty;
    dr["ParticipantGender"] = string.Empty;
    dr["ParticipantGrade"] = string.Empty;
    dt.Rows.Add(dr);

    //Store the DataTable in ViewState
    ViewState["CurrentTable"] = dt;

    gvParticipant.DataSource = dt;
    gvParticipant.DataBind();
}
```

Data is populated in a DataTable which is stored in ViewState

```
private void AddNewRowToGrid()
{
    int rowIndex = 0;
    //Check if view state is not null
    if (ViewState["CurrentTable"] != null)
    {
        DataTable dtCurrentTable = (DataTable)ViewState["CurrentTable"];  //Retrieve datatable from view state
        DataRow drCurrentRow = null;
        int rowCount = dtCurrentTable.Rows.Count;
        int MaxParticipants = Convert.ToInt32(Session["MaximumParticipants"]);
        if (rowCount < MaxParticipants)
        {
            if (dtCurrentTable.Rows.Count > 0)
            {
                for (int i = 1; i <= dtCurrentTable.Rows.Count; i++)
                {                                                   //Retrieve values inputted in textbox and dropdowns
                    TextBox txtpname = (TextBox)gvParticipant.Rows[rowIndex].Cells[2].FindControl("txtParticipantName");
                    DropDownList ddlpgender = (DropDownList)gvParticipant.Rows[rowIndex].Cells[3].FindControl("ddlParticipantGender");
                    DropDownList ddlpgrade = (DropDownList)gvParticipant.Rows[rowIndex].Cells[4].FindControl("ddlParticipantGrade");

                    drCurrentRow = dtCurrentTable.NewRow();  //Create new row and add each row into data table
                    drCurrentRow["FestName"] = Session["festname"].ToString();
                    drCurrentRow["EventName"] = Session["eventname"].ToString();

                    dtCurrentTable.Rows[i - 1]["FestName"] = Session["festname"].ToString();
                    dtCurrentTable.Rows[i - 1]["EventName"] = Session["eventname"].ToString();
                    dtCurrentTable.Rows[i - 1]["ParticipantName"] = txtpname.Text;
                    dtCurrentTable.Rows[i - 1]["ParticipantGender"] = ddlpgender.SelectedValue;
                    dtCurrentTable.Rows[i - 1]["ParticipantGrade"] = ddlpgrade.SelectedValue;

                    rowIndex++;
                }
                dtCurrentTable.Rows.Add(drCurrentRow); //Add Created Rows to the DataTable
                ViewState["CurrentTable"] = dtCurrentTable; //Save DataTable back to ViewState
                gvParticipant.DataSource = dtCurrentTable; //Bind GridView with New Row
                gvParticipant.DataBind();
            }
            SetPreviousData();
        }
        else
        {
            lblErrorMessage.Visible = true;
            lblErrorMessage.Text = "Participant Limit Reached!";
```

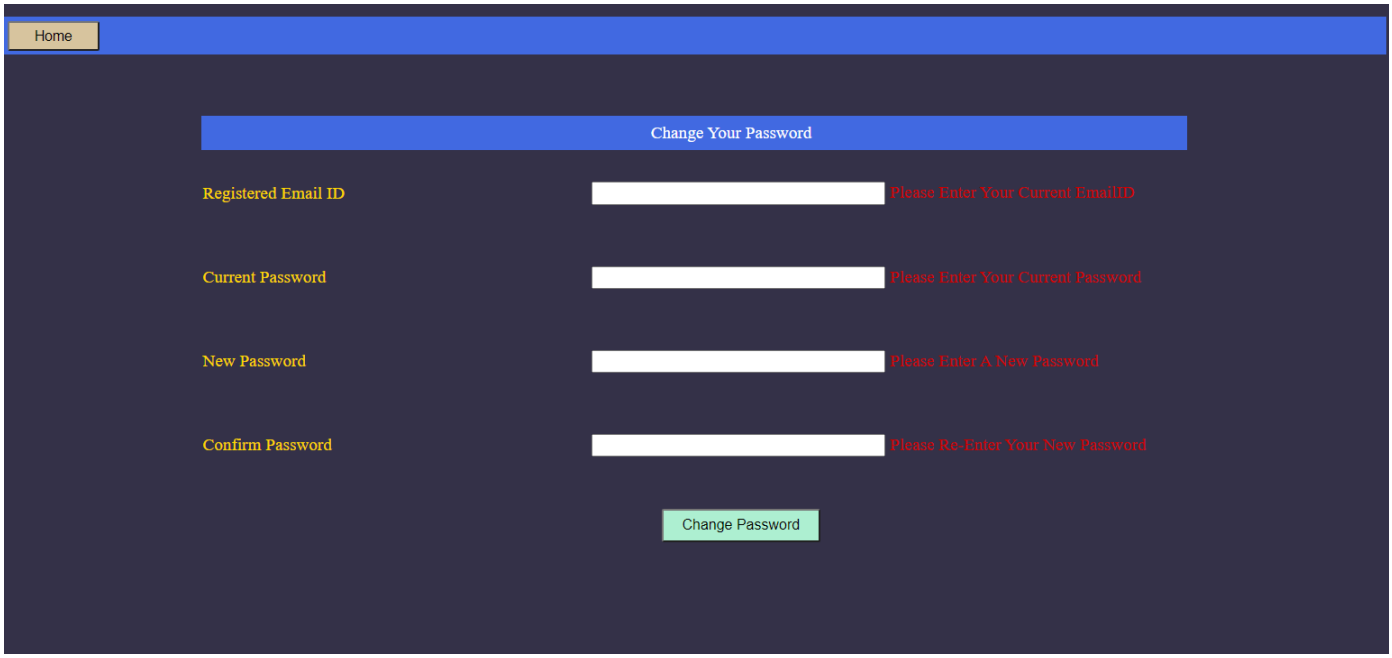
DataTable is fetched from ViewState variable

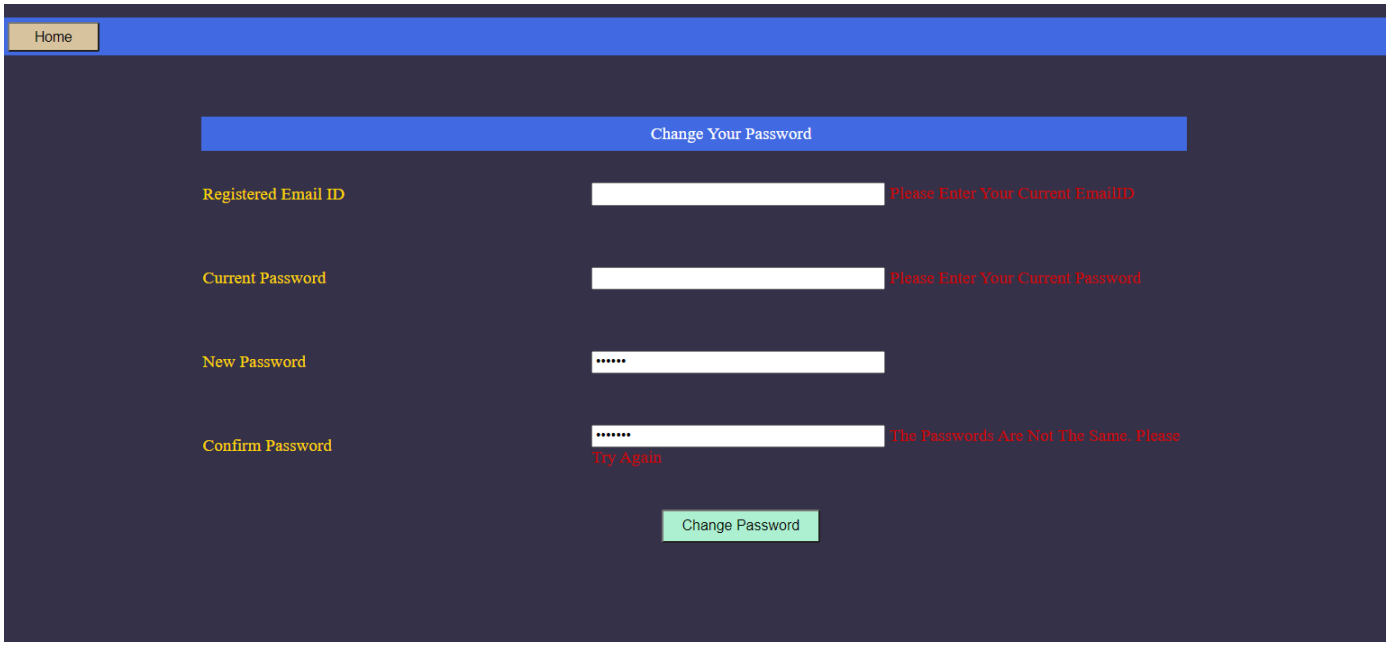
New Row is added to DataTable based on values entered in textbox and dropdowns

Adding created row to DataTable then storing DataTable to ViewState and re-binding the GridView

Client-Side Validations

To avoid a trip to the server in case of invalid values, allowing the user to make changes on-the-spot.





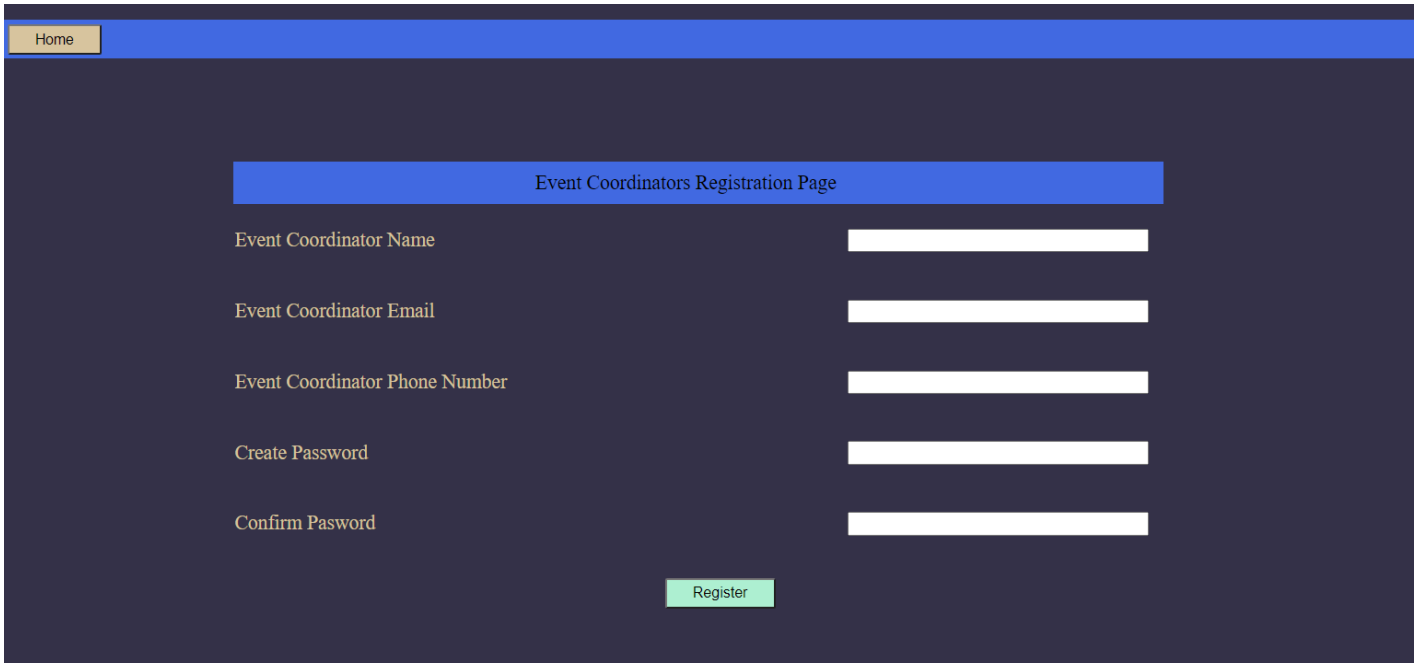
```
<td class="auto-style4">
    <asp:TextBox ID="txtConfirmPassword" Width="264px" runat="server" TextMode="Password" ></asp:TextBox>
    <asp:RequiredFieldValidator ID="RequiredFieldValidator4" Display="Dynamic" ForeColor="#CC0000"
ControlToValidate="txtConfirmPassword" runat="server" ErrorMessage="Please Re-Enter Your New Password"
        SetFocusOnError="True"></asp:RequiredFieldValidator>
  <asp:CompareValidator ID="CompareValidator2" Display="Dynamic" ForeColor="#CC0000"
ControlToCompare="txtNewPassword" ControlToValidate="txtConfirmPassword" runat="server"
        ErrorMessage="The Passwords Are Not The Same. Please Try Again" SetFocusOnError="True"></asp:CompareValidator>
</td>
```

Required Field Validator checks whether textbox is empty or not

Compare Validator checks whether values in the 2 textboxes are same

Outputs error message if values are not the same

Event Coordinator Registration Page –



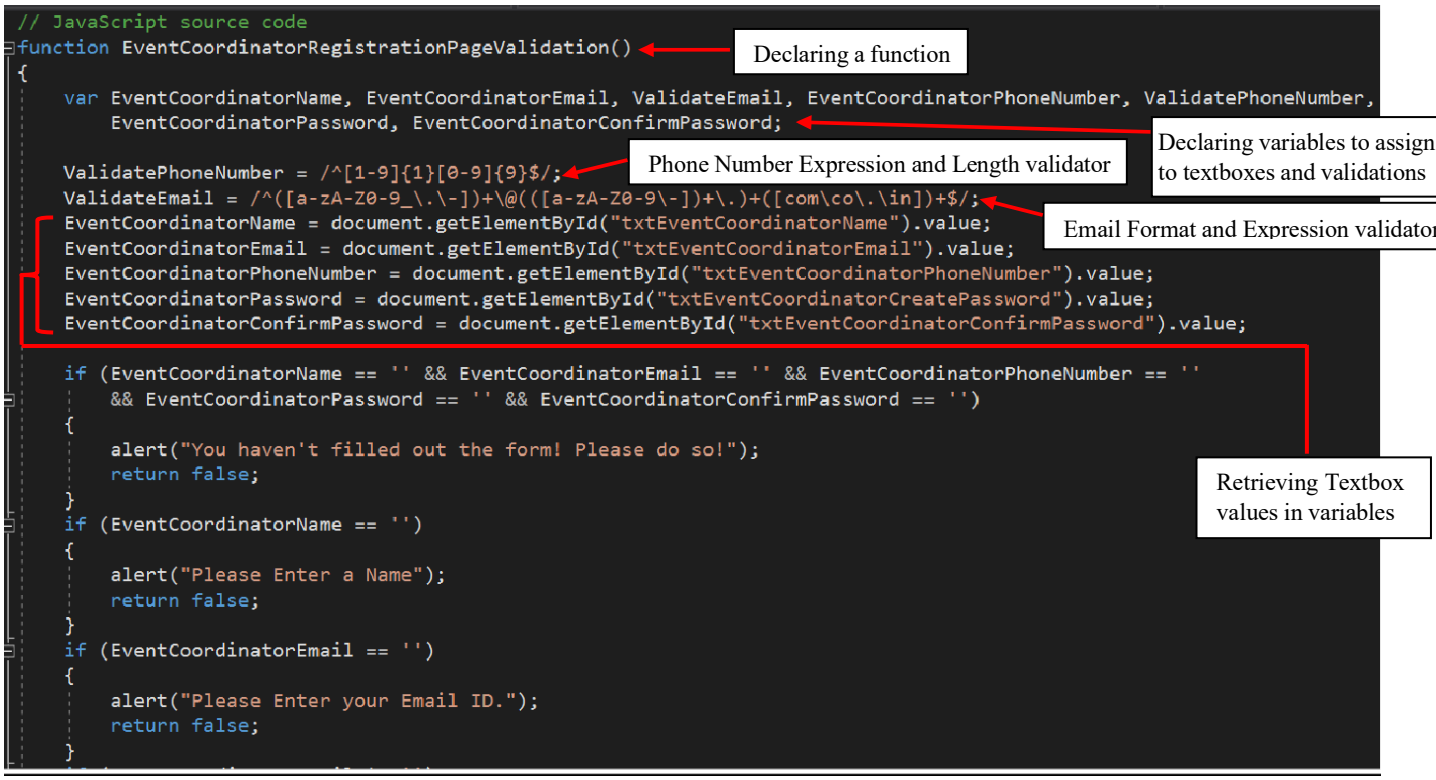
Javascript Validation

Refer to javascript file within script tags in source code

```
<script src="EventCoordinatorRegistrationPageValidation.js" type="text/javascript"></script>
```

OnClientClick property, the method is run and called back to the source code

```
<asp:Button ID="BtnRegisterEventCoordinator" runat="server" OnClientClick="return EventCoordinatorRegistrationPageValidation();"
```


// JavaScript source code
function EventCoordinatorRegistrationPageValidation()
{
var EventCoordinatorName, EventCoordinatorEmail, ValidateEmail, EventCoordinatorPhoneNumber, ValidatePhoneNumber,
EventCoordinatorPassword, EventCoordinatorConfirmPassword;
ValidatePhoneNumber = /^[1-9]{1}[0-9]{9}$/;
ValidateEmail = /^([a-zA-Z0-9_\.\-])+\@(([a-zA-Z0-9\-])+\.)+([com\co\.\in])+$/;
EventCoordinatorName = document.getElementById("txtEventCoordinatorName").value;
EventCoordinatorEmail = document.getElementById("txtEventCoordinatorEmail").value;
EventCoordinatorPhoneNumber = document.getElementById("txtEventCoordinatorPhoneNumber").value;
EventCoordinatorPassword = document.getElementById("txtEventCoordinatorCreatePassword").value;
EventCoordinatorConfirmPassword = document.getElementById("txtEventCoordinatorConfirmPassword").value;
if (EventCoordinatorName == '' && EventCoordinatorEmail == '' && EventCoordinatorPhoneNumber == ''
&& EventCoordinatorPassword == '' && EventCoordinatorConfirmPassword == '')
{
alert("You haven't filled out the form! Please do so!");
return false;
}
if (EventCoordinatorName == '')
{
alert("Please Enter a Name");
return false;
}
if (EventCoordinatorEmail == '')
{
alert("Please Enter your Email ID.");
return false;
}
Declaring a function
Declaring variables to assign to textboxes and validations
Phone Number Expression and Length validator
Email Format and Expression validator
Retrieving Textbox values in variables

```
if (EventCoordinatorEmail != '')
{
    if (!EventCoordinatorEmail.match(ValidateEmail))
    {
        alert("Invalid Email ID.");
        return false;
    }
}
if (EventCoordinatorPhoneNumber == '')
{
    alert("Please Enter Your Phone Number!");
    return false;
}
if (EventCoordinatorPhoneNumber != '')
{
    if (!EventCoordinatorPhoneNumber.match(ValidatePhoneNumber))
    {
        alert("Please Enter A Valid Mobile Number!");
        return false;
    }
}
if (EventCoordinatorPassword == '')
{
    alert("Please Create a Password.");
    return false;
}
if (EventCoordinatorPassword != '' && EventCoordinatorConfirmPassword == '')
{
    alert("Please Confirm the Password.");
    return false;
}
```

Validating Email ID and Phone Number using custom validators

If any of the listed validations are not passed by the user, a pop-up message will be displayed indicating the error

```
    if (EventCoordinatorPassword != EventCoordinatorConfirmPassword)
    {
        alert("The Passwords don't match. Please try again");
        return false;
    }

    return true;
}
```

Return false to prevent the user from proceeding further and registering themselves

Return true if the user has passed all the validation checks, allowing them to register

Server-Side Validations

Server-Side validations are used to validate the data before the data is inserted and saved in the database.

