FIT5032 Design Report

Major Application Development

{High Distinction}

VMed MRI Imaging Group

VINCENT: 30052586

Table of Contents

1. Web Application Title and Description

Title: V-Med MRI Imaging Group | Empowering Footballer's Futures Through Advanced MRI Imaging

Description: V-Med MRI Imaging Group offers a range of MRI imaging procedures including Lumbar Spine MRI, Cardiac MRI, Knee MRI, and Shoulder MRI for footballers across Australia. We are committed to providing our patients with high-resolution MRI imaging, personalized health reports, and recovery solutions that get them back on the field faster.

2. User Stories and Use Case Diagram

User Stories:

1. As a patient, I want to book an appointment with a doctor, so that I can receive the medical care and advice I need in terms of MRI imaging services.
2. As a doctor, I want to send an email to my patients, so that I can effectively communicate important information, updates, and reminders related to their MRI imaging results and healthcare.
3. As a patient, I want to navigate the location of a clinic on a map, so that I can easily find and reach the healthcare facility for my appointment or medical needs.
4. As a patient, I want to give a rating of a doctor's imaging service, so that I can provide feedback on my experience and help other patients make informed decisions about their healthcare.
5. As an admin, I want to send an email to all patients, so that I can efficiently communicate important updates, announcements, or general healthcare information to them.
6. As a patient, I want to receive a detailed and transparent breakdown of the costs associated with my MRI imaging services, so that I can make informed financial decisions regarding my healthcare and plan for any necessary expenses.
7. As a patient, I want to visually present my past appointment records, so that I can have a clear and accessible overview of my healthcare history, track my medical progress, and share this information with healthcare providers when necessary.

Use Case Diagram:

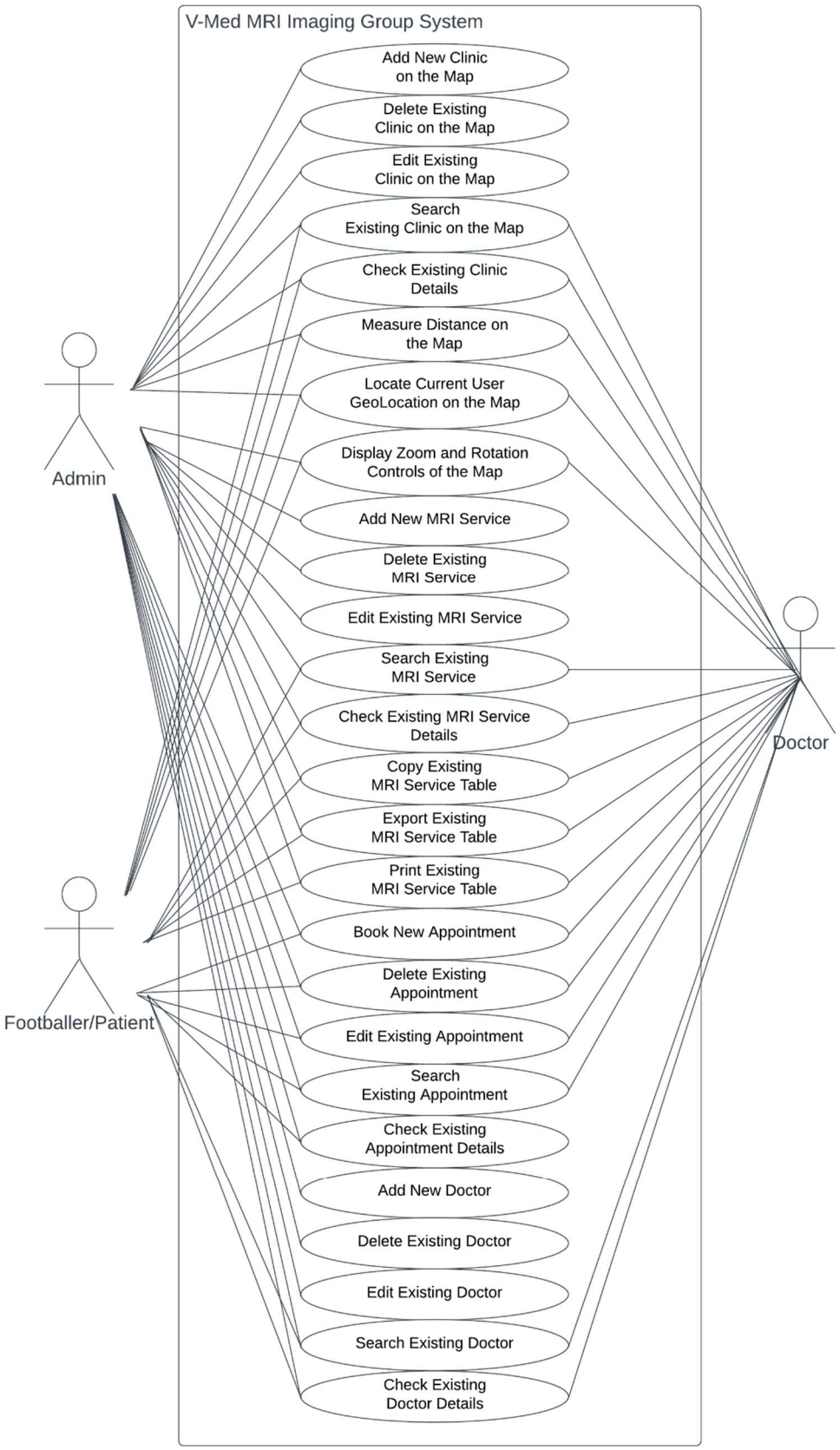


Figure 1. Use Case Diagram Part 1

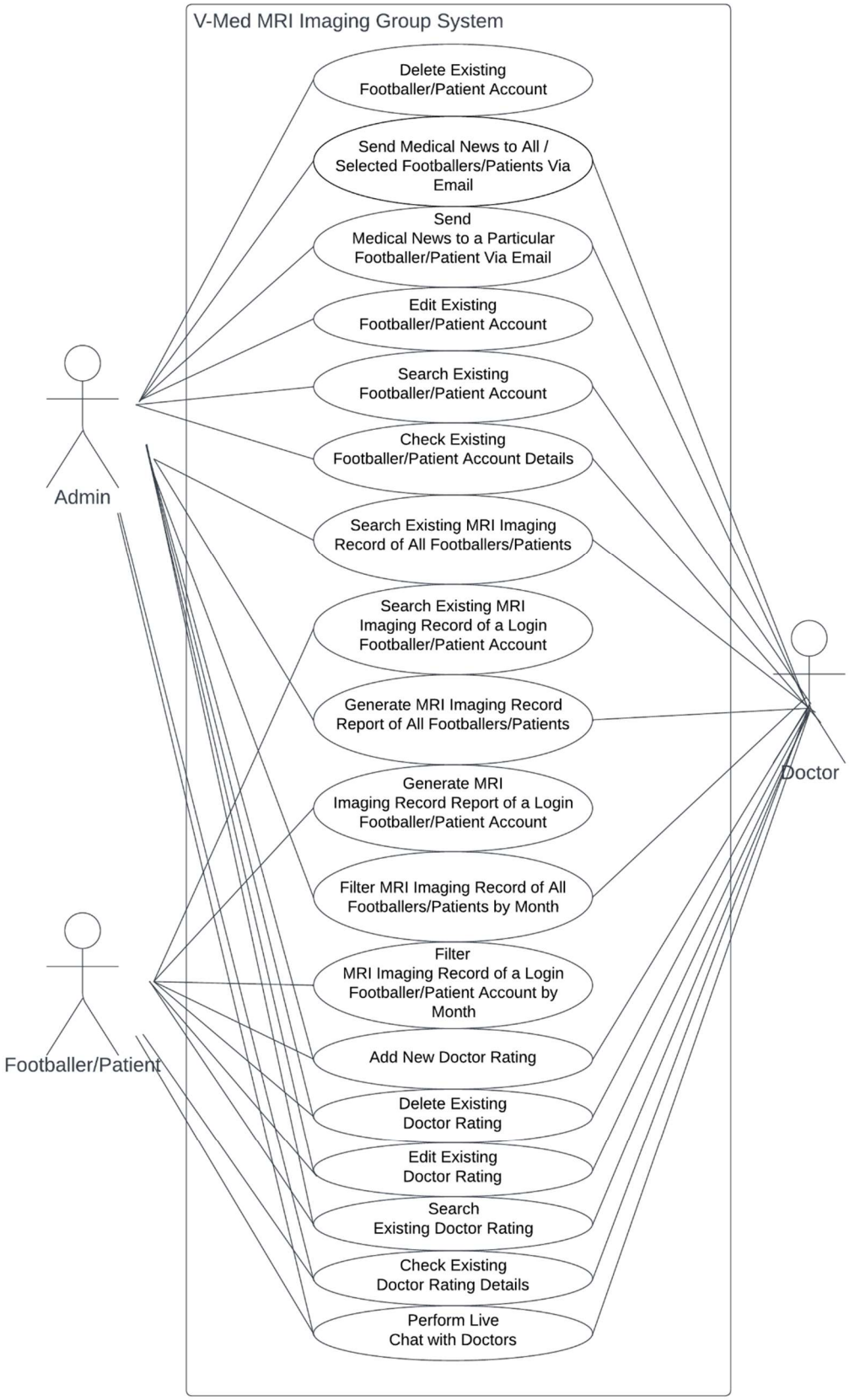


Figure 2. Use Case Diagram Part 2

3. Block/Functional Diagram

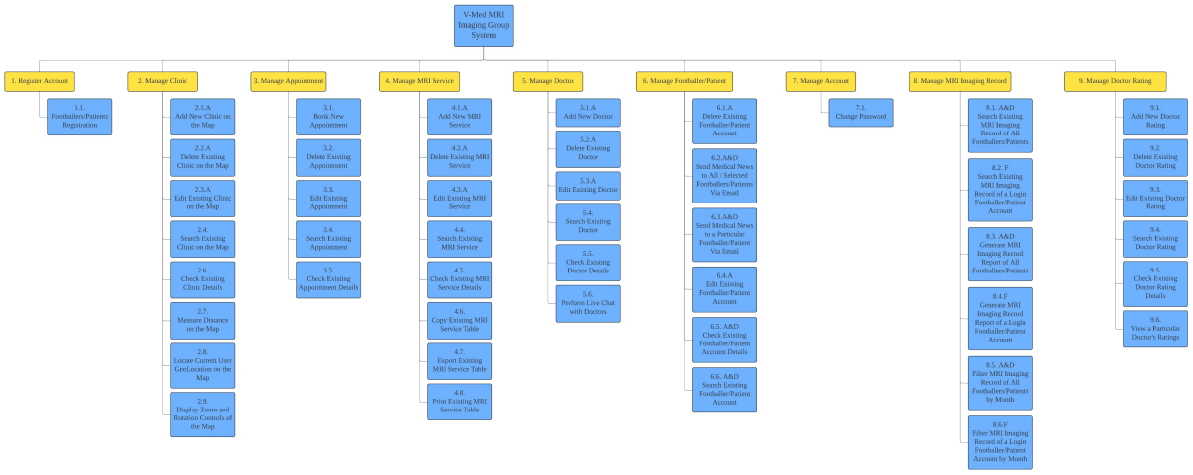


Figure 3. Block/Functional Diagram Full Screenshot

Note: A represents the function can only be performed by role Admin; D represents the function can only be performed by role Doctor; F represents the function can only be performed by role Footballer/Patient. In addition, the block/functional diagram above covers all the business requirements (BRs) up to (E.4). Since the screenshot is too small, please refer to Appendix A for the cropped version.

