

Project Report: Unicom TIC Management System

Author: Mishari

1. Project Overview

This document outlines the key features, technologies, and development journey of the Unicom TIC Management System, a desktop application designed to manage academic and administrative data for an educational institution.

Key Features Implemented:

- **Secure Role-Based Login System:** The application features a robust login screen that authenticates users and assigns them one of four roles: Admin, Student, Staff, or Lecturer.
- **Dynamic, Role-Based Dashboards:** The main dashboard is customized based on the user's role. Admins have access to a full "Manage" menu, while Students have a restricted view with access only to their personal information.
- **Full CRUD Functionality:** Admins have complete Create, Read, Update, and Delete (CRUD) capabilities for all core entities through dedicated management forms.
- **Modular Management System:** The application includes separate, easy-to-use modules for managing:
 - User Accounts (including role assignment)
 - Courses and Subjects (with relational links)
 - Student Records (with course enrollment)
 - Rooms and Timetables
- **Advanced Marks Entry System:** A master-detail form allows administrators to manage exams and efficiently enter marks for all students enrolled in the relevant subject.
- **Personalized Student Views:** Students can log in to view their own academic timetable and a personalized list of their exam marks, ensuring data privacy.
- **Portable Database:** Utilizes a file-based SQLite database (unicomtic.db), allowing the entire application to be run from a single folder with no complex setup.

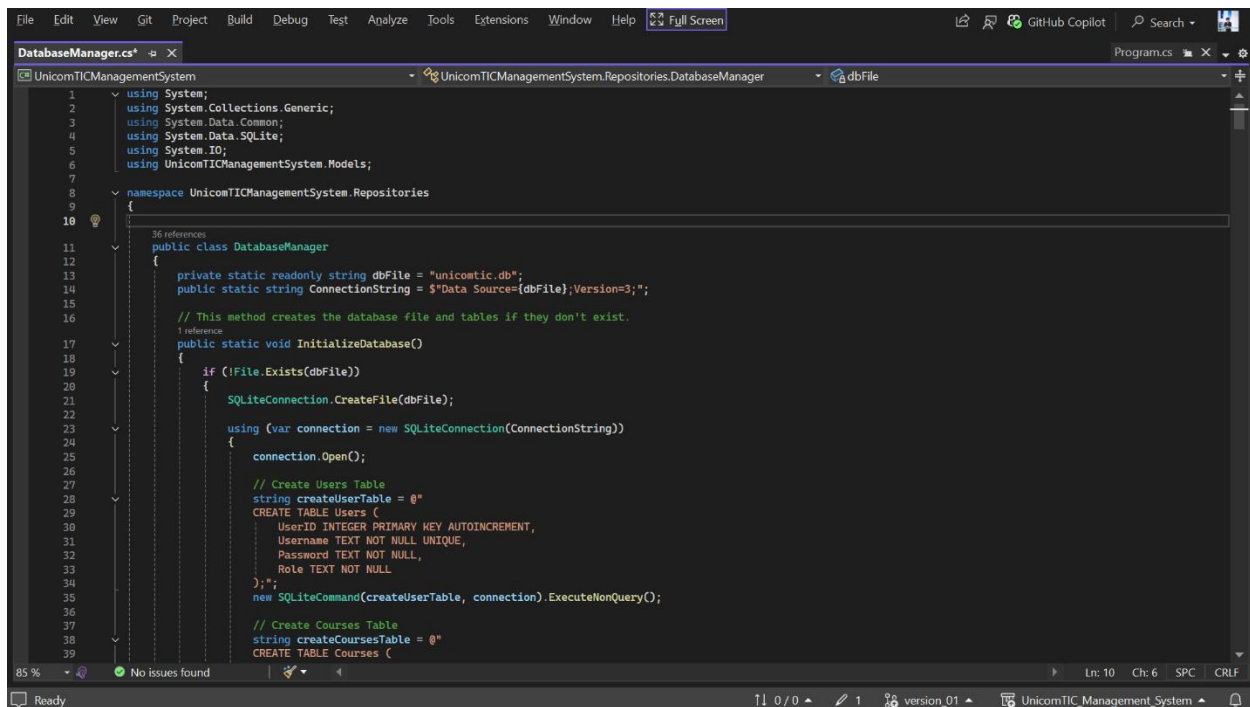
Technologies Used:

- **Programming Language:** C#
- **Framework:** .NET Framework
- **Application Type:** Windows Forms (WinForms) Desktop Application
- **Database:** SQLite
- **IDE:** Microsoft Visual Studio
- **Version Control:** Git & GitHub

Challenges Faced and Solutions:

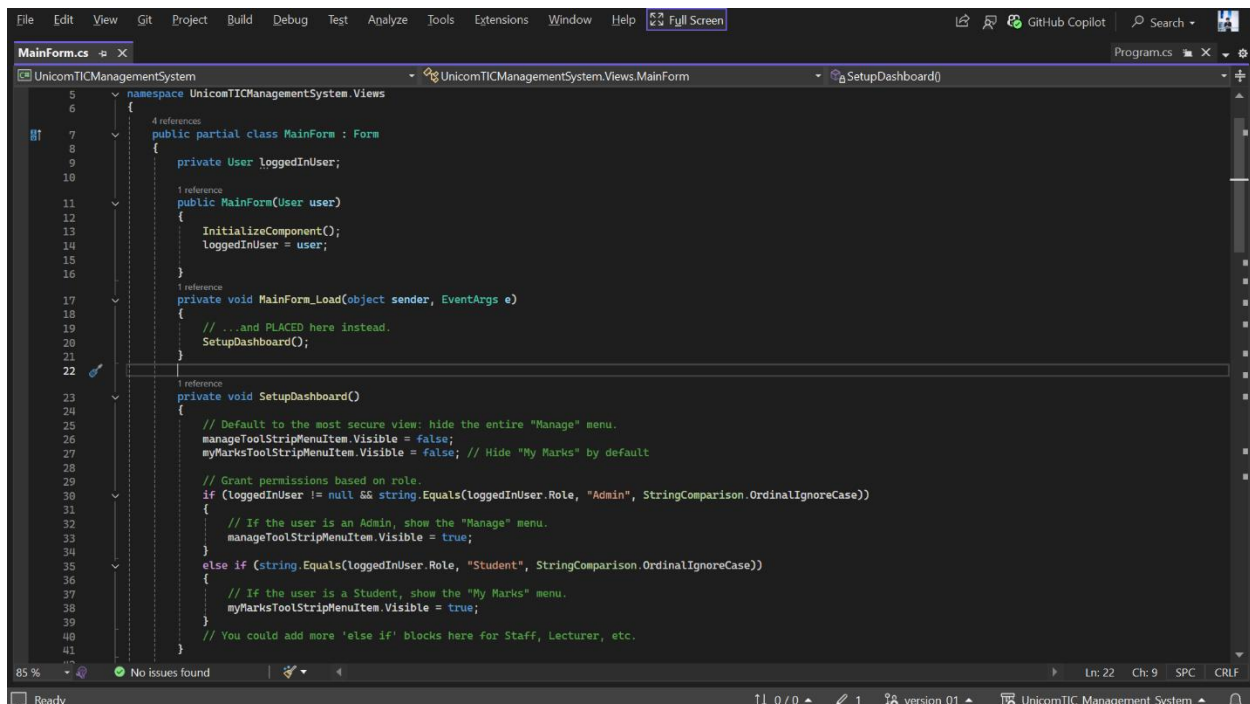
- **Challenge: UI Elements not updating correctly after database changes.**
 - **Solution:** Implemented dedicated LoadData() methods (e.g., LoadUsersGrid()). After any action like adding or deleting, this method is called to refresh the DataSource of the DataGridView, ensuring the UI always reflects the current state of the database.
- **Challenge: Application crashing when moved to a new computer (SQLite.Interop.dll not found).**
 - **Solution:** Diagnosed the issue as a platform architecture mismatch. The problem was permanently fixed by setting the project's **Platform Target** to **x86** in the project properties. This ensures compatibility between the application and the SQLite library on any machine.
- **Challenge: Forms crashing in the Visual Studio Designer after being moved.**
 - **Solution:** Identified that complex code (like creating a controller) in a form's constructor can cause the designer to fail. The logic was refactored by moving all data-loading and controller-initialization code from the constructor into the Form_Load event, which is not run by the designer.