```
public override int LoginCheck()
{
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["mycon"].ToString());
    con.Open();
    SqlCommand cmd = new SqlCommand("SELECT * FROM EventCoordinatorCredentials WHERE EventCoordinatorEmail=" +
        "'" + this.UserEmail + "' and EventCoordinatorPassword='" + this.UserPassword + "'", con);
    cmd.Parameters.AddWithValue("@EventCoordinatorEmail", this.UserEmail);
    cmd.Parameters.AddWithValue("@EventCoordinatorPassword", this.UserPassword);
    SqlDataAdapter da = new SqlDataAdapter(cmd);
    DataTable dt = new DataTable();
    da.Fill(dt);
    if (dt.Rows.Count > 0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

SqlCommand is run to compare user input with existing values

If rows in DataTable dt is greater than 1, it indicates that record exists and returns 1

If no rows exist, method returns int value of 0

```
public override bool CheckEmail()
{
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["mycon"].ToString());
    con.Open();
    SqlCommand cmd = new SqlCommand("SELECT EventCoordinatorEmail FROM EventCoordinatorCredentials " +
        "WHERE EventCoordinatorEmail='" + this.UserEmail + "'", con);
    SqlDataAdapter da = new SqlDataAdapter(cmd);
    DataTable dt = new DataTable();
    da.Fill(dt);
    if (dt.Rows.Count > 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

This method checks whether email already exists during registration

If rows in DataTable dt is greater than 1, it indicates that email exists and returns true

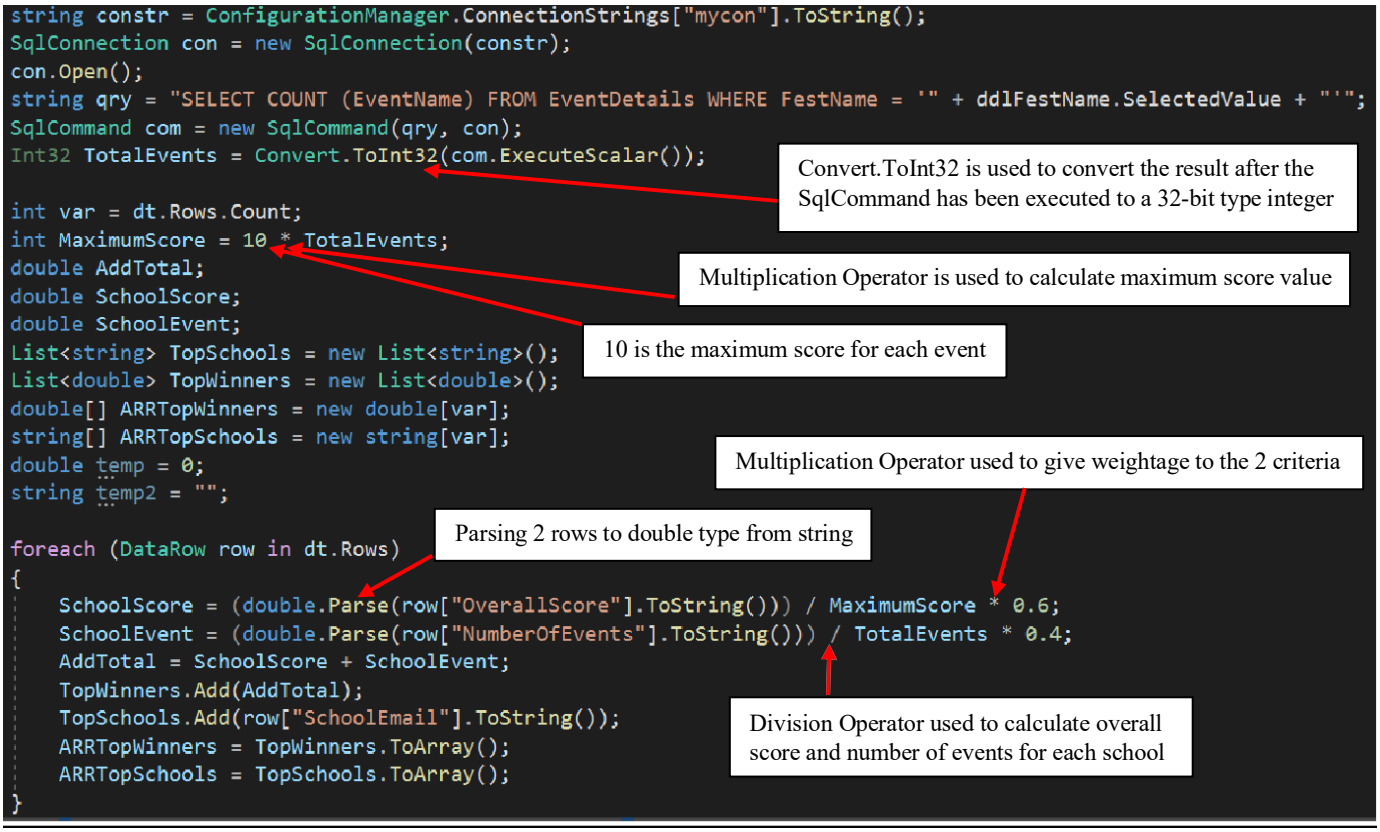
If no rows exist, method returns bool value false

Parsing and Mathematical Operations

Parsing was used to mostly convert string type values to int or vice-versa and mathematical operations were used to calculate weighted overall score of schools for different events.


```
protected void btnSubmitFeedback_Click(object sender, EventArgs e)
{
    SchoolFeedbackClass objFeedback = new SchoolFeedbackClass(int.Parse(rblRateFest.SelectedItem.Text), int.Parse(rblOrganizedFest.SelectedItem.Text),
    int Result = objFeedback.SubmitFeedback();
    if (Result > 0)
    {
        lblSubmitFeedbackMessage.Visible = true;
        lblSubmitFeedbackMessage.Text = "You have successfully submitted your feedback!";
    }
    else
    {
        lblSubmitFeedbackMessage.Visible = true;
        lblSubmitFeedbackMessage.Text = "Feedback failed to submit. Please try again!";
    }
}
```

Int.Parse is used to convert the selected value from the radio button list to int type data



Exception Handling (try-catch methods)

Try and Catch blocks have been implemented throughout the application that handle runtime exceptions and display user-friendly error to the user.

Forgot Password Page –

```
            mailclient.Send(mail);
            dr.Close();
            dr.Dispose();
            cmd.ExecuteReader();
            cmd.Dispose();
            con.Close();
            lblMessage.Visible = true;
            lblMessage.Text = "Your New Password has been sent to your email!";
            txtAdminEmail.Text = string.Empty;
        }
        else
        {
            lblMessage.Visible = true;
            lblMessage.Text = "Email ID does not exist in our database.";
            txtAdminEmail.Text = string.Empty;
            con.Close();
            return;
        }
    }
    catch (Exception ex)
    {
        lblMessage.Visible = true;
        lblMessage.Text = "An Error Occured: " + ex.Message.ToString();
    }
    finally
    {
        cmd.Dispose();
    }
```

Code to send email was written within the try brace

Catch braces used to store the error in an exception variable ex and displays the error in a label by converting the message to string

Data Structures

1D arrays were used for storing eligible grades when creating an event –


```
1 reference
public string[] selectedIndexesOfCheckBoxList(CheckBoxList cblEligibleGrades)
{
    List<string> selectedIndexes = new List<string>();

    foreach (ListItem item in cblEligibleGrades.Items)
    {
        if (item.Selected)
        {
            selectedIndexes.Add(item.Value);
        }
    }

    return selectedIndexes.ToArray();
}
```

A 1-D string array is declared

For each item in the CheckBox List, if the value is selected, it gets added to the array

The selected values in the array are then returned and stored in the database

And for storing scores and number of events for calculating top 3 winners –

```
int var = dt.Rows.Count;
int MaximumScore = 10 * TotalEvents;
double AddTotal;
double SchoolScore;
double SchoolEvent;
List<string> TopSchools = new List<string>();
List<double> TopWinners = new List<double>();
double[] ARRTopWinners = new double[var];
string[] ARRTopSchools = new string[var];
double temp = 0;
string temp2 = "";

foreach (DataRow row in dt.Rows)
{
    SchoolScore = (double.Parse(row["OverallScore"].ToString())) / MaximumScore * 0.6;
    SchoolEvent = (double.Parse(row["NumberOfEvents"].ToString())) / TotalEvents * 0.4;
    AddTotal = SchoolScore + SchoolEvent;
    TopWinners.Add(AddTotal);
    TopSchools.Add(row["SchoolEmail"].ToString());
    ARRTopWinners = TopWinners.ToArray();
    ARRTopSchools = TopSchools.ToArray();
}
```

Lists gets populated, for example, with all schools from a row in a DataTable

Parallel 1-D arrays declared to store scores (double data type) and corresponding school names (string data type) in the two arrays

Length of Array specified according to number of rows in DataTable

2D Arrays such as DataSets and DataTables were used to retrieve rows and columns from a database table and display or store values in GridViews.

```
SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["mycon"].ToString());
DataTable ecinputscores = new DataTable();
DataRow dr;

ecinputscores.Columns.Add(new DataColumn("FestName", typeof(string)));
ecinputscores.Columns.Add(new DataColumn("SchoolEmail", typeof(string)));
ecinputscores.Columns.Add(new DataColumn("EventName", typeof(string)));
ecinputscores.Columns.Add(new DataColumn("Score", typeof(string)));

foreach (GridViewRow row in gvInputScore.Rows)
{
    ecinputscores.Rows.Add();
    int i = 0;
    for (i = 0; i < row.Cells.Count; i++)
    {
        if (i == 3)
        {
            DropDownList ddlScore = (DropDownList)gvInputScore.Rows[row.RowIndex].Cells[3].FindControl("ddlScore");
            ecinputscores.Rows[row.RowIndex][i] = ddlScore.SelectedValue;
        }
        else
        {
            ecinputscores.Rows[row.RowIndex][i] = row.Cells[i].Text;
        }
    }
}
```

Declaring a new DataTable in a variable

Adding columns to the declared DataTable

Adding values to the declared columns for each row in GridView

Adding DropDown selected value in Score Column

Adding DataBound values in respective columns

Sorting the Array

Bubble sort technique was used to sort the arrays by repeatedly swapping adjacent elements in both arrays according to the scores then fetching the emails of the corresponding scores.

```
for (int i = 0; i <= ARRTopWinners.Length - 1; i++)
{
    for (int j = i + 1; j < ARRTopWinners.Length; j++)
    {
        if (ARRTopWinners[i] < ARRTopWinners[j])
        {
            temp = ARRTopWinners[i];
            ARRTopWinners[i] = ARRTopWinners[j];
            ARRTopWinners[j] = temp;

            temp2 = ARRTopSchools[i];
            ARRTopSchools[i] = ARRTopSchools[j];
            ARRTopSchools[j] = temp2;
        }
    }
}
```

If the current score (i) is smaller than the value of i+1, the 2 scores swap places using temp variable

If the scores are swapped, then so are the emails of the corresponding schools

User-Defined Methods

User-Defined methods were used to allow modular programming, making each method independent of one another, allowing for greater flexibility.

Event Coordinator Registration

```
protected void BtnRegisterEventCoordinator_Click(object sender, EventArgs e)
{
    EventCoordinatorClass objRegister = new EventCoordinatorClass();
    objRegister.EventCoordinatorName = txtEventCoordinatorName.Text;
    objRegister.EventCoordinatorPhoneNumber = txtEventCoordinatorPhoneNumber.Text;
    objRegister.UserEmail = txtEventCoordinatorEmail.Text;
    objRegister.UserPassword = txtEventCoordinatorCreatePassword.Text;

    bool Result = objRegister.CheckEmail();
    int Result2 = objRegister.AddEventCoordinatorDetails();
    if (Result == false)
    {
        if (Result2 > 0)
        {
            lblRegisterEventCoordinatorMessage.Visible = false;
            Response.Redirect("EventCoordinatorLoginPage.aspx");
        }
        else
        {
            lblRegisterEventCoordinatorMessage.Visible = true;
            lblRegisterEventCoordinatorMessage.Text = "Registration Unsuccessful. Please try again!";
        }
    }
    else
    {
        lblRegisterEventCoordinatorMessage.Visible = true;
        lblRegisterEventCoordinatorMessage.Text = "Email ID already exists.";
    }
}
```

Retrieving the return value of the 2 user-defined method

```
public int AddEventCoordinatorDetails()
{
    try
    {
        string qry = "insert into EventCoordinatorCredentials values(@EventCoordinatorName, " +
            "@EventCoordinatorEmail, @EventCoordinatorPhoneNumber, @EventCoordinatorPassword)";
        SqlCommand cmd = new SqlCommand(qry, con);
        cmd.Parameters.AddWithValue("@EventCoordinatorName", this.EventCoordinatorName);
        cmd.Parameters.AddWithValue("@EventCoordinatorEmail", this.UserEmail);
        cmd.Parameters.AddWithValue("@EventCoordinatorPhoneNumber", this.EventCoordinatorPhoneNumber);
        cmd.Parameters.AddWithValue("@EventCoordinatorPassword", this.UserPassword);
        con.Open();
        int Result = cmd.ExecuteNonQuery();
        cmd.Dispose();
        return Result;
    }
    catch (Exception e)
    {
        return 0;
    }
}
```

Method definition with no input parameter and int return type

Method returns int value stored in Result variable

Method returns 0 in case of an exception

```
public int AddEligibleGrades(string eventname, string[] eligiblegrades)
{
    try
    {
        DataTable eligibletd = new DataTable();
        DataRow dr;
        eligibletd.Columns.Add(new DataColumn("event_name", typeof(string)));
        eligibletd.Columns.Add(new DataColumn("eligible_grades", typeof(string)));

        for (int i= 0; i < eligiblegrades.Length; i++)
        {

            eligibletd.Rows.Add(eventname, eligiblegrades[i]);
        }
        con.Open();
        SqlBulkCopy objbulk = new SqlBulkCopy(con);
        objbulk.DestinationTableName = "EventEligibleGrades";
        objbulk.ColumnMappings.Add("event_name", "EventName");
        objbulk.ColumnMappings.Add("eligible_grades", "EligibleGrades");
        objbulk.WriteToServer(eligibletd);
        con.Close();
        return 1;
    }

    catch (Exception e)
    {
        return 0;
    }
}
```

Method definition with 2 string input parameters and int return type

Iterations ('for' and 'for each' loops)

For and ForEach loops were used to loop through each element in rows and arrays.

Registering Participants (For and ForEach loops)

```
foreach (GridViewRow row in gvParticipant.Rows)
{
    participantdetailstd.Rows.Add();
    int i = 0;
    for (i = 0; i < row.Cells.Count-1; i++)
    {
        if (i == 2)
        {
            TextBox txtpname = (TextBox)gvParticipant.Rows[row.RowIndex].Cells[2].FindControl("txtParticipantName");
            participantdetailstd.Rows[row.RowIndex][i] = txtpname.Text;
        }
        else if (i ==3)
        {
            DropDownList ddlpgender = (DropDownList)gvParticipant.Rows[row.RowIndex].Cells[3].FindControl("ddlParticipantGender");
            participantdetailstd.Rows[row.RowIndex][i] = ddlpgender.SelectedValue;
        }
        else if (i == 4)
        {
            DropDownList ddlpgrade = (DropDownList)gvParticipant.Rows[row.RowIndex].Cells[4].FindControl("ddlParticipantGrade");
            participantdetailstd.Rows[row.RowIndex][i] = ddlpgrade.SelectedValue;
        }
        else
        {
            participantdetailstd.Rows[row.RowIndex][i] = row.Cells[i].Text;
        }
    }
    participantdetailstd.Rows[row.RowIndex][i] = lblSchoolEmail.Text;
}
```

ForEach loop that loops through each row in GridView, retrieves the values then adds them to a DataTable

For loop that loops through the cell values for each row until row.Cells.Count-1 because the last cell value is the delete button present in Gridview

Calculating Top 3 winners (For and ForEach loops)

Foreach loop fetches total score and number of events for each school from DataTable and populates parallel arrays with weighted score and school emails.

```
foreach (DataRow row in dt.Rows)
{
    SchoolScore = (double.Parse(row["OverallScore"].ToString())) / MaximumScore * 0.6;
    SchoolEvent = (double.Parse(row["NumberOfEvents"].ToString())) / TotalEvents * 0.4;
    AddTotal = SchoolScore + SchoolEvent;
    TopWinners.Add(AddTotal);
    TopSchools.Add(row["SchoolEmail"].ToString());
    ARRTopWinners = TopWinners.ToArray();
    ARRTopSchools = TopSchools.ToArray();
}
for (int i = 0; i <= ARRTopWinners.Length - 1; i++)
{
    for (int j = i + 1; j < ARRTopWinners.Length; j++)
    {
        if (ARRTopWinners[i] < ARRTopWinners[j])
        {
            temp = ARRTopWinners[i];
            ARRTopWinners[i] = ARRTopWinners[j];
            ARRTopWinners[j] = temp;

            temp2 = ARRTopSchools[i];
            ARRTopSchools[i] = ARRTopSchools[j];
            ARRTopSchools[j] = temp2;
        }
    }
}
LblSchoolName1.Visible = true;
LblSchoolName2.Visible = true;
LblSchoolName3.Visible = true;
btnCalculateWinners.Visible = false;

LblSchoolName1.Text = ARRTopSchools[0].ToString();
LblSchoolName2.Text = ARRTopSchools[1].ToString();
LblSchoolName3.Text = ARRTopSchools[2].ToString();
```

Foreach loop to fetch total score and number of events for each school from DataTable.

Nested for loop used to implement bubble sort to find the top 3 winners based on weighted scores.

Nested and Complex If...Else statements

Nested and Complex If Else statements were used to ensure that each condition is dependent on each other and the record would only be inserted/saved if all conditions are met, otherwise, outputs an error message.

Creating Event (Nested Statements)

```
protected void BtnAddEvent_Click(object sender, EventArgs e)
{
    EventClass objAddEvent = new EventClass(ddlFestName.SelectedValue, txtEventName.Text, txtEventDescription.Text,
    ddlEventCoordinatorEmail.SelectedValue, ddlNumberOfParticipants.SelectedValue, txtDateOfTheEvent.Text, ddlTimeOfTheEvent.SelectedValue);
    bool Result = objAddEvent.CheckEventExists();
    if (Result == false)
    {
        int Result2 = objAddEvent.AddEventDetails();
        if (Result2 > 0)
        {
            string[] chkboxgrade = selectedIndexesOfCheckBoxList(cblEligibleGrades);
            int Result3 = objAddEvent.AddEligibleGrades(txtEventName.Text, chkboxgrade);

            if (Result3 > 0)
            {
                lblAddAnEvent.Visible = true;
                lblAddAnEvent.Text = "You have successfully added an event!";
            }

            else
            {
                lblAddAnEvent.Visible = true;
                lblAddAnEvent.Text = "The event was not added successfully. Please try again!";
            }
        }
        else
        {
            lblAddAnEvent.Visible = true;
            lblAddAnEvent.Text = "The event was not added successfully. Please try again!";
        }
    }
    else
    {
        lblAddAnEvent.Visible = true;
        lblAddAnEvent.Text = "The Event already exists!";
    }
}
```

IF condition that returns a bool false value, indicating that event name doesn't exist

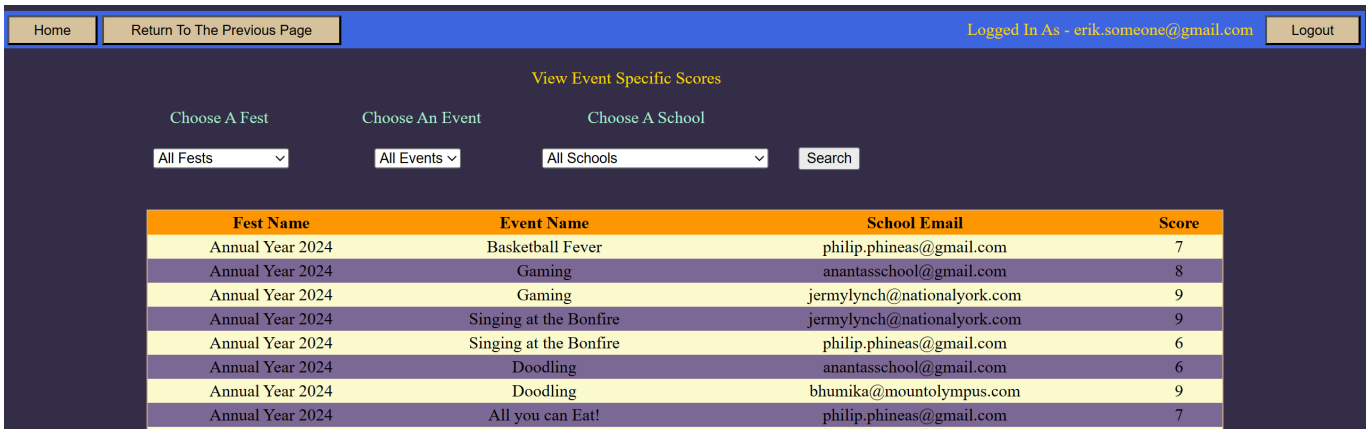
IF condition that adds the event details entered by the user into the database if value of variable Result2 is greater than 0.

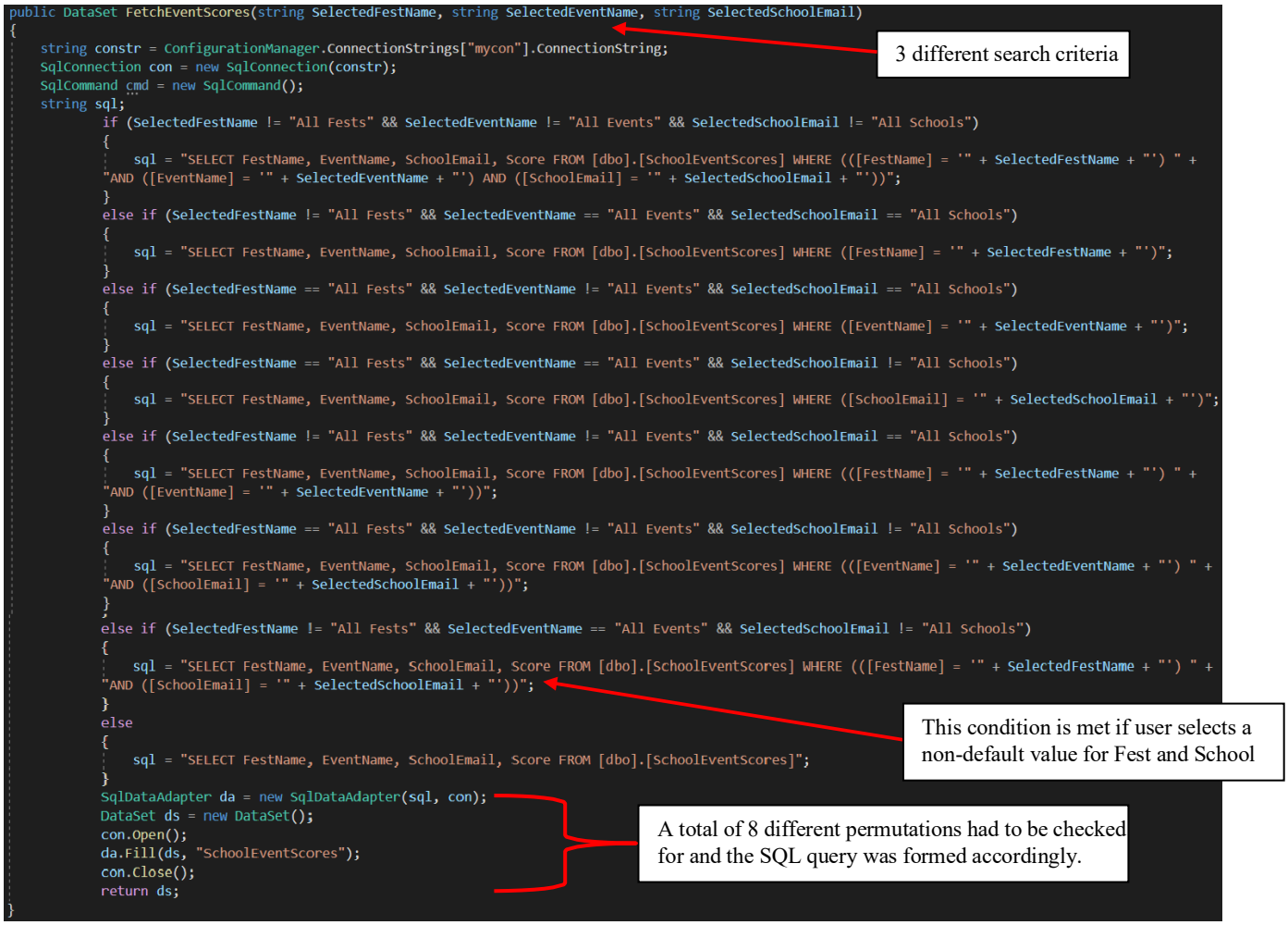
IF condition that runs only when the 1st two IF conditions are met.

IF condition which checks whether the eligible grades are added into the database.

Search Criteria (Complex statements)

Multiple IF…ELSE IF statements were used for the SQL SEARCH query based on filters selected to fetch the required records from database.