4. Your Selected Approach when Constructing the Application

In this project, I have adopted two approaches namely Database First approach and Code First approach with Database First approach plays most of the role. There are several reasons to choose Database First approach to finish my project. Firstly, this approach allows me to use Trigger in calculating the average rating of the doctors (BR (D.3) Rating). To illustrate, by using Trigger codes, whenever a new rating data is being added, deleted, or edited by any users, the average rating field in the Doctor entity will automatically be updated. The second reason is that time constraints are a crucial factor in this project. Since I have no experience with C# and have a better understanding in SQL query and schema, I think it is the best choice to go with Database First approach because it allows me to automatically generate model classes, controllers, and views based on the current database schema. The next reason is that I need to manipulate and report the data to fulfill BR (E.3) Chart as well as to remove all appointments related to a particular user (or other data which has a foreign key attached to it), which can be easily achieved by using ON DELETE CASCADE command. Finally, this approach also allows me to handle the versioning of my project because I have a text file that contains the SQL query to generate the entire model. Below is my SQL query to generate the entire model:

```
-- Create the Clinic Table
CREATE TABLE Clinic (
    ClinicID INT PRIMARY KEY IDENTITY(1,1),
    ClinicName VARCHAR(100) NOT NULL,
    Address VARCHAR(255) NOT NULL,
    Latitude DECIMAL(9, 6) NOT NULL,
    Longitude DECIMAL(9, 6) NOT NULL,
    ContactNumber VARCHAR(15) NOT NULL
);

-- Create the Doctor Table
CREATE TABLE Doctor (
    DoctorID INT PRIMARY KEY IDENTITY(1,1),
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Specialization VARCHAR(100) NOT NULL,
    ContactNumber VARCHAR(15) NOT NULL,
    Email VARCHAR(100) NOT NULL,
    ClinicID INT, -- Reference to Clinic
    AverageRating DECIMAL(3, 2), -- To store the average rating
    FOREIGN KEY (ClinicID) REFERENCES Clinic(ClinicID)
);

-- Create the Rating Table
CREATE TABLE Rating (
    RatingID INT PRIMARY KEY IDENTITY(1,1),
    DoctorID INT NOT NULL, -- Reference to Doctor
    RatingValue DECIMAL(3, 2) NOT NULL, -- Rating value, e.g., 4.5
    Comment TEXT, -- Optional comment or review
    FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID)
);

-- Create the Service Table
CREATE TABLE Service (
    ServiceID INT PRIMARY KEY IDENTITY(1,1),
    ServiceName VARCHAR(100) NOT NULL,
    Description VARCHAR(500),
    Cost DECIMAL(10, 2) NOT NULL
);

-- Create the Appointment Table
CREATE TABLE Appointment (
    AppointmentID INT PRIMARY KEY IDENTITY(1,1),
    DoctorID INT NOT NULL,
    ServiceID INT NOT NULL,
    AppointmentDate DATE NOT NULL,
    Notes VARCHAR(500),
    FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID),
    FOREIGN KEY (ServiceID) REFERENCES Service(ServiceID)
);
```

```
-- Create the Trigger to calculate and update the average rating for each doctor
-- This must be in a separate batch
GO

CREATE TRIGGER UpdateDoctorAverageRating
ON Rating
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    DECLARE @DoctorID INT;

    -- Check if any records were inserted or updated
    IF EXISTS (SELECT * FROM inserted)
    BEGIN
        SELECT TOP 1 @DoctorID = DoctorID FROM inserted;

        -- Update the AverageRating for the affected doctor
        UPDATE Doctor
        SET AverageRating = (
            SELECT AVG(RatingValue)
            FROM Rating
            WHERE DoctorID = @DoctorID
        )
        WHERE DoctorID = @DoctorID;
    END
    ELSE IF EXISTS (SELECT * FROM deleted)
    BEGIN
        -- Handle DELETE operation
        -- Get the DoctorID from the first deleted record (assuming it's the same for all)
        SELECT TOP 1 @DoctorID = DoctorID FROM deleted;

        -- Update the AverageRating for the affected doctor
        UPDATE Doctor
        SET AverageRating = (
            SELECT AVG(RatingValue)
            FROM Rating
            WHERE DoctorID = @DoctorID
        )
        WHERE DoctorID = @DoctorID;
    END
END;
```

After entering the SQL codes above, I will need to manually link Appointment entity to the AspNetUsers by using UserId as the foreign key with the codes below.

```
[UserId] NVARCHAR (128) NOT NULL,
CONSTRAINT [FK_dbo.Appointment.AspNetUsers_UserId] FOREIGN KEY ([UserId]) REFERENCES [dbo].[AspNetUsers] ([Id]) ON DELETE CASCADE
```

Note: In the SQL query above, I forgot to include ON DELETE CASCADE command in all the foreign keys. I have updated my database during the testing and checking process (i.e., this is also another reason why I choose this approach – only need to add ON DELETE CASCADE and "Update" database).

Another approach is Code First approach. This approach was adopted when I want to add more registration required fields such as FirstName, LastName, Gender, and PhoneNumber. To illustrate, I need to use migration to update my database (i.e., AspNetUsers entity) as shown in Figure 4 below.

5. Class Diagram or Entity Relationship Diagram

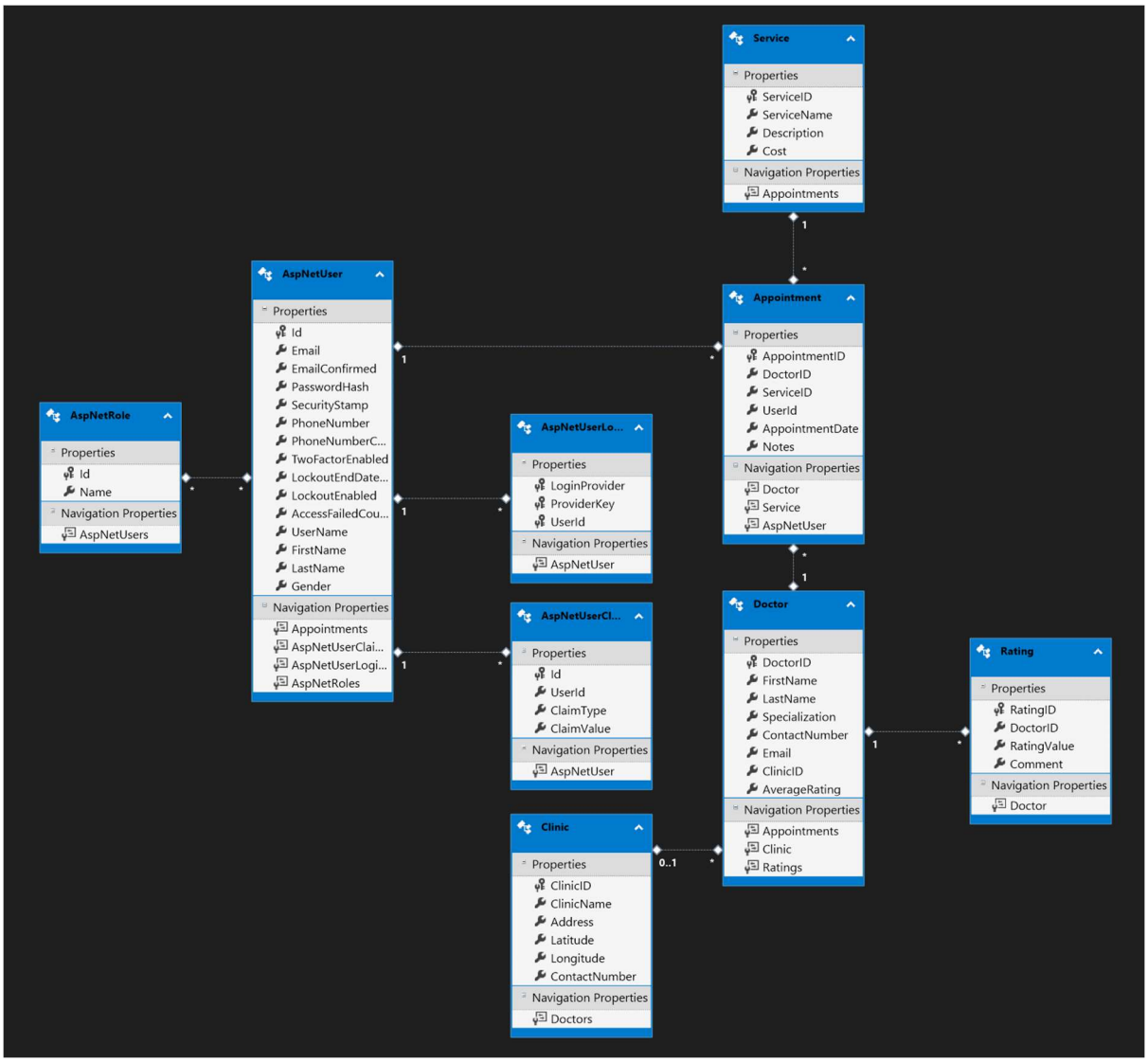


Figure 4. Entity Relationship (ER) Diagram with Crows Feet Notation Screenshot

Figure 4 above does not cover the email features. In this project, both email business requirements (BR (D.2) and (E.1)) have been achieved by using EmailController.cs, Views/Email/BulkEmail.cshtml, and Views/Email/Index.cshtml (i.e., there is no model involved). Below is the screenshot of the Email class diagram.

