**Prototype 1 – Main Page:**

**Overview**

The prototype for the **main page** of the traffic test application in C# focuses on a user-friendly navigation system between various forms, such as the Practice Page, Mock Test Page, Instructions Page, and Progress Page. Initially, button clicks were implemented to open new forms, with early iterations addressing navigation issues, such as incorrect form references. The logic evolved to introduce an Instructions Page before accessing the Mock Test. Further development included a new **Progress Page** that tracks user performance. This was achieved by incorporating a **dictionary to store test scores** and passing it as a parameter to the Progress Page constructor. Each iteration refined the flow of the main page, ensuring smoother navigation and enhanced functionality for users.

**Development and Debugging:**

**Iteration 1:**  
A screenshot of a computer

Description automatically generated

**Problem:** How to Map buttons so that they open a new form?  
**Solution:** Applying the function with the Click Keyword with the name of the button.  
**Code:**   
 ***private void Practice\_Click(object sender, EventArgs e)***

***{***

***PracticePage nextForm = new PracticePage();***

***nextForm.Show();***

***this.Hide();***

***}***

Opens new form  
 ***PracticePage nextForm = new PracticePage();***

***nextForm.Show();***

**Iteration 2:**

**Obstacle**: The Mock Test Page opening the Practice Page  
**Fix:** Refrence Wasnt Named Right Hence Causing the Practice Page to open Rather Than Mock test Page  
  
**Code:**

*private void Mock\_test\_Click(object sender, EventArgs e)*

*{*

*MockTest\_Page nextForm = new MockTest\_Page();*

*nextForm.Show();*

*this.Hide();*

*}*

**Iteration 3:**

**Update**: The Mock Test Page button now opens the Instruction form

***private void Mock\_test\_Click(object sender, EventArgs e)***

***{***

***InstructionsForm nextForm = new InstructionsForm();***

***nextForm.Show();***

***this.Hide();***

***}*  
Iteration 4:**

**Update:** The New Progress\_Page constructor in my new logic which will require a Dictionary<int, int> argument that contains the scores for each test. This dictionary keeps track of which tests have been attempted and their respective scores. I need to pass this dictionary when creating an instance of Progress\_Page. In your MockTest\_Page, Ialready have a testScores dictionary that tracks the scores for each test. So, I can pass that dictionary when calling the constructor.  
  
**Code:**  
***private void Progress\_Click(object sender, EventArgs e)***

***{***

***Progress\_Page nextForm = new Progress\_Page(GlobalData.TestScores);***

***nextForm.Show();***

***this.Hide();***

***}***

**Review:**

|  |  |
| --- | --- |
| **Success Criteria** | **Result** |
| **Does the system allow users to mark topics as completed?** |  |
| **Does the system persist the checkbox state even when the user navigates away from the page?** |  |
| **Can users view all signs related to a specific topic when clicking on it?** |  |
| **Are all traffic signs displayed with appropriate images and descriptions?** |  |
| **Is the progress of each topic updated immediately when the user marks it as completed or incomplete?** |  |
| **Can the user view flagged questions separately?** |  |
| **Does the progress page show a complete list of all completed topics?** |  |
| **Does the system display dynamic images and descriptions of signs from the correct topic?** |  |
| **Is the user interface intuitive for selecting topics and viewing signs?** |  |
| **Can the user easily toggle between topics and view progress?** |  |
| **Are checkboxes clearly visible and easy to interact with?** |  |
| **Can users navigate seamlessly between different pages (topics, progress, signs)?** |  |
| **Is there a way for users to track and view their overall progress on a single page?** |  |
| **Are the images of signs clear and appropriately sized?** |  |
| **Does the interface visually distinguish between completed and incomplete topics?** |  |
| **Can users undo or reset their progress on specific topics if needed?** |  |
| **Does the app load quickly without significant delays?** |  |
| **Is there any lag when switching between topics or viewing images?** |  |
| **Are sign images loaded efficiently without causing performance issues?** |  |
| **Does the app handle high volumes of data, such as a large number of signs or topics, without crashing?** |  |
| **Is the application using a global variable or persistent data structure for storing progress?** |  |
| **Is there error handling in place for missing images or incorrect data states?** |  |
| **Are images and descriptions dynamically loaded from the appropriate file location?** |  |
| **Is the progress stored in a way that can be saved or transferred between sessions?** |  |
| **Is the layout clean, with proper spacing and alignment for readability?** |  |
| **Is the text legible and appropriately styled for clarity?** |  |
| **Are progress indicators (checkmarks, completion percentages) clearly visible?** |  |
| **Is the theme of the page customizable or consistent with the overall project design?** |  |
| **Is the app interactive, allowing the user to engage with different features (checkboxes, navigation, image display)?** |  |
| **Does the system update in real-time when a topic is marked as complete?** |  |
| **Does the progress page update automatically to reflect changes in the checkbox states?** |  |
| **Can users interact with the images, such as zooming in for better visibility of signs?** |  |
| **Does the system handle missing or corrupted image files gracefully (e.g., showing a placeholder)?** |  |
| **Does the app prevent crashes or unexpected behavior from faulty inputs or actions?** |  |
| **Are there clear error messages or notifications if something goes wrong (e.g., missing data or images)?** |  |
| **Is the system flexible enough to add new topics or signs in the future?** |  |
| **Can additional content, such as multimedia (videos, animations), be added without major redesigns?** |  |
| **Can more complex features (like quizzes or assessments) be easily integrated into the current framework?** |  |

**Prototype 2 – Practice Page:**

**Overview:**

The Practice Page is an interactive quiz-based learning platform designed to help users practice and test their knowledge on various topics. Users can select from multiple predefined tests, each containing a unique set of questions. The page presents one question at a time, offering multiple answer options for users to choose from. After submitting an answer, users receive immediate feedback on its correctness or view their results at the end of the test. The system tracks user progress, showing the number of questions answered out of the total, and provides a final score summary upon completion. The interface is simple, intuitive, and user-friendly, allowing for easy navigation between questions. By offering a self-paced learning experience, the Practice Page encourages users to improve their understanding of key concepts while tracking their performance. This approach makes it an effective tool for education, exam preparation, and skill development.

**Development and Debugging:**

**Iteration 1:**



**Problem**: How to Map the start button to start the quiz?  
  
**Solution**: Use A Function class to call another class which clears the form from everything and loads in questions   
 ***Controls.Clear();***

**Problem**: Tracker to track the question number you are on and iterating to the next question?  
  
**Solution**: Introduce a Tracker Label ***private Label trackerLabel;***

The total number of questions in the test is determined by the `selectedTest.Questions.Count` property, which retrieves the size of the `Questions` collection in the `selectedTest` object. This count is used to ensure the test ends when the current question index (`currentQuestionIndex`) reaches the total number of questions

***if (selectedTest == null || currentQuestionIndex >= selectedTest.Questions.Count*)**

**To update the tracker label:** that displays the current question number Each `Question` object in the `Questions` collection contains the question text and its possible options.

***trackerLabel.Text = $"Question {currentQuestionIndex + 1} of {selectedTest.Questions.Count}";***

**Iterate through the options**: A for loop is used for the collection selectedTest.Questions contains Question objects, where each question has its text (currentQuestion.Text) and a list of options (currentQuestion.Options).   
  
**Code:**

***for (int i = 0; i < currentQuestion.Options.Count; i++)***

***{***

***RadioButton optionButton = new RadioButton***

***{***

***Text = currentQuestion.Options[i],***

***AutoSize = true,***

***Tag = i // Store the option index***

***};***

***optionButton.Location = new Point((ClientSize.Width - optionButton.Width) / 2, yPosition);***

***Controls.Add(optionButton);***

***yPosition += 30;***

***}***

**Iteration 2:**

**Problem**: I need buttons to navigate between questions but in a smoother way, ensuring I don't navigate to a non-existent question.

**Solution**: I use the currentQuestionIndex variable to keep track of the current question I'm viewing. This variable updates when I navigate between questions, ensuring the correct question from selectedTest.Questions is loaded.

nextButton for Moving Forward by incrementing currentQuestionIndex which calls LoadQuestion() to load the next question  
  
**Code:**

***private void NextButton\_Click(object sender, EventArgs e)***

***{***

***currentQuestionIndex++;***

***LoadQuestion();***

***}***

ensures the subsequent question is displayed as long as currentQuestionIndex is less than selectedTest.Questions.Count.

previousButton for Moving Backward by decrementing currentQuestionIndex which calls LoadQuestion() to load the previous question  
  
**Code:**

***private void PreviousButton\_Click(object sender, EventArgs e)***

***{***

***if (currentQuestionIndex > 0)***

***{***

***currentQuestionIndex--;***

***LoadQuestion();***

***}***

***}***

Navigation backward is only allowed when currentQuestionIndex is greater than 0, ensuring the user doesn't navigate to a non-existent question.

**Problem**: I need to see how many questions I've gotten right and which option is correct. How will the scoring mechanism work?  
**Solution**: I will compare the selected option with the correct answer for each question. Each Question object has a property called CorrectOptionIndex, which stores the index of the correct option in the Options list.

**Code:**

***new Question***

***{***

***Text = "What is the capital of France?",***

***Options = new List<string> { "Berlin", "Madrid", "Paris", "Rome" },***

***CorrectOptionIndex = 2 // "Paris" is the correct option***

***}***

When the user clicks the Next button, iterations through the Controls collection are done to check which RadioButton is selected. The Tag property of the selected RadioButton holds the index of the selected option by using foreach:

***foreach (Control control in Controls)***

***{***

***if (control is RadioButton radioButton && radioButton.Checked)***

***{***

***int selectedOption = (int)radioButton.Tag;***

***...***

***}***

***}***

The Scoring Mechanism works by comparing the user's selected option (selectedOption) with the correct option (CorrectOptionIndex) for the current question. If they match, the score is incremented:

***if (selectedOption == questions[currentQuestionIndex].CorrectOptionIndex)***

***{***

***score++; // Increment the score for a correct answer***

***}***

At the end of the quiz, the ShowScore() method is called. This method clears the form and displays the user's total score out of the total number of questions:

***Label scoreLabel = new Label***

***{***

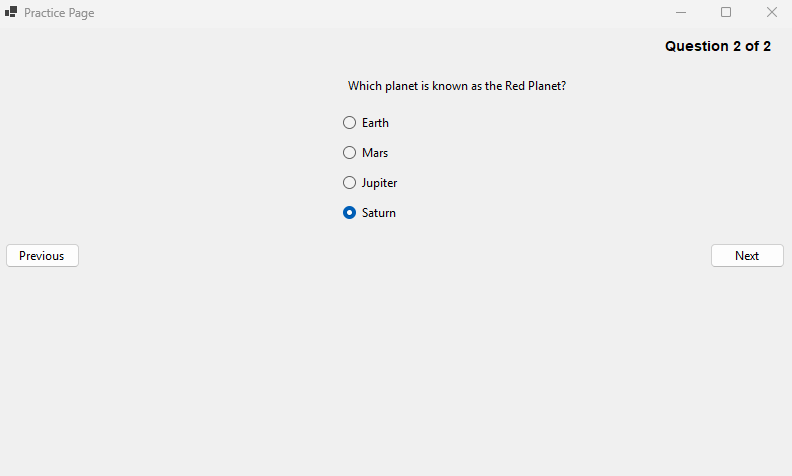
***Text = $"Quiz Completed!\nYour score: {score} out of {questions.Count}",***