```
public DataSet FetchEventScores(string SelectedFestName, string SelectedEventName, string SelectedSchoolEmail)
{
    string constr = ConfigurationManager.ConnectionStrings["mycon"].ConnectionString;
    SqlConnection con = new SqlConnection(constr);
    SqlCommand cmd = new SqlCommand();
    string sql;
            if (SelectedFestName != "All Fests" && SelectedEventName != "All Events" && SelectedSchoolEmail != "All Schools")
            {
                sql = "SELECT FestName, EventName, SchoolEmail, Score FROM [dbo].[SchoolEventScores] WHERE (([FestName] = '" + SelectedFestName + "') " +
            "AND ([EventName] = '" + SelectedEventName + "') AND ([SchoolEmail] = '" + SelectedSchoolEmail + "'))";
            }
            else if (SelectedFestName != "All Fests" && SelectedEventName == "All Events" && SelectedSchoolEmail == "All Schools")
            {
                sql = "SELECT FestName, EventName, SchoolEmail, Score FROM [dbo].[SchoolEventScores] WHERE ([FestName] = '" + SelectedFestName + "')";
            }
            else if (SelectedFestName == "All Fests" && SelectedEventName != "All Events" && SelectedSchoolEmail == "All Schools")
            {
                sql = "SELECT FestName, EventName, SchoolEmail, Score FROM [dbo].[SchoolEventScores] WHERE ([EventName] = '" + SelectedEventName + "')";
            }
            else if (SelectedFestName == "All Fests" && SelectedEventName == "All Events" && SelectedSchoolEmail != "All Schools")
            {
                sql = "SELECT FestName, EventName, SchoolEmail, Score FROM [dbo].[SchoolEventScores] WHERE ([SchoolEmail] = '" + SelectedSchoolEmail + "')";
            }
            else if (SelectedFestName != "All Fests" && SelectedEventName != "All Events" && SelectedSchoolEmail == "All Schools")
            {
                sql = "SELECT FestName, EventName, SchoolEmail, Score FROM [dbo].[SchoolEventScores] WHERE (([FestName] = '" + SelectedFestName + "') " +
            "AND ([EventName] = '" + SelectedEventName + "'))";
            }
            else if (SelectedFestName == "All Fests" && SelectedEventName != "All Events" && SelectedSchoolEmail != "All Schools")
            {
                sql = "SELECT FestName, EventName, SchoolEmail, Score FROM [dbo].[SchoolEventScores] WHERE (([EventName] = '" + SelectedEventName + "') " +
            "AND ([SchoolEmail] = '" + SelectedSchoolEmail + "'))";
            }
            else if (SelectedFestName != "All Fests" && SelectedEventName == "All Events" && SelectedSchoolEmail != "All Schools")
            {
                sql = "SELECT FestName, EventName, SchoolEmail, Score FROM [dbo].[SchoolEventScores] WHERE (([FestName] = '" + SelectedFestName + "') " +
            "AND ([SchoolEmail] = '" + SelectedSchoolEmail + "'))";
            }
            else
            {
                sql = "SELECT FestName, EventName, SchoolEmail, Score FROM [dbo].[SchoolEventScores]";
            }
            SqlDataAdapter da = new SqlDataAdapter(sql, con);
            DataSet ds = new DataSet();
            con.Open();
            da.Fill(ds, "SchoolEventScores");
            con.Close();
            return ds;
}
```

OOPs concept (Polymorphism, Encapsulation, Inheritance)

Polymorphism

Polymorphism was used to define multiple implementations of methods with the same method name. Both method overriding and method overloading techniques were used in the web-application.

```
public SchoolFeedbackClass(string _FestName, string _SchoolEmail)
{
    this.FestName = _FestName;
    this.SchoolEmail = _SchoolEmail;
}
public SchoolFeedbackClass(string FestName, string SchoolEmail, int RateFest, int RateOrganizedFest,
    string RateFutureFest, string SchoolLikes, string SchoolDislikes, string GeneralFeedback)
{
    this.FestName = FestName;
    this.SchoolEmail = SchoolEmail;
    this.RateFest = RateFest;
    this.RateOrganizedFest = RateOrganizedFest;
    this.RateFutureFest = RateFutureFest;
    this.SchoolLikes = SchoolLikes;
    this.SchoolDislikes = SchoolDislikes;
    this.GeneralFeedback = GeneralFeedback;
}
```

Defining overloaded constructors with the same name but different set of parameters

```
protected void btnSubmitFeedback_Click(object sender, EventArgs e)
{
    SchoolFeedbackClass objFeedback = new SchoolFeedbackClass(ddlFestName.SelectedValue, lblSchoolEmail.Text,
    int.Parse(rblRateFest.SelectedItem.Text), int.Parse(rblOrganizedFest.SelectedItem.Text),
    rblFutureFests.SelectedItem.Text, txtSchoolLikes.Text, txtSchoolDislikes.Text, txtSchoolThoughts.Text);

    int Result = objFeedback.SubmitFeedback();
    if (Result > 0)
    {
        lblMessage.Visible = true;
        lblMessage.Text = "You have successfully submitted your feedback!";
    }
    else
    {
        lblMessage.Visible = true;
        lblMessage.Text = "Feedback failed to submit. Please try again!";
    }
}
```

SchoolFeedbackClass constructor is called with 8 parameters to add school feedback into the database

```
private void Search()
{
    SchoolFeedbackClass objSchoolSearch = new SchoolFeedbackClass(ddlFestName.SelectedValue,
    ddlSchoolEmail.SelectedValue);
    DataSet ds = objSchoolSearch.FetchFeedbackDetails();
    if (ds.Tables[0].Rows.Count > 0)
    {
        lblErrorMessage.Visible = false;
        gvViewFeedback.Visible = true;
        gvViewFeedback.DataSource = ds;
        gvViewFeedback.DataBind();
    }
    else
    {
        gvViewFeedback.Visible = false;
        lblErrorMessage.Visible = true;
        lblErrorMessage.Text = "No Data was available for the selected search criteria.";
    }
}
```

Another constructor of SchoolFeedbackClass is called with 2 parameters to display results based on search criteria

```
public int SubmitFeedback()
{
    try
    {
        string qry = "insert into SchoolFeedback values(@FestName, @SchoolEmail, @FestRate," +
            "@OrganizationOfFestRate, @ReturnRate, @LikesAboutFestOrEvent, " +
            "@DislikesAboutFestOrEvent, @GeneralThoughts)";
        SqlCommand cmd = new SqlCommand(qry, con);
        cmd.Parameters.AddWithValue("@FestName", this.FestName);
        cmd.Parameters.AddWithValue("@SchoolEmail", this.UserEmail);
        cmd.Parameters.AddWithValue("@FestRate", this.RateFest);
        cmd.Parameters.AddWithValue("@OrganizationOfFestRate", this.RateOrganizedFest);
        cmd.Parameters.AddWithValue("@ReturnRate", this.RateFutureFest);
        cmd.Parameters.AddWithValue("@LikesAboutFestOrEvent", this.SchoolLikes);
        cmd.Parameters.AddWithValue("@DislikesAboutFestOrEvent", this.SchoolDislikes);
        cmd.Parameters.AddWithValue("@GeneralThoughts", this.GeneralFeedback);

        con.Open();
        int Result = cmd.ExecuteNonQuery();
        cmd.Dispose();
        con.Close();
        return Result;
    }
    catch (Exception e)
    {
        return 0;
    }
}
```

The values passed in the constructors are used to insert school feedback into the database

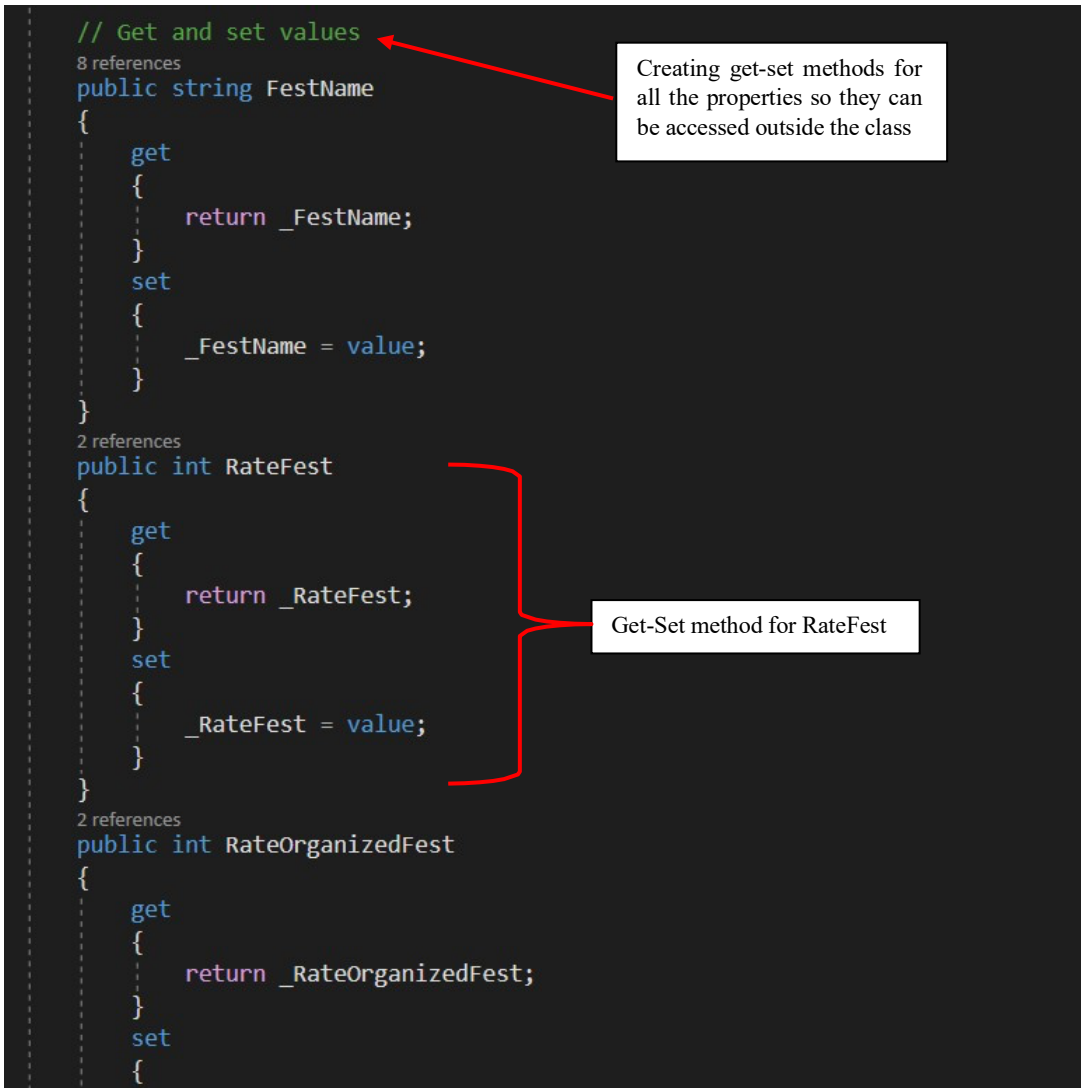
Encapsulation

Encapsulation has been used throughout the code for all classes. Private state is maintained by encapsulating classes with private properties to restrict direct access and manipulation from outside the class. However, the properties are made accessible by declaring public get/set methods for each of them. This allows for abstraction as any changes made within the class have minimal impact on the code outside.

```
public class SchoolFeedbackClass
{
    //Declaring Feedback Variables
    private string _SchoolEmail;
    private string _FestName;
    private int _RateFest;
    private int _RateOrganizedFest;
    private string _RateFutureFest;
    private string _SchoolLikes;
    private string _SchoolDislikes;
    private string _GeneralFeedback;
```

Class is SchoolFeedbackClass

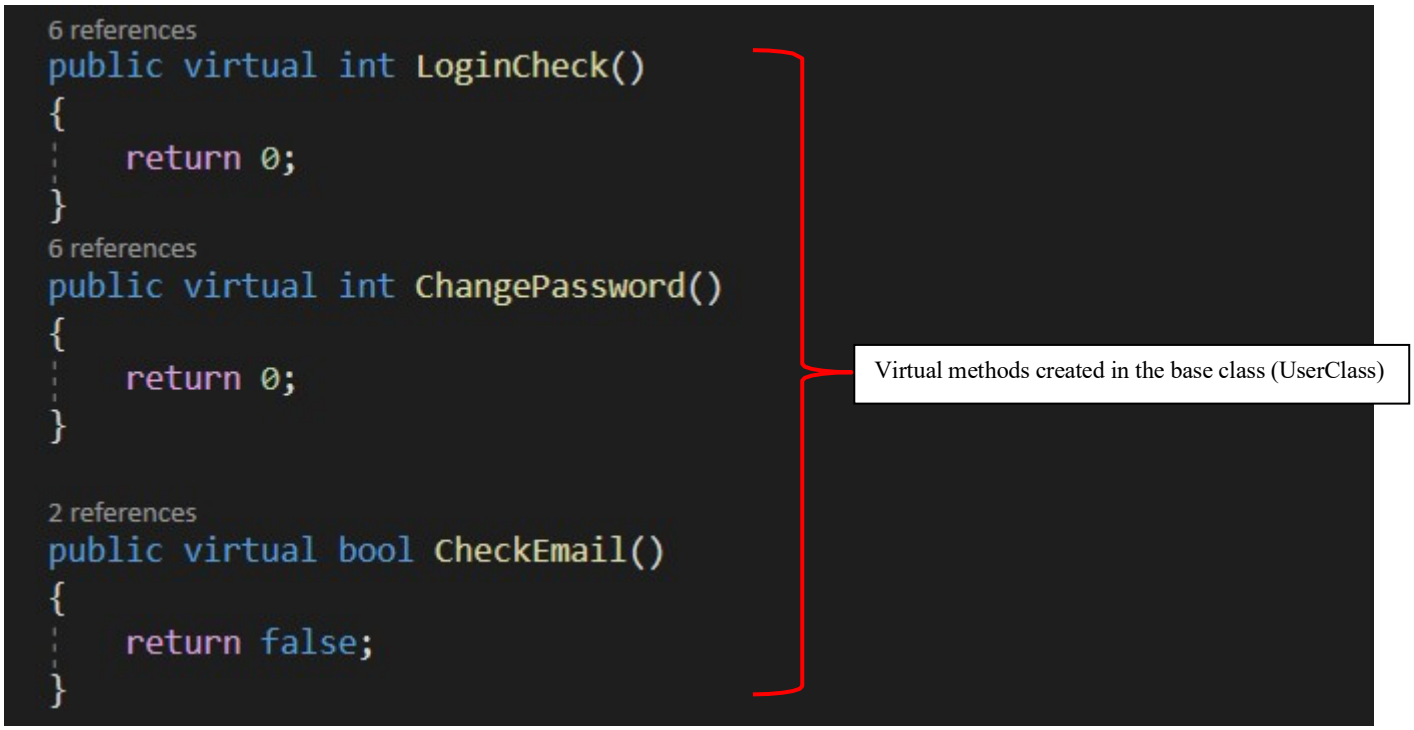
Creating private properties within the class with their associated data types



Inheritance

Inheritance was used as it allows the reusability of code by sharing common code amongst several derived classes. It also reduces the redundancy of the code and provides extensibility of code by overriding the base class functionality.


public class UserClass
{
protected string _UserEmail;
protected string _UserPassword;
protected string _CurrentPassword;
protected string _NewPassword;
20 references
public string UserEmail
{
get
{
return _UserEmail;
}
set
{
_UserEmail = value;
}
}
13 references
public string UserPassword
{
get
{
return _UserPassword;
}
set
{
_UserPassword = value;
}
}
Base Class is UserClass
Properties in the base class are protected so they can be accessed by the derived classes


6 references
public virtual int LoginCheck()
{
return 0;
}
6 references
public virtual int ChangePassword()
{
return 0;
}
2 references
public virtual bool CheckEmail()
{
return false;
}
Virtual methods created in the base class (UserClass)

```
public class SchoolClass : UserClass
{
    //Declaring School Registration Variables
    private string _SchoolName;
    private string _SchoolAddress;
    private string _SchoolContactPerson;
    private string _SchoolPhoneNumber;
    private string _EventName;

    // Get and set values
    public string SchoolName
    {
        get
        {
            return _SchoolName;
        }
        set
        {
            _SchoolName = value;
        }
    }
```

SchoolClass inherits UserClass

Private Properties that are solely a part of the SchoolClass

Check Email Method –

```
public override bool CheckEmail()
{
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["mycon"].ToString());
    con.Open();
    SqlCommand cmd = new SqlCommand("SELECT SchoolEmail FROM SchoolCredentials WHERE SchoolEmail='" + this.UserEmail + "'", con);
    SqlDataAdapter da = new SqlDataAdapter(cmd);
    DataTable dt = new DataTable();
    da.Fill(dt);
    if (dt.Rows.Count > 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

SchoolClass overrides the virtual method from UserClass

```
protected void BtnRegister_Click(object sender, EventArgs e)
{
    SchoolClass objSchoolRegister = new SchoolClass();
    objSchoolRegister.SchoolName = txtSchoolName.Text;
    objSchoolRegister.SchoolAddress = txtSchoolAddress.Text;
    objSchoolRegister.SchoolPhoneNumber = txtSchoolPhoneNumber.Text;
    objSchoolRegister.SchoolContactPerson = txtSchoolContactPersonName.Text;
    objSchoolRegister.UserEmail = txtSchoolContactEmail.Text;
    objSchoolRegister.UserPassword = txtCreatePassword.Text;

    bool Result = objSchoolRegister.CheckEmail();
    if (Result == false)
    {
        int Result2 = objSchoolRegister.AddSchoolDetails();
        if (Result2 > 0)
        {
            lblErrorMessage.Visible = false;
            Response.Redirect("SchoolLoginPage.aspx");
```