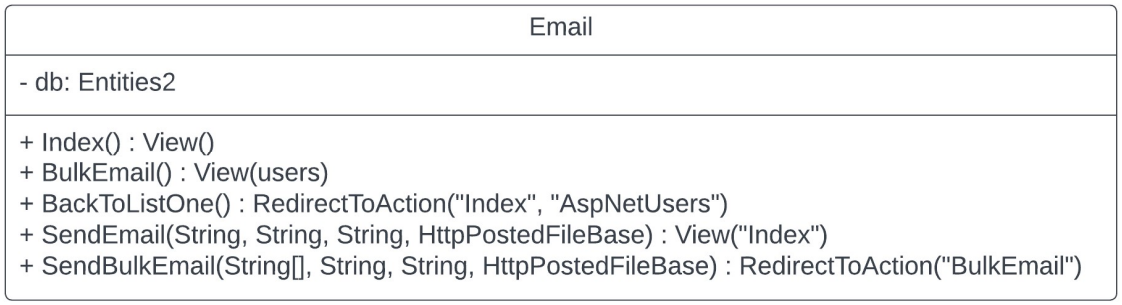


Figure 5. Email Class Diagram Screenshot

6. Data Dictionary

Table 1. Data Dictionary Table

Data Dictionary Table		
Entity: Service		
Field Name	**Data Type**	**Justification**
ServiceID	INT	Type INT for efficient indexing of the primary key constraint
ServiceName	VARCHAR(100)	To accommodate reasonably long service names
Description	VARCHAR(500)	To accommodate more detailed descriptions
Cost	DECIMAL(10, 2)	The cost of the service with two decimal places for precision (i.e., include cents)
Entity: Appointment		
Field Name	**Data Type**	**Justification**
AppointmentID	INT	Type INT for efficient indexing of the primary key constraint
AppointmentDate	DATE	To simplify the project because TIME will make it too complex
Notes	VARCHAR(500)	To accommodate more detailed notes
Entity: Doctor		
Field Name	**Data Type**	**Justification**
DoctorID	INT	Type INT for efficient indexing of the primary key constraint
FirstName	VARCHAR(50)	To accommodate reasonably long first names
LastName	VARCHAR(50)	To accommodate reasonably long last names
Specialization	VARCHAR(100)	To accommodate reasonably long specializations
ContactNumber	VARCHAR(10)	To accommodate standard phone numbers (e.g., 0466598833)
Email	VARCHAR(100)	To accommodate long email addresses
AverageRating	DECIMAL(3, 2)	To store the average rating of the doctor with two decimal places for precision
Entity: Rating		
Field Name	**Data Type**	**Justification**
RatingID	INT	Type INT for efficient indexing of the primary key constraint
RatingValue	DECIMAL(3, 2)	To store the rating value with two decimal places for precision
Comment	TEXT	TEXT data type is suitable for accommodating longer comments (e.g., 2 paragraphs of complaint)
Entity: Clinic		
Field Name	**Data Type**	**Justification**

ClinicID		Type INT for efficient indexing of the primary key constraint
ClinicName	VARCHAR(100)	To accommodate a reasonably long name
Address	VARCHAR(255)	To accommodate longer addresses
Latitude	DECIMAL(9, 6)	To store latitude with a high level of precision
Longitude	DECIMAL(9, 6)	To store latitude with a high level of precision
ContactNumber	VARCHAR(10)	To accommodate standard phone numbers (e.g., 0466598833)

7. Mockup Prototypes and Implementation with User Registration and Authentication

Mockup Prototypes Design:

Figure 6. Mock-up Prototype Design BR (C.3), (D.3) and (C.1) - Admin Doctor Page [Part 1]

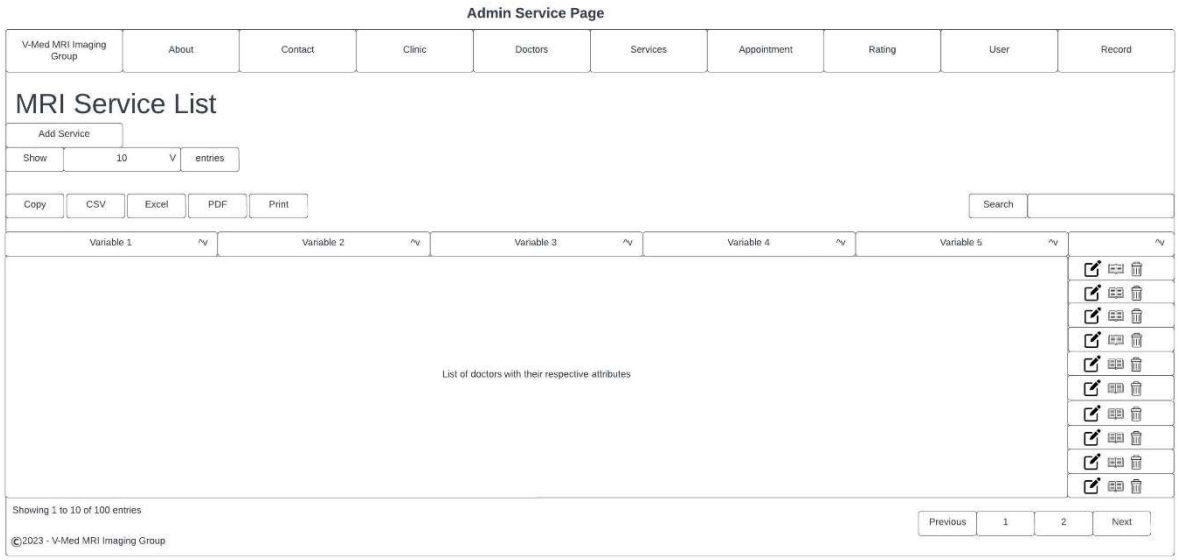


Figure 7. Mock-up Prototype Design BR (C.3), (D.3), (E.4) and (C.1) - Admin Service Page [Part 2]

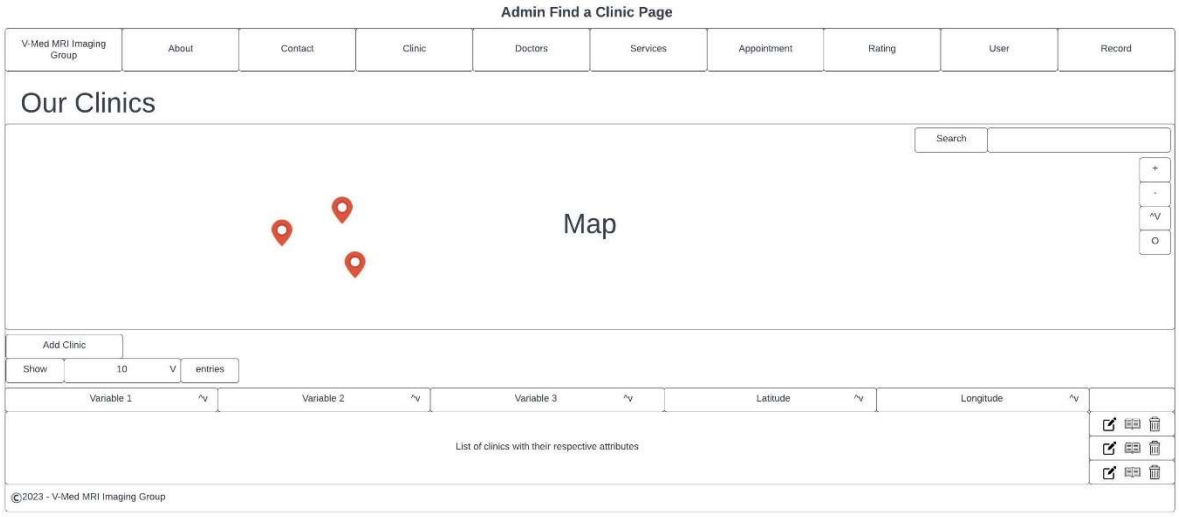


Figure 8. Mock-up Prototype Design BR (E.3) - Admin Clinic Page

Implementation with User Registration and Authentication:

In this project, all the user registration and authentication has been implemented in the AccountController.cs and Register.cshtml with Javascript.

Git URL: https://github.com/vinctechgeek/FIT5032_VMedProject

Final User Registration Implementation:

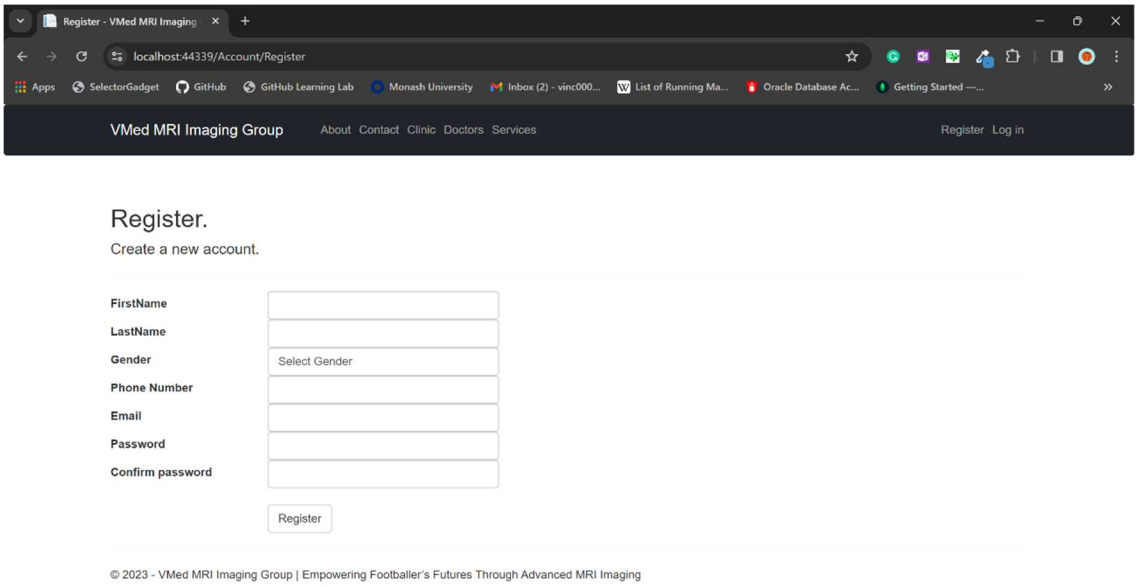


Figure 9. Registration Page Screenshot Part 1

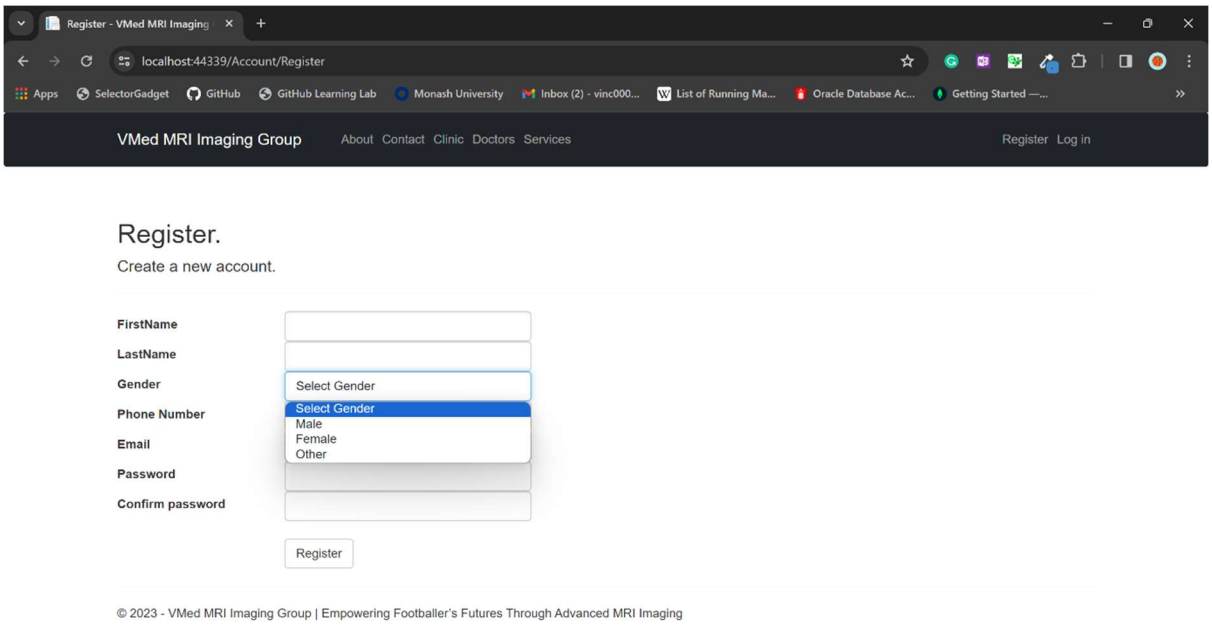


Figure 10. Registration Page Screenshot Part 2

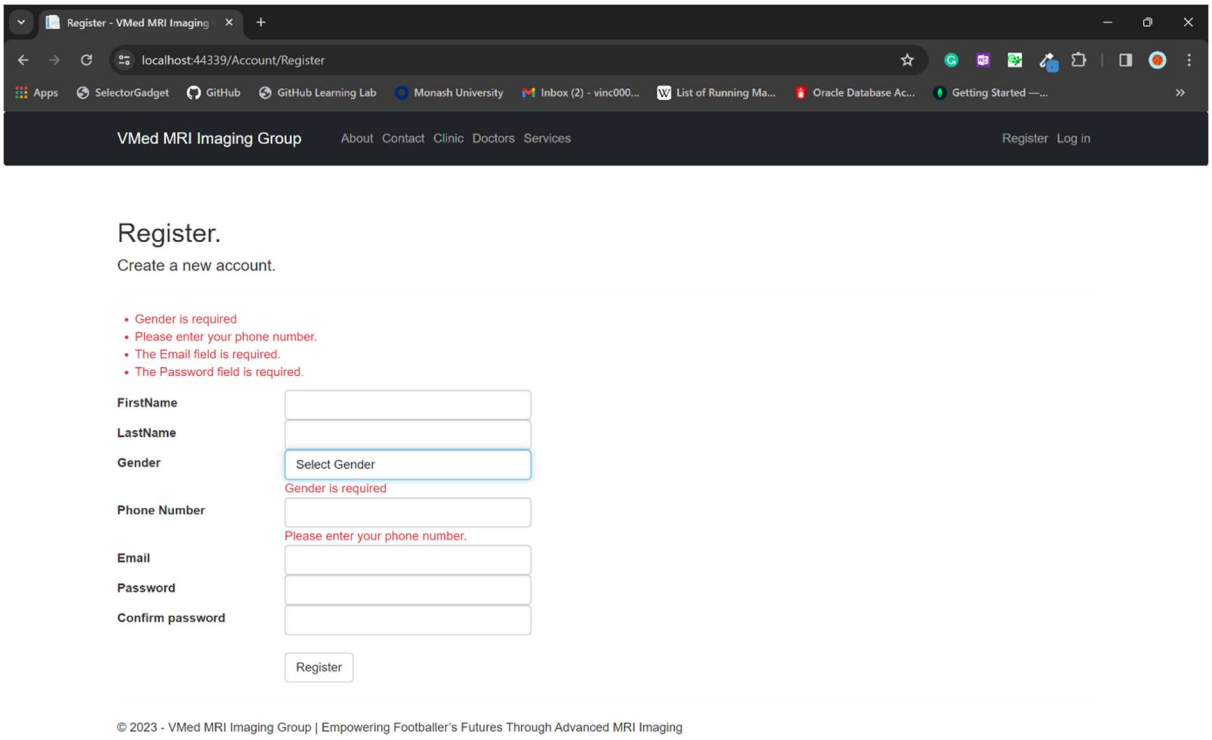


Figure 11. Registration Page Screenshot Part 3 (Validation)

Note: To improve the security, I have also enabled Javascript to automatically delete any characters that are not numbers, whitespaces, and letters (i.e., symbols such as < or * will be automatically removed when typing).

Final User Authentication Implementation:

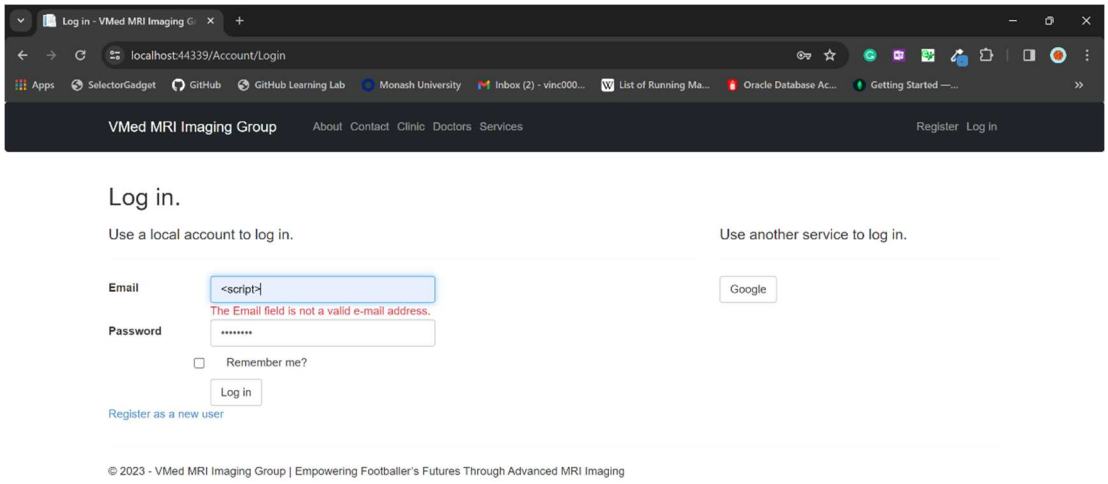


Figure 12. Email Validation Screenshot

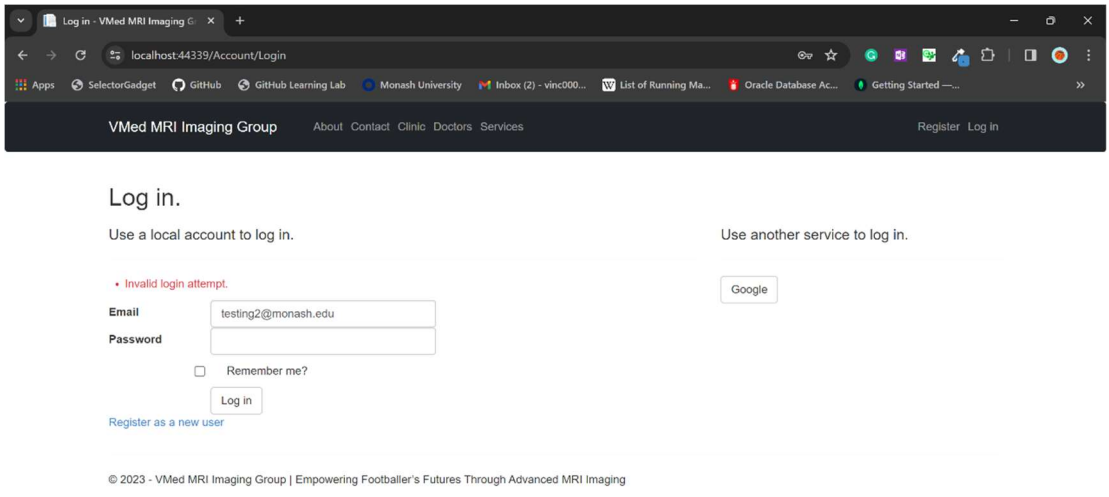


Figure 13. Invalid Login Input Validation (i.e., Username or Password is wrong)

Implementation (Codes):

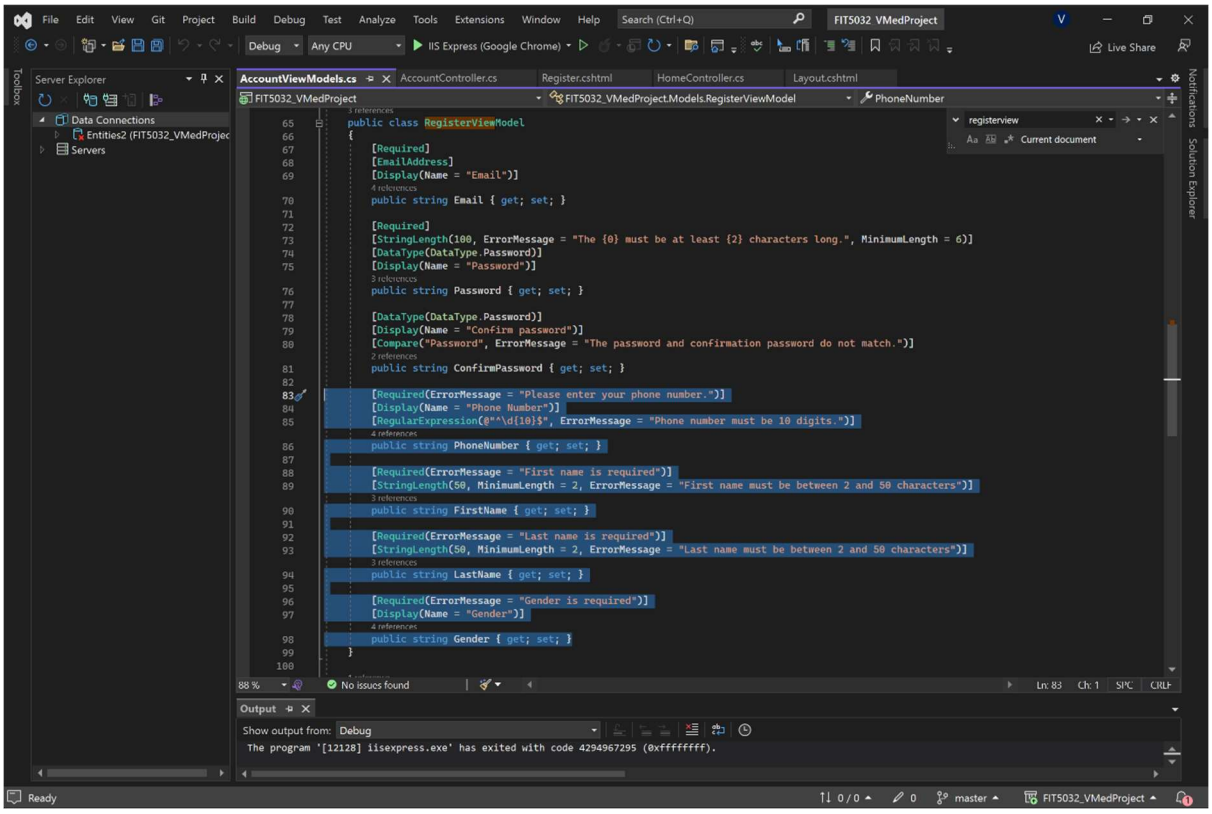


Figure 14. User Registration Model Validation (AccountViewModels.cs)