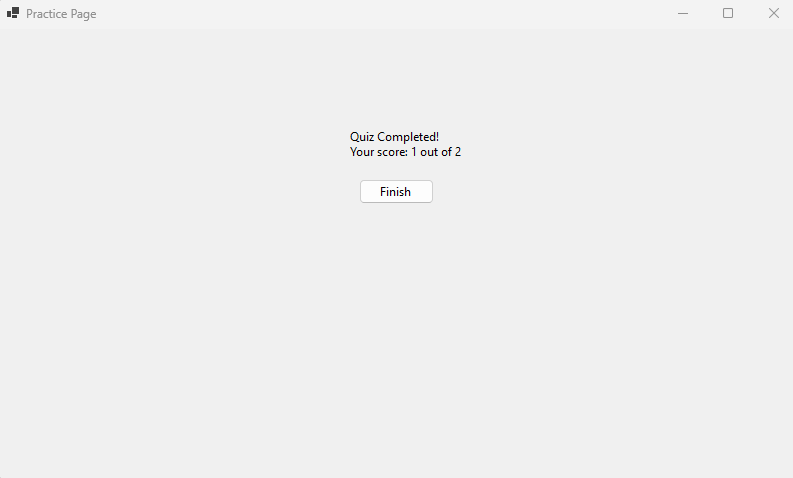
***AutoSize = true***

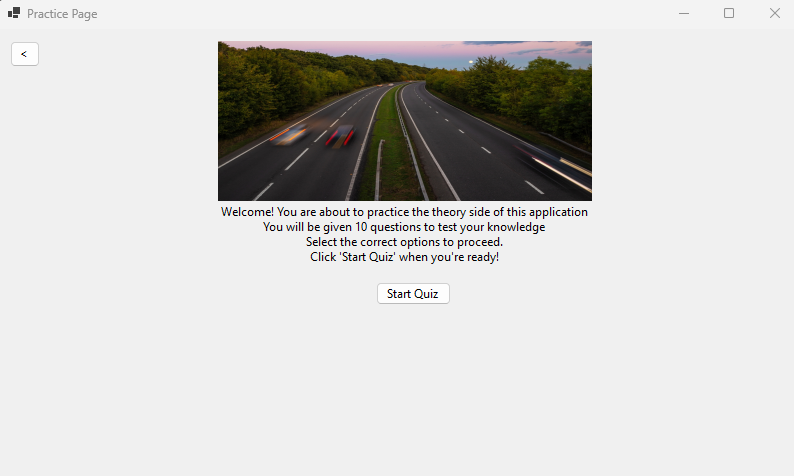
***};***

***A screenshot of a computer

Description automatically generated***

******

****

**Iteration 3:**  
  
**Problem:** How to Have Multiple Tests with different Questions?  
 **Solution:** I am implementing multiple tests with different questions by organizing the questions into a List<Test>, where each Test object contains its own set of questions.

**Code:**

***public class Test***

***{***

***public List<Question> Questions { get; set; } = new List<Question>();***

***}***

***public class Question***

***{***

***public string? Text { get; set; }***

***public List<string>? Options { get; set; }***

***public int CorrectOptionIndex { get; set; }***

***}***

This structure allows each Test to encapsulate its own unique set of questions.

**Problem:** How to store these tests?  
  
**Solution:** I am creating multiple tests by using a List<Test> to store multiple tests, with each Test initialized with a unique set of questions.

**Code:**

***tests = new List<Test>***

***{***

***new Test***

***{***

***Questions = new List<Question>***

***{***

***new Question***

***{***

***Text = "What is the capital of France?",***

***Options = new List<string> { "Berlin", "Madrid", "Paris", "Rome" },***

***CorrectOptionIndex = 2***

***},***

***new Question***

***{***

***Text = "Which planet is known as the Red Planet?",***

***Options = new List<string> { "Earth", "Mars", "Jupiter", "Saturn" },***

***CorrectOptionIndex = 1***

***},***

***}***

***},***

***new Test***

***{***

***Questions = new List<Question>***

***{***

***new Question***

***{***

***Text = "What is the largest ocean on Earth?",***

***Options = new List<string> { "Atlantic", "Indian", "Arctic", "Pacific" },***

***CorrectOptionIndex = 3***

***},***

***new Question***

***{***

***Text = "What is the square root of 64?",***

***Options = new List<string> { "6", "7", "8", "9" },***

***CorrectOptionIndex = 2***

***},***

***}***

***},***

***// More tests can be added here***

***};***

**Problem:** How to make the tests Random and Dynamic at the same time? Why To Choose Tests at Random?   
  
**Solution:** I am making the test selection random by using the Random class in the StartQuizButton\_Click method.

**Code:  
 *private void StartQuizButton\_Click(object? sender, EventArgs e)***

***{***

***Random random = new Random(); // Create a new instance of the Random class***

***selectedTest = tests[random.Next(tests.Count)]; // Randomly select a test***

***currentQuestionIndex = 0;***

***score = 0;***

***LoadQuestion(); // Start loading the first question***

***}***

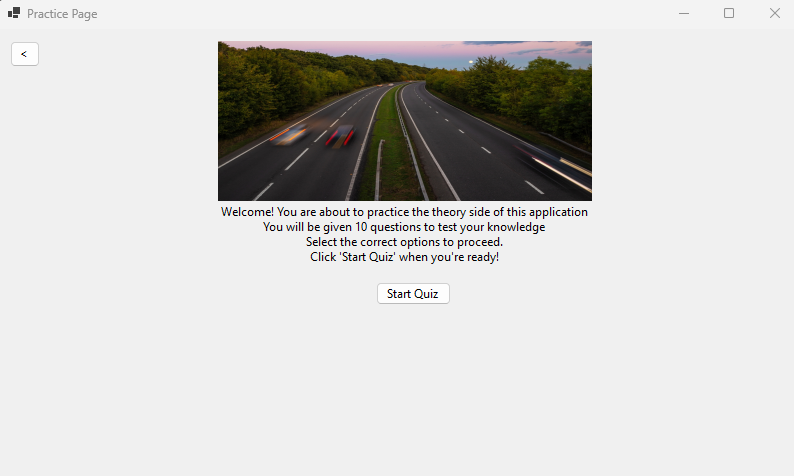
I use the Random class to generate pseudo-random numbers. The selection is done by random.Next(tests.Count), which generates a random integer between 0 (inclusive) and tests.Count (exclusive). This ensures that the index is valid within the bounds of the tests list, so a test is picked from the list.

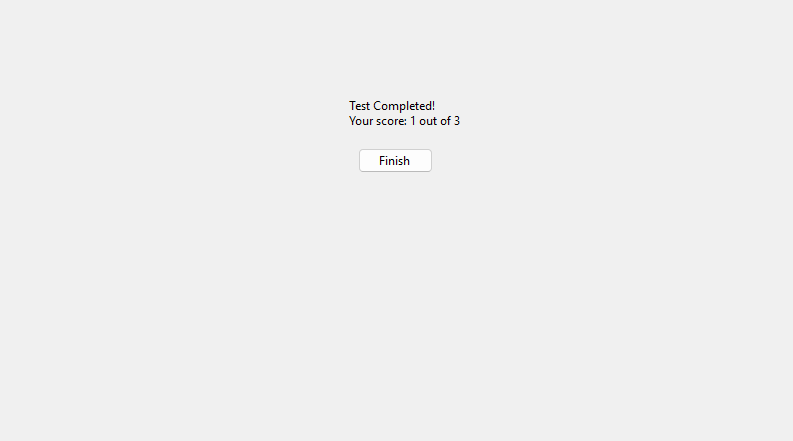
***selectedTest = tests[random.Next(tests.Count)];***

Dynamic Selection is ensured whenever the StartQuizButton\_Click method is executed (when the user starts the quiz), a new random test is selected.  
This ensures variety in the quiz experience every time it starts.

The Use of Choosing Random test is to ensure that the quiz feels fresh for the user by not always starting with the same test and to evenly distribute the selection across all available tests

**Obstacle**: When the test is End and the Quiz is started by the User it shows me the final result of the test rather than starting a new one



******

**Fix**: The issue arises because the currentQuestionIndex and score variables are not reset after completing the first test. When I click the "Start Test" button again, the previous values persist, causing the program to skip to the score display.

**Code:  
*private void StartQuizButton\_Click(object? sender, EventArgs e)***

***{***

***// Reset test-specific variables***

***currentQuestionIndex = 0;***

***score = 0;***

***// Randomly select one test***

***Random random = new Random();***

***selectedTest = tests[random.Next(tests.Count)];***

***InitializeTimer();***

***LoadQuestion();***

***}***

**Obstacle:** Facing the same test which the user has just attempted  
  
**Fix:** I will change the random selection logic to ensure I don't repeat the same test as the previous one. The previousTest variable is used to store the last selected test.

**Code:**

***private void StartQuizButton\_Click(object? sender, EventArgs e)***

***{***

***// Randomly select a test that is not the same as the previous one***

***Random random = new Random();***

***Test? newTest;***

***do***

***{***

***newTest = tests[random.Next(tests.Count)];***

***} while (newTest == previousTest); // Repeat until a different test is selected***

***previousTest = newTest; // Update the previous test***

***//...(Below Logic Remains the same)***

***}***

**Problem:** showing the user What test They are attempting at the moment.  
  
**Solution:** To display the name or index of the current test, adding a label that updates whenever a new test is selected

Modify the StartQuizButton\_Click method to update the label when a test is selected

***// Update the test name label int testIndex = tests.IndexOf(selectedTest) + 1; // Test index starts from 1 testNameLabel.Text = $"Test {testIndex}";***

Ensure the Label is Visible During the Quiz by adding the testNameLabel to the form in the LoadQuestion method to ensure it remains visible when navigating through questions:

***// Display the current question Question currentQuestion = selectedTest.Questions[currentQuestionIndex]; Label questionLabel = new Label { Text = currentQuestion.Text, AutoSize = true }; questionLabel.Location = new Point((ClientSize.Width - questionLabel.Width) / 2, 50); Controls.Add(questionLabel);*Iteration 4:**

**Update:** I am changing the logic of random tests to user choice, as this is a practice mode and that logic is more suitable for the mock test page. I am also replacing the StartQuiz class with StartTest.  
  
**Code:**  
 ***private void StartTest(int testIndex)***

***{***

***selectedTest = tests[testIndex];***

***currentQuestionIndex = 0;***

***score = 0;***

***// Update the test name label***

***testNameLabel.Text = $"Test {testIndex + 1}";***

***LoadQuestion();***

***}***

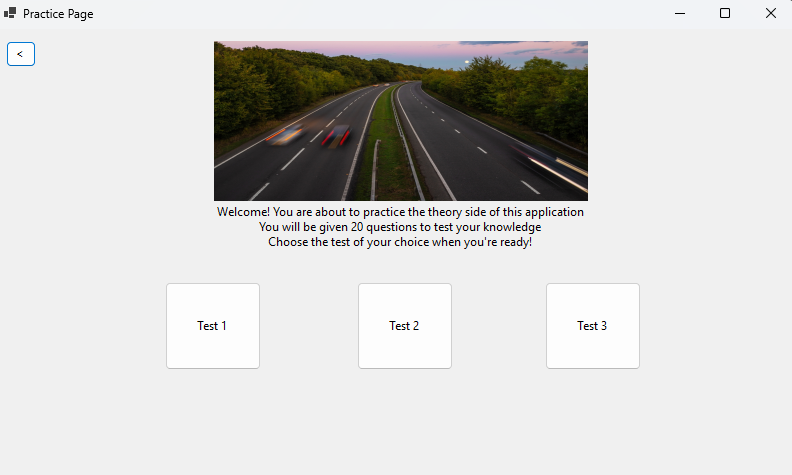
***// Update the test name label***

***testNameLabel.Text = $"Test {testIndex + 1}";***

***LoadQuestion();***

***}***

***}***

******

***Problem:*** What if I want to quit the practice? A Quit button will help me quit at any given time.

***Solution:*** I will add a Quit button, which, when clicked, takes me back to the initial screen.  
  
**Code:**  
 ***private void Quit\_Click(object sender, EventArgs e)***

***{***

***// Reset test-related variables***

***selectedTest = null;***

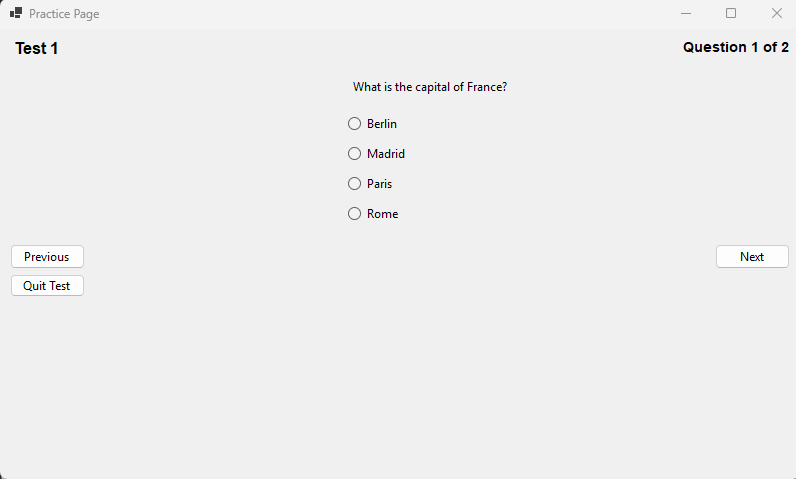
***previousTest = null;***

***currentQuestionIndex = 0;***

***score = 0;***

***// Return to the introduction screen***

***ShowIntroduction();***

***}  
  
*Problem:** What if the user wants to see the answer to the question right there?  
  