2. Code Samples (Screenshots)



```
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Full Screen
DatabaseManager.cs
UnicomTICManagementSystem
UnicomTICManagementSystem.Repositories.DatabaseManager
dbFile
1 using System;
2 using System.Collections.Generic;
3 using System.Data.Common;
4 using System.Data.SQLite;
5 using System.IO;
6 using UnicomTICManagementSystem.Models;
7
8 namespace UnicomTICManagementSystem.Repositories
9 {
10
11     36 references
12     public class DatabaseManager
13     {
14         private static readonly string dbFile = "unicomtic.db";
15         public static string ConnectionString = $"Data Source={dbFile};Version=3;";
16
17         // This method creates the database file and tables if they don't exist.
18         1 reference
19         public static void InitializeDatabase()
20         {
21             if (!File.Exists(dbFile))
22             {
23                 SQLiteConnection.CreateFile(dbFile);
24
25                 using (var connection = new SQLiteConnection(ConnectionString))
26                 {
27                     connection.Open();
28
29                     // Create Users Table
30                     string createUserTable = @"
31                     CREATE TABLE Users (
32                         UserID INTEGER PRIMARY KEY AUTOINCREMENT,
33                         Username TEXT NOT NULL UNIQUE,
34                         Password TEXT NOT NULL,
35                         Role TEXT NOT NULL
36                     );";
37                     new SQLiteCommand(createUserTable, connection).ExecuteNonQuery();
38
39                     // Create Courses Table
40                     string createCoursesTable = @"
41                     CREATE TABLE Courses (
42
43             Ln: 10 Ch: 6 SPC CRLF
44 Ready 0/0 1 version_01 UnicomTIC_Management_System
```

Figure 1: The InitializeDatabase method, which creates the entire database schema and seeds it with default users.



```
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Full Screen
MainForm.cs
UnicomTICManagementSystem
UnicomTICManagementSystem.Views.MainForm
SetupDashboard()
5 namespace UnicomTICManagementSystem.Views
6 {
7     4 references
8     public partial class MainForm : Form
9     {
10         private User loggedInUser;
11
12         1 reference
13         public MainForm(User user)
14         {
15             InitializeComponent();
16             loggedInUser = user;
17
18         1 reference
19         private void MainForm_Load(object sender, EventArgs e)
20         {
21             // ...and PLACED here instead.
22             SetupDashboard();
23
24         1 reference
25         private void SetupDashboard()
26         {
27             // Default to the most secure view: hide the entire "Manage" menu.
28             manageToolStripMenuItem.Visible = false;
29             myMarksToolStripMenuItem.Visible = false; // Hide "My Marks" by default
30
31             // Grant permissions based on role.
32             if (loggedInUser != null && string.Equals(loggedInUser.Role, "Admin", StringComparison.OrdinalIgnoreCase))
33             {
34                 // If the user is an Admin, show the "Manage" menu.
35                 manageToolStripMenuItem.Visible = true;
36
37             }
38             else if (string.Equals(loggedInUser.Role, "Student", StringComparison.OrdinalIgnoreCase))
39             {
40                 // If the user is a Student, show the "My Marks" menu.
41                 myMarksToolStripMenuItem.Visible = true;
42
43             }
44             // You could add more 'else if' blocks here for Staff, Lecturer, etc.
45
46             Ln: 22 Ch: 9 SPC CRLF
47 Ready 0/0 1 version_01 UnicomTIC_Management_System
```