Calling method by instantiating the object of SchoolClass

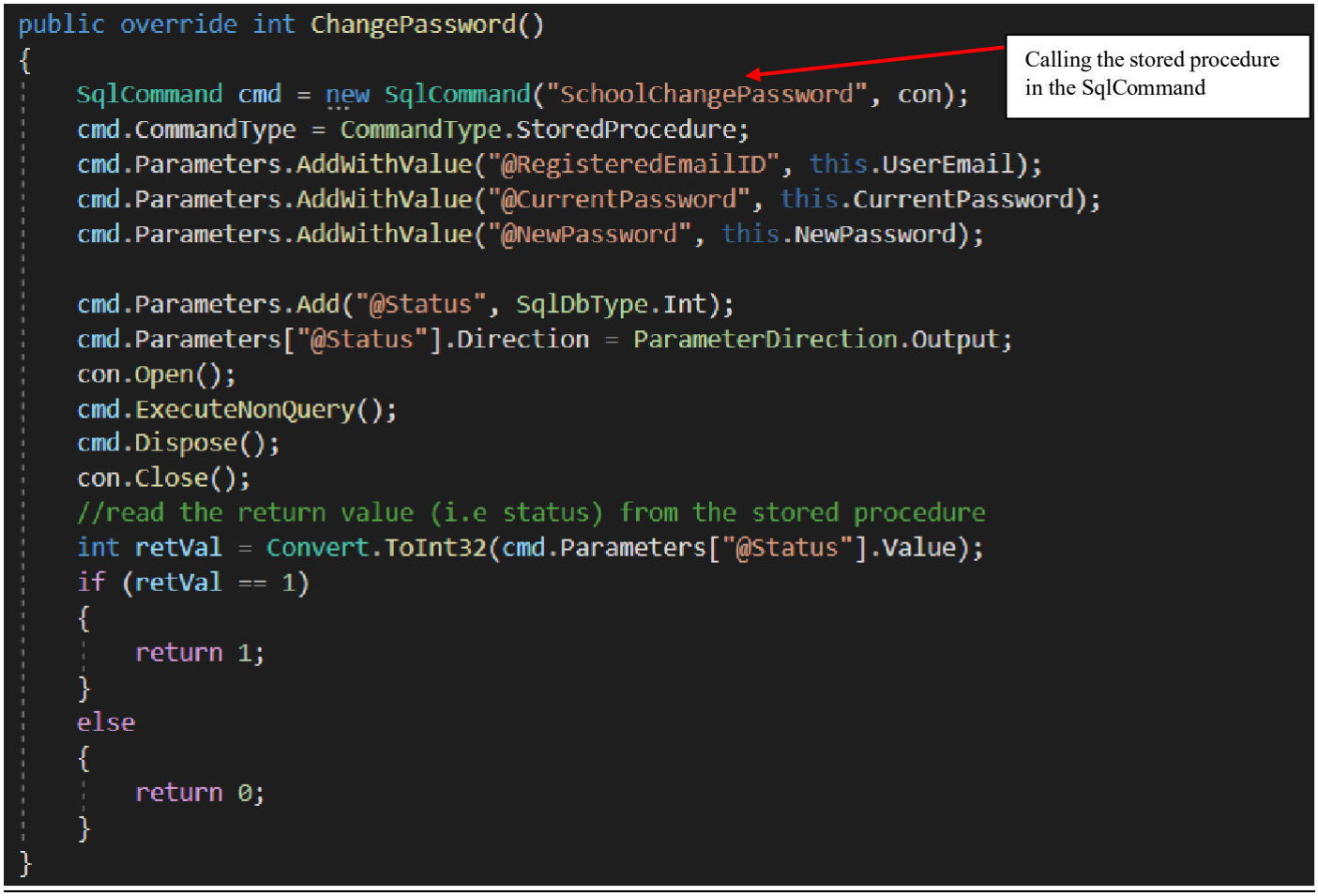
Stored Procedure

Stored procedures were used for changing the user password. A stored procedure also improves performance of the database operations and stores and executes SQL queries directly on the server.

```
CREATE PROCEDURE AdminChangePassword
(
        @RegisteredEmailID              NVARCHAR(50),
        @CurrentPassword                NVARCHAR(50),
        @NewPassword                    NVARCHAR(50),
        @Status                         int OUTPUT
)
AS
BEGIN

IF EXISTS(SELECT * FROM AdminLoginCredentials WHERE AdminEmail COLLATE Latin1_general_CS_AS=@RegisteredEmailID AND [AdminPassword]
COLLATE Latin1_general_CS_AS=@CurrentPassword)
    BEGIN
            UPDATE AdminLoginCredentials SET [AdminPassword]=@NewPassword WHERE AdminEmail=@RegisteredEmailID
            SET @Status=1
    END
ELSE
    BEGIN
            SET @Status=0
    END
END
RETURN @Status
```

Creating custom parameters

Returns the value of Status to the method in the class




```
public override int ChangePassword()
{
    SqlCommand cmd = new SqlCommand("SchoolChangePassword", con);
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.Parameters.AddWithValue("@RegisteredEmailID", this.UserEmail);
    cmd.Parameters.AddWithValue("@CurrentPassword", this.CurrentPassword);
    cmd.Parameters.AddWithValue("@NewPassword", this.NewPassword);

    cmd.Parameters.Add("@Status", SqlDbType.Int);
    cmd.Parameters["@Status"].Direction = ParameterDirection.Output;
    con.Open();
    cmd.ExecuteNonQuery();
    cmd.Dispose();
    con.Close();
    //read the return value (i.e status) from the stored procedure
    int retVal = Convert.ToInt32(cmd.Parameters["@Status"].Value);
    if (retVal == 1)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

StringBuilder, StmpClient and MailClient

These 3 techniques were used to send email with new password. StringBuilder allows repeated modifications of a string and doesn't create a new object in the memory as it dynamically expands it to accommodate the modified string. Mail Client and StmpClient was used to send email using an STMP server.

```
if (dr.HasRows)
{
    dr.Read();
    //Generate unique code
    UniqueCode = Convert.ToString(System.Guid.NewGuid());
    //Updating the AdminPassword column in the database for the Admin Email that the user has entered in the Textbox
    cmd = new SqlCommand("UPDATE AdminLoginCredentials set AdminPassword=@UniqueCode WHERE AdminEmail=@AdminEmail", con);
    cmd.Parameters.AddWithValue("@UniqueCode", UniqueCode);
    cmd.Parameters.AddWithValue("@AdminEmail", txtAdminEmail.Text.Trim());

    StringBuilder strBody = new StringBuilder();
    strBody.Append("Please Find Your New Password written in the subject of this email. It is recommended that you change your password immediately.");
    //                                                          Sender Gmail          Reciever of the email       Subject          Body
    System.Net.Mail.MailMessage mail = new System.Net.Mail.MailMessage("isweathebot@gmail.com", dr["AdminEmail"].ToString(), UniqueCode, strBody.ToString());
    //pasing the Gmail credentials to send the email
    //                                                                    Sender Gmail           Sender Password
    System.Net.NetworkCredential mailAuthenticaion = new System.Net.NetworkCredential("isweathebot@gmail.com", "ISWEAbotresetyourpassword");
    //                                                                  Port Number
    System.Net.Mail.SmtpClient mailclient = new System.Net.Mail.SmtpClient("smtp.gmail.com", 587);
    mailclient.EnableSsl = true;
    mailclient.UseDefaultCredentials = false;
    mailclient.Credentials = mailAuthenticaion;
    mail.IsBodyHtml = true;
    mailclient.Send(mail);
    dr.Close();
    dr.Dispose();
    cmd.ExecuteReader();
    cmd.Dispose();
    con.Close();
    lblMessage.Visible = true;
    lblMessage.Text = "Your New Password has been sent to your email!";
    txtAdminEmail.Text = string.Empty;
```

Generates a 32-digit Unique Code using Guild.NewGuild

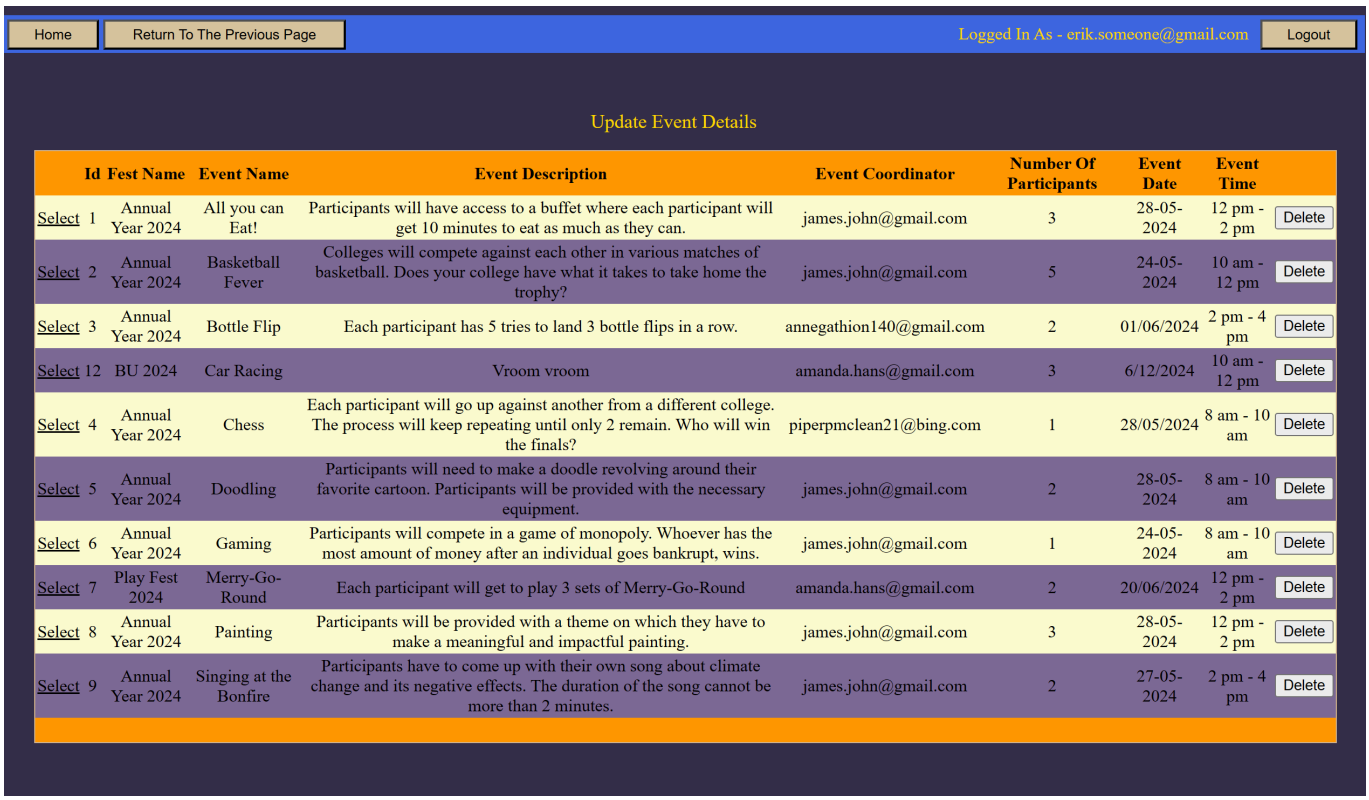
Updates the Password column with the Unique Code for that user

Uses StmpClient and MailClient to send the email with the new password

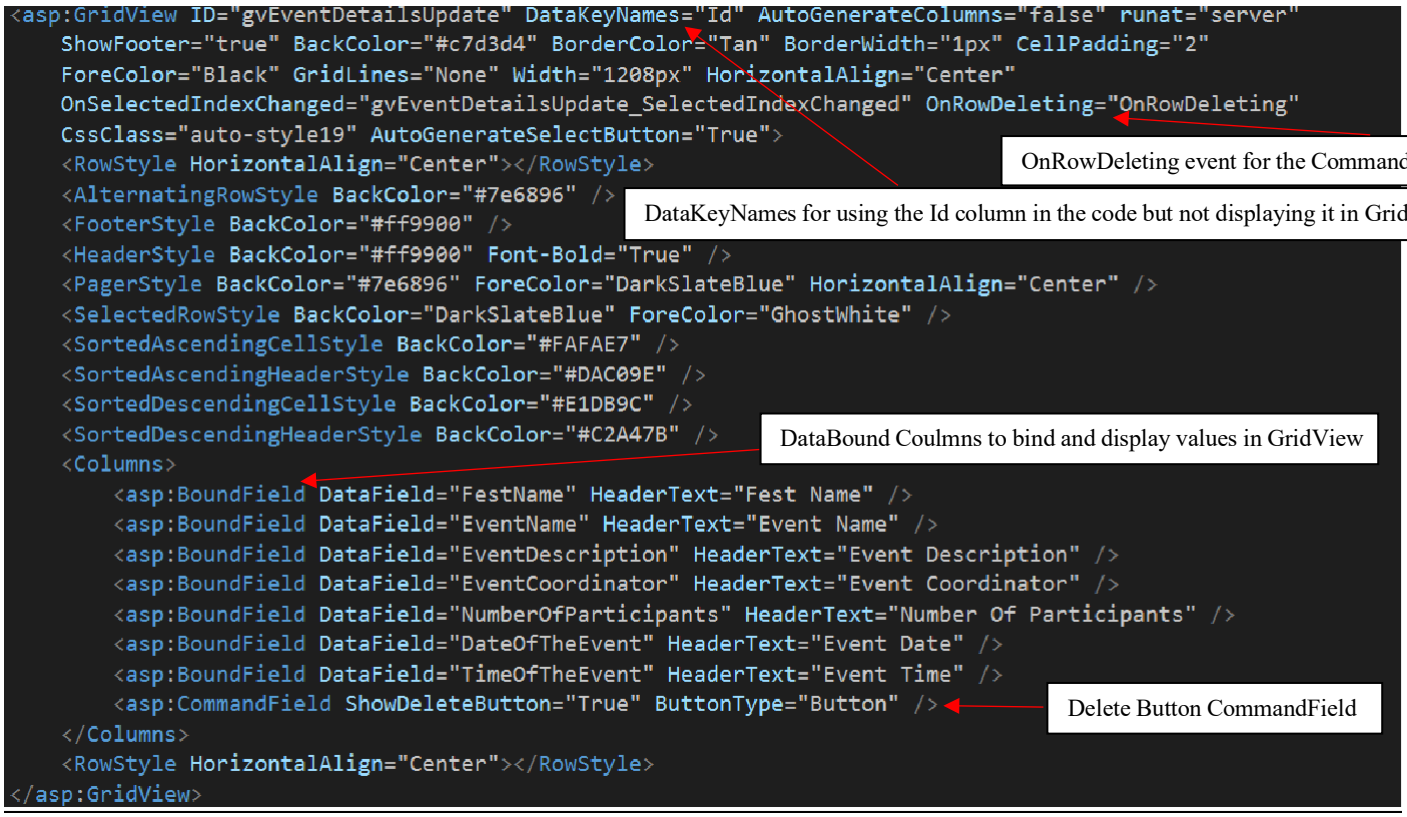
StringBuilder to concat the content of the mail

ASP.NET GridView controls

The main functionalities in the application requires displaying values from data-tables to GridViews. Hence, many GridView controls and techniques have been used in the application such as DataKeyNames, RowDatBound, RowCreation, ItemTemplates and more.



	Id	Fest Name	Event Name	Event Description	Event Coordinator	Number Of Participants	Event Date	Event Time	
Select	1	Annual Year 2024	All you can Eat!	Participants will have access to a buffet where each participant will get 10 minutes to eat as much as they can.	james.john@gmail.com	3	28-05-2024	12 pm - 2 pm	Delete
Select	2	Annual Year 2024	Basketball Fever	Colleges will compete against each other in various matches of basketball. Does your college have what it takes to take home the trophy?	james.john@gmail.com	5	24-05-2024	10 am - 12 pm	Delete
Select	3	Annual Year 2024	Bottle Flip	Each participant has 5 tries to land 3 bottle flips in a row.	annegathion140@gmail.com	2	01/06/2024	2 pm - 4 pm	Delete
Select	12	BU 2024	Car Racing	Vroom vroom	amanda.hans@gmail.com	3	6/12/2024	10 am - 12 pm	Delete
Select	4	Annual Year 2024	Chess	Each participant will go up against another from a different college. The process will keep repeating until only 2 remain. Who will win the finals?	piperpmclean21@bing.com	1	28/05/2024	8 am - 10 am	Delete
Select	5	Annual Year 2024	Doodling	Participants will need to make a doodle revolving around their favorite cartoon. Participants will be provided with the necessary equipment.	james.john@gmail.com	2	28-05-2024	8 am - 10 am	Delete
Select	6	Annual Year 2024	Gaming	Participants will compete in a game of monopoly. Whoever has the most amount of money after an individual goes bankrupt, wins.	james.john@gmail.com	1	24-05-2024	8 am - 10 am	Delete
Select	7	Play Fest 2024	Merry-Go-Round	Each participant will get to play 3 sets of Merry-Go-Round	amanda.hans@gmail.com	2	20/06/2024	12 pm - 2 pm	Delete
Select	8	Annual Year 2024	Painting	Participants will be provided with a theme on which they have to make a meaningful and impactful painting.	james.john@gmail.com	3	28-05-2024	12 pm - 2 pm	Delete
Select	9	Annual Year 2024	Singing at the Bonfire	Participants have to come up with their own song about climate change and its negative effects. The duration of the song cannot be more than 2 minutes.	james.john@gmail.com	2	27-05-2024	2 pm - 4 pm	Delete




```
<asp:GridView ID="gvEventDetailsUpdate" DataKeyNames="Id" AutoGenerateColumns="false" runat="server"
    ShowFooter="true" BackColor="#c7d3d4" BorderColor="Tan" BorderWidth="1px" CellPadding="2"
    ForeColor="Black" GridLines="None" Width="1208px" HorizontalAlign="Center"
    OnSelectedIndexChanged="gvEventDetailsUpdate_SelectedIndexChanged" OnRowDeleting="OnRowDeleting"
    CssClass="auto-style19" AutoGenerateSelectButton="True">
    <RowStyle HorizontalAlign="Center"></RowStyle>
    <AlternatingRowStyle BackColor="#7e6896" />
    <FooterStyle BackColor="#ff9900" />
    <HeaderStyle BackColor="#ff9900" Font-Bold="True" />
    <PagerStyle BackColor="#7e6896" ForeColor="DarkSlateBlue" HorizontalAlign="Center" />
    <SelectedRowStyle BackColor="DarkSlateBlue" ForeColor="GhostWhite" />
    <SortedAscendingCellStyle BackColor="#FAFAE7" />
    <SortedAscendingHeaderStyle BackColor="#DAC09E" />
    <SortedDescendingCellStyle BackColor="#E1DB9C" />
    <SortedDescendingHeaderStyle BackColor="#C2A47B" />
    <Columns>
        <asp:BoundField DataField="FestName" HeaderText="Fest Name" />
        <asp:BoundField DataField="EventName" HeaderText="Event Name" />
        <asp:BoundField DataField="EventDescription" HeaderText="Event Description" />
        <asp:BoundField DataField="EventCoordinator" HeaderText="Event Coordinator" />
        <asp:BoundField DataField="NumberOfParticipants" HeaderText="Number Of Participants" />
        <asp:BoundField DataField="DateOfTheEvent" HeaderText="Event Date" />
        <asp:BoundField DataField="TimeOfTheEvent" HeaderText="Event Time" />
        <asp:CommandField ShowDeleteButton="True" ButtonType="Button" />
    </Columns>
    <RowStyle HorizontalAlign="Center"></RowStyle>
</asp:GridView>
```

```
protected void OnRowDeleting(object sender, GridViewDeleteEventArgs e)
{
    //Get the value of column from the DataKeys using the RowIndex.
    int Id = Convert.ToInt32(gvEventDetailsUpdate.DataKeys[e.RowIndex].Values[0]);
    string constr = ConfigurationManager.ConnectionStrings["mycon"].ConnectionString;
    using (SqlConnection con = new SqlConnection(constr))
    {
        using (SqlCommand cmd = new SqlCommand("DELETE FROM EventDetails WHERE Id = @Id"))
        {
            cmd.Parameters.AddWithValue("@Id", Id);
            cmd.Connection = con;
            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
        }
    }
    this.BindGrid();
}
```

OnRowDeleting event, the row for which the user clicks the delete button gets deleted from the database using the Id column

```
protected void gvParticipant_RowCreated(object sender, GridViewRowEventArgs e)
{
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
        DataTable dt = (DataTable)ViewState["CurrentTable"];
        LinkButton lb = (LinkButton)e.Row.FindControl("LinkButton1");
        if (lb != null)
        {
            if (dt.Rows.Count > 1)
            {
                if (e.Row.RowIndex == dt.Rows.Count - 1)
                {
                    lb.Visible = false;
                }
            }
            else
            {
                lb.Visible = false;
            }
        }
    }
}
```

RowCreated event is used when creating each row in the GridView control

Registering Participants Source Code

```
<asp:TemplateField HeaderText="Participant Grade">
            <ItemTemplate>
                <asp:DropDownList ID="ddlParticipantGrade" runat="server">
                <asp:ListItem>Grade 6</asp:ListItem>
                <asp:ListItem>Grade 7</asp:ListItem>
                <asp:ListItem>Grade 8</asp:ListItem>
                <asp:ListItem>Grade 9</asp:ListItem>
                <asp:ListItem>Grade 10</asp:ListItem>
                <asp:ListItem>Grade 11</asp:ListItem>
                <asp:ListItem>Grade 12</asp:ListItem>
                </asp:DropDownList>
            </ItemTemplate>
    <FooterStyle HorizontalAlign="Right" />
            <FooterTemplate>
    <asp:Button ID="btnAddANewParticipant" runat="server" Text="Add A New Participant"
                onclick="btnAddANewParticipant_Click" />
    <asp:Button ID="btnSave" runat="server" Text="Save"
                onclick="btnSave_Click" /> </FooterTemplate>
        </asp:TemplateField>
```

ItemTemplate fields to make controls editable

FooterTemplate to display buttons to add participants or save data

Manipulation of GridView and Calender Control

GridView and Calendar Controls were manipulated. When user selects a row in GridView, the background colour changes. Similarly, when the user selects a date from the calendar, the background colour for that date changes as well. This allows the user to easily distinguish the selected value from the rest of the values/elements.