**Solution:** I will add a "Show Answer" button, which, when clicked, highlights the correct answer in green.

**Code:**  
 ***private void showAnswerButton\_Click(object sender, EventArgs e)***

***{***

***if (selectedTest == null || currentQuestionIndex >= selectedTest.Questions.Count)***

***{***

***MessageBox.Show("No question loaded to show the answer.");***

***return;***

***}***

***// Find the correct answer for the current question***

***Question currentQuestion = selectedTest.Questions[currentQuestionIndex];***

***int correctOptionIndex = currentQuestion.CorrectOptionIndex;***

***// Iterate through controls to find the RadioButton with the correct answer***

***foreach (Control control in Controls)***

***{***

***if (control is RadioButton radioButton && (int)radioButton.Tag == correctOptionIndex)***

***{***

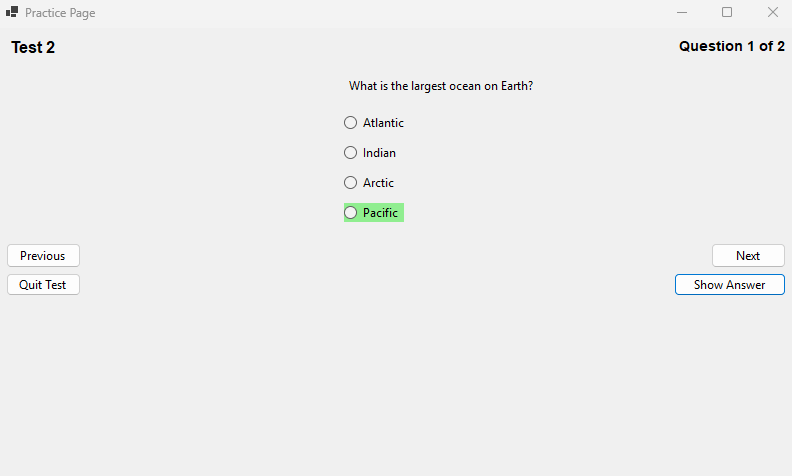
***radioButton.BackColor = Color.LightGreen; // Highlight the correct answer***

***break;***

***}***

***}***

***}***

******

**Iteration 5:**

Problem: How to add pictorial Questions?  
Solution: I will introduce a picture box to display the picture and set a proper location so that it doesn’t overlap with any button.  
  
**Code:**  
 ***// Initialize PictureBox for question images***

***questionPictureBox = new PictureBox***

***{***

***SizeMode = PictureBoxSizeMode.Zoom,***

***Size = new Size(300, 200), // Default size***

***Visible = false, // Initially hidden; only shown if there's an image***

***Location = new Point((ClientSize.Width - 300) / 2, 200) // Centered***

***};***

Edit load questions function to display and show the picture if it exists  
**Code: *// Display image if available***

***if (currentQuestion.Image != null)***

***{***

***questionPictureBox.Image = currentQuestion.Image;***

***questionPictureBox.Visible = true;***

***}***

***else***

***{***

***questionPictureBox.Visible = false;***

***}***Make the image property global and give proper path of the image as well   
**Code: *public Image? Image { get; set; } // New property for image  
 Image =Image.FromFile("E:\\CLIENT\\WindowsFormsApp1\\STOP\_SIGN\_PIC.jpg") // Add a valid image file path***

******

**Review:**

|  |  |
| --- | --- |
| **Success Criteria** | **Result** |
| **Does the system allow users to mark topics as completed?** |  |
| **Does the system persist the checkbox state even when the user navigates away from the page?** |  |
| **Can users view all signs related to a specific topic when clicking on it?** |  |
| **Are all traffic signs displayed with appropriate images and descriptions?** |  |
| **Is the progress of each topic updated immediately when the user marks it as completed or incomplete?** |  |
| **Can the user view flagged questions separately?** |  |
| **Does the progress page show a complete list of all completed topics?** |  |
| **Does the system display dynamic images and descriptions of signs from the correct topic?** |  |
| **Is the user interface intuitive for selecting topics and viewing signs?** |  |
| **Can the user easily toggle between topics and view progress?** |  |
| **Are checkboxes clearly visible and easy to interact with?** |  |
| **Can users navigate seamlessly between different pages (topics, progress, signs)?** |  |
| **Is there a way for users to track and view their overall progress on a single page?** |  |
| **Are the images of signs clear and appropriately sized?** |  |
| **Does the interface visually distinguish between completed and incomplete topics?** |  |
| **Can users undo or reset their progress on specific topics if needed?** |  |
| **Does the app load quickly without significant delays?** |  |
| **Is there any lag when switching between topics or viewing images?** |  |
| **Are sign images loaded efficiently without causing performance issues?** |  |
| **Does the app handle high volumes of data, such as a large number of signs or topics, without crashing?** |  |
| **Is the application using a global variable or persistent data structure for storing progress?** |  |
| **Is there error handling in place for missing images or incorrect data states?** |  |
| **Are images and descriptions dynamically loaded from the appropriate file location?** |  |
| **Is the progress stored in a way that can be saved or transferred between sessions?** |  |
| **Is the layout clean, with proper spacing and alignment for readability?** |  |
| **Is the text legible and appropriately styled for clarity?** |  |
| **Are progress indicators (checkmarks, completion percentages) clearly visible?** |  |
| **Is the theme of the page customizable or consistent with the overall project design?** |  |
| **Is the app interactive, allowing the user to engage with different features (checkboxes, navigation, image display)?** |  |
| **Does the system update in real-time when a topic is marked as complete?** |  |
| **Does the progress page update automatically to reflect changes in the checkbox states?** |  |
| **Can users interact with the images, such as zooming in for better visibility of signs?** |  |
| **Does the system handle missing or corrupted image files gracefully (e.g., showing a placeholder)?** |  |
| **Does the app prevent crashes or unexpected behavior from faulty inputs or actions?** |  |
| **Are there clear error messages or notifications if something goes wrong (e.g., missing data or images)?** |  |
| **Is the system flexible enough to add new topics or signs in the future?** |  |
| **Can additional content, such as multimedia (videos, animations), be added without major redesigns?** |  |
| **Can more complex features (like quizzes or assessments) be easily integrated into the current framework?** |  |

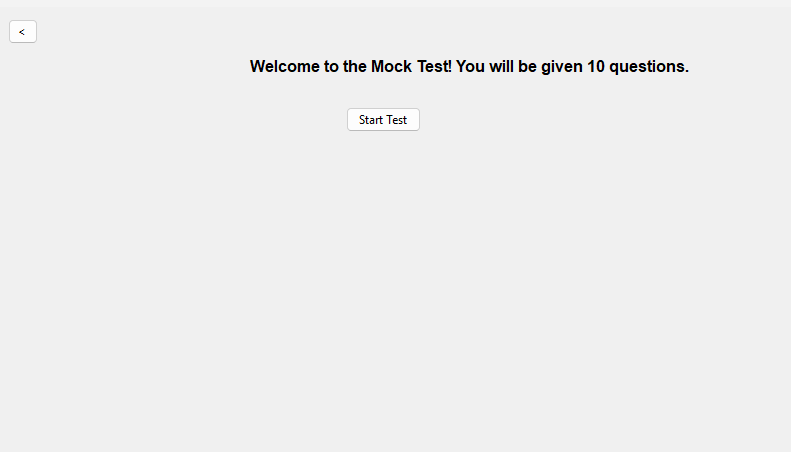
**Prototype 3 - Mock Test Page:**

**Overview:**

The Mock Test Page simulates a time-limited quiz, with a 57-minute countdown timer. Users can flag difficult questions for later review, and navigate through the test even without answering flagged questions. The test tracks answers, highlighting incorrect ones in red and correct ones in green in the results. A checkbox must be selected to confirm the user has read the instructions before starting the test. The page shows the time remaining and progress, and detailed feedback is provided on incorrect answers after the test ends.

**Development and Debugging:**

**Iteration 1:**



Everything is the same as the practice page except there is a timer on  
  
**Problem**: How to Implement a CountDown?

**Solution:** I will implement the countdown timer by initializing a Timer in the StartQuizButton\_Click method.

**Code:  
 *private void InitializeTimer()***

***{***

***if (quizTimer == null)***

***{***

***quizTimer = new Timer();***

***quizTimer.Interval = 1000; // 1-second intervals***

***quizTimer.Tick += QuizTimer\_Tick; // Event handler when timer ticks***

***}***

***timeRemaining = 30; // Set the initial countdown time to 30 seconds***

***quizTimer.Start(); // Start the timer***

***timerLabel = new Label***

***{***

***Text = $"Time Left: {timeRemaining} seconds",***

***AutoSize = true,***

***Font = new Font("Arial", 10, FontStyle.Bold),***

***ForeColor = Color.Red,***

***Location = new Point(10, 10)***

***};***

***Controls.Add(timerLabel); // Add the timer label to the form***

***}***

**Iteration 2:**

**Problem:** How will the timer would be Updated? This timer should Stop the Test when the timer runs out:

**Solution:** I will create the QuizTimer\_Tick method, which is triggered every second. It will update the timeRemaining value and display the remaining time on the timerLabel. If the timer reaches zero, the test will stop, and the score will be shown.

**Code: *private void QuizTimer\_Tick(object? sender, EventArgs e)***

***{***

***timeRemaining--; // Decrease the time remaining by 1 second***

***if (timeRemaining <= 0)***

***{***

***quizTimer.Stop(); // Stop the timer when it reaches 0***

***ShowScore(); // Show the score when the time runs out***

***}***

***else***

***{***

***timerLabel.Text = $"Time Left: {timeRemaining} seconds"; // Update the label***

***}***

***}***

The timer starts when the test begins with a countdown of 30 seconds. The quizTimer\_Tick event handler decreases the remaining time every second and updates the UI. Once the countdown reaches zero, the test is stopped by calling quizTimer.Stop(), and the ShowScore method is invoked to display the final score.

**Obstacle:** Severity Code Description Project File Line Suppression State Details Error (active) CS0104 'Timer' is an ambiguous reference between 'System.Windows.Forms.Timer' and 'System.Threading.Timer' WinFormsApp1 **Fix:** The error occurs because both System.Windows.Forms.Timer and System.Threading.Timer are referenced, and the compiler doesn't know which one to use. To resolve this ambiguity, I need to explicitly qualify the Timer class with its namespace. For example, if I'm working with a Windows Forms application, I will use:

System.Windows.Forms.Timer quizTimer = new System.Windows.Forms.Timer();

This will ensure that the correct Timer class is used in the context of my Windows Forms application.