Figure 2: The SetupDashboard method, which implements role-based access control.

```

ExamForm.cs
UnicomTICManagementSystem.Views.ExamForm
dgvExams_CellClick(object sender, DataGridViewCellEventArgs e)

56 LoadExamsGrid(); // Refresh the list of exams
57 txtExamName.Clear();
58
59 // This is the key event for the Master-Detail view.
60 // When an exam is clicked, we load the marks for it.
61
62 private void dgvExams_CellClick(object sender, DataGridViewCellEventArgs e)
63 {
64     if (e.RowIndex >= 0 && dgvExams.CurrentRow != null)
65     {
66         int selectedExamId = Convert.ToInt32(dgvExams.CurrentRow.Cells["ExamID"].Value);
67
68         // Get the marks data and store it in our class-level list
69         _currentMarks = _controller.GetMarksForExam(selectedExamId);
70
71         // Set this as the data source for the second grid
72         dgvMarks.DataSource = _currentMarks;
73
74         // Configure the marks grid for editing
75         SetupMarksGrid();
76     }
77
78
79 private void SetupMarksGrid()
80 {
81     dgvMarks.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
82     // Hide ID columns
83     if (dgvMarks.Columns["MarkID"] != null) dgvMarks.Columns["MarkID"].Visible = false;
84     if (dgvMarks.Columns["StudentID"] != null) dgvMarks.Columns["StudentID"].Visible = false;
85     if (dgvMarks.Columns["ExamID"] != null) dgvMarks.Columns["ExamID"].Visible = false;
86
87     // Make columns read-only, except for the 'Score' column
88     if (dgvMarks.Columns["StudentName"] != null) dgvMarks.Columns["StudentName"].ReadOnly = true;
89     if (dgvMarks.Columns["Score"] != null) dgvMarks.Columns["Score"].ReadOnly = false;
90
91
92 private void btnSaveChanges_Click(object sender, EventArgs e)
93 {
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Figure 3: The event handler for the master exams grid (dgvExams). This code demonstrates a master-detail UI pattern where selecting an exam in one grid triggers a database query to populate a second grid with a detailed list of students and their marks.

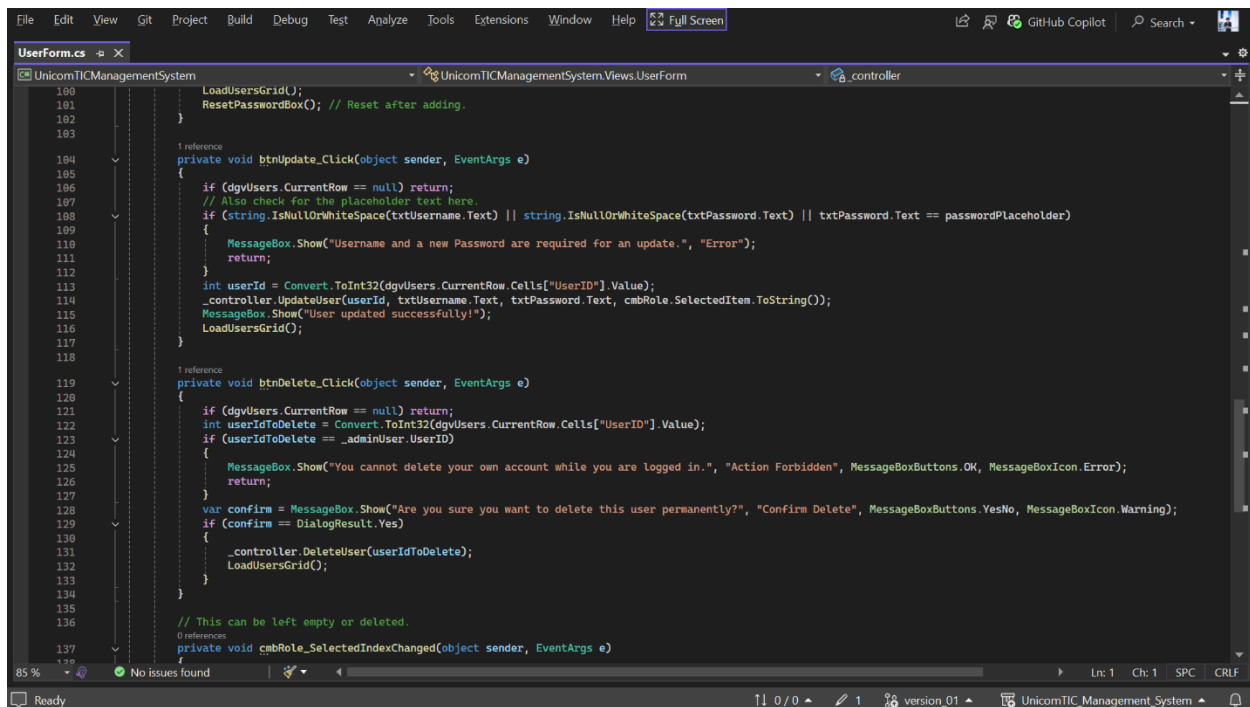
```