```
            <asp:Calendar ID="Calendar1" runat="server" BackColor="#FF9912" OnSelectionChanged="Calendar1_SelectionChanged">
    <SelectedDayStyle BackColor="#343148" Font-Bold="True" />
    <SelectorStyle BackColor="#FFCC66" />
    <TodayDayStyle BackColor="#03A89E" ForeColor="Black" />
    <OtherMonthDayStyle ForeColor="#006B38" />
    <NextPrevStyle Font-Size="9pt" ForeColor="#FFFFCC" />
            </asp:Calendar>
<asp:TextBox ID="txtDateOfTheEvent" runat="server" ReadOnly="True"></asp:TextBox>
```

Different BackColor and ForeColor's used for ease of access in terms of displaying present date, next month's dates etc

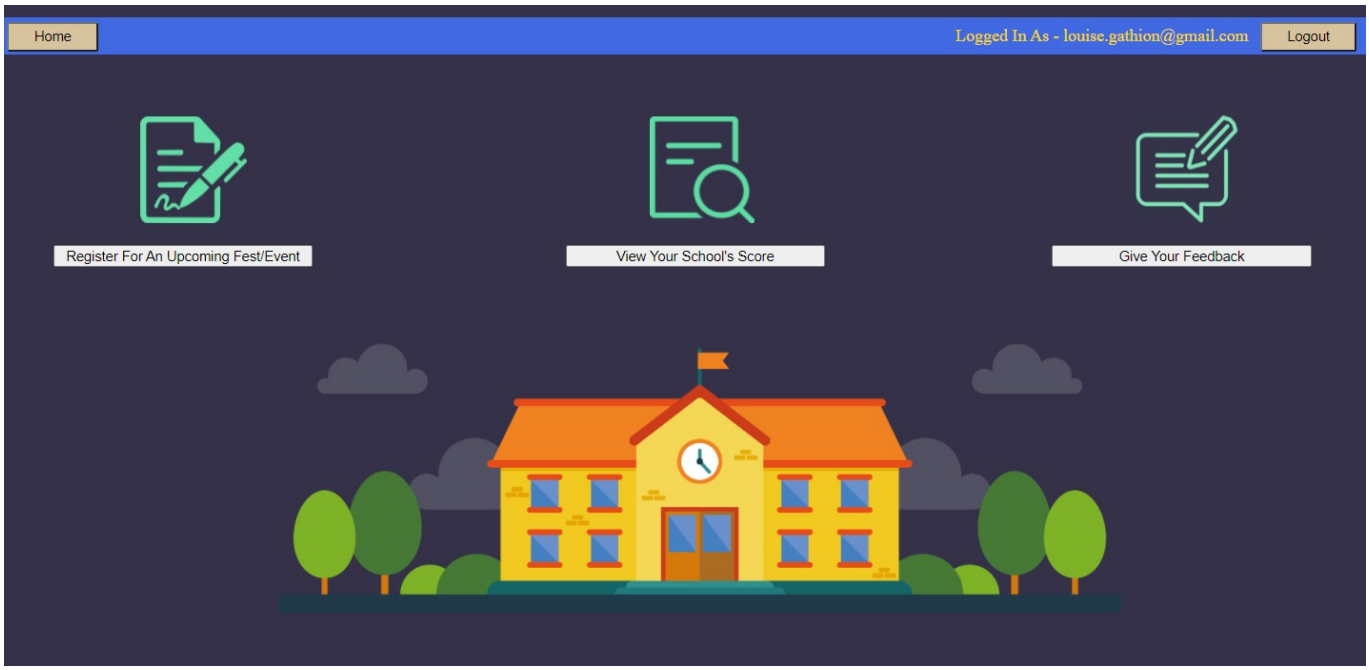
```
<asp:GridView ID="gvEventDetails" runat="server" AutoGenerateColumns="False" Height="200px" Width="1124px" BorderColor="T
    <RowStyle HorizontalAlign="Center"></RowStyle>
    <AlternatingRowStyle BackColor="#7e6896" />
    <FooterStyle BackColor="#ff9900" />
    <HeaderStyle BackColor="#ff9900" Font-Bold="True" />
    <PagerStyle BackColor="#7e6896" ForeColor="DarkSlateBlue" HorizontalAlign="Center" />
    <SelectedRowStyle BackColor="DarkSlateBlue" ForeColor="GhostWhite" />
    <SortedAscendingCellStyle BackColor="#FAFAE7" />
    <SortedAscendingHeaderStyle BackColor="#DAC09E" />
    <SortedDescendingCellStyle BackColor="#E1DB9C" />
    <SortedDescendingHeaderStyle BackColor="#C2A47B" />
```

When user selects a row, its BackColor and ForeColor colors both change to make it easier to distinguish from other rows

Role-Based access

Upon successful login, each user gets redirected to their user-specific homepage from where they are shown different functionalities on their user-specific homepage, depending on the user type.

School Home Page


Home
Logged In As - louise.gathion@gmail.com
Logout
Register For An Upcoming Fest/Event
View Your School's Score
Give Your Feedback

Code-Behind

```
protected void btnHome_Click(object sender, EventArgs e)
{
    Response.Redirect("HomePage.aspx");
}
0 references
protected void btnLogoutSchool_Click(object sender, EventArgs e)
{
    Response.Redirect("SchoolLoginPage.aspx");
}

0 references
protected void btnRegisterEvent_Click(object sender, EventArgs e)
{
    lblSchoolEmail.Text = Session["SchoolEmail"].ToString();
    Response.Redirect("SchoolEventDetailsPage.aspx");
}

0 references
protected void btnViewScore_Click(object sender, EventArgs e)
{
    lblSchoolEmail.Text = Session["SchoolEmail"].ToString();
    Response.Redirect("SchoolViewScoresPage.aspx");
}

0 references
protected void btnGiveFeedback_Click(object sender, EventArgs e)
{
    lblSchoolEmail.Text = Session["SchoolEmail"].ToString();
    Response.Redirect("SchoolFeedbackPage.aspx");
}
```

If user clicks on View Your School's Score button, they get redirected to that page

Session Variables

Session Variables were used in multiple instances as they allow the application to retain values across multiple pages. For example, user Email ID is stored in a session variable so that it can be displayed on every page. Similarly, when a school selects an event to register for, the fest and event name are stored in a session variable so they can be displayed in a GridView on the next page.

Event Coordinator Choosing Event

Code-Behind

```
protected void btnNextScreen_Click(object sender, EventArgs e)
{
    Session["ChosenEvent"] = ddlEventName.SelectedValue;
    Session["ChosenFest"] = ddlFestName.SelectedValue;
    Response.Redirect("EventCoordinatorInputScoresPage.aspx");
}
```

Storing selected Fest and Event in session variables and passing them on to the next page

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        lblEventCoordinatorEmail.Text = Session["EventCoordinatorEmail"].ToString();

        string constr = ConfigurationManager.ConnectionStrings["mycon"].ToString();
        // connection string
        SqlConnection con = new SqlConnection(constr);
        con.Open();
        string qry = "SELECT DISTINCT SchoolEmail, FestName, EventName FROM ParticipantDetails " +
            "WHERE FestName = '" + Session["ChosenFest"].ToString() + "' " +
            "AND EventName = '" + Session["ChosenEvent"].ToString() + "' ";
        SqlCommand com = new SqlCommand(qry, con);
        SqlDataAdapter da = new SqlDataAdapter(com);
        DataSet ds = new DataSet();
        da.Fill(ds);  // fill dataset
        gvInputScore.DataSource = ds;   //assigning datasource to the GridView
        gvInputScore.DataBind();  //binding GridView
    }
}
```

Populating GridView based on the value in the Session Variable by converting them to string

```
protected void BtnLoginAdmin_Click(object sender, EventArgs e)
{
    UserClass objAdminLogin = new AdminClass();
    objAdminLogin.UserEmail = txtAdminEmail.Text;
    objAdminLogin.UserPassword = txtAdminPassword.Text;

    int Result = objAdminLogin.LoginCheck();
    if (Result == 1)
    {
        lblErrorMessage.Visible = false;
        Session["AdminEmail"] = txtAdminEmail.Text;
        Response.Redirect("AdminHomePage.aspx");
    }
```

Storing Admin Email ID in a session variable by retrieving the email entered in the

```
protected void Page_Load(object sender, EventArgs e)
{
    lblAdminEmail.Text = Session["AdminEmail"].ToString();
}
```

Retrieving the Email ID stored in the session variable by converting it to string and displaying it in a label

Date-Time Functions

Date-Time functions were implemented using Range Validators. They are used to ensure that the date of the event lies between 1 to 3 months from the current date. The dates were converted to ShortDateString type date and were parsed in order to be compared.

Creating Event –

Declaring Range Validator

```
<asp:RangeValidator ID="RangeValidator" runat="server" EnableClientScript="true" Display="Dynamic"
    ControlToValidate="txtDateOfTheEvent" ForeColor="Red" SetFocusOnError="True"
    Visible="False">The Event has to be registered 1 to 3 months in advance.</asp:RangeValidator>
<br />
 <asp:TextBox ID="txtDateOfTheEvent" runat="server" Width="252px" ReadOnly="True"></asp:TextBox>
```

```
protected void Calendar1_SelectionChanged(object sender, EventArgs e)
{
    RangeValidator.ControlToValidate = "txtDateOfTheEvent";
    RangeValidator.Type = ValidationDataType.Date;
    RangeValidator.MinimumValue = DateTime.Now.AddMonths(1).ToShortDateString();
    RangeValidator.MaximumValue = DateTime.Now.AddMonths(4).ToShortDateString();
    RangeValidator.Visible = false;

    if (DateTime.Parse(Calendar1.SelectedDate.ToShortDateString()) > DateTime.Parse(RangeValidator.MinimumValue))
    {
        if (DateTime.Parse(Calendar1.SelectedDate.ToShortDateString()) < DateTime.Parse(RangeValidator.MaximumValue))
        {
            RangeValidator.Visible = false;
            txtDateOfTheEvent.Text = Calendar1.SelectedDate.ToShortDateString();
        }
        else
        {
            RangeValidator.Visible = true;
            RangeValidator.Text = "The Event has to be registered 1 to 3 months in advance.";
        }
    }
    else
    {
        RangeValidator.Visible = true;
        RangeValidator.Text = "The Event has to be registered 1 to 3 months in advance.";
    }
}
```

Declaring Minimum and Maximum dates

Using 2 IF statements to ensure selected date is greater than minimum value but lesser than maximum value

Testing

Manual Testing

Login – (Same for all Users)

Test Case	Method Of Testing	Expected Outcome
Abnormal Login Condition	User enters incorrect Email	Output message "Invalid Login Credentials. Please Try Again!"
Abnormal Login Condition	User enters incorrect Password	Output message "Invalid Login Credentials. Please Try Again!"
Normal Login Condition	User enters the correct Email and Password	Redirect user to their user-specific homepage.

Change Password – (Same for all Users)

Test Case	Method Of Testing	Expected Outcome
Abnormal Condition	User inputs incorrect Email and/or Old Password	Output message "Current Email ID or Old Password is incorrect. Please try again."
Abnormal Condition	User inputs different passwords in the new and confirm passwords textbox	Output message "The Passwords Are Not The Same. Please Try Again."
Normal Condition	User inputs existing Email and Old Password, along with matching text in new and confirm password textboxes.	Output message "Your password has been changed successfully!" along with updating the password in the database.

Reset Password – (Same for all Users)

Test Case	Method Of Testing	Expected Outcome
Abnormal Condition	User inputs incorrect Email	Output message "Email ID does not exist in our database."
Normal Condition	User inputs existing Email	Output message "Your New Password has been sent to your email!" and a new randomly generated password gets sent to the entered Email address. The password is also updated in the database.

Creating A Fest

Test Case	Method Of Testing	Expected Outcome
Abnormal Condition	Admin does not input a Fest Name	Output message "Please Enter A Fest Name!"
Abnormal Condition	Admin inputs a Fest Name that already exists in the database	Output message "This Fest already exists in our database."
Normal Condition	Admin inputs a new Fest Name	Output message "You have successfully created a new Fest!"

Creating An Event

Test Case	Method Of Testing	Expected Outcome
Abnormal Condition	Admin does not enter all the required fields	Output message "The event was not added successfully. Please try again!"
Abnormal Condition	Event Name already exists	Output message "The Event Name already exists. Please input a new name."
Abnormal Condition	The selected date for the Event does not lie between 1 to 3 months from the current date.	Output message "The Event has to be registered 1 to 3 months in advance."
Normal Condition	Admin enters all required fields, the event name does not already exist, and the date for the Event lies between 1 to 3 months from the current date.	Output message "You have successfully added an event!" and insert data in the allotted data-table in the database.

Updating Event Details

Test Case	Method Of Testing	Expected Outcome
Abnormal Condition	Admin does not enter all the required fields	Output message depending on the field(s).
Abnormal Condition	Event Name already exists	Output message "The Event Name already exists. Please input a new name."
Abnormal Condition	The selected date for the Event does not lie between 1 to 3 months from the current date.	Output message "The Event has to be registered 1 to 3 months in advance."

Normal Condition	Admin enters all required fields, the event name does not already exist, and the date for the Event lies between 1 to 3 months from the current date.	Output message "You have successfully added an event!" and insert data in the allotted data-table in the database.

Deleting an Event

Test Case	Method Of Testing	Expected Outcome
Normal Condition	Admin clicks on the Delete button next to the row the user wants to delete	The record is deleted from both the GridView and the data table containing the record.

Registering Participants For an Event by the School

Test Case	Method Of Testing	Expected Outcome
Abnormal Condition	School tries to register for the same event more than once.	Output message "You have already registered for this event.".
Abnormal Condition	User does not input a Participant Name.	Output message "Required.".
Abnormal Condition	User tries to add more participants than allowed according to the event criteria.	Output message "Participant Limit Reached!".
Normal Condition	User inputs Participant Names for all rows and the number of participants doesn't exceed the criteria.	Output message "You have successfully added all participants!".

Calculating Top 3 Winners

Test Case	Method Of Testing	Expected Outcome
Normal Condition	Admin clicks on the Calculate Top 3 Winners Button	A bubble sort algorithm is run where the 2 criteria are given weightage and basis this, a calculation is done for each school and the total values are added together and stored in an array. Finally, the Top 3 values are taken and their corresponding schools are fetched and displayed.

Inputting Event Scores by Event Coordinators

Test Case	Method Of Testing	Expected Outcome
Abnormal Condition	The scores for the selected event have already been given	Output message “The event scores have already been given.”.
Normal Condition	Event Coordinator inputs a score for all schools	Output message “You have successfully added the scores for all schools!”. The record gets inserted in the database.

Updating Event Scores by Event Coordinators

Test Case	Method Of Testing	Expected Outcome
Normal Condition	Event Coordinator selects a school and updates score through a DropDownList.	The score gets updated for the school in the database.

Schools register for an Event

Test Case	Method Of Testing	Expected Outcome
Abnormal Condition	User clicks on the “Register for this event!” button without selecting an event from the GridView	Output message “Please select an event.”.
Abnormal Condition	User tries to register for the same event more than once	Output message “You have already registered for this event.”.
Normal Condition	User selects a row from the GridView first, then clicks on the register for this event button	Redirects the user to the page where the user can register their participants for that selected event

Registering Participants for the Event

Test Case	Method Of Testing	Expected Outcome
Abnormal Condition	User does not enter values in all the required fields	Output message, depending on the field.
Abnormal Condition	User tries to add more participants than allowed for the event	Output message “Participant Limit Reached!”.

Normal Condition	User enters values in all the fields and selects eligible grades and clicks on the Save button	Output message "You have successfully added all participants!" and insert the records into a data-table in the database

Giving Feedback by Schools

Test Case	Method Of Testing	Expected Outcome
Abnormal Condition	User does not input or select a value for all the required fields	Output message, depending on the field.
Normal Condition	User inputs and selects a value for all required fields. User clicks on the Submit Feedback button	Output message "You have successfully submitted your feedback!" and inserts the feedback into a data-table in the database

Populating GridView based on the search criteria selected in the dropdowns (Search Method)
For demonstration purposes, this test plan is in regards to AdminViewEventSpecificScoresPage.

Test Case	Method Of Testing	Expected Outcome
Abnormal Condition	User selects a combination of non-default values, the data for which record(s) do not exist in the database. For example, choosing a Fest that does not contain any events.	Output error message "No data is available for the selected search criteria.".
Normal Condition	User selects default values in each of the drop downs then clicks on the Search button	Populates GridView based on the values selected in the dropdown.
Normal Condition	User selects a non-default value in one or more dropdown(s) and clicks on the Search button	Populates GridView according to the value(s) selected in the dropdown(s)

Compatibility on different browsers

Method Of Testing	Expected Outcome
The web-application will be run on different web browsers through visual studio	The web-application runs as expected and all functions and features are working as they should

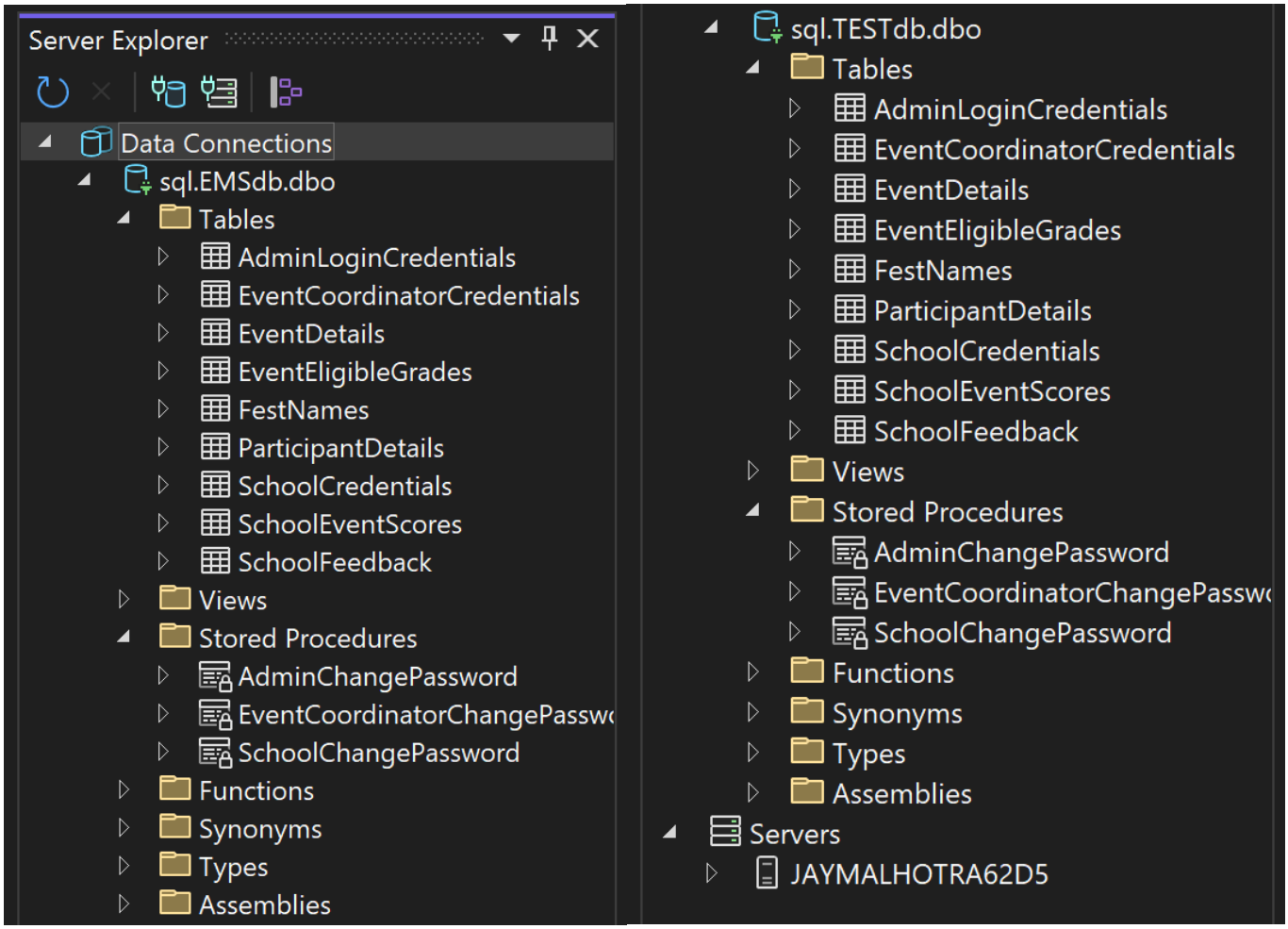
Integration Testing

Testing Environment:
The tests were executed within a controlled environment using Microsoft Test Test Project in Visual Studio. This environment was chosen for its robust testing frameworks and seamless integration with .NET projects, which our application is based on.

Testing Approach:

- **Automated Testing:** Utilizing the MS Test framework, automated tests were designed to simulate real-world operations and interactions between different software modules.
- **Test Coverage:** Each class file associated with the application was tested with an average of five test methods to cover a broad range of scenarios. This approach ensured that all critical functionalities were thoroughly validated.
- **Test Methods:** Approximately 25 test methods were implemented to address various functionalities across the modules. These methods tested functions including but not limited to database operations, user input handling, and response accuracy to API calls.

Test Database Setup:
Firstly, a dedicated test database was established to mimic the production environment but in a controlled manner that allows for repeatable testing without affecting live data. This test database was structured to mirror the actual application database with similar schemas and test data injected specifically for test scenarios.



Connecting the test database through a connection string in the configuration file –

```
    <connectionStrings>
  <add name="mycon" connectionString="Data Source=192.168.0.101;Initial Catalog=EMSdb;Persist Security Info=True;User ID=sa;Password=Slogo74484@;"
       <add name ="test" connectionString="Data Source=192.168.0.101;Initial Catalog=TESTdb;Persist Security Info=True;User ID=sa;Password=Slogo7
 </connectionStrings>
</configuration>
```

However, the test project kept throwing an System.NullReferenceException error, meaning that the test database could not be found through the connection string “test”. Therefore, we decided to hardcode the database string by inputting the data source directly into the SQL commands responsible for initializing the connections.