**private System.Windows.Forms.Timer quizTimer; // Timer object**

In the class of IntializeTimer we need to change the qualification  
 **Code:**

**private void InitializeTimer()**

**{**

**if (quizTimer == null)**

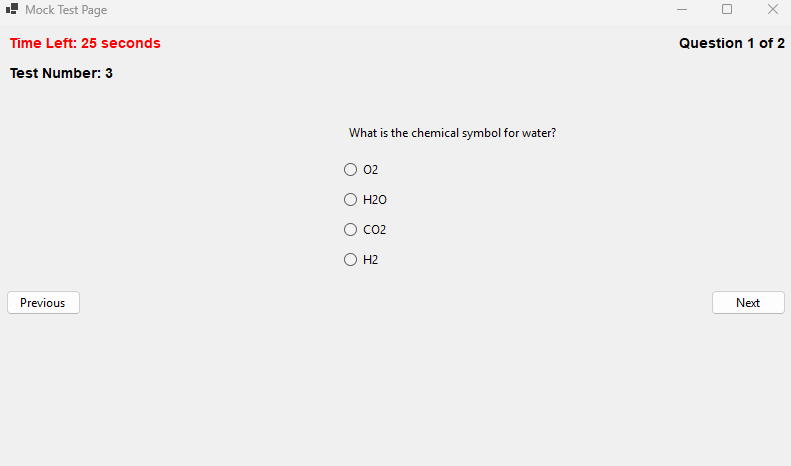
**{**

**quizTimer = new System.Windows.Forms.Timer(); // Explicit namespace usage**

**quizTimer.Interval = 1000; // 1 second intervals**

**quizTimer.Tick += QuizTimer\_Tick;**

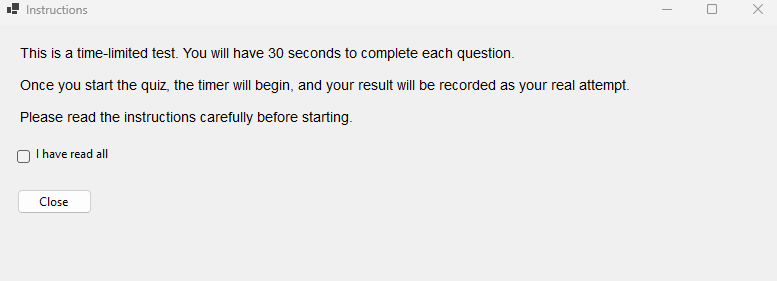
**}**



**Iteration 3:**

**Problem:** I need to make sure that I know this is a time-limited test and that I can't start until I have read the instructions carefully. To achieve this, I will clearly display a message stating that the test is time-limited, and ensure the "Start Test" button is only enabled once I check the "I have read all of the instructions" checkbox.

**Solution**: A new form is opened when the "Mock Test" button is clicked on the main page, containing the instructions that I must read before I can start the quiz. The Mock Test page form won't open until I have checked the "I have read all of the instructions" checkbox.



**Iteration 4:**  
Update: Increase the Timer from 30 sec to 57 Minutes by changing the Initialize Timer Code  
**Code:  
 *private void InitializeTimer()***

***{***

***if (quizTimer == null)***

***{***

***quizTimer = new System.Windows.Forms.Timer();***

***quizTimer.Interval = 1000; // 1-second intervals***

***quizTimer.Tick += QuizTimer\_Tick;***

***}***

***timeRemaining = 57 \* 60; // 57 minutes in seconds***

***quizTimer.Start();  
  
 ...//remains the same***

***}***

Added new function to change the format  
**Code:**  
 ***private string FormatTime(int seconds)***

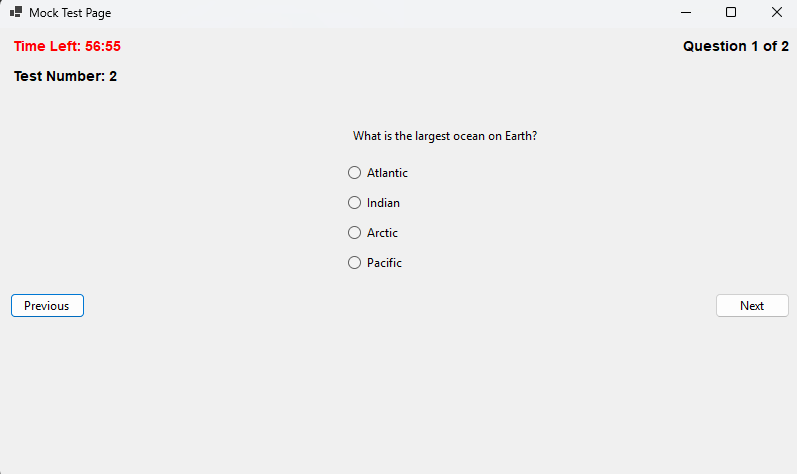
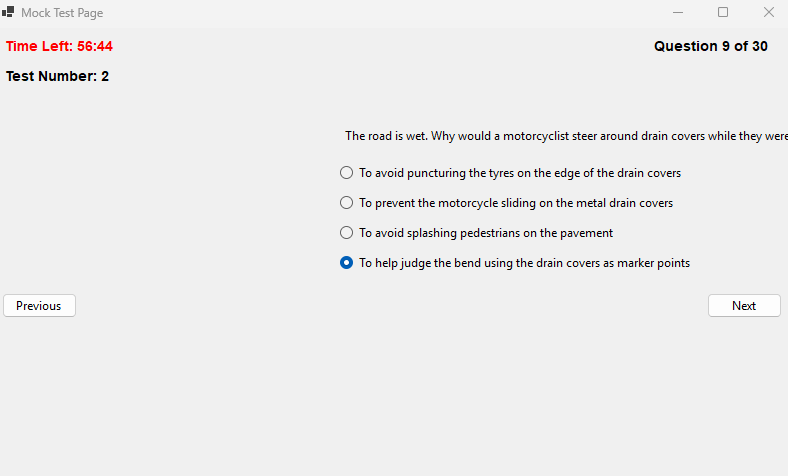
***{***

***int minutes = seconds / 60;***

***int remainingSeconds = seconds % 60;***

***return $"{minutes:D2}:{remainingSeconds:D2}";***

***}***

******Adding real traffic questions by putting 30 Questions in 3 mock test each  
  


**Iteration 5:**

**Problem:** If a user wants to mark a question they dont understand how would they?  
  
**Solution**: I will introduce a "Flag" button that allows me to flag questions I want to know the answer to. These flagged questions will then be displayed on the progress page. I can flag and unflag questions as needed. Additionally, I will introduce a list to save the selected flagged questions for easy tracking.  
 ***private List<int> flaggedQuestions = new List<int>();***  
  
Make a new Class for it  
  
**Code:  
 *private void Flag\_Click(object? sender, EventArgs e)***

***{***

***// Toggle flag for the current question***

***if (flaggedQuestions.Contains(currentQuestionIndex))***

***{***

***flaggedQuestions.Remove(currentQuestionIndex);***

***flagButton.Text = "Flag";***

***}***

***else***

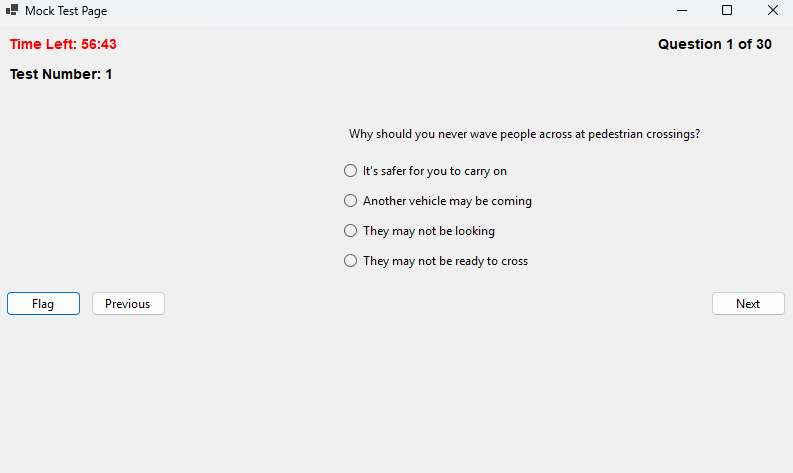
***{***

***flaggedQuestions.Add(currentQuestionIndex);***

***flagButton.Text = "Unflag";***

***}***

***}***

****

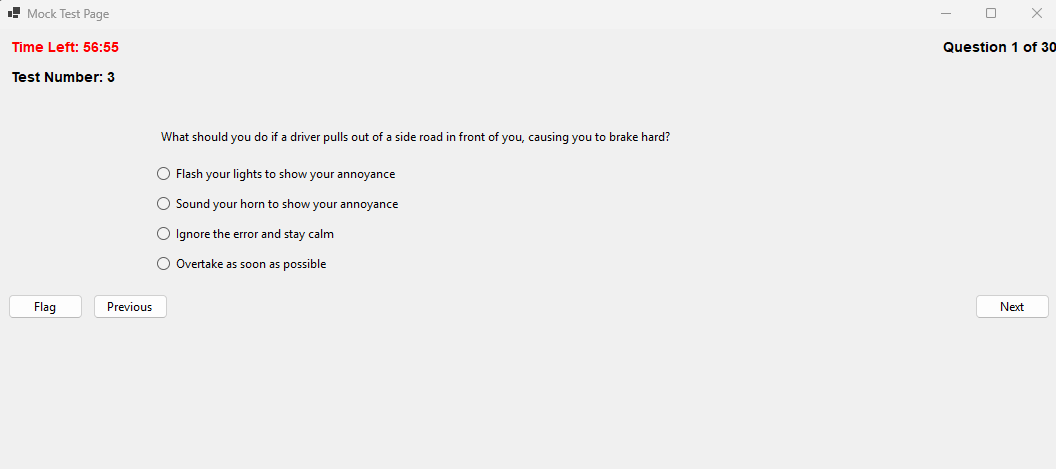
Made Questions Global as well so it could be accessed by the Progress page(Flagged questions Form)  
  
 ***public static List<List<Question>> AllTests { get; set; } = new List<List<Question>>***

***{***

***new List<Question>***

***{***

***new Question{Text = "Why should you never wave people across at pedestrian crossings?",Options = new List<string>{"It's safer for you to carry on","Another vehicle may be coming","They may not be looking","They may not be ready to cross"},CorrectOptionIndex = 1},***

**  
Problem:** when the user is flagging the question they are still forced to select an option to move ahead  
  
**Solution:** Change the Next Click Class to include an if which checks if the question is flagged or not ***private void Next\_Click(object? sender, EventArgs e)***

***{***

***// Check if the current question is flagged***

***if (flaggedQuestions.Contains(currentQuestionIndex))***

***{***

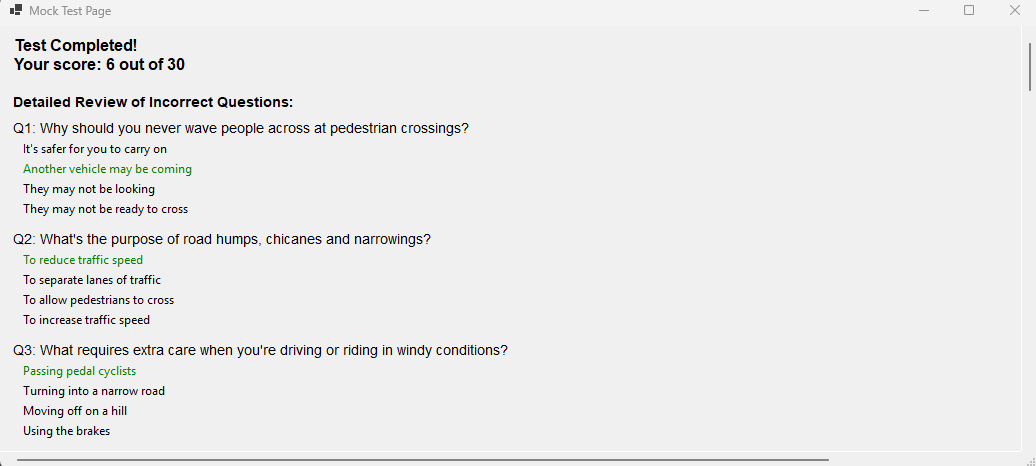
***// Proceed to the next question without requiring an answer***

***currentQuestionIndex++;***

***LoadQuestion();***

***return;***

***}***

**Update:** The Show screen also needs to show what questions the user has gotten right or wrong  
 **Problem:** Showing all the questions without showing user choice

**Solution:** Make and implement a Dictionary which stores the users choice

**Code:**

***private Dictionary<int, int> userAnswers = new Dictionary<int, int>();***

implement it in next click class so the users choice can be stored

**Code:  
 *// Store the user's answer***

***userAnswers[currentQuestionIndex] = selectedOption;***I will change the ShowScore class to incorporate all of the new changes by creating a loop that goes through all of the incorrect questions answered by me. In this loop, red will represent the user’s choice that was wrong, while green will represent the correct answer. This will provide a clear visual representation of the mistakes made and the correct answers.