DatabaseManager.cs
UnicomTICManagementSystem.Repositories.DatabaseManager
dbFile

580 public static List<Mark> GetMarksForExam(int examId)
581 {
582     var marks = new List<Mark>();
583     var examSubjectIdQuery = "SELECT SubjectID FROM Exams WHERE ExamID = @examId";
584     int subjectId;
585
586     // First, find out which subject this exam belongs to.
587     using (var connection = new SQLiteConnection(connectionString))
588     {
589         connection.Open();
590         using (var command = new SQLiteCommand(examSubjectIdQuery, connection))
591         {
592             command.Parameters.AddWithValue("@examId", examId);
593             var result = command.ExecuteScalar(); // Gets a single value
594             if (result == null || result == DBNull.Value) return marks; // No subject found, return empty list
595             subjectId = Convert.ToInt32(result);
596         }
597
598         // Now, get all students in that subject's course and LEFT JOIN their marks for this exam.
599         string query = @"
600 SELECT s.StudentID, s.Name, m.MarkID, m.Score
601 FROM Students s
602 LEFT JOIN Marks m ON s.StudentID = m.StudentID AND m.ExamID = @examId
603 WHERE s.CourseID = (SELECT CourseID FROM Subjects WHERE SubjectID = @subjectId)";
604
605         using (var command = new SQLiteCommand(query, connection))
606         {
607             command.Parameters.AddWithValue("@examId", examId);
608             command.Parameters.AddWithValue("@subjectId", subjectId);
609             using (var reader = command.ExecuteReader())
610             {
611                 while (reader.Read())
612                 {
613                     marks.Add(new Mark
614                     {
615                         StudentID = reader.GetInt32(0),
616                         StudentName = reader.GetString(1),
617                         MarkID = reader.IsDBNull(2) ? 0 : reader.GetInt32(2), // MarkID is 0 if no mark exists yet
618                         ExamID = examId,
619                         // Score can be null if no mark has been entered
620                     });
621                 }
622             }
623         }
624     }
625     return marks;
626 }

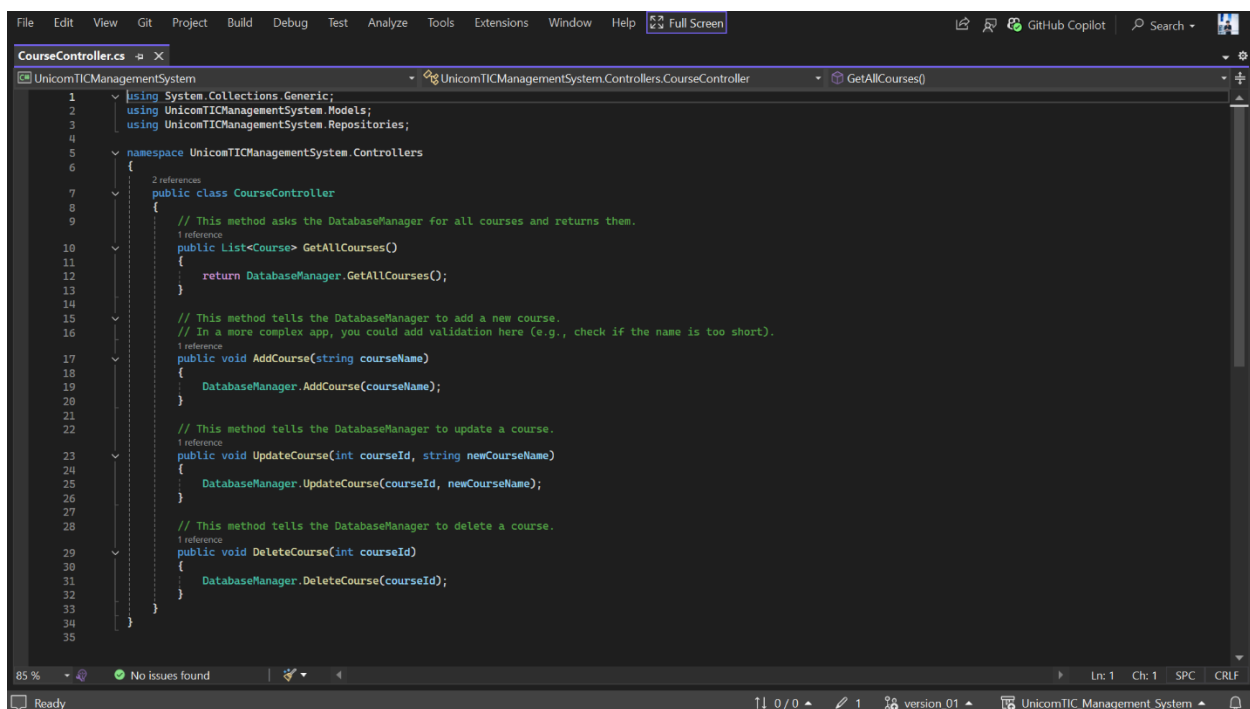
```

Figure 4: A repository method featuring a complex SQL LEFT JOIN query. This query efficiently combines data from multiple tables (Students and Marks) to build a comprehensive list for the marks entry screen.