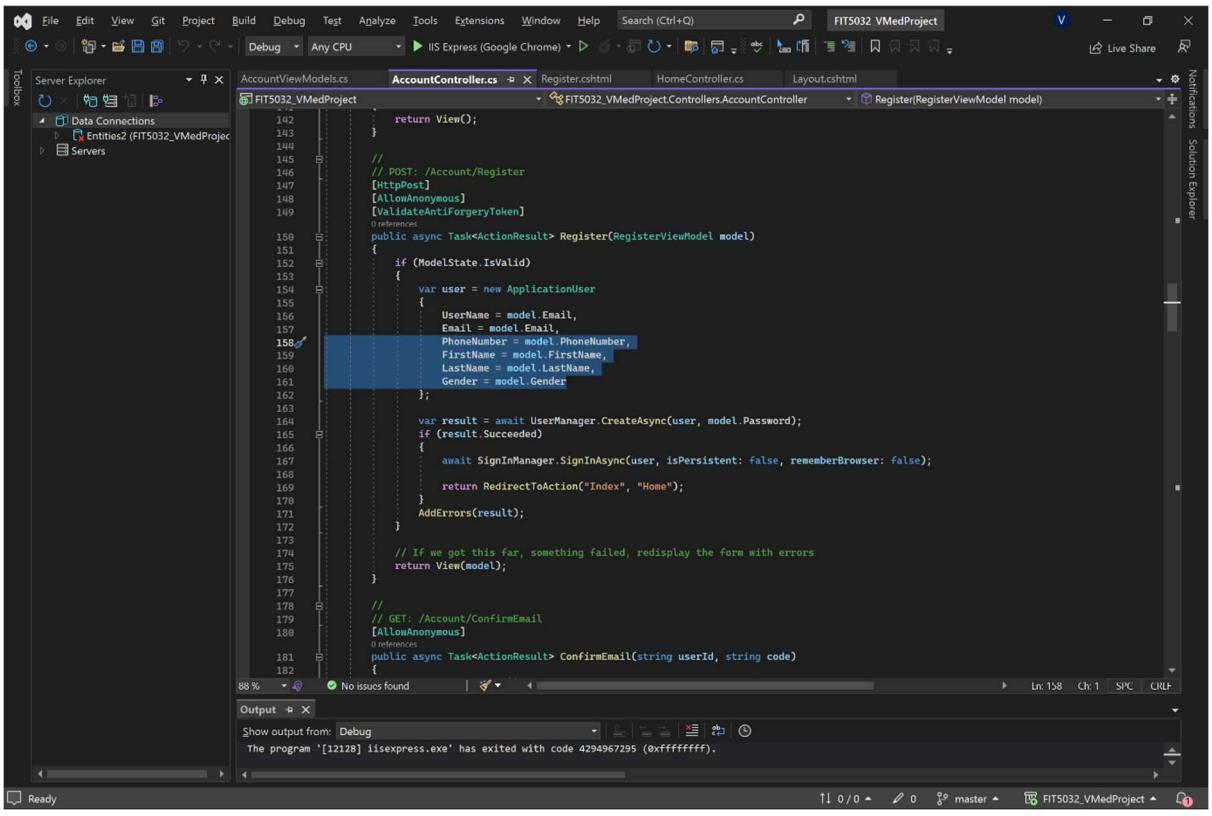


Figure 15. User Registration with Additional Fields Request (AccountController.cs)

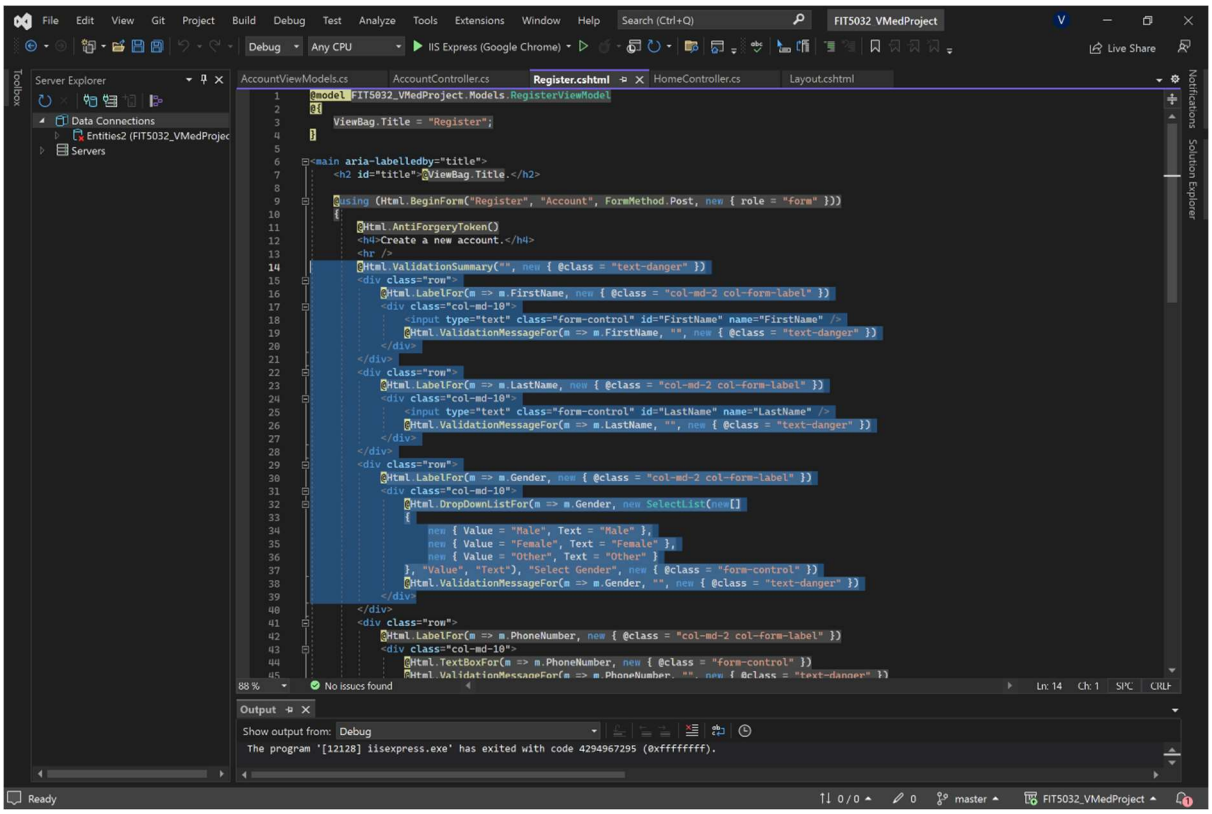


Figure 16. User Registration New View Part 1 (Register.cshtml)

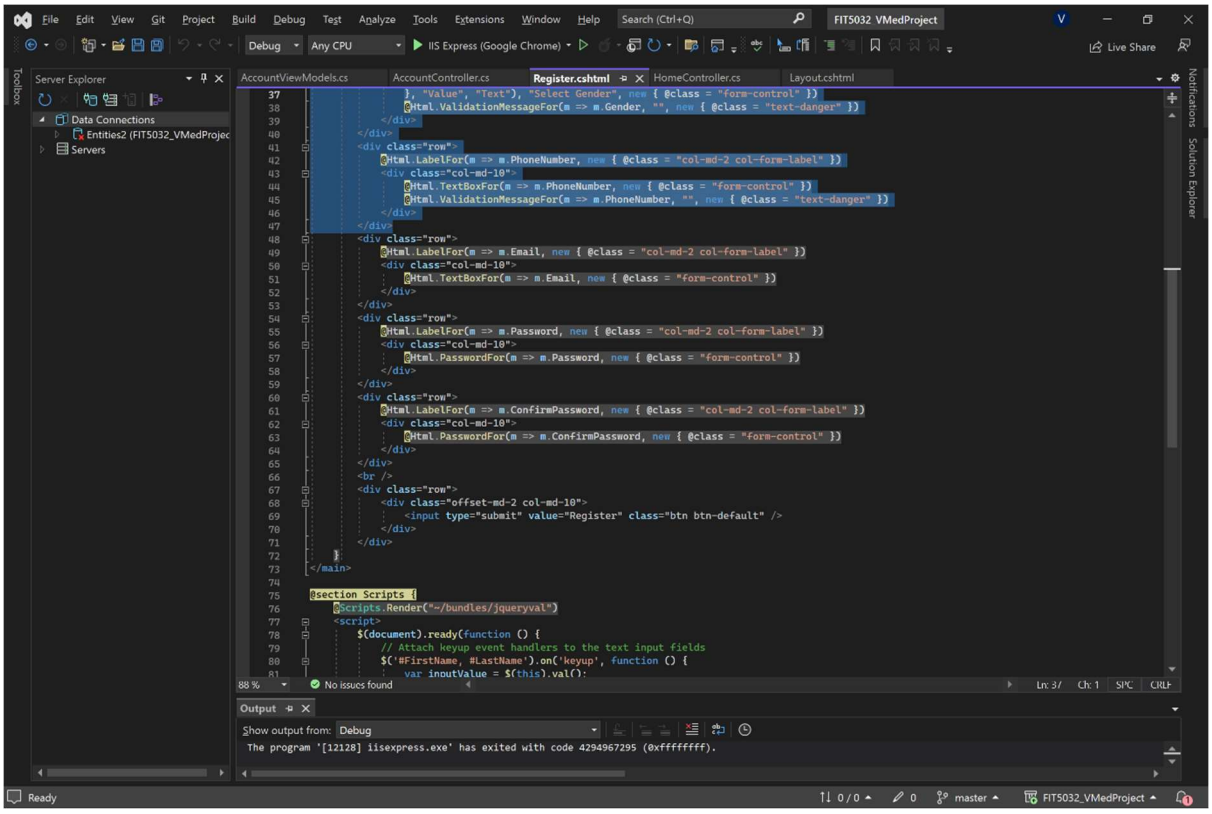


Figure 17. User Registration New View Part 2 (Register.cshtml)