**Code:  
*// Display detailed results for incorrect answers***

***Label reviewLabel = new Label***

***{***

***Text = "Review of Incorrect Questions:",***

***AutoSize = true,***

***Font = new Font("Arial", 10, FontStyle.Bold),***

***Location = new Point(10, yPosition)***

***};***

***scrollablePanel.Controls.Add(reviewLabel);***

***yPosition = reviewLabel.Bottom + 10;***

***// Loop through the questions and display only incorrect answers***

***for (int i = 0; i < selectedTest.Count; i++)***

***{***

***Question question = selectedTest[i];***

***int userAnswer = -1; // Default value for no answer***

***bool isCorrect = false;***

***// Determine user's selected answer***

***if (userAnswers.ContainsKey(i))***

***{***

***userAnswer = userAnswers[i];***

***isCorrect = userAnswer == question.CorrectOptionIndex;***

***}***

***// Skip correctly answered questions***

***if (isCorrect)***

***continue;***

***// Display the question***

***Label questionLabel = new Label***

***{***

***Text = $"Q{i + 1}: {question.Text}",***

***AutoSize = true,***

***Font = new Font("Arial", 10),***

***Location = new Point(10, yPosition)***

***};***

***scrollablePanel.Controls.Add(questionLabel);***

***yPosition = questionLabel.Bottom + 5;***

***// Display the options***

***for (int j = 0; j < question.Options.Count; j++)***

***{***

***Label optionLabel = new Label***

***{***

***Text = question.Options[j],***

***AutoSize = true,***

***Location = new Point(20, yPosition)***

***};***

***// Highlight the user's incorrect answer in red***

***if (j == userAnswer)***

***{***

***optionLabel.ForeColor = Color.Red;***

***}***

***// Highlight the correct answer in green***

***else if (j == question.CorrectOptionIndex)***

***{***

***optionLabel.ForeColor = Color.Green;***

***}***

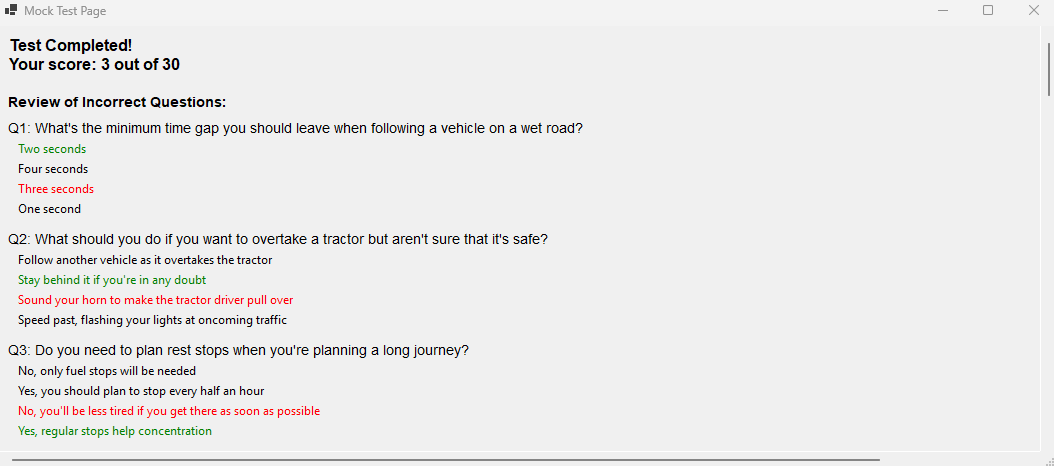
***scrollablePanel.Controls.Add(optionLabel);***

***yPosition = optionLabel.Bottom + 5;***

***}***

***yPosition += 10; // Add spacing between questions***

***}***

****

**Review:**

|  |  |
| --- | --- |
| **Success Criteria** | **Result** |
| **Does the system allow users to mark topics as completed?** |  |
| **Does the system persist the checkbox state even when the user navigates away from the page?** |  |
| **Can users view all signs related to a specific topic when clicking on it?** |  |
| **Are all traffic signs displayed with appropriate images and descriptions?** |  |
| **Is the progress of each topic updated immediately when the user marks it as completed or incomplete?** |  |
| **Can the user view flagged questions separately?** |  |
| **Does the progress page show a complete list of all completed topics?** |  |
| **Does the system display dynamic images and descriptions of signs from the correct topic?** |  |
| **Is the user interface intuitive for selecting topics and viewing signs?** |  |
| **Can the user easily toggle between topics and view progress?** |  |
| **Are checkboxes clearly visible and easy to interact with?** |  |
| **Can users navigate seamlessly between different pages (topics, progress, signs)?** |  |
| **Is there a way for users to track and view their overall progress on a single page?** |  |
| **Are the images of signs clear and appropriately sized?** |  |
| **Does the interface visually distinguish between completed and incomplete topics?** |  |
| **Can users undo or reset their progress on specific topics if needed?** |  |
| **Does the app load quickly without significant delays?** |  |
| **Is there any lag when switching between topics or viewing images?** |  |
| **Are sign images loaded efficiently without causing performance issues?** |  |
| **Does the app handle high volumes of data, such as a large number of signs or topics, without crashing?** |  |
| **Is the application using a global variable or persistent data structure for storing progress?** |  |
| **Is there error handling in place for missing images or incorrect data states?** |  |
| **Are images and descriptions dynamically loaded from the appropriate file location?** |  |
| **Is the progress stored in a way that can be saved or transferred between sessions?** |  |
| **Is the layout clean, with proper spacing and alignment for readability?** |  |
| **Is the text legible and appropriately styled for clarity?** |  |
| **Are progress indicators (checkmarks, completion percentages) clearly visible?** |  |
| **Is the theme of the page customizable or consistent with the overall project design?** |  |
| **Is the app interactive, allowing the user to engage with different features (checkboxes, navigation, image display)?** |  |
| **Does the system update in real-time when a topic is marked as complete?** |  |
| **Does the progress page update automatically to reflect changes in the checkbox states?** |  |
| **Can users interact with the images, such as zooming in for better visibility of signs?** |  |
| **Does the system handle missing or corrupted image files gracefully (e.g., showing a placeholder)?** |  |
| **Does the app prevent crashes or unexpected behavior from faulty inputs or actions?** |  |
| **Are there clear error messages or notifications if something goes wrong (e.g., missing data or images)?** |  |
| **Is the system flexible enough to add new topics or signs in the future?** |  |
| **Can additional content, such as multimedia (videos, animations), be added without major redesigns?** |  |
| **Can more complex features (like quizzes or assessments) be easily integrated into the current framework?** |  |

**Prototype 4 - Progress Page:**

**Overview:**

The project tracks and displays user progress in mock and practice tests using dynamic progress bars. It calculates progress based on the number of correct answers and the total questions per test, with scores stored globally. The interface also allows for flagging questions and displays a timer to track completed topics, updating the user's progress continuously. This system enhances user engagement by providing real-time feedback on test performance and completed content.

**Development and Debugging:**

**Iteration 1:**

**Problem:** How can I track the progress of the user, including how many tests they have taken and their scores on those tests (graphically)? How will the score percentage be calculated?

**Solution:** To implement a graphical progress bar for the progress page, I can use the ProgressBar control in Windows Forms and a Label to display the score percentage. I can create a dynamic progress bar for each test, where the value of the progress bar reflects the user's score as a percentage. I will calculate the score percentage based on the number of correct answers (e.g., 10 questions per test, or adjust this to the actual number of questions in the test).

Next to each progress bar, a Score Display label will show the score percentage or indicate if the test has not been attempted. When navigating to the Progress\_Page, I will pass a dictionary containing the scores for each test to ensure all progress is displayed correctly. **Code:**

***private void InitializeProgressBar()***

***{***

***Controls.Clear();***

***Label titleLabel = new Label***

***{***

***Text = "Progress Overview",***

***Font = new Font("Arial", 14, FontStyle.Bold),***

***AutoSize = true,***

***Location = new Point((ClientSize.Width - 200) / 2, 20)***

***};***

***Controls.Add(titleLabel);***

***int yPosition = titleLabel.Bottom + 30;***

***// Display progress bar***

***ProgressBar progressBar = new ProgressBar***

***{***

***Minimum = 0,***

***Maximum = 100,***

***Value = testScores.ContainsKey(testNumber)***

***? (testScores[testNumber] \* 100) / 10***

***: 0, // Assuming each test has 10 questions***

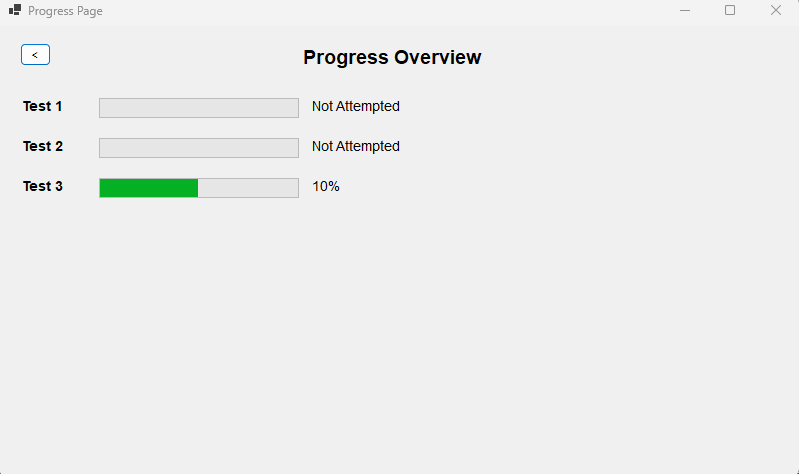
***Size = new Size(200, 20),***

***Location = new Point(100, yPosition)***

***};***

***Controls.Add(progressBar);***

**Problem:** How to Integrate it with the Mock test page?  
  
**Solution:** To integrate with my MockTest\_Page, updating the ShowScore method so it can save the score in a global object. Then, pass the scores to the Progress\_Page.  
  