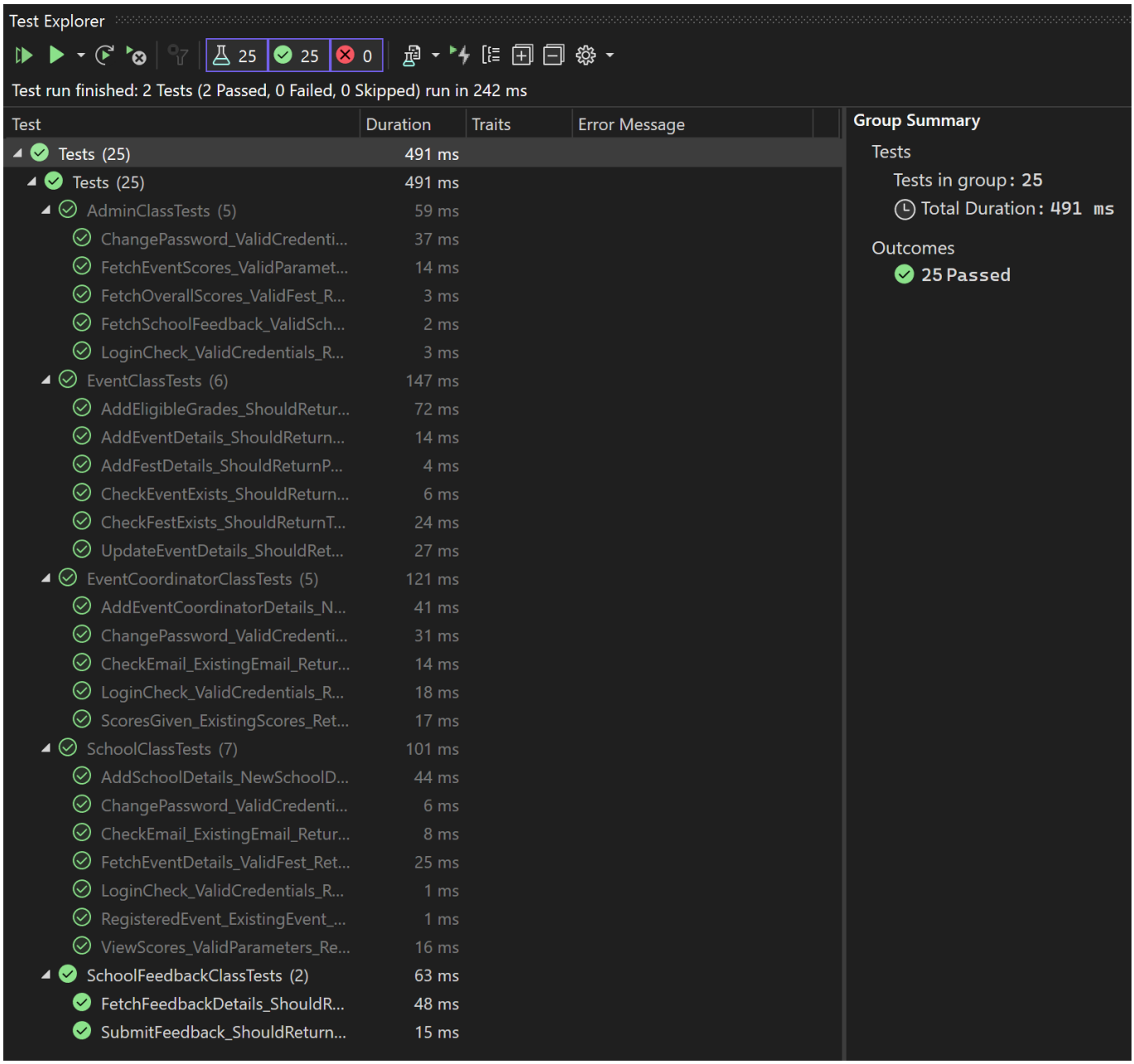
Execution of Tests:

Test Initialization: Before the commencement of any test, initial setup was performed which included populating the test database with predefined data suitable for each test case.

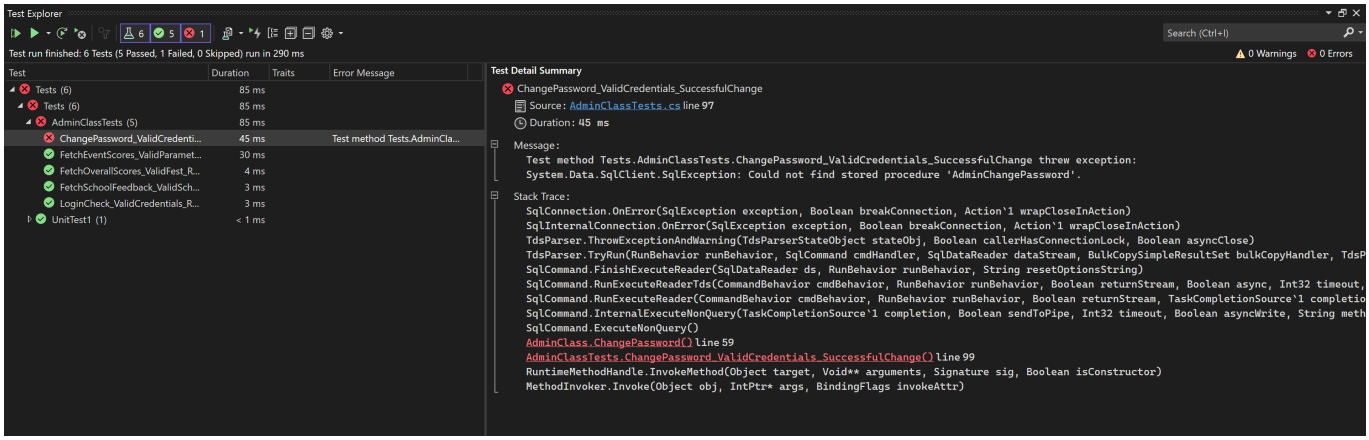
```
[TestClass]
0 references
public class AdminClassTests
{
    private static SqlConnection? _testConnection;
    private static AdminClass? _adminUnderTest;

    [ClassInitialize]
    0 references
    public static void SetUp(TestContext context)
    {
        try
        {
            string connectionString = "Data Source=192.168.0.101;Initial Catalog=TESTdb;Persist Security Info=True;User ID=sa;Password=Slogo74484@;";
            _testConnection = new SqlConnection(connectionString);
            _testConnection.Open();

            // Inserting Test Data
            SqlCommand insertAdminCreds = new SqlCommand(
                "INSERT INTO AdminLoginCredentials (AdminEmail, AdminPassword) " +
                "VALUES ('admin@test.com', 'adminPassword')",
                _testConnection);
            insertAdminCreds.ExecuteNonQuery();

            SqlCommand insertCoordinator = new SqlCommand(
                "INSERT INTO EventCoordinatorCredentials (EventCoordinatorName, EventCoordinatorEmail, EventCoordinatorPhoneNumber, EventCoordinatorPassword) " +
                "VALUES ('John Doe', 'john.doe@example.com', '555-1234', 'pass1234')",
                _testConnection);
            insertCoordinator.ExecuteNonQuery();

            SqlCommand insertSchool = new SqlCommand(
                "INSERT INTO SchoolCredentials (SchoolName, SchoolEmail, SchoolPassword, SchoolAddress, SchoolContactPerson, SchoolPhoneNumber) " +
```

Running the Tests: Tests were executed systematically with breakpoints to ensure that each integrated component functioned as expected within the system. This included testing APIs, middleware, and interaction with the backend database.

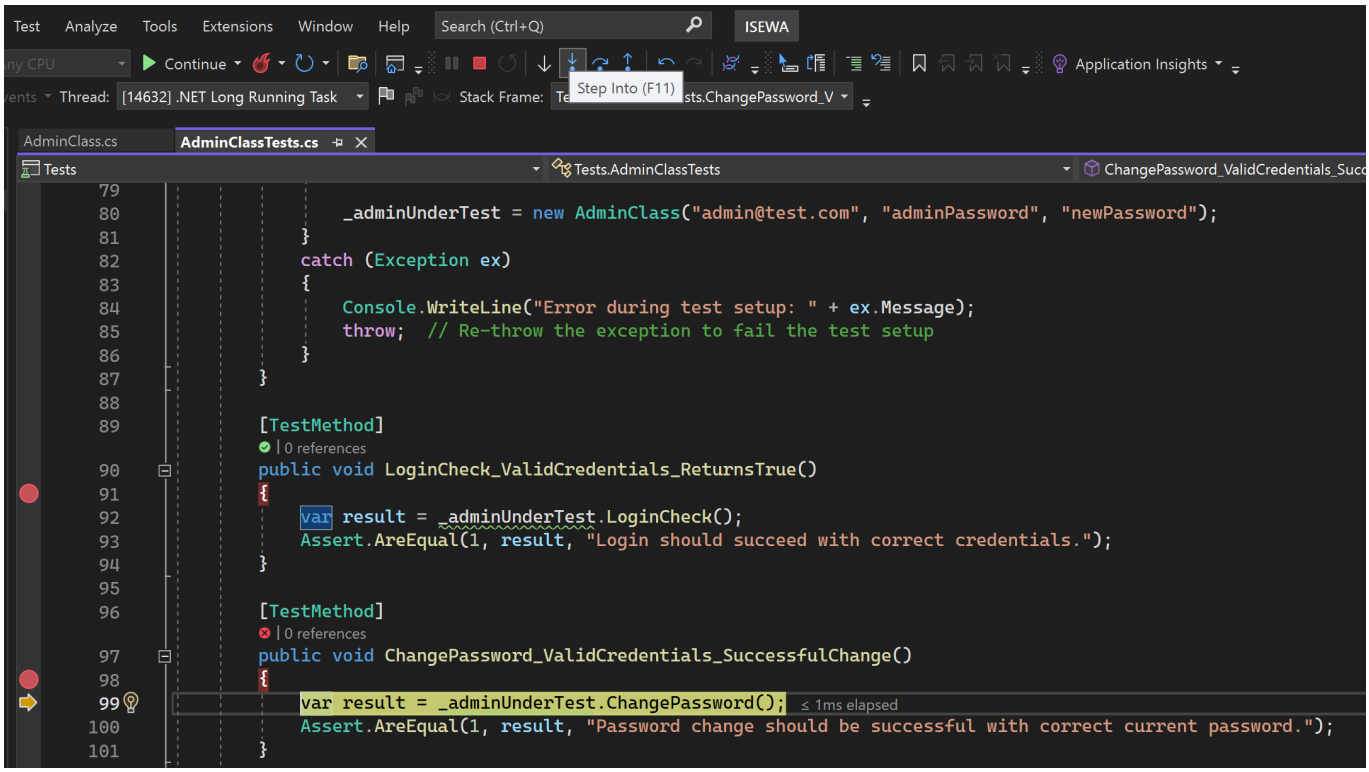
```
[TestMethod]
0 references
public void LoginCheck_ValidCredentials_ReturnsTrue()
{
    var result = _adminUnderTest.LoginCheck();
    Assert.AreEqual(1, result, "Login should succeed with correct credentials.");
}

[TestMethod]
0 references
public void ChangePassword_ValidCredentials_SuccessfulChange()
{
    var result = _adminUnderTest.ChangePassword();
    Assert.AreEqual(1, result, "Password change should be successful with correct current password.");
}

[TestMethod]
0 references
public void FetchEventScores_ValidParameters_ReturnsDataSet()
{
    var result = _adminUnderTest.FetchEventScores("TestFest", "TestEvent", "school@test.com");
    Assert.IsTrue(result.Tables[0].Rows.Count > 0, "Should return event scores data for valid parameters.");
}

[TestMethod]
0 references
public void FetchOverallScores_ValidFest_ReturnsDataSet()
{
    var result = _adminUnderTest.FetchOverallScores("TestFest");
    Assert.IsTrue(result.Tables[0].Rows.Count > 0, "Should return overall scores for the fest.");
}
```

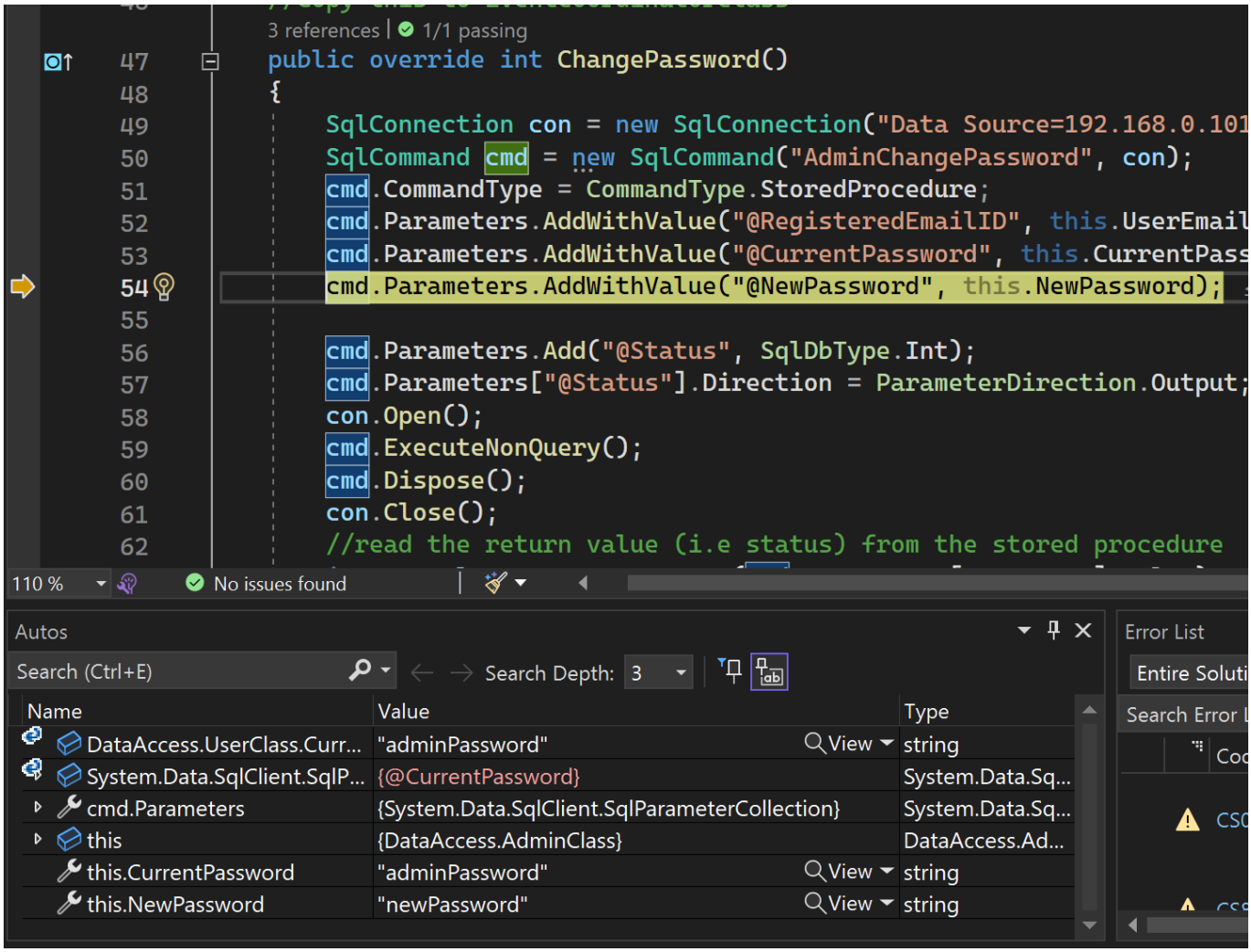


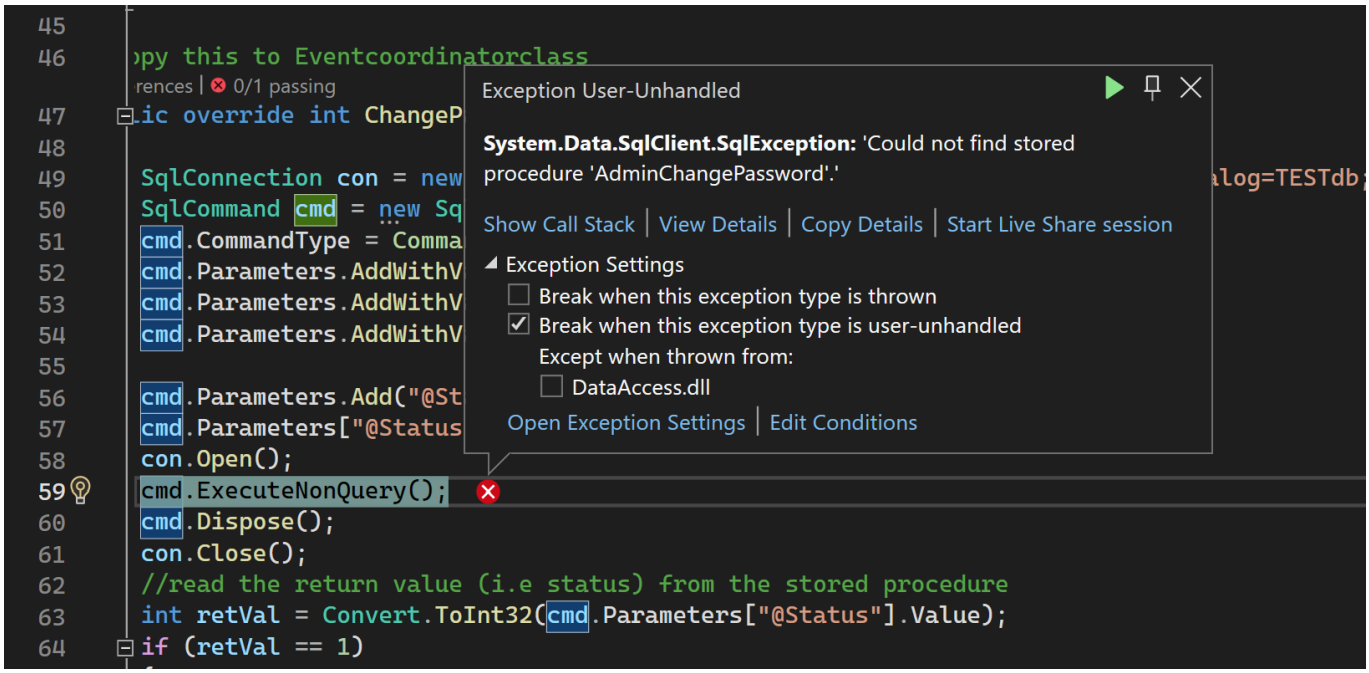
Monitoring and Logging: Test executions were monitored, and results were logged to facilitate error detection and to provide insights for debugging. Breakpoints were used to pinpoint exactly what values were being passed and where the error took place.



Using breakpoints and debugging the tests –







Cleaning up the class by resetting the test environment –

```
[ClassCleanup]
0 references
public static void Cleanup()
{
    try
    {
        // Clean up all test data to ensure tests are independent
        var cleanupCommand = new SqlCommand("DELETE FROM SchoolEventScores; DELETE FROM SchoolFeedback;
        cleanupCommand.ExecuteNonQuery();

        // Close the connection
        _testConnection.Close();
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error during test cleanup: " + ex.Message);
        throw;  // Re-throw to highlight issues in cleanup
    }
}
```

Deployment

The deployment phase of our project is currently localized to a development environment setup, utilizing Microsoft's Internet Information Services (IIS) and Visual Studio for hosting and managing the web application. Alongside this, a local SQL database is integrated to handle data storage and retrieval, ensuring that all functionalities are tested and optimized in a controlled environment. This approach aligns with the agile development model we have adopted, which emphasizes iterative development and frequent testing. Each component of the application—from backend logic to front-end presentation—has been incrementally developed and refined through multiple sprints. As of now, the application has not been deployed in a production environment; it remains hosted locally to facilitate further development, testing, and immediate access for debugging and enhancements. The final deployment to a live server will be considered upon completion of all testing phases and final approval of the project's stakeholders.

Project Estimation and Costing

COCOMO Model Application

In the realm of project management and software development, accurate estimation of time and resources is pivotal. The Constructive Cost Model (COCOMO) is a widely utilized algorithmic software cost estimation model that was originally formulated by Dr. Barry Boehm. The essence of COCOMO is to estimate the effort and cost of a software project by using a mathematical formula based on the number of lines of code (LOC).

In our project, we are specifically leveraging the COCOMO model for its detailed and structured approach to estimate the cost and schedule based on the actual code written. While our primary development methodology is Agile, which covers various phases of the software development lifecycle through iterative sprints and frequent evaluations, we integrate the COCOMO model solely for its calculation capabilities. This integration helps avoid redundancy, as the Agile model already addresses many aspects covered by the traditional COCOMO phases such as planning, requirements specification, and post-deployment maintenance. By focusing on the calculation part of COCOMO, we utilize a quantitative measure to assess and refine our development efforts, ensuring efficiency and alignment with the project’s timelines and budget constraints.

Calculation Based on Lines of Source Code

In applying the COCOMO model, we used Visual Studio's built-in code metrics calculator to find the number of lines of executable code within our project's scope. We have calculated figures for the Organic, Semi-Detached, and Embedded development modes. The line of source code consists of 9,639 lines, however, we will be using the number of lines of executable code, which is 1,780.