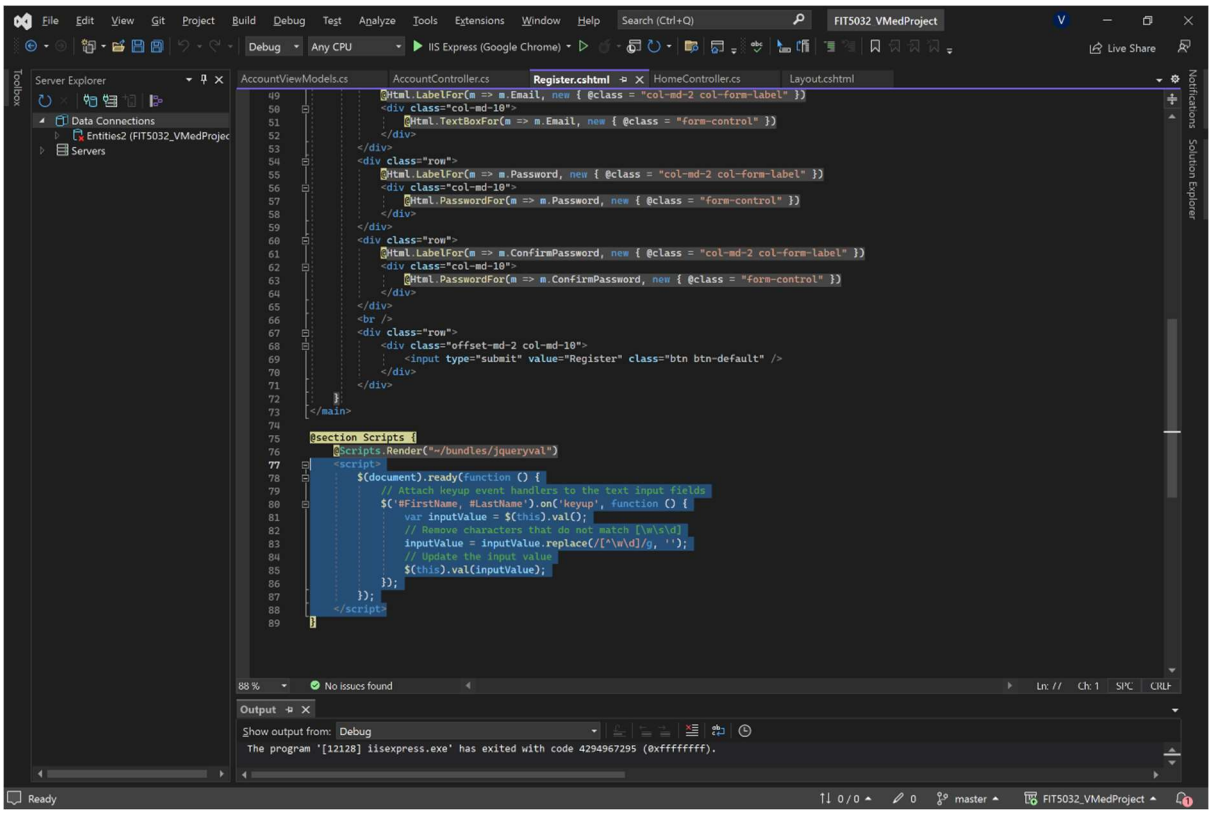


Figure 18. User Registration New View Part 3 – Javascript Validation as Mentioned in the Note Above (Register.cshtml)

8. Usability Design Review

The usability evaluation is based on Donald Norman principles.

Visibility

Visibility in design refers to making important elements, controls, and information easily visible and apparent to users (Norman, 1988; Educative, 2023). In this web application, there will be three roles and each role will have their respective visible elements, controls, and information. In addition, “Administrator” role will have more visibility than “Footballer” and “Doctor” roles. For example, “Administrator” role can add, delete, edit, and view details of MRI services whereas “Footballer” and “Doctor” roles can only view the listed MRI services.

Figure 19. Visibility Principle (Norman)

Feedback

Feedback is about providing users with clear and timely information about the results of their actions (Norman, 1988; Educative, 2023). Whenever an administrator or doctor managed to send either email with attachment (BR (D.2)) or bulk emails (BR (E.1)), there will be a message of "Email sent successfully!" as shown in Figure 20 below.

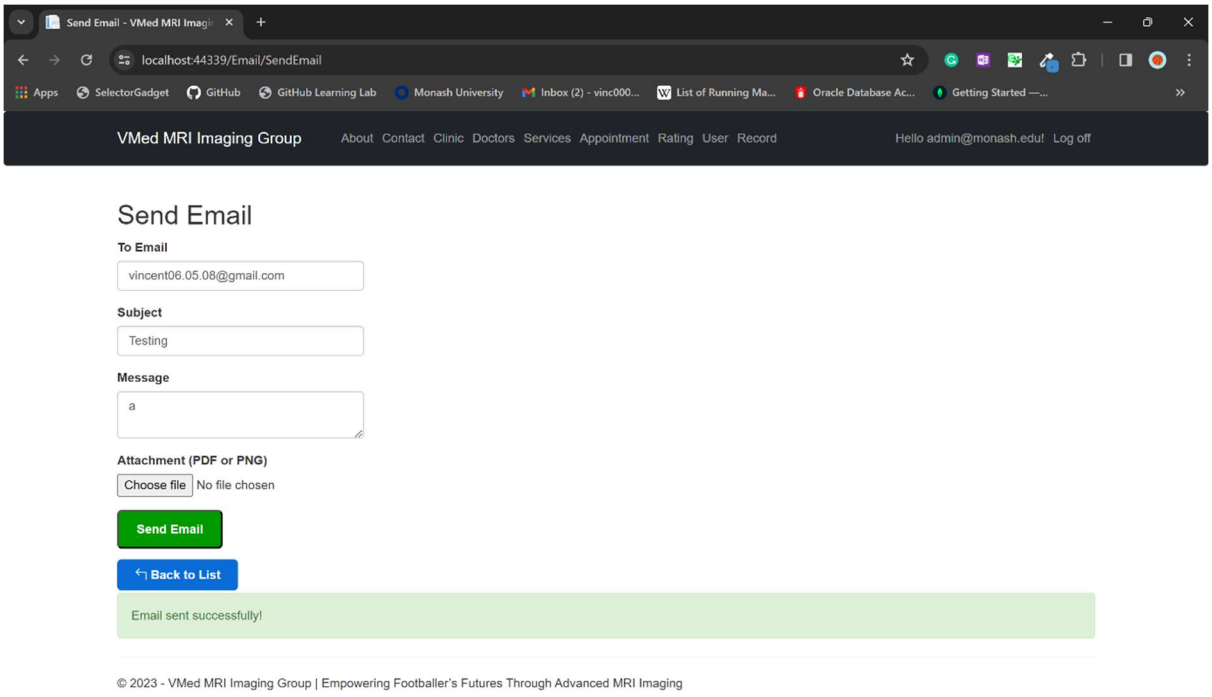


Figure 20. Feedback Principle (Norman)

Affordance

Affordance refers to the perceived or intuitive properties of an object or interface that suggest how it should be used (Norman, 1988; Educative, 2023). All the clickable buttons will be designed in such a way that can be perceived by users such as "Book Appointment" button in the Home page.

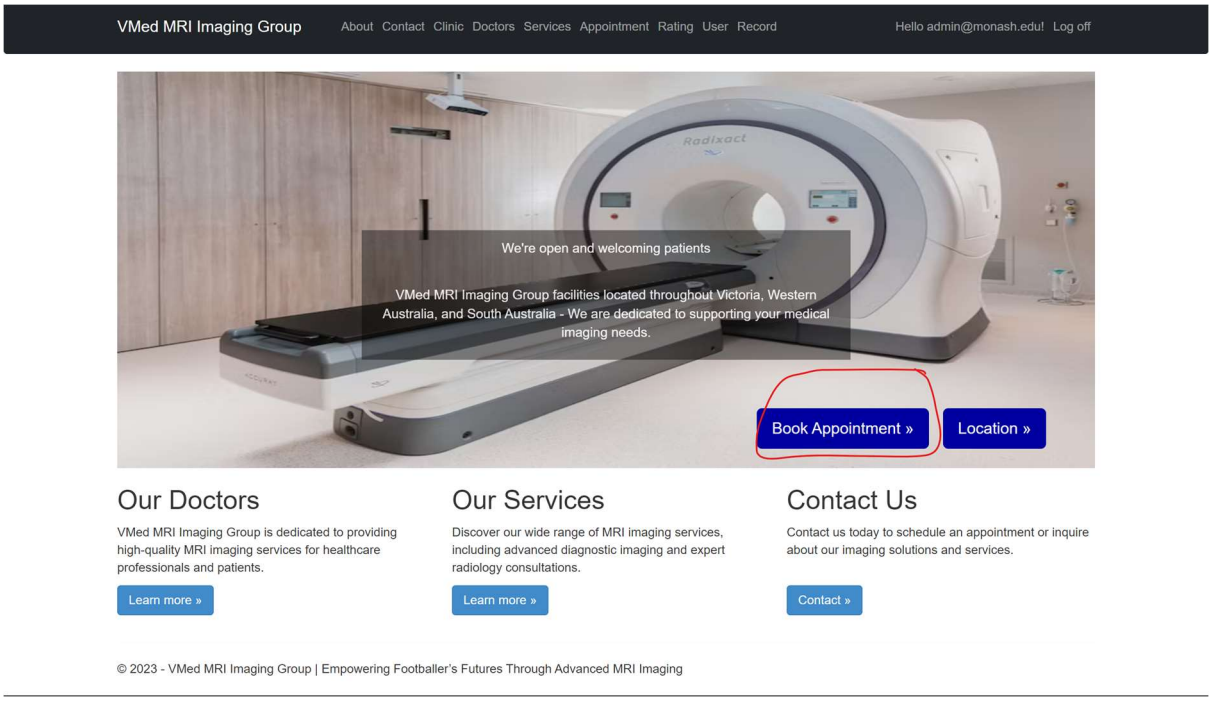


Figure 21. "Book Appointment" Button Satisfies Affordance Principle (Norman)

Constraint

Constraints are limitations or restrictions placed on a design to prevent users from making errors or taking unintended actions (Norman, 1988; Educative, 2023). As briefly discussed in "Visibility" section, the "Footballer" role will have a lot of constraint especially in terms of CRUD activity within the web application.

Figure 22. Administrator Screen: Constraint Principle (Norman) Part 1

Figure 23. Footballer Screen: Constraint Principle (Norman) Part 1

Consistency

Consistency in design ensures that elements, actions, and terminology remain the same or follow predictable patterns across an interface (Norman, 1988; Educative, 2023). It will be an icon that represents deleting (i.e., bin) and editing (i.e., pencil and paper) when the "Administrator" role wants to perform a specific CRUD activity. All of these icons are consistent with other web applications.