**Code:**  
***private Dictionary<int, int> testScores = new Dictionary<int, int>(); // Global dictionary to track scores***  
***// Save the score for the current test if (!testScores.ContainsKey(testNumber)) testScores[testNumber] = score;***



**Iteration 2:**

**Update**: To Dynamically calculate the progress bar based on the number of questions in the test  
  
***Code:  
Dictionary<int, int> testQuestions = new Dictionary<int, int>***

***{***

***{ 1, 10 }, // Test 1 has 10 questions***

***{ 2, 8 }, // Test 2 has 8 questions***

***{ 3, 12 } // Test 3 has 12 questions***

***};***

***// Display progress bar***

***ProgressBar progressBar = new ProgressBar***

***{***

***Minimum = 0,***

***Maximum = 100,***

***Value = testScores.ContainsKey(testNumber) && testQuestions.ContainsKey(testNumber)***

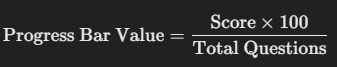
***? (testScores[testNumber] \* 100) / testQuestions[testNumber]***

***: 0, // Default to 0 if the test number is not found***

***Size = new Size(200, 20),***

***Location = new Point(100, yPosition)***

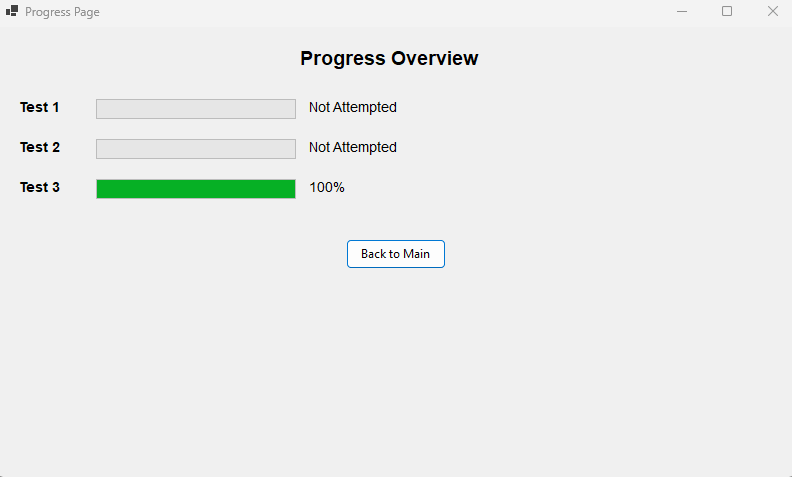
***};***

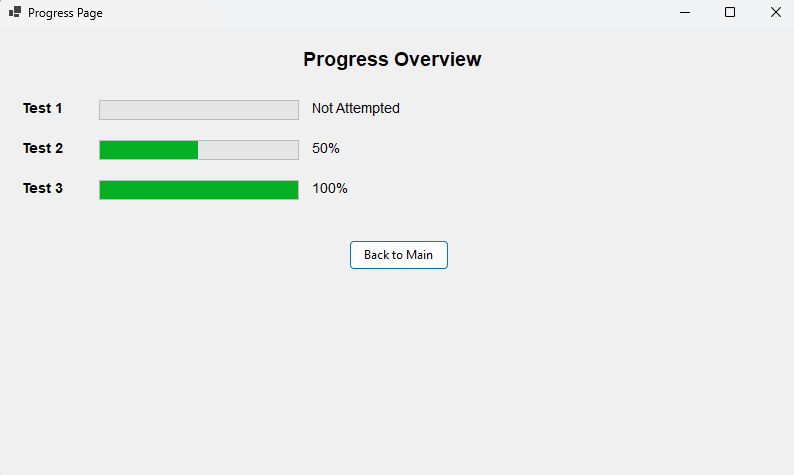
If the test score for the current test (testNumber) exists in testScores and the total number of questions for that test exists in testQuestions, calculate the percentage:   
  
Otherwise, default to 0. This logic ensures the progress bar adjusts according to the actual number of questions in the test rather than a fixed assumption of 10.  
  
**Problem:** The text next to the progress bar is 10%/20% it is being calculated by Multiplying the score by 10,Assuming that the number of question is 10  
the percentage should be calculated with the dynamic question logic now  
 ***Text = testScores.ContainsKey(testNumber)***

***? $"{testScores[testNumber] \* 10}%"***

***: "Not Attempted",***

**Solution:** I can use a Label that dynamically calculates and displays the percentage based on the testScores and testQuestions.   
Calculate Percentage percentage is dynamically calculated now using the formula ****

If the score or total number of questions is missing, it defaults to 0%.  
The progress bar value is now set using the percentage.  
A Label is added to display the percentage next to the progress bar.  
The logic works for any number of questions and any score.  




**Iteration 3:**

**Update:** It only shows the progress of the mock test right now I want to change so that so it also shows the progress of practice page  
I have to add A variable to score practice score globally  
**Code:**  
 ***public static class GlobalData***

***{***

***public static Dictionary<int, int> TestScores { get; set; } = new Dictionary<int, int>();***

***public static Dictionary<int, int> PracticeScores { get; set; } = new Dictionary<int, int>(); // Practice test scores***

***}***I then have to store the scores aswell **Code:  
 *private void ShowScore()***

***{***

***// Clear the form***

***Controls.Clear();***

***// Save the score globally with the test index***

***if (selectedTest != null)***

***{***

***int testIndex = tests.IndexOf(selectedTest) + 1; // Test index starts from 1***

***GlobalData.PracticeScores[testIndex] = score; // Save the score for this test***

***}***I also changed the label to better identify the tests and show progress under  
**Code:**

***// Add mock test progress***

***Label mockProgressLabel = new Label***

***{***

***Text = "Mock Test Progress",***

***..//remains the same***

***};***

***foreach (var testNumber in testQuestions.Keys)***

***{***

***AddProgressRow("Test", testNumber, mockTestScores, ref yPosition);***

***}***

***// Add practice test progress***

***Label practiceProgressLabel = new Label***

***{***

***Text = "Practice Test Progress",***

***..//remains the same***

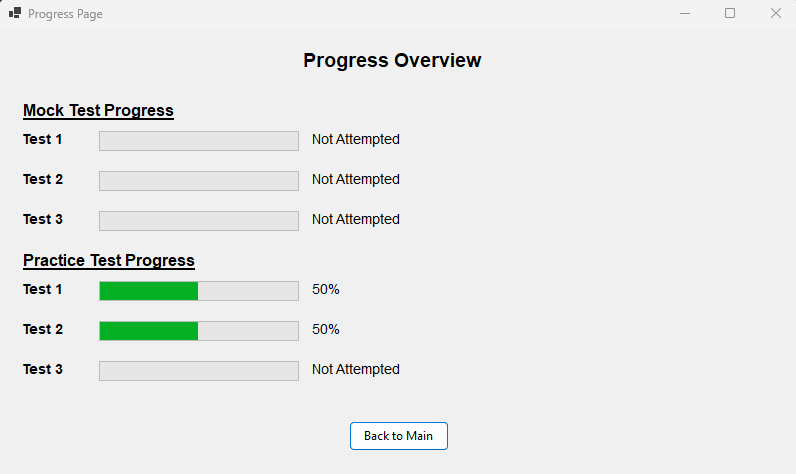
***};***

***foreach (var testNumber in testQuestions.Keys)***

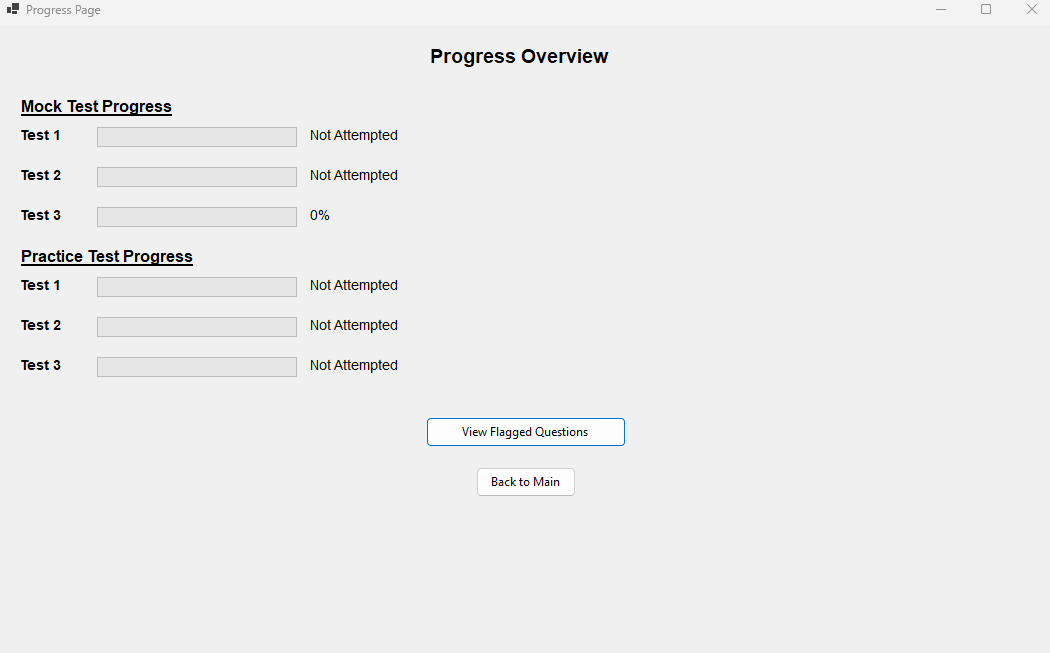
***{***

***AddProgressRow("Test", testNumber, practiceTestScores, ref yPosition);***

***}***

****Iteration 4:**

Update: Show the Flagged Questions here under a Flag Question Button,This Button would open a new form on which every flagged question alongside its answer would be there.

******

**Iteration 5:**

**Problem:** After adding the checkbox for completed topics, how can I track what the user has completed? And if they go back and complete it later, how will that be updated?

**Solution**: I will introduce a timer that checks what the user has selected to be completed. This timer will run every second. If the user hasn't completed any topics, it will display "None."  
add a timer and the topics completed  
  
**Code:**

***topicsCompletedLabel = new Label***

***{***

***Text = "Topics Completed: None",***

***Font = new Font("Arial", 12, FontStyle.Bold),***

***AutoSize = true,***

***Location = new Point(20, yPosition )***

***};***

***Controls.Add(topicsCompletedLabel);***

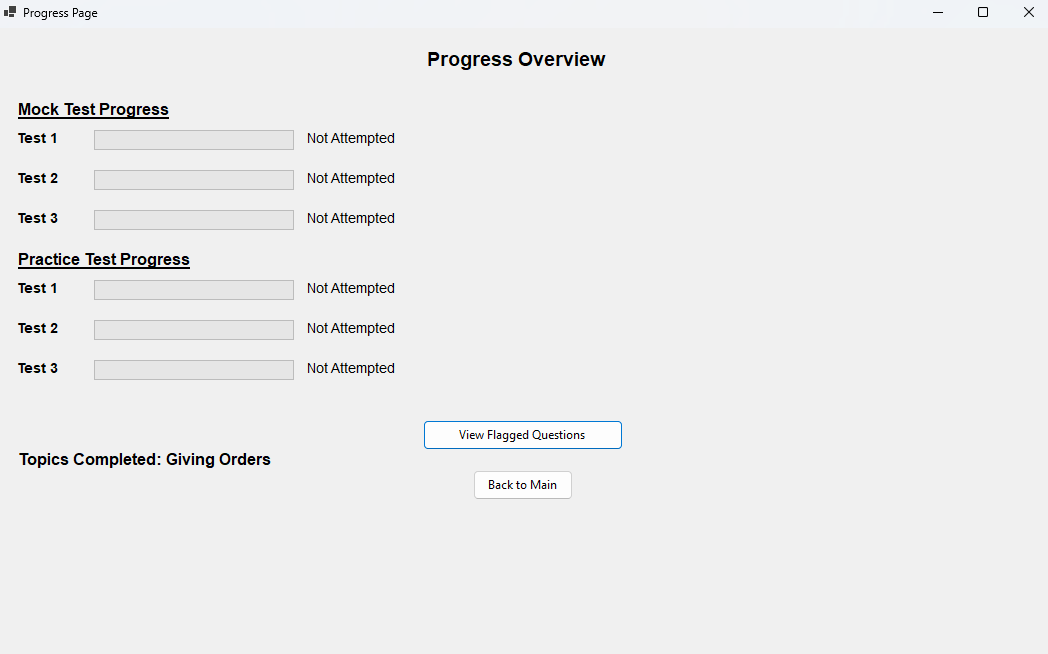
***System.Windows.Forms.Timer updateTimer = new System.Windows.Forms.Timer***