Organic Model Calculation

- **Effort:**
  - Formula: Effort = $a1 \times (KLOC)^{a2}$
  - Constants: $a1$=2.4, $a2$=1.05
  - Calculation: $2.4 \times (1.780)^{1.05} \approx 4.4$ Person-Months (PM)
- **Productivity:**
  - Formula: Productivity = $\frac{KLOC}{Effort}$
  - Calculation: $\frac{1.780}{4.4} \approx 0.405$ KLOC/PM
- **Development Time:**
  - Formula: Development Time = $b1 \times (Effort)^{b2}$
  - Constants: $b1$=2.5, $b2$=0.38
  - Calculation: $2.5 \times (4.4)^{0.38} \approx 4.4$ months
- **Staff:**
  - Formula: Staff = $\frac{Effort}{Development\ Time}$
  - Calculation: $\frac{4.4}{4.4} \approx 1$ person
- **Cost:**
  - Assumption: $1,000 per person per month
  - Calculation: $1 \times 4.4 \times 1{,}000 = \$4{,}400$

Semi-Detached Model

- **Effort:**
  - Constants: $a1$=3.0, $a2$=1.12
  - Calculation: $3.0 \times (1.780)^{1.12} \approx 6.51$ PM
- **Productivity:**
  - Calculation: $\frac{1.780}{6.51} \approx 0.273$ KLOC/PM
- **Development Time:**
  - Constants: $b1$=2.5, $b2$=0.35
  - Calculation: $2.5 \times (6.51)^{0.35} \approx 5.65$ months
- **Staff:**
  - Calculation: $\frac{6.51}{5.65} \approx 1.15$ people
- **Cost:**
  - Calculation: $1.15 \times 5.65 \times 1{,}000 = \$6{,}500$

Embedded Model

- **Effort:**
  - Constants: $a1$=3.6, $a2$=1.2
  - Calculation: $3.6 \times (1.780)^{1.2} \approx 8.20$ PM
- **Productivity:**
  - Calculation: $\frac{1.780}{8.20} \approx 0.217$ KLOC/PM
- **Development Time:**
  - Constants: $b1$=2.5, $b2$=0.32

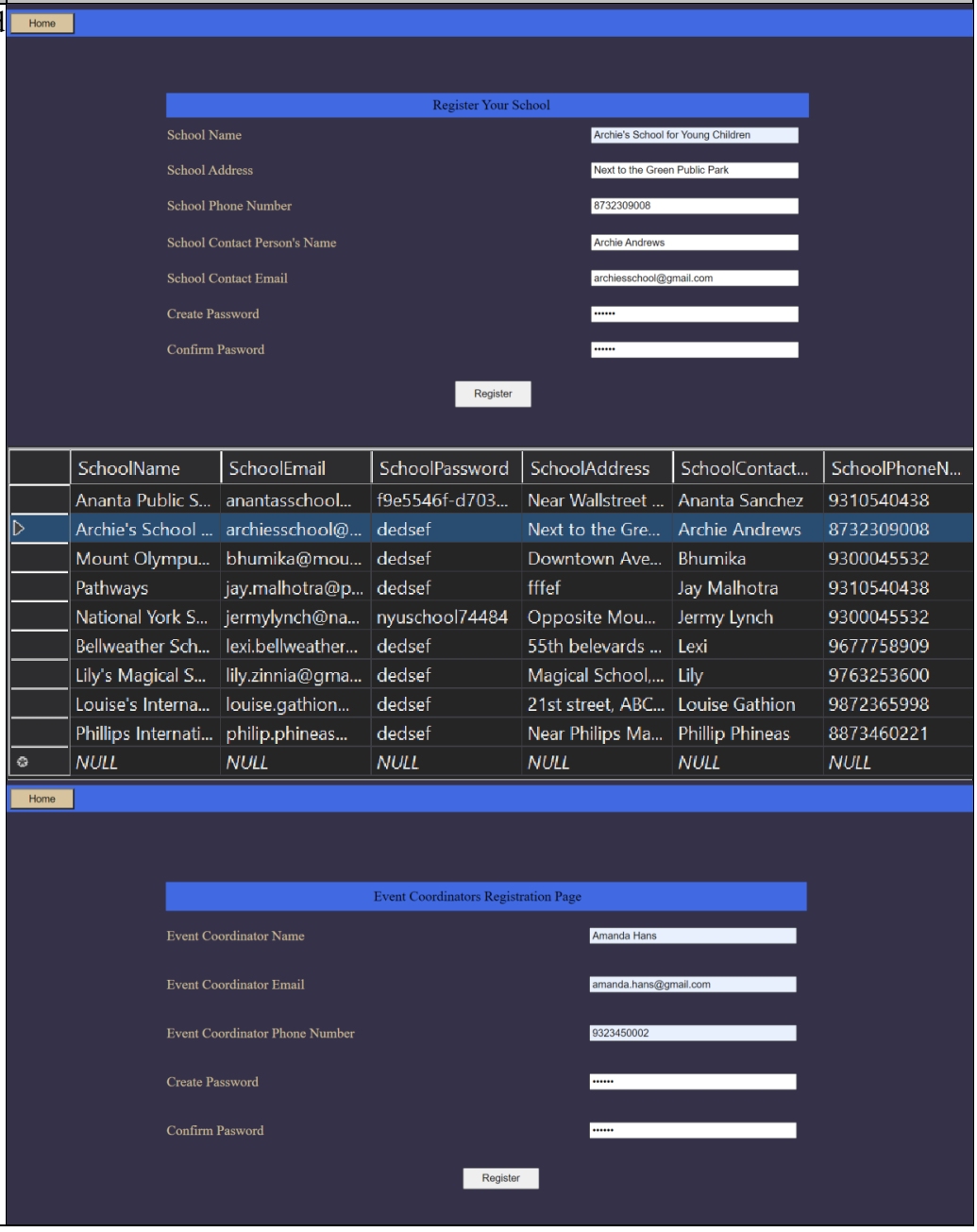
    - Calculation: $2.5 \times (8.20)^{0.32} \approx 6.89$ months
- **Staff:**
    - Calculation: $\frac{8.20}{6.89} \approx 1.19$ people
- **Cost:**
    - Calculation: $1.19 \times 6.89 \times 1{,}000 = \$8{,}200$

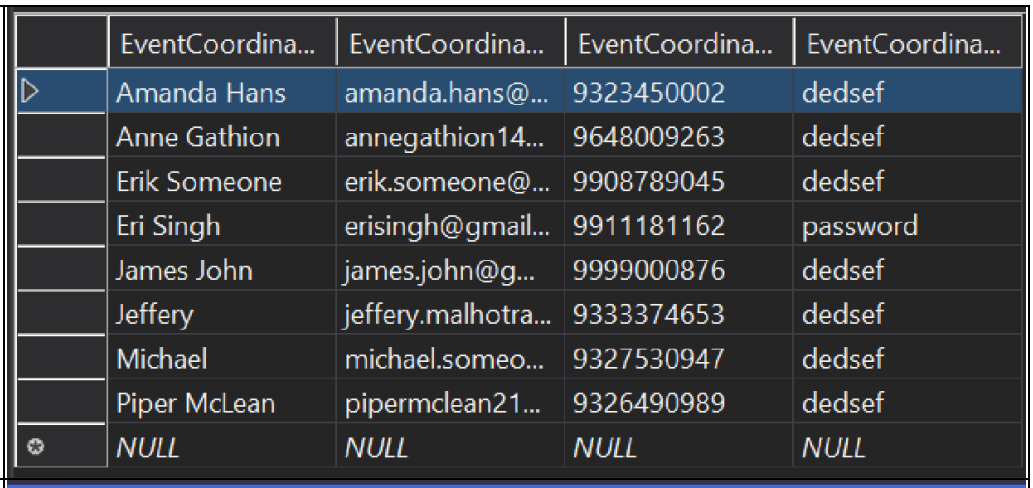
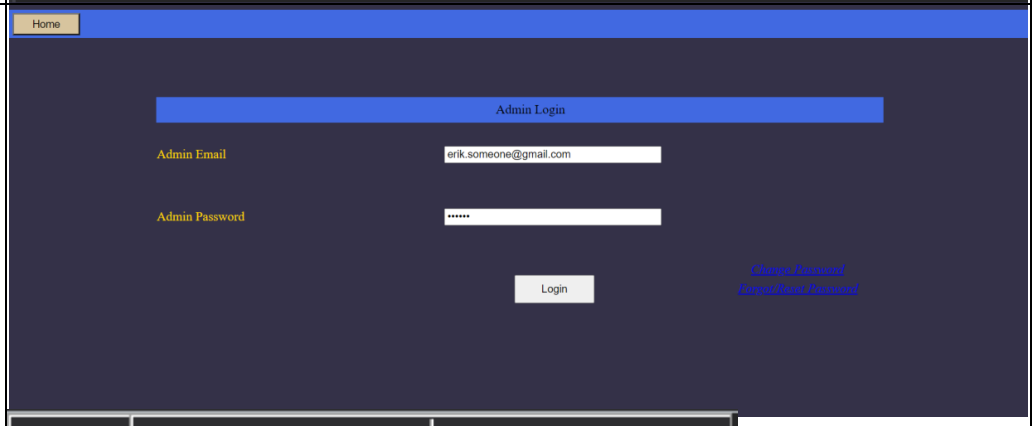
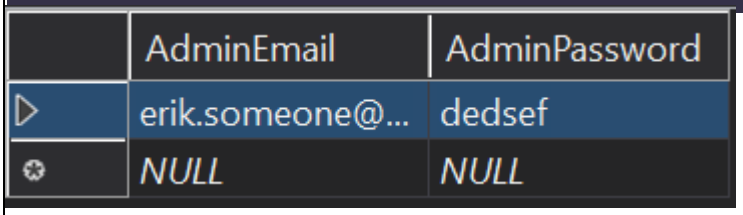
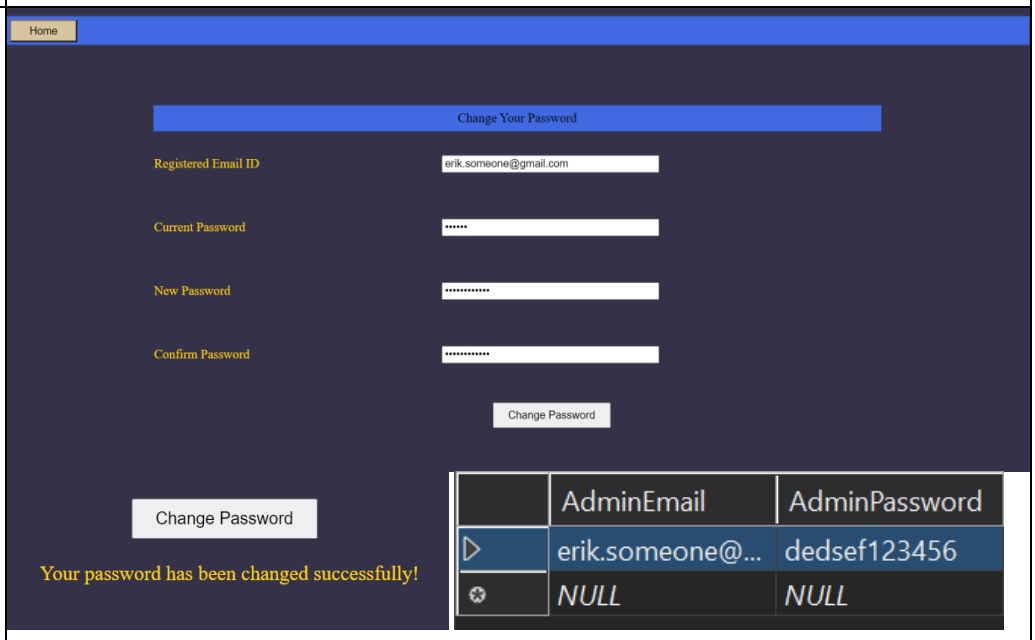
Summary

The calculations indicate that the Organic model is the most realistic and economical for the project scope, involving a straightforward development of an event management system. The estimated cost and effort are significantly more reasonable than initially calculated using larger-scale parameters.

Review

In the review section of the report, we will conduct a thorough comparison of the final product against the predefined success criteria established during the initial stages of the project. This comparison is critical to determine if the product meets all the functional and non-functional requirements specified by the stakeholders. We will assess each feature and functionality of the product to ensure compliance with the criteria, highlighting any discrepancies, areas of improvement, and features that have been successfully implemented. This evaluation will not only validate the effectiveness and efficiency of the product but also ensure that it aligns well with the user needs and business goals. The insights gained from this review will be instrumental in making informed decisions about any necessary adjustments or future enhancements to the product.

Success Criteria	Evidence
Sign up by schools and event coordinators. Schools and Event Coordinators can register on the application. Records are inserted in the database.	

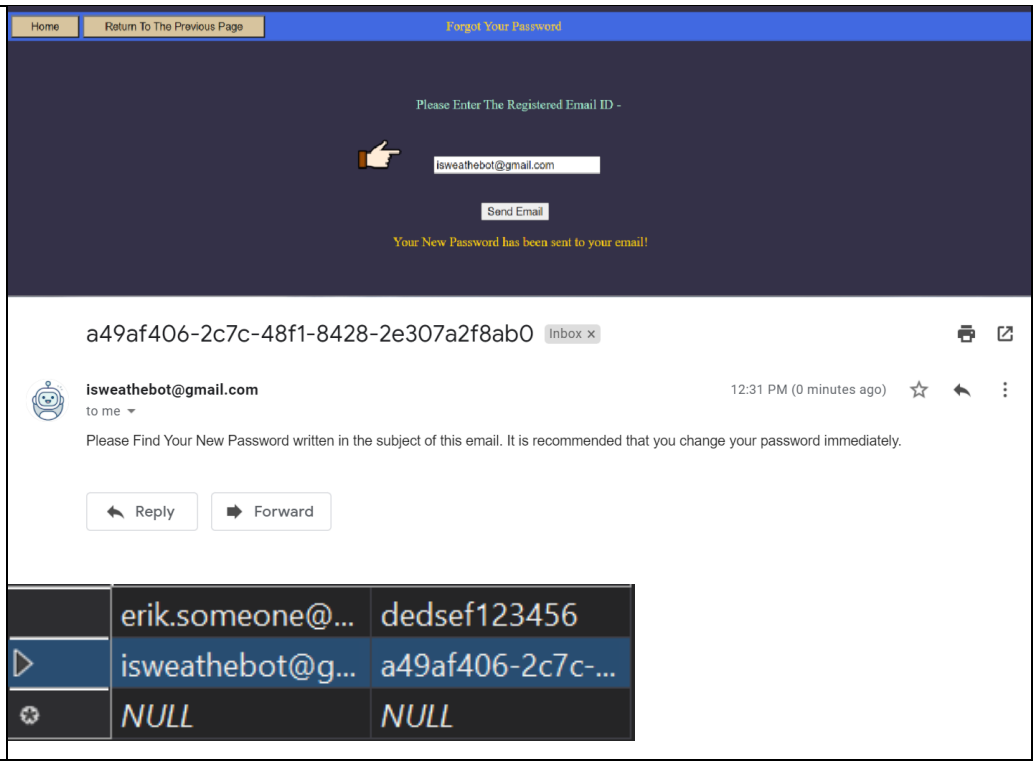
	
Login Functionality All authenticated users can login by crosschecking record from the database.	 
Change/Reset Password Functionality Password is changed and updated in the database.	

	EventCoordina...	EventCoordina...	EventCoordina...	EventCoordina...
▷	Amanda Hans	amanda.hans@...	9323450002	dedsef
	Anne Gathion	annegathion14...	9648009263	dedsef
	Erik Someone	erik.someone@...	9908789045	dedsef
	Eri Singh	erisingh@gmail...	9911181162	password
	James John	james.john@g...	9999000876	dedsef
	Jeffery	jeffery.malhotra...	9333374653	dedsef
	Michael	michael.someo...	9327530947	dedsef
	Piper McLean	pipermclean21...	9326490989	dedsef
✪	*NULL*	*NULL*	*NULL*	*NULL*

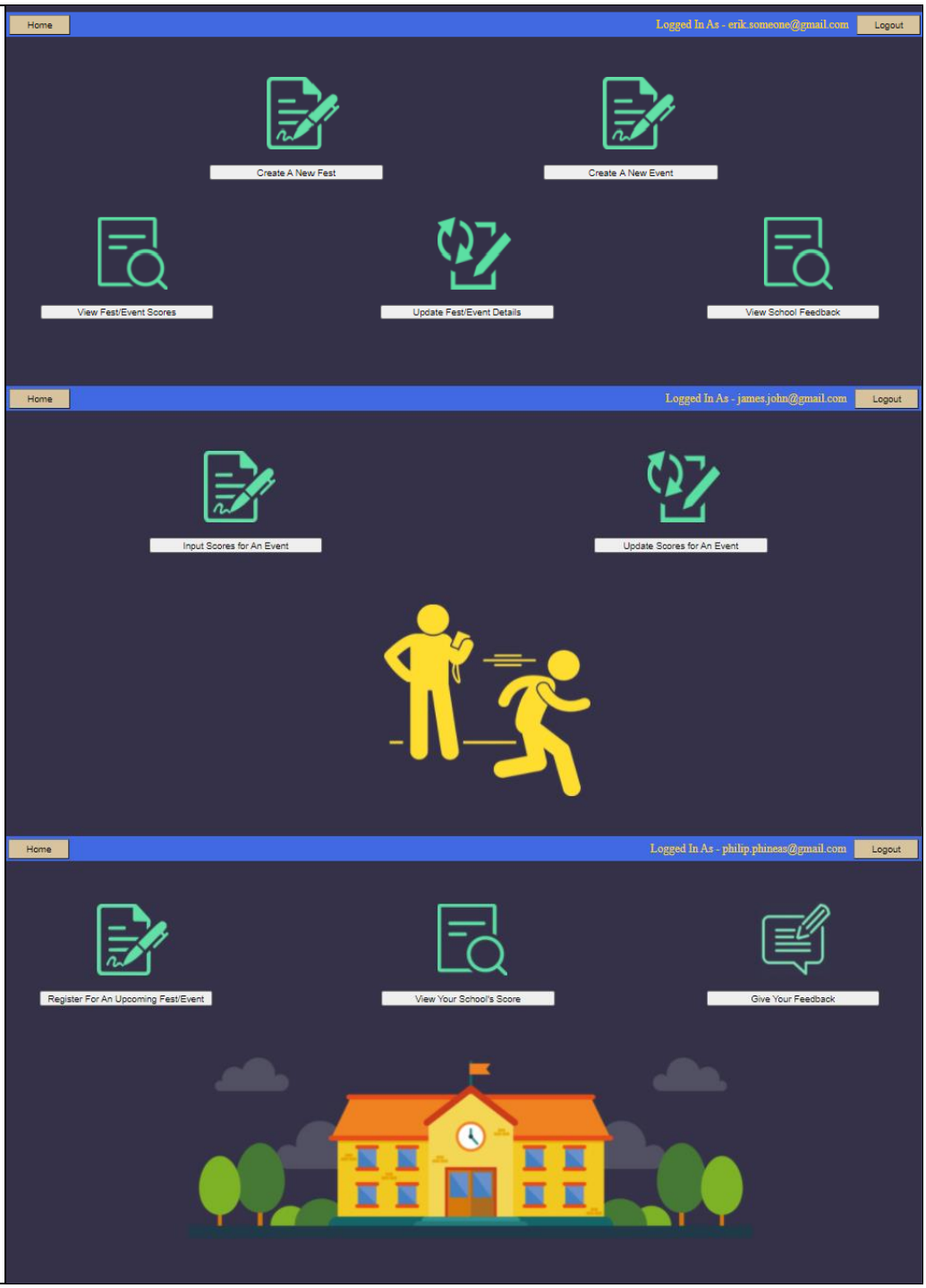
	AdminEmail	AdminPassword
▷	erik.someone@...	dedsef
✪	*NULL*	*NULL*

	AdminEmail	AdminPassword
▷	erik.someone@...	dedsef123456
✪	*NULL*	*NULL*

User receives a random generated password in the subject of the mail.



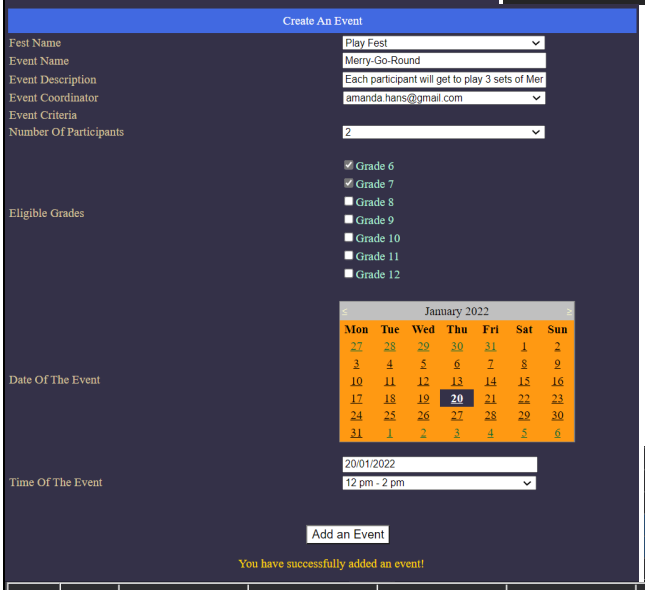
	erik.someone@...	dedsef123456
▶	isweathebot@g...	a49af406-2c7c-...
	NULL	NULL

Role Based Access User is redirected to their user-specific homepage depending on their role.	

Fest and Event Creation along with Criteria

Creating a Fest, event and event criteria.