```
100 LoadUsersGrid();
101 ResetPasswordBox(); // Reset after adding.
102
103
104 1 reference
105 private void btnUpdate_Click(object sender, EventArgs e)
106 {
107     if (dgvUsers.CurrentRow == null) return;
108     // Also check for the placeholder text here.
109     if (string.IsNullOrEmpty(txtUsername.Text) || string.IsNullOrEmpty(txtPassword.Text) || txtPassword.Text == passwordPlaceholder)
110     {
111         MessageBox.Show("Username and a new Password are required for an update.", "Error");
112         return;
113     }
114     int userId = Convert.ToInt32(dgvUsers.CurrentRow.Cells["UserID"].Value);
115     _controller.UpdateUser(userId, txtUsername.Text, txtPassword.Text, cmbRole.SelectedItem.ToString());
116     MessageBox.Show("User updated successfully!");
117     LoadUsersGrid();
118 }
119
120 1 reference
121 private void btnDelete_Click(object sender, EventArgs e)
122 {
123     if (dgvUsers.CurrentRow == null) return;
124     int userIdToDelete = Convert.ToInt32(dgvUsers.CurrentRow.Cells["UserID"].Value);
125     if (userIdToDelete == _adminUser.UserID)
126     {
127         MessageBox.Show("You cannot delete your own account while you are logged in.", "Action Forbidden", MessageBoxButtons.OK, MessageBoxIcon.Error);
128         return;
129     }
130     var confirm = MessageBox.Show("Are you sure you want to delete this user permanently?", "Confirm Delete", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);
131     if (confirm == DialogResult.Yes)
132     {
133         _controller.DeleteUser(userIdToDelete);
134         LoadUsersGrid();
135     }
136 }
137
138 // This can be left empty or deleted.
139 0 references
140 private void cmbRole_SelectedIndexChanged(object sender, EventArgs e)
```

Figure 5: A security check within the User Management form that prevents an administrator from deleting their own account. This demonstrates defensive programming by anticipating and preventing potentially harmful user actions.



```
1 using System.Collections.Generic;
2 using UnicomTICManagementSystem.Models;
3 using UnicomTICManagementSystem.Repositories;
4
5 namespace UnicomTICManagementSystem.Controllers
6 {
7     2 references
8     public class CourseController
9     {
10         // This method asks the DatabaseManager for all courses and returns them.
11         1 reference
12         public List<Course> GetAllCourses()
13         {
14             return DatabaseManager.GetAllCourses();
15         }
16
17         // This method tells the DatabaseManager to add a new course.
18         // In a more complex app, you could add validation here (e.g., check if the name is too short).
19         1 reference
20         public void AddCourse(string courseName)
21         {
22             DatabaseManager.AddCourse(courseName);
23         }
24
25         // This method tells the DatabaseManager to update a course.
26         1 reference
27         public void UpdateCourse(int courseId, string newCourseName)
28         {
29             DatabaseManager.UpdateCourse(courseId, newCourseName);
30         }
31
32         // This method tells the DatabaseManager to delete a course.
33         1 reference
34         public void DeleteCourse(int courseId)
35         {
36             DatabaseManager.DeleteCourse(courseId);
37         }
38     }
39 }
```

Figure 6: The CourseController.cs class, demonstrating the MVC pattern. This controller handles all business logic for course management, connecting the user interface to the database repository.