***{***

***Interval = 1000 // Check for updates every second***

***};***

***updateTimer.Tick += (s, e) => UpdateTopicsCompleted();***

***updateTimer.Start();***  
****Review:**

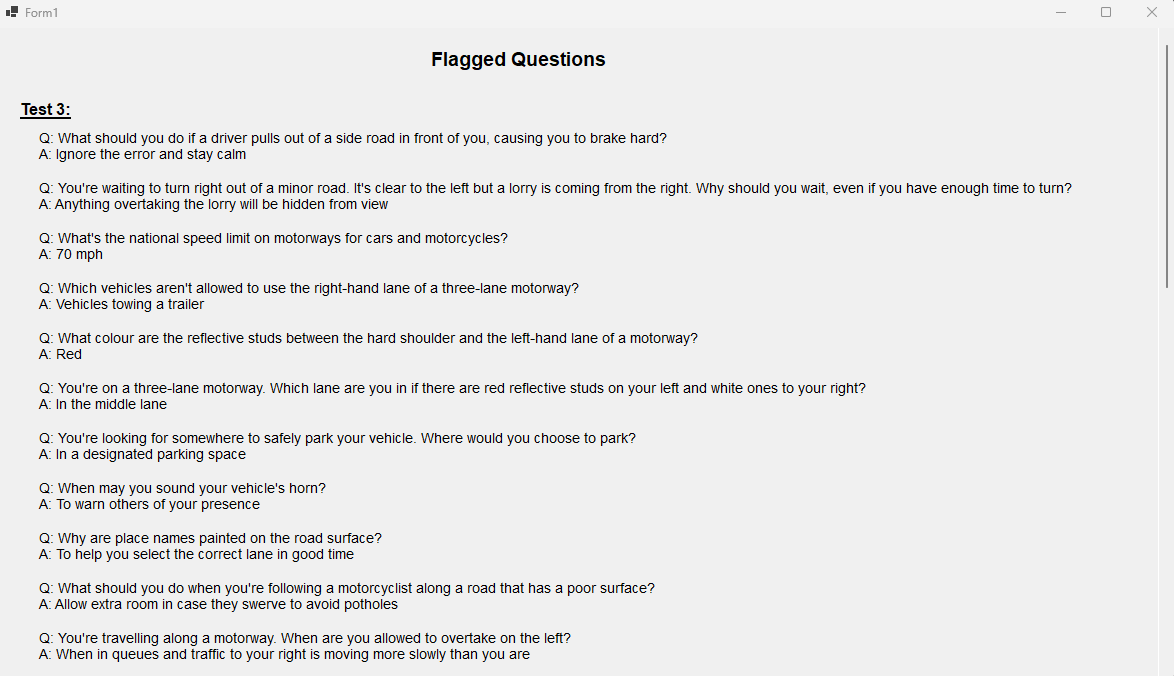
|  |  |
| --- | --- |
| **Success Criteria** | **Result** |
| **Does the system allow users to mark topics as completed?** |  |
| **Does the system persist the checkbox state even when the user navigates away from the page?** |  |
| **Can users view all signs related to a specific topic when clicking on it?** |  |
| **Are all traffic signs displayed with appropriate images and descriptions?** |  |
| **Is the progress of each topic updated immediately when the user marks it as completed or incomplete?** |  |
| **Can the user view flagged questions separately?** |  |
| **Does the progress page show a complete list of all completed topics?** |  |
| **Does the system display dynamic images and descriptions of signs from the correct topic?** |  |
| **Is the user interface intuitive for selecting topics and viewing signs?** |  |
| **Can the user easily toggle between topics and view progress?** |  |
| **Are checkboxes clearly visible and easy to interact with?** |  |
| **Can users navigate seamlessly between different pages (topics, progress, signs)?** |  |
| **Is there a way for users to track and view their overall progress on a single page?** |  |
| **Are the images of signs clear and appropriately sized?** |  |
| **Does the interface visually distinguish between completed and incomplete topics?** |  |
| **Can users undo or reset their progress on specific topics if needed?** |  |
| **Does the app load quickly without significant delays?** |  |
| **Is there any lag when switching between topics or viewing images?** |  |
| **Are sign images loaded efficiently without causing performance issues?** |  |
| **Does the app handle high volumes of data, such as a large number of signs or topics, without crashing?** |  |
| **Is the application using a global variable or persistent data structure for storing progress?** |  |
| **Is there error handling in place for missing images or incorrect data states?** |  |
| **Are images and descriptions dynamically loaded from the appropriate file location?** |  |
| **Is the progress stored in a way that can be saved or transferred between sessions?** |  |
| **Is the layout clean, with proper spacing and alignment for readability?** |  |
| **Is the text legible and appropriately styled for clarity?** |  |
| **Are progress indicators (checkmarks, completion percentages) clearly visible?** |  |
| **Is the theme of the page customizable or consistent with the overall project design?** |  |
| **Is the app interactive, allowing the user to engage with different features (checkboxes, navigation, image display)?** |  |
| **Does the system update in real-time when a topic is marked as complete?** |  |
| **Does the progress page update automatically to reflect changes in the checkbox states?** |  |
| **Can users interact with the images, such as zooming in for better visibility of signs?** |  |
| **Does the system handle missing or corrupted image files gracefully (e.g., showing a placeholder)?** |  |
| **Does the app prevent crashes or unexpected behavior from faulty inputs or actions?** |  |
| **Are there clear error messages or notifications if something goes wrong (e.g., missing data or images)?** |  |
| **Is the system flexible enough to add new topics or signs in the future?** |  |
| **Can additional content, such as multimedia (videos, animations), be added without major redesigns?** |  |
| **Can more complex features (like quizzes or assessments) be easily integrated into the current framework?** |  |

**Prototype 5 - Flagged Questions Page:**

**Overview:**

In Iteration 1, the goal is to display all flagged questions from a test on a separate page. This involves iterating through a collection of flagged questions, retrieving the question text and its correct answer, and dynamically displaying them in a scrollable panel for easy viewing. Each flagged question is represented by a label showing its text and the correct answer, and the layout adjusts the position of each question as it is added to the panel.

**Development and Debugging:  
Iteration 1:**  
For getting all the flagged questions and showing them on one page



**Code: *foreach (int questionIndex in test.Value)***

***{***

***Question question = GlobalData.AllTests[test.Key - 1][questionIndex]; // Get the flagged question***

***Label questionLabel = new Label***

***{***

***Text = $"Q: {question.Text}\nA: {question.Options[question.CorrectOptionIndex]}",***

***Font = new Font("Arial", 10, FontStyle.Regular),***

***AutoSize = true,***

***Location = new Point(40, yPosition)***

***};***

***scrollablePanel.Controls.Add(questionLabel);***

***yPosition += 50;***

***}*Review:**

|  |  |
| --- | --- |
| **Success Criteria** | **Result** |
| **Does the system allow users to mark topics as completed?** |  |
| **Does the system persist the checkbox state even when the user navigates away from the page?** |  |
| **Can users view all signs related to a specific topic when clicking on it?** |  |
| **Are all traffic signs displayed with appropriate images and descriptions?** |  |
| **Is the progress of each topic updated immediately when the user marks it as completed or incomplete?** |  |
| **Can the user view flagged questions separately?** |  |
| **Does the progress page show a complete list of all completed topics?** |  |
| **Does the system display dynamic images and descriptions of signs from the correct topic?** |  |
| **Is the user interface intuitive for selecting topics and viewing signs?** |  |
| **Can the user easily toggle between topics and view progress?** |  |
| **Are checkboxes clearly visible and easy to interact with?** |  |
| **Can users navigate seamlessly between different pages (topics, progress, signs)?** |  |
| **Is there a way for users to track and view their overall progress on a single page?** |  |
| **Are the images of signs clear and appropriately sized?** |  |
| **Does the interface visually distinguish between completed and incomplete topics?** |  |
| **Can users undo or reset their progress on specific topics if needed?** |  |
| **Does the app load quickly without significant delays?** |  |
| **Is there any lag when switching between topics or viewing images?** |  |
| **Are sign images loaded efficiently without causing performance issues?** |  |
| **Does the app handle high volumes of data, such as a large number of signs or topics, without crashing?** |  |
| **Is the application using a global variable or persistent data structure for storing progress?** |  |
| **Is there error handling in place for missing images or incorrect data states?** |  |
| **Are images and descriptions dynamically loaded from the appropriate file location?** |  |
| **Is the progress stored in a way that can be saved or transferred between sessions?** |  |
| **Is the layout clean, with proper spacing and alignment for readability?** |  |
| **Is the text legible and appropriately styled for clarity?** |  |
| **Are progress indicators (checkmarks, completion percentages) clearly visible?** |  |
| **Is the theme of the page customizable or consistent with the overall project design?** |  |
| **Is the app interactive, allowing the user to engage with different features (checkboxes, navigation, image display)?** |  |
| **Does the system update in real-time when a topic is marked as complete?** |  |
| **Does the progress page update automatically to reflect changes in the checkbox states?** |  |
| **Can users interact with the images, such as zooming in for better visibility of signs?** |  |
| **Does the system handle missing or corrupted image files gracefully (e.g., showing a placeholder)?** |  |
| **Does the app prevent crashes or unexpected behavior from faulty inputs or actions?** |  |
| **Are there clear error messages or notifications if something goes wrong (e.g., missing data or images)?** |  |
| **Is the system flexible enough to add new topics or signs in the future?** |  |
| **Can additional content, such as multimedia (videos, animations), be added without major redesigns?** |  |
| **Can more complex features (like quizzes or assessments) be easily integrated into the current framework?** |  |

**Prototype 6 - Traffic Signs Page:**

**Overview:**

This prototype aims to enhance a user's learning experience by tracking progress on traffic signs topics and displaying relevant sign images with descriptions. It includes two main features:

1. **Progress Tracking:**Users can mark topics as completed via checkboxes. Their progress is stored in a persistent dictionary, ensuring that their selections are remembered even when they navigate away from the page. The system dynamically updates the checkbox states to reflect the user's progress.
2. **Sign Study Interface:**For each traffic signs topic, a dedicated page displays related signs with images and descriptions. A DataGridView is utilized to show these signs in a scrollable format, with an image column and a text column for additional information. This interface allows users to visually study and learn the traffic signs.

The prototype efficiently tracks user progress and provides an interactive study interface for traffic signs, helping users easily learn and monitor their advancement.

**Development and Debugging:**

**Iteration 1:**  
**Problem:** How to know what topics are completed by the user?  
**Solution:** I will add a checkbox under every topic so that when the user completes a topic, they can check it. This allows the user to track their progress and mark topics as completed.

**Code:**  
***public static List<string> CompletedTopics { get; private set; } = new List<string>();***

***public Traffic\_Signs\_page()***

***{***

***InitializeComponent();***

***}***

***private void Giving\_Order\_Complete\_CheckedChanged(object sender, EventArgs e)***

***{***

***UpdateCompletedTopics("Giving Orders", Giving\_Order\_Complete.Checked);***

***}***

***private void Warning\_Signs\_Complete\_CheckedChanged(object sender, EventArgs e)***

***{***

***UpdateCompletedTopics("Warning Signs", Warning\_Signs\_Complete.Checked);***

***}***

***private void Direction\_Signs\_Complete\_CheckedChanged(object sender, EventArgs e)***

***{***

***UpdateCompletedTopics("Direction Signs", Direction\_Signs\_Complete.Checked);***

***}***

***private void Information\_Signs\_Complete\_CheckedChanged(object sender, EventArgs e)***

***{***

***UpdateCompletedTopics("Information Signs", Information\_Signs\_Complete.Checked);***

***}***

***private void Road\_Work\_Complete\_CheckedChanged(object sender, EventArgs e)***

***{***

***UpdateCompletedTopics("Road Work Signs", Road\_Work\_Complete.Checked);***

***}***

***private void UpdateCompletedTopics(string topic, bool isCompleted)***

***{***

***if (isCompleted)***

***{***

***if (!CompletedTopics.Contains(topic))***

***{***

***CompletedTopics.Add(topic);***

***}***

***}***

***else***

***{***

***CompletedTopics.Remove(topic);***

***}***

***}***

**Obstacle:** when the user goes back to the menu and open traffic page again the checkbox becomes empty  
  
**Solution:** The issue arises because the state of the checkboxes (whether they are checked or not) is not being preserved when navigating away from the Traffic\_Signs\_page and reopening it. This happens because each time the form is opened, a new instance of Traffic\_Signs\_page is created, and the checkboxes are reset to their default state (unchecked).