	Id	FestName
	2	Annual Year 2020
	6	Annual Year 2021
	1	Fun Fest!
	4	Get-Set-Go!
▷	8	Play Fest
✪	NULL	NULL



	Dummy Event	Grade 7	68
	Dummy Event	Grade 6	69
	Dummy Event	Grade 7	70
	Merry-Go-Round	Grade 6	71
▷	Merry-Go-Round	Grade 7	72
✪	NULL	NULL	NULL

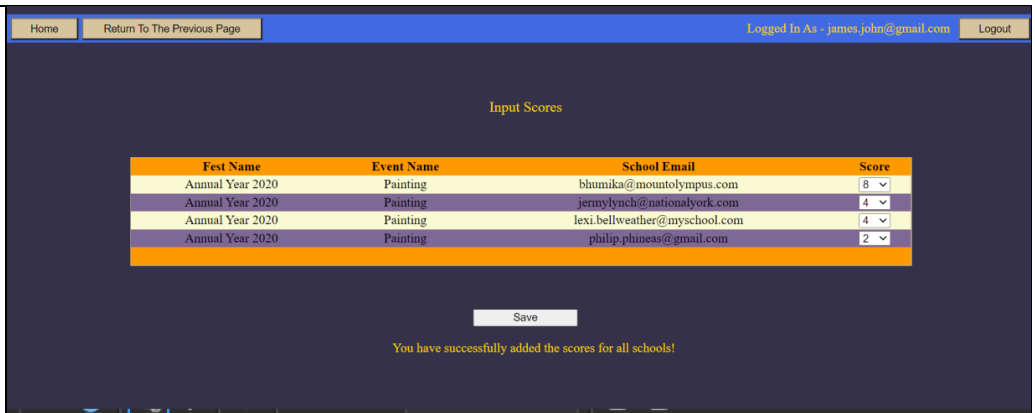
	Id	FestName	EventName	EventDescripti...	EventCoordina...	NumberOfParti...	DateOfTheEvent	TimeOfTheEvent
	1	Annual Year 2020	All you can Eat!	Participants will ...	james.john@g...	3	28-11-2021	12 pm - 2 pm
	9	Annual Year 2020	Basketball Fever	Schools will co...	james.john@g...	5	24-11-2021	10 am - 12 pm
	16	Annual Year 2020	Bottle Flip	Each participant...	annegathion14...	2	01/01/2022	2 pm - 4 pm
	3	Annual Year 2020	Doodling	Participants will ...	james.john@g...	2	28-11-2021	8 am - 10 am
	26	Annual Year 2020	Dummy Event	Description	jeffery.malhotra...	1	31/12/2021	10 am - 12 pm
	4	Annual Year 2020	Gaming	Participants will ...	james.john@g...	1	24-11-2021	8 am - 10 am
▷	27	Play Fest	Merry-Go-Round	Each participant...	amanda.hans@...	2	20/01/2022	12 pm - 2 pm
	5	Annual Year 2020	Painting	Participants will ...	james.john@g...	3	28-11-2021	12 pm - 2 pm
	6	Annual Year 2020	Singing at the B...	Participants hav...	james.john@g...	2	27-11-2021	2 pm - 4 pm
	15	Fun Fest!	Skating	Participants will ...	michael.someo...	2	31/12/2021	10 am - 12 pm
✪	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

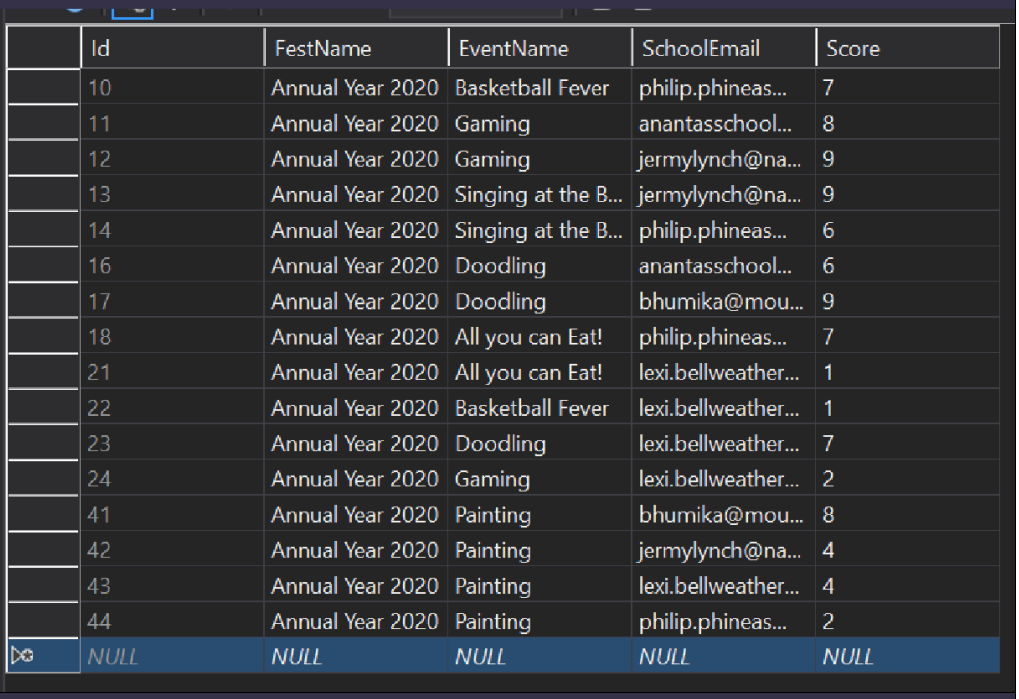
Registering for the Event

Selecting the event and registering participants according to the event criteria.

Inputting Scores for individual events

Choosing Fest and Event name then inputting scores.

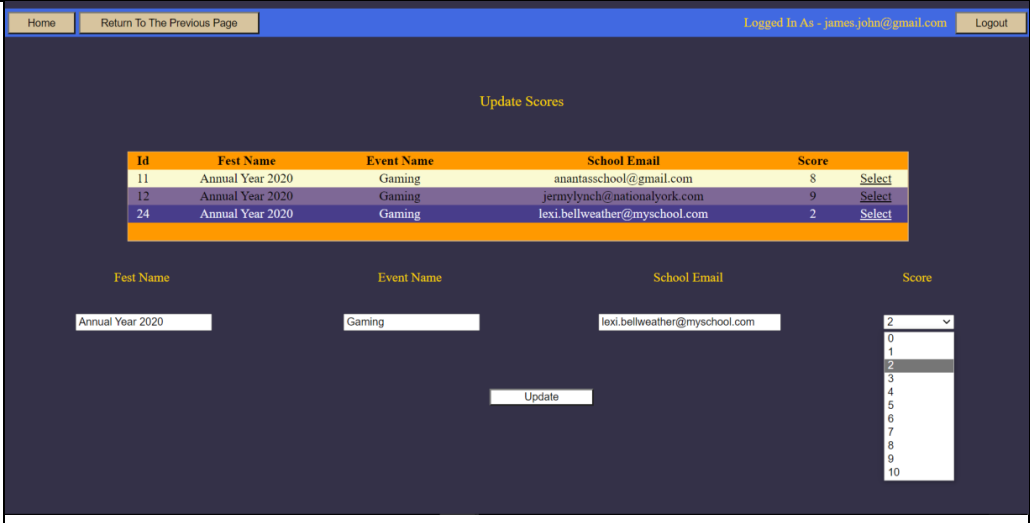


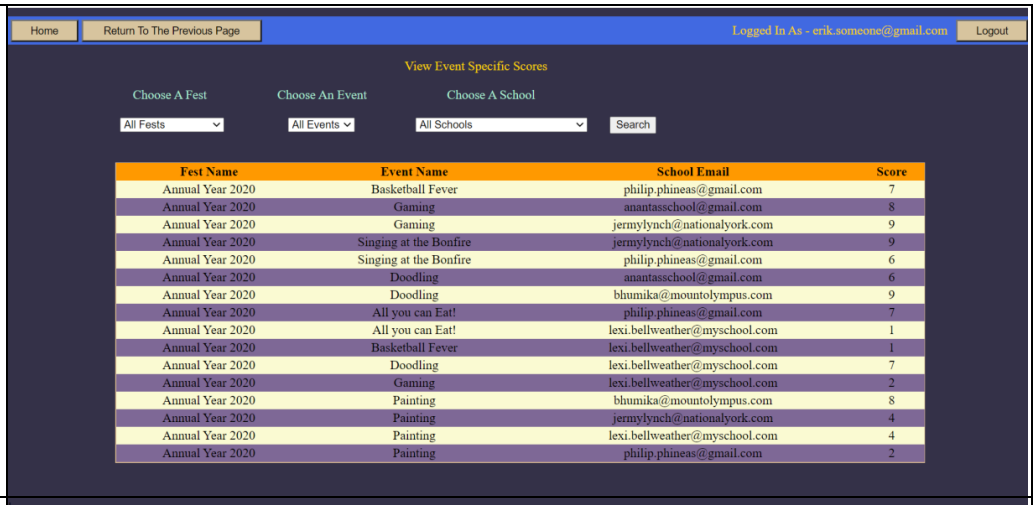


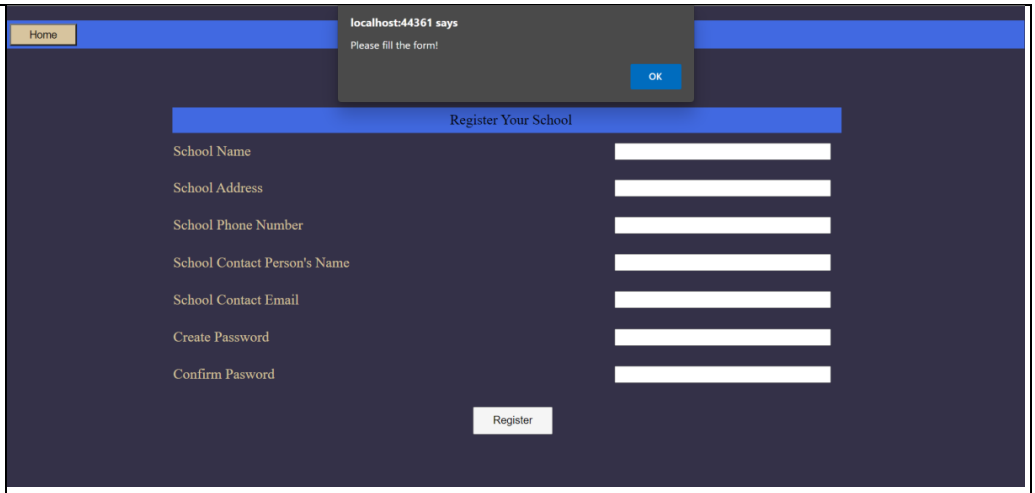
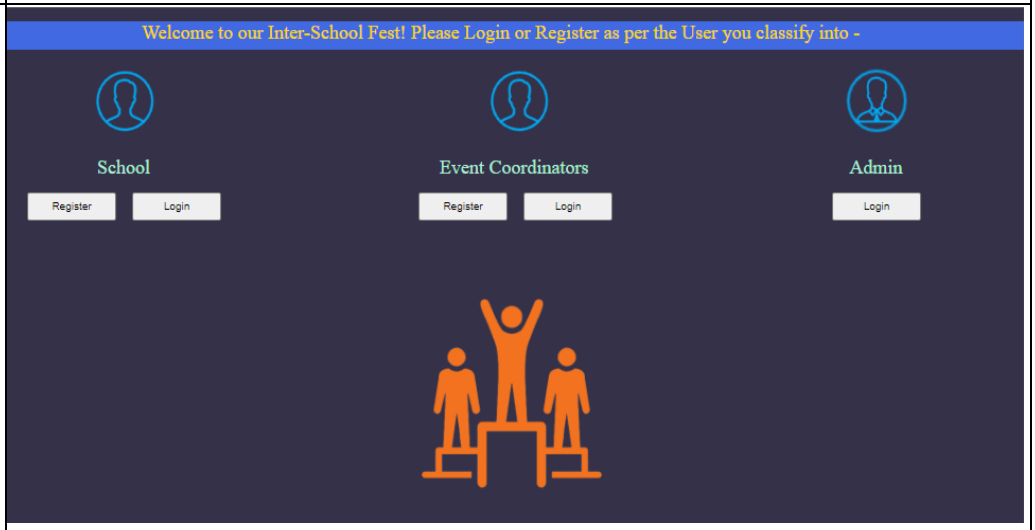
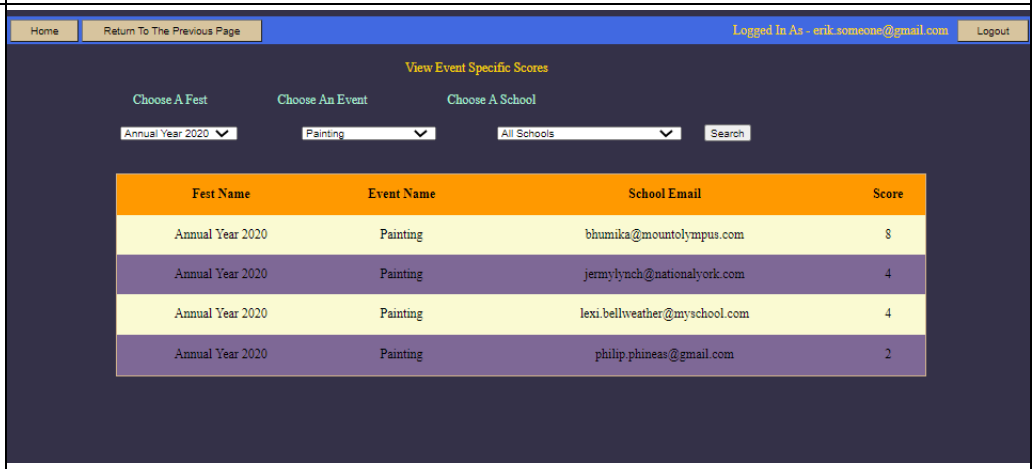
Id	FestName	EventName	SchoolEmail	Score
10	Annual Year 2020	Basketball Fever	philip.phineas...	7
11	Annual Year 2020	Gaming	anantasschool...	8
12	Annual Year 2020	Gaming	jermylynch@na...	9
13	Annual Year 2020	Singing at the B...	jermylynch@na...	9
14	Annual Year 2020	Singing at the B...	philip.phineas...	6
16	Annual Year 2020	Doodling	anantasschool...	6
17	Annual Year 2020	Doodling	bhumika@mou...	9
18	Annual Year 2020	All you can Eat!	philip.phineas...	7
21	Annual Year 2020	All you can Eat!	lexi.bellweather...	1
22	Annual Year 2020	Basketball Fever	lexi.bellweather...	1
23	Annual Year 2020	Doodling	lexi.bellweather...	7
24	Annual Year 2020	Gaming	lexi.bellweather...	2
41	Annual Year 2020	Painting	bhumika@mou...	8
42	Annual Year 2020	Painting	jermylynch@na...	4
43	Annual Year 2020	Painting	lexi.bellweather...	4
44	Annual Year 2020	Painting	philip.phineas...	2
NULL	NULL	NULL	NULL	NULL

Updating Scores for individual events

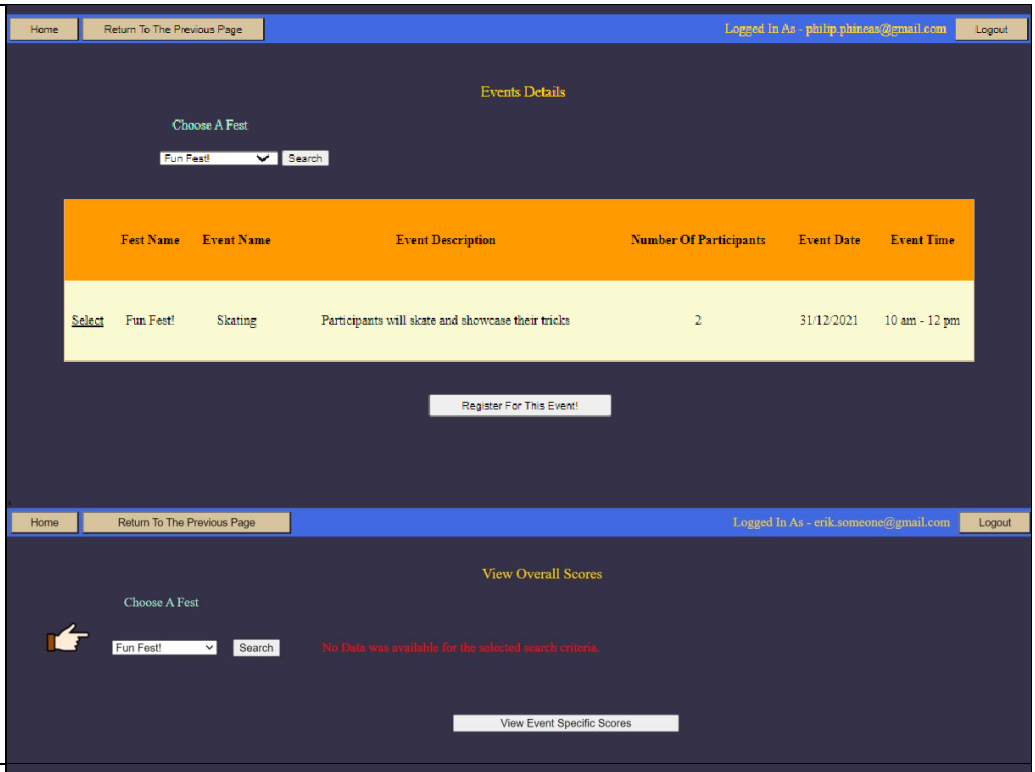
Selecting row and changing score from dropdown.



View Scores of the events Viewing scores in Gridview.	
Calculation of scores in real time for leader board GridView adds scores for all events school took part in for each school.	
Top 3 winners Calculated and displayed in labels.	

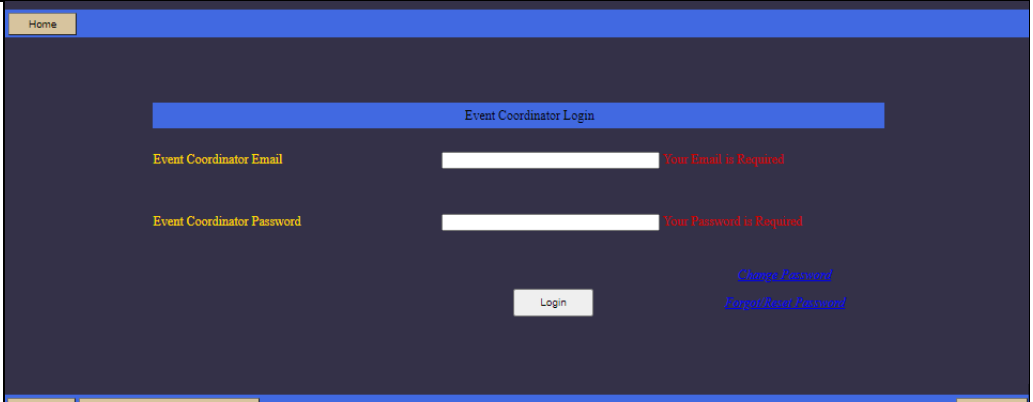
User Friendly Error messages JavaScript pop-up is displayed when user tries to submit a blank form.	
Intuitive and Professional User Interface Homepage consisting of a user-friendly image, bright colours, big fonts and organized layout.	
Search based on Filters GridView is populated according to search criteria.	

If no record is available, error message is displayed.

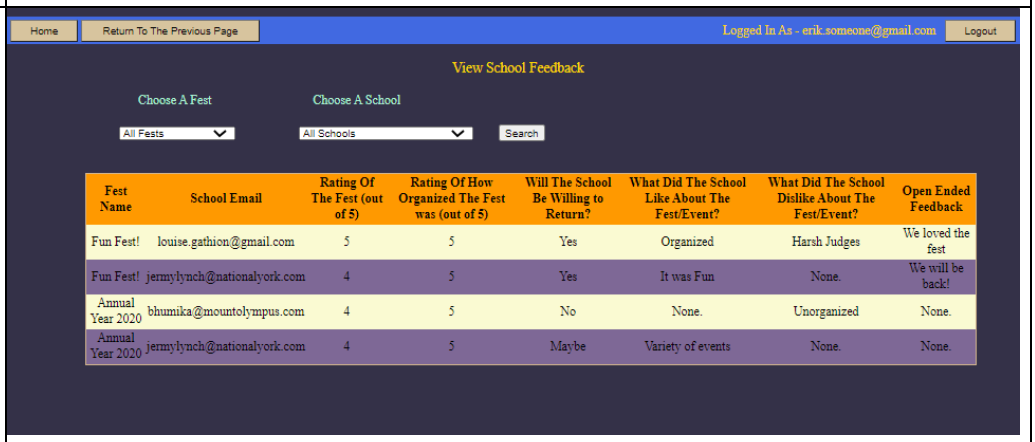


Validation

Required Field Validators.



Validation ensuring no duplicate data is inserted.

Feedback Page Schools have option to leave feedback.	
Viewing School Feedback Admin can view feedback.	

Scalability

Object-Oriented Programming concepts were applied throughout the application to ensure scalability and give scope for adding new features to the application and making the code reusable. Through encapsulation, any new functionality added in one of the classes will have minimal impact on the overall code and other classes, enhancing the flexibility of the code and ensuring abstraction. Moreover, any changes would be an extension to the already- existing code or can re-use/modify the existing code. New users and related functionality can also be easily implemented by referring to the already-existing code, databases and classes.

Future Development

Recommendations for future development play a crucial role in enhancing the functionality and user experience of the application. Based on the observed needs during the project lifecycle, the following enhancements are proposed:

1. **Automated Reminder Emails**: It is recommended to implement a feature for the admin to send automated reminder emails to all participating schools about the event details. There might be instances where schools forget the event dates, leading to delays or absences. Automated reminders would help mitigate this issue and ensure better participation.
2. **Customizable Weightage for Scoring**: Currently, the weightage assigned to the criteria for judging the events is fixed. To provide more flexibility and adaptability, it is suggested that admins should have the capability to adjust the weightage for individual fests or events based on their specific requirements and standards.
3. **Performance Statistics for Schools**: Schools would benefit from being able to access and view performance statistics. This feature would enable schools to compare their scores year on year and analyze performance trends across different events, facilitating strategic preparations and improvements.
4. **Event Rubrics Upload and Access**: To streamline the preparation process for schools and reduce administrative overhead, a feature should be added that allows the admin to upload rubrics for each event directly onto the platform. Schools could then download these rubrics at their convenience, ensuring they are well-prepared and that the event criteria are transparent.
5. **File Uploads for Event Submissions**: For events that require submissions prior to the event day, such as video projects, the application should include a provision for schools to upload their files directly. This would simplify the submission process, ensure timely receipt of materials, and centralize the management of submissions.

Implementing these recommendations would not only enhance the efficiency of event management but also significantly improve user satisfaction by addressing specific needs and feedback from the stakeholders involved.