To fix this, I will store the state of the checkboxes (whether they are checked or unchecked) in a persistent location, such as a global variable, a file, or a database. When the Traffic\_Signs\_page is reopened, I can retrieve the saved state and update the checkboxes accordingly.  
 **Code:**  
 ***// Static dictionary to store checkbox states***

***private static Dictionary<string, bool> CheckboxStates = new Dictionary<string, bool>***

***{***

***{ "Giving Orders", false },***

***{ "Warning Signs", false },***

***{ "Direction Signs", false },***

***{ "Information Signs", false },***

***{ "Road Work Signs", false }***

***};***

***public Traffic\_Signs\_page()***

***{***

***InitializeComponent();***

***RestoreCheckboxStates();***

***}***

***private void RestoreCheckboxStates()***

***{***

***// Restore each checkbox state from the static dictionary***

***Giving\_Order\_Complete.Checked = CheckboxStates["Giving Orders"];***

***Warning\_Signs\_Complete.Checked = CheckboxStates["Warning Signs"];***

***Direction\_Signs\_Complete.Checked = CheckboxStates["Direction Signs"];***

***Information\_Signs\_Complete.Checked = CheckboxStates["Information Signs"];***

***Road\_Work\_Complete.Checked = CheckboxStates["Road Work Signs"];***

***// Debug logging for validation***

***Console.WriteLine("Checkbox States Restored:");***

***foreach (var state in CheckboxStates)***

***{***

***Console.WriteLine($"{state.Key}: {state.Value}");***

***}***

***}***

***private void UpdateCompletedTopics(string topic, bool isCompleted)***

***{***

***if (isCompleted)***

***{***

***if (!CompletedTopics.Contains(topic))***

***{***

***CompletedTopics.Add(topic);***

***}***

***}***

***else***

***{***

***CompletedTopics.Remove(topic);***

***}***

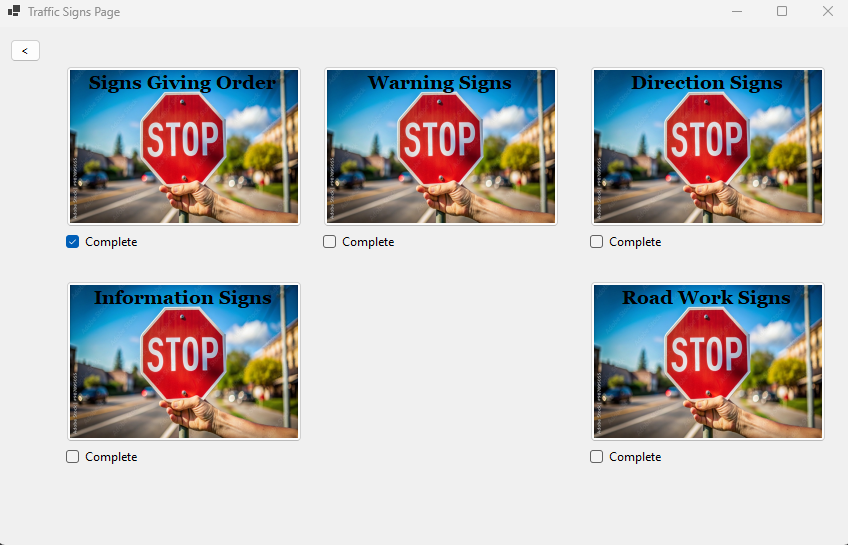
***// Update the static dictionary with the checkbox state***

***CheckboxStates[topic] = isCompleted;***

***// Debug logging for validation***

***Console.WriteLine($"Updated Checkbox State: {topic} = {isCompleted}");***

***}***

******

**Iteration 2:**

**Problem:** How to put pictures with text so the user can study signs from?  
  
**Solution:** I will add a new page under each topic so that the user can see what signs belong to that particular topic. When the user clicks on a topic, they will be directed to a page that displays all the relevant signs under that topic.

Code:  
 ***private void Orders\_Signs\_Click(object sender, EventArgs e)***

***{***

***Orders\_Signs nextForm = new Orders\_Signs();***

***nextForm.Show();***

***this.Hide();***

***}***

**Problem:** How to put pictures with text?  
  
**Solution:** Implementation of a datagrid with scroll wheel

Code:  
 ***private void InitializeGrid()***

***{***

***// Create and configure DataGridView***

***DataGridView signsGridView = new DataGridView***

***{***

***Dock = DockStyle.Fill,***

***AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill,***

***RowTemplate = { Height = 100 }, // Adjust row height to fit images***

***AllowUserToAddRows = false,***

***ReadOnly = true***

***};***

***// Add columns***

***DataGridViewImageColumn imageColumn = new DataGridViewImageColumn***

***{***

***HeaderText = "Sign Image",***

***Name = "ImageColumn",***

***ImageLayout = DataGridViewImageCellLayout.Zoom // Adjust image display***

***};***

***signsGridView.Columns.Add(imageColumn);***

***DataGridViewTextBoxColumn infoColumn = new DataGridViewTextBoxColumn***

***{***

***HeaderText = "Information",***

***Name = "InfoColumn"***

***};***

***signsGridView.Columns.Add(infoColumn);***

***// Add data***

***AddSignRow(signsGridView, "Signs with red circles are mostly prohibitive.\r\nPlates below signs qualify their message.", "Blank.png");***

***AddSignRow(signsGridView, "Entry to 20 mph zone", "Entry\_to\_20\_mph\_zone.png");***

***AddSignRow(signsGridView, "End of 20 mph zone", "End\_of\_20\_mph\_zone.png");***

***if (signsGridView.Rows.Count > 1) // Ensure the row exists***

***{***

***signsGridView.Rows[0].DefaultCellStyle.Font = new Font("Arial", 10, FontStyle.Bold);***

***}***

***// Add DataGridView to form***

***Controls.Add(signsGridView);***

***}***

***private void AddSignRow(DataGridView grid, string info, string imagePath)***

***{***

***// Load image***

***string appDirectory = AppDomain.CurrentDomain.BaseDirectory;***

***string imageFullPath = System.IO.Path.Combine(appDirectory, "Signs\_Giving\_Order", imagePath);***

***if (!System.IO.File.Exists(imageFullPath))***

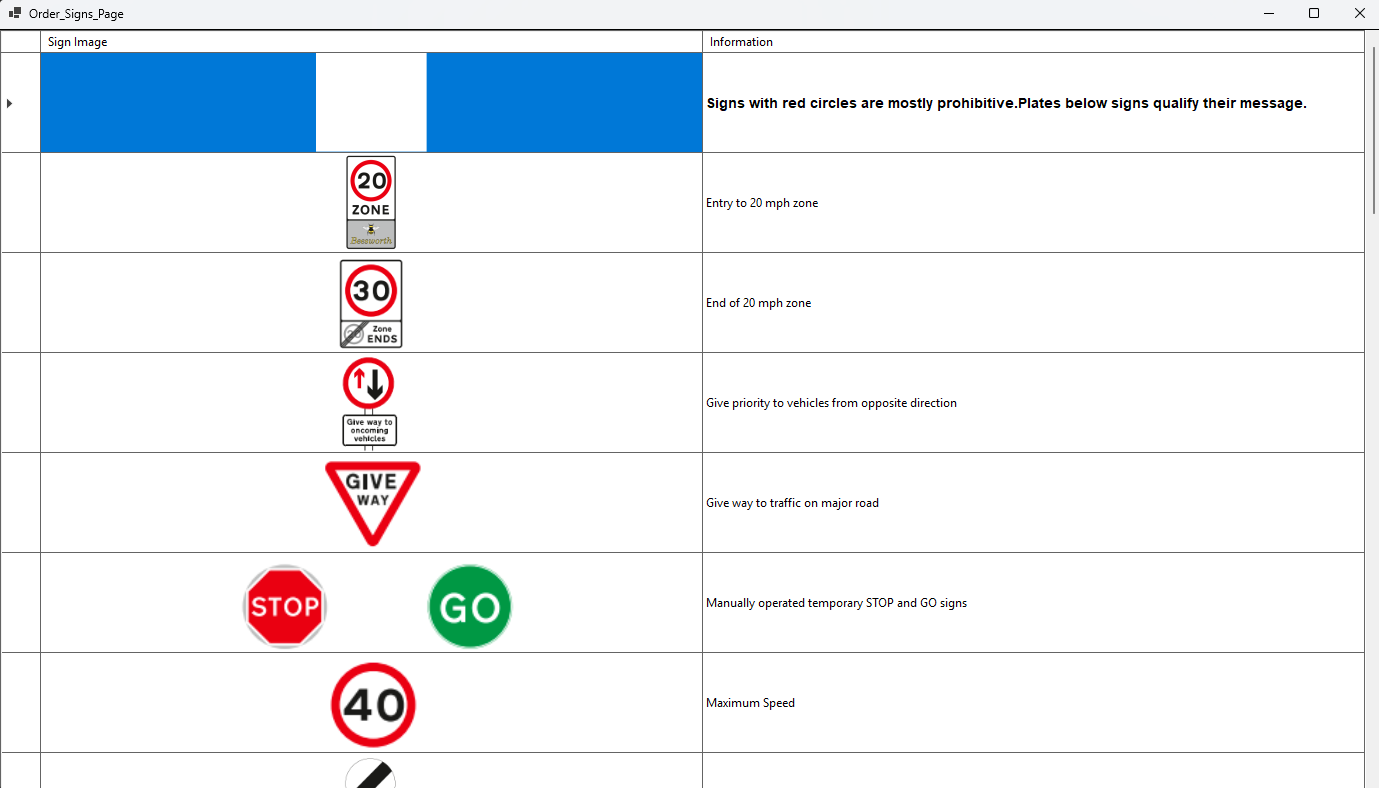
***throw new System.IO.FileNotFoundException($"Image file not found: {imageFullPath}");***

***Image signImage = Image.FromFile(imageFullPath);***

***// Add row***

***grid.Rows.Add(signImage, info);***

***}***

******

**Review:**

|  |  |
| --- | --- |
| **Success Criteria** | **Result** |
| **Does the system allow users to mark topics as completed?** |  |
| **Does the system persist the checkbox state even when the user navigates away from the page?** |  |
| **Can users view all signs related to a specific topic when clicking on it?** |  |
| **Are all traffic signs displayed with appropriate images and descriptions?** |  |
| **Is the progress of each topic updated immediately when the user marks it as completed or incomplete?** |  |
| **Can the user view flagged questions separately?** |  |
| **Does the progress page show a complete list of all completed topics?** |  |
| **Does the system display dynamic images and descriptions of signs from the correct topic?** |  |
| **Is the user interface intuitive for selecting topics and viewing signs?** |  |
| **Can the user easily toggle between topics and view progress?** |  |
| **Are checkboxes clearly visible and easy to interact with?** |  |
| **Can users navigate seamlessly between different pages (topics, progress, signs)?** |  |
| **Is there a way for users to track and view their overall progress on a single page?** |  |
| **Are the images of signs clear and appropriately sized?** |  |
| **Does the interface visually distinguish between completed and incomplete topics?** |  |
| **Can users undo or reset their progress on specific topics if needed?** |  |
| **Does the app load quickly without significant delays?** |  |
| **Is there any lag when switching between topics or viewing images?** |  |
| **Are sign images loaded efficiently without causing performance issues?** |  |
| **Does the app handle high volumes of data, such as a large number of signs or topics, without crashing?** |  |
| **Is the application using a global variable or persistent data structure for storing progress?** |  |
| **Is there error handling in place for missing images or incorrect data states?** |  |
| **Are images and descriptions dynamically loaded from the appropriate file location?** |  |
| **Is the progress stored in a way that can be saved or transferred between sessions?** |  |
| **Is the layout clean, with proper spacing and alignment for readability?** |  |
| **Is the text legible and appropriately styled for clarity?** |  |
| **Are progress indicators (checkmarks, completion percentages) clearly visible?** |  |
| **Is the theme of the page customizable or consistent with the overall project design?** |  |
| **Is the app interactive, allowing the user to engage with different features (checkboxes, navigation, image display)?** |  |
| **Does the system update in real-time when a topic is marked as complete?** |  |
| **Does the progress page update automatically to reflect changes in the checkbox states?** |  |
| **Can users interact with the images, such as zooming in for better visibility of signs?** |  |
| **Does the system handle missing or corrupted image files gracefully (e.g., showing a placeholder)?** |  |
| **Does the app prevent crashes or unexpected behavior from faulty inputs or actions?** |  |
| **Are there clear error messages or notifications if something goes wrong (e.g., missing data or images)?** |  |
| **Is the system flexible enough to add new topics or signs in the future?** |  |
| **Can additional content, such as multimedia (videos, animations), be added without major redesigns?** |  |
| **Can more complex features (like quizzes or assessments) be easily integrated into the current framework?** |  |