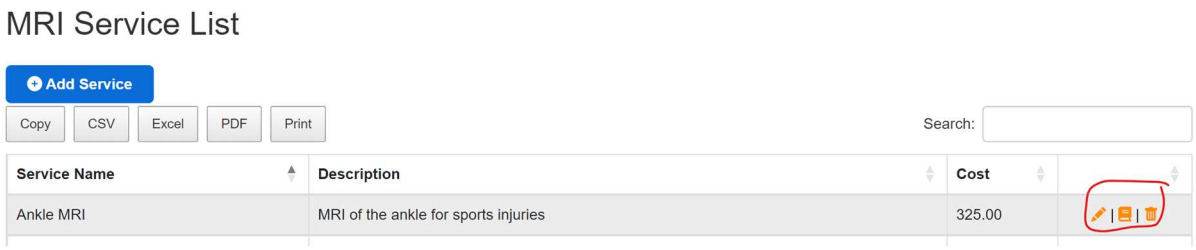


Figure 24. Consistent Icons - Consistency Principle (Norman)

9. Development Methodology

For this portfolio submission, I used the Code & Fix development methodology. This approach involves an iterative process of coding, testing, and refining to achieve the desired outcomes. To illustrate, in week 8, I just noticed in the studio class that I have done a wrong Database First approach implementation in which I have create two separate databases. Due to this mistake, my web application cannot connect the UserId in the AspNetUsers entity to my Appointment entity. Therefore, I need to redo the SQL query steps to generate the database schema or model.

Furthermore, as the Code & Fix model encourages rapid iterations, I continuously tested and evaluated the code as I progressed, making improvements and adjustments as needed. This iterative process allowed me to identify and address issues early in the development cycle. To illustrate, in the process of completing DataTable and Mapbox, the provided examples in Moodle are outdated in which all CDN versions cannot be used anymore as well as some parts of the provided codes (e.g., location.js in the Map Post Activity). Therefore, I need to continuously evaluate the Javascript code to achieve my desired visual representation of the clinics on a map and interactive DataTables.

Finally, while the Code & Fix model does not emphasize comprehensive documentation upfront, I made sure to document the changes, updates, and rationale behind decisions made

during the development process in a separate Word document, which also help me in handling the versioning of my project as discussed in the next section (available upon request).

10. Versioning

For this project, I utilized Git as my version control system. Git is a widely recognized and versatile tool for tracking changes in code and collaborating with others. Throughout the development process, I regularly committed changes to the Git repository. Each commit was accompanied by a descriptive message explaining the purpose and context of the modifications made which allows for a clear and organized version history.

Proof of using Git:

Git URL: https://github.com/vinctechgeek/FIT5032_VMedProject

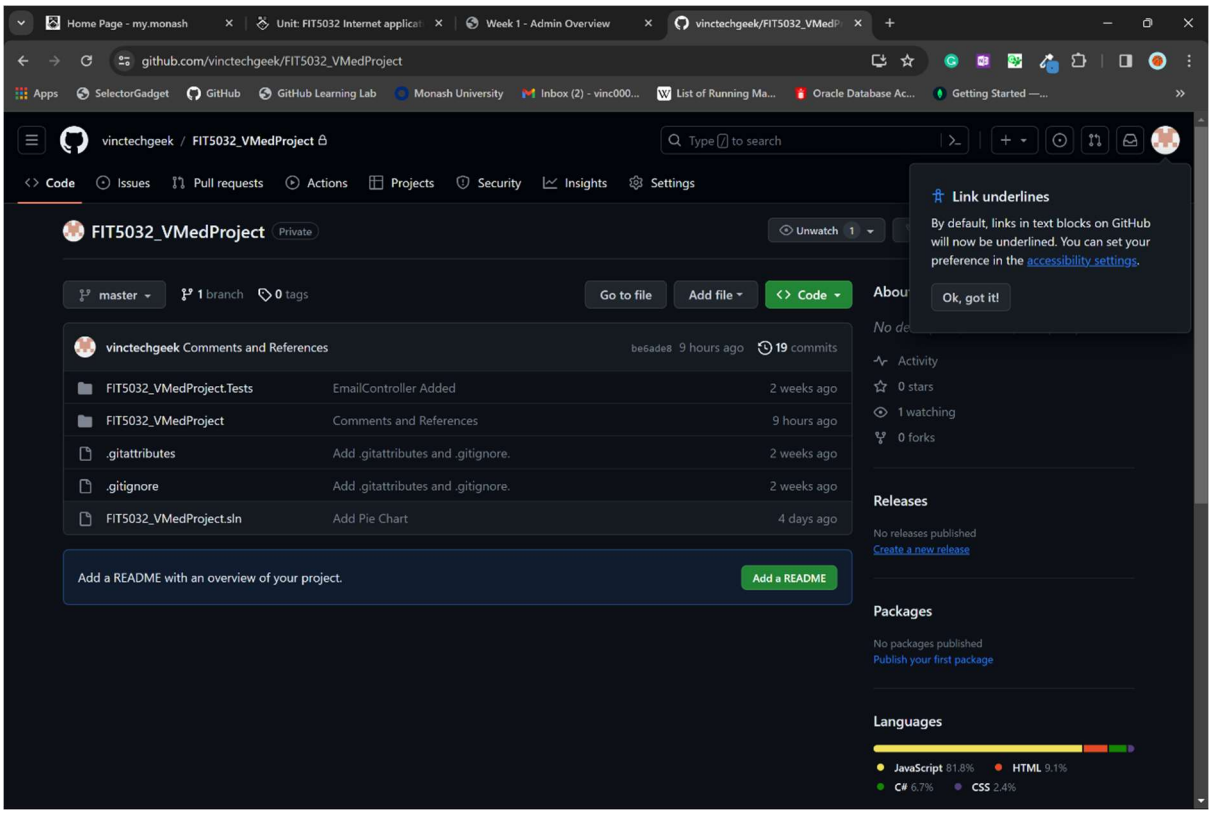


Figure 25. Proof of using Git Screenshot

The second approach is to store the foundation codes in a separate location. To illustrate, I have a folder called Database materials to store all the SQL queries to generate my whole model. The schema SQL query can be observed in section 4 above.

Proof:

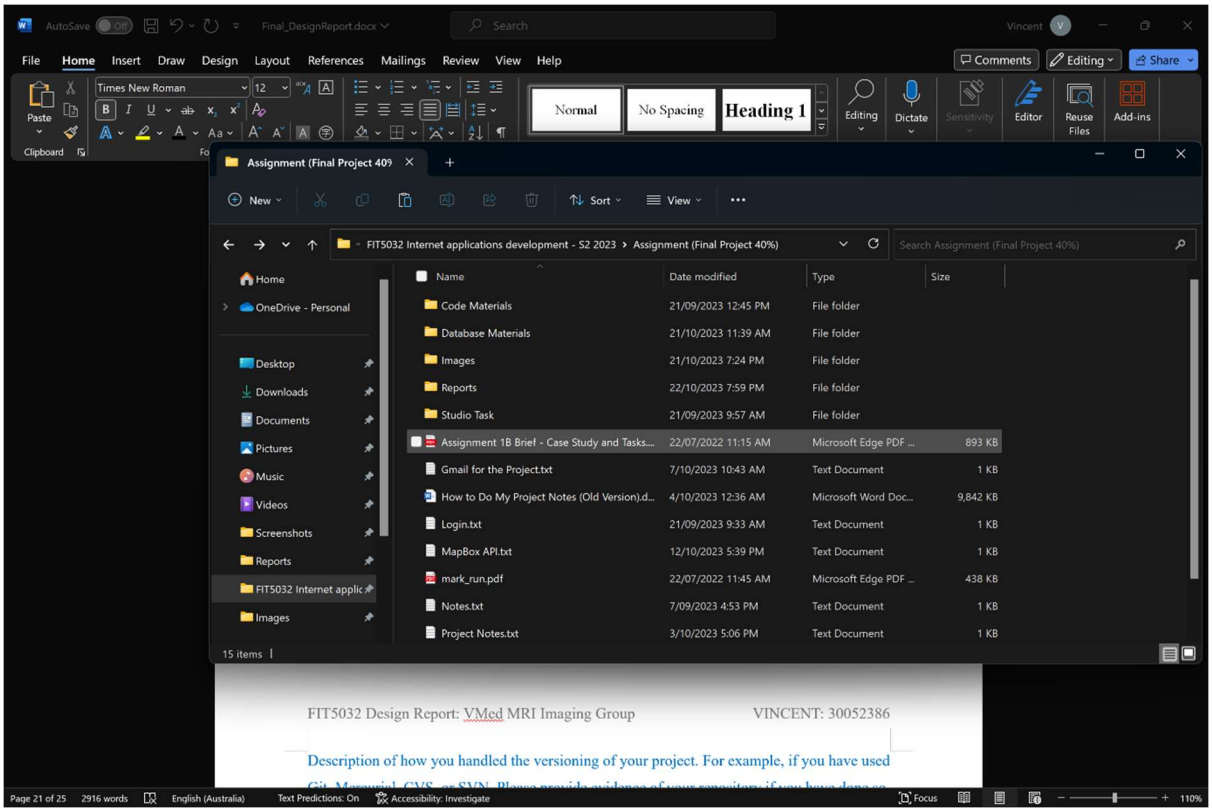


Figure 26. Foundation Codes Local Repository Screenshot

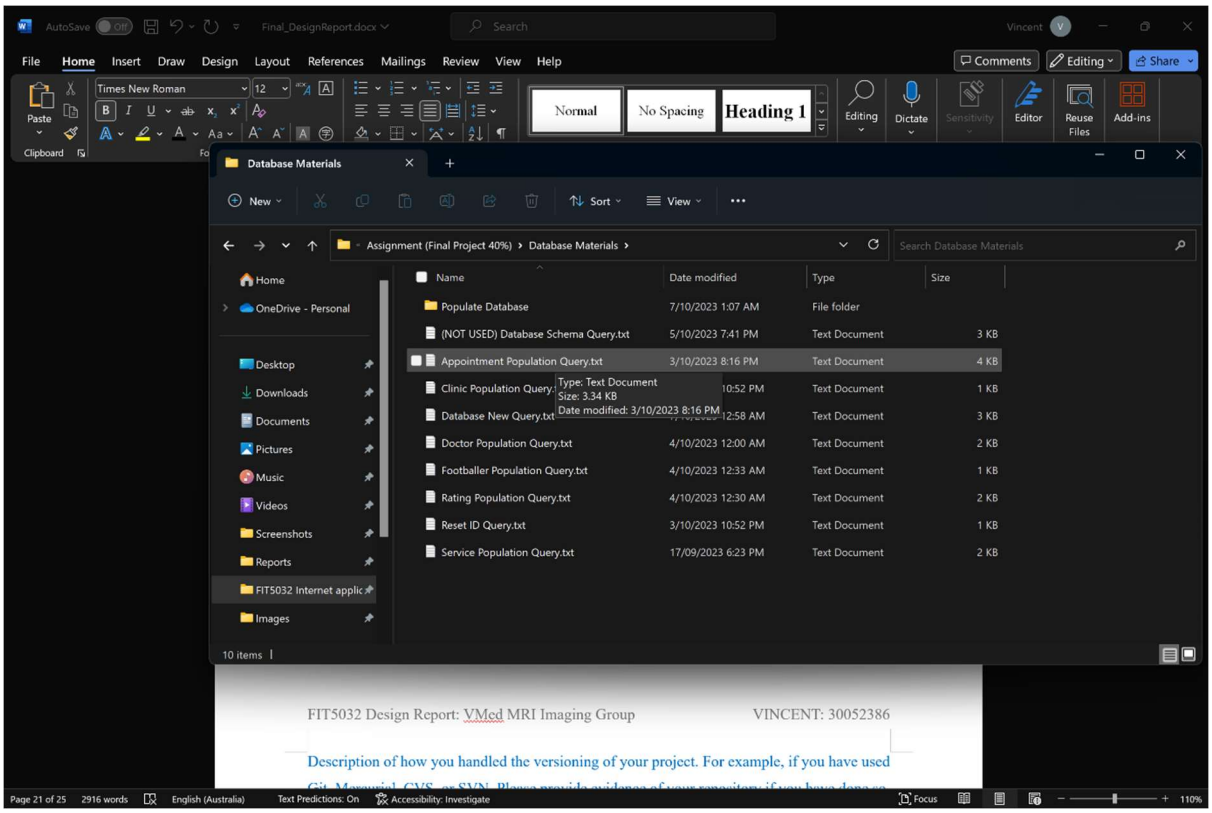


Figure 27. Database Materials Folder Screenshot

Finally, as discussed in previous section, I also created a Word document to document the changes, updates, and rationale behind decisions made.

Proof:

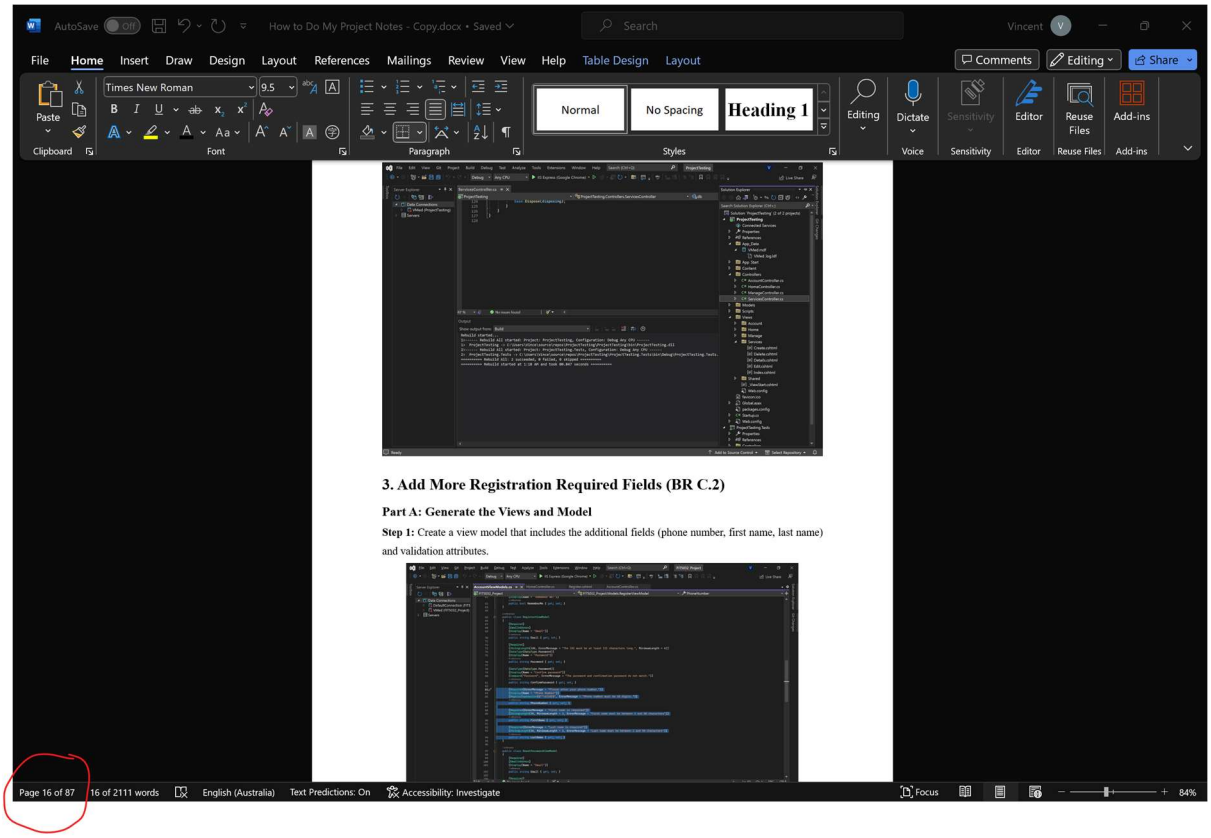


Figure 28. Word Document Notes Screenshot

11. Innovation and Research

User Story: [Enhancement of BR (D.3) Rating]

As a patient, I want to provide a review of a doctor's MRI service from my personal experience, so that I can help the community in making informed decisions about their choice of a doctor to consult.

Description:

For the implementation of the new web application, a significant innovation aimed at improving the user experience centres around the enhancement of the "Doctor MRI Service Reviews" feature (by enhancing BR (D.3) Rating). This enhancement was designed to encourage patients to provide comprehensive and insightful reviews of a doctor's MRI service based on their personal experiences. The primary goal of this innovation is to facilitate

informed decision-making within the community regarding the selection of a doctor to consult for MRI services.

Implementation:

Git URL: https://github.com/vinctechgeek/FIT5032_VMedProject

Final Presentation:

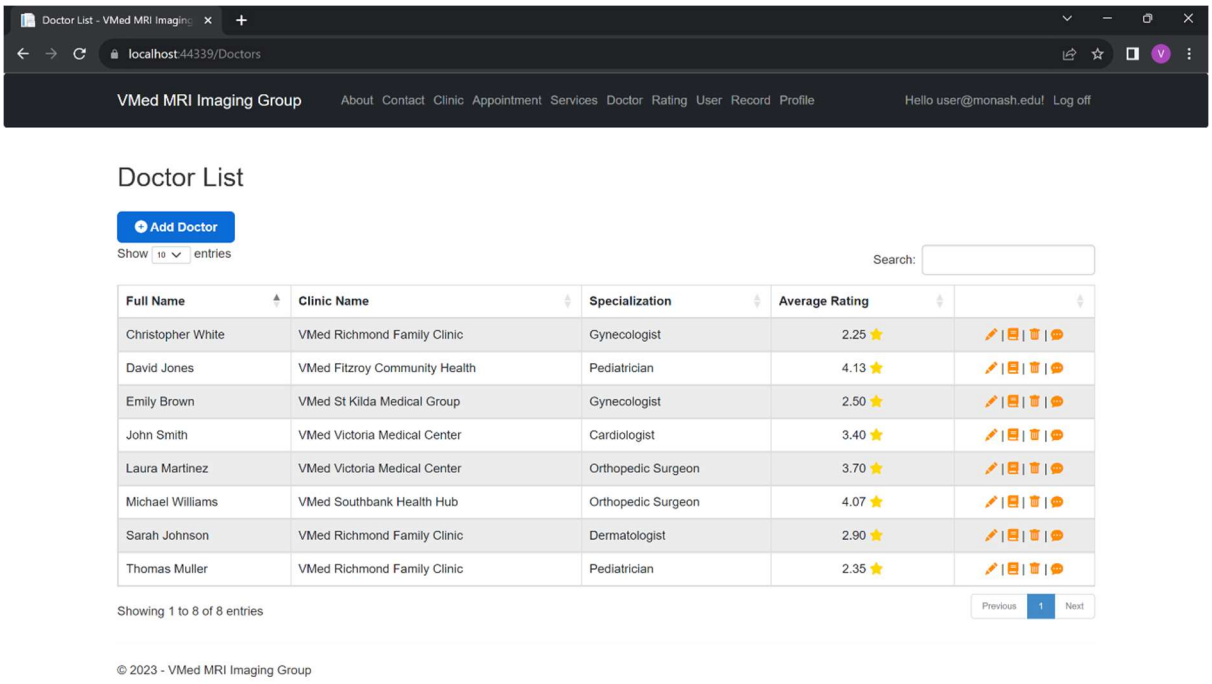


Figure 29. Review Implementation Screenshot [Part 1]

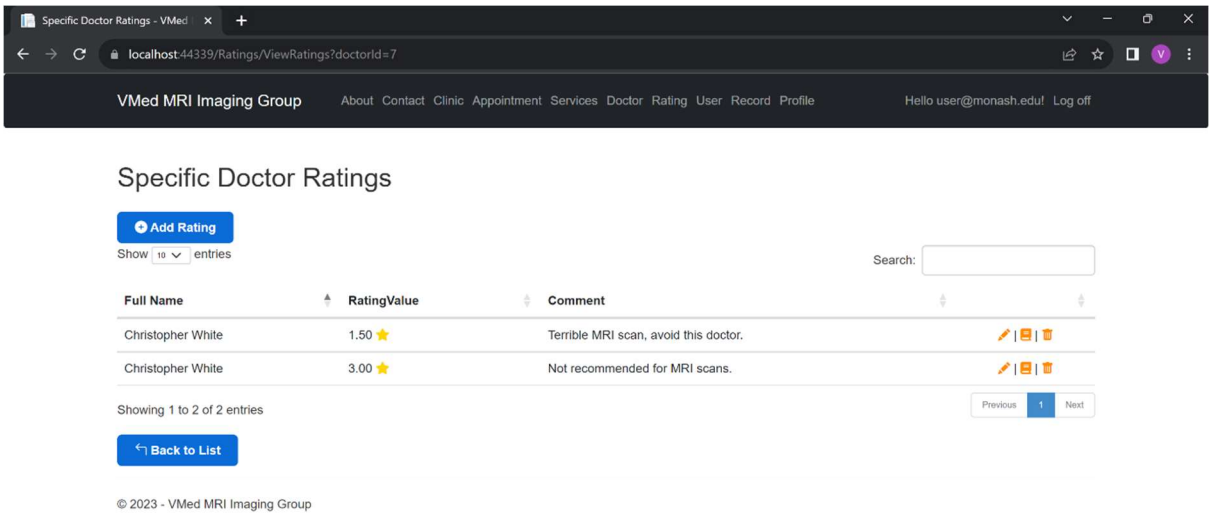


Figure 30. Review Implementation Screenshot [Part 2]

Controller: Full Implementation Refer to DoctorsController.cs and RatingsController.cs in the Git Repository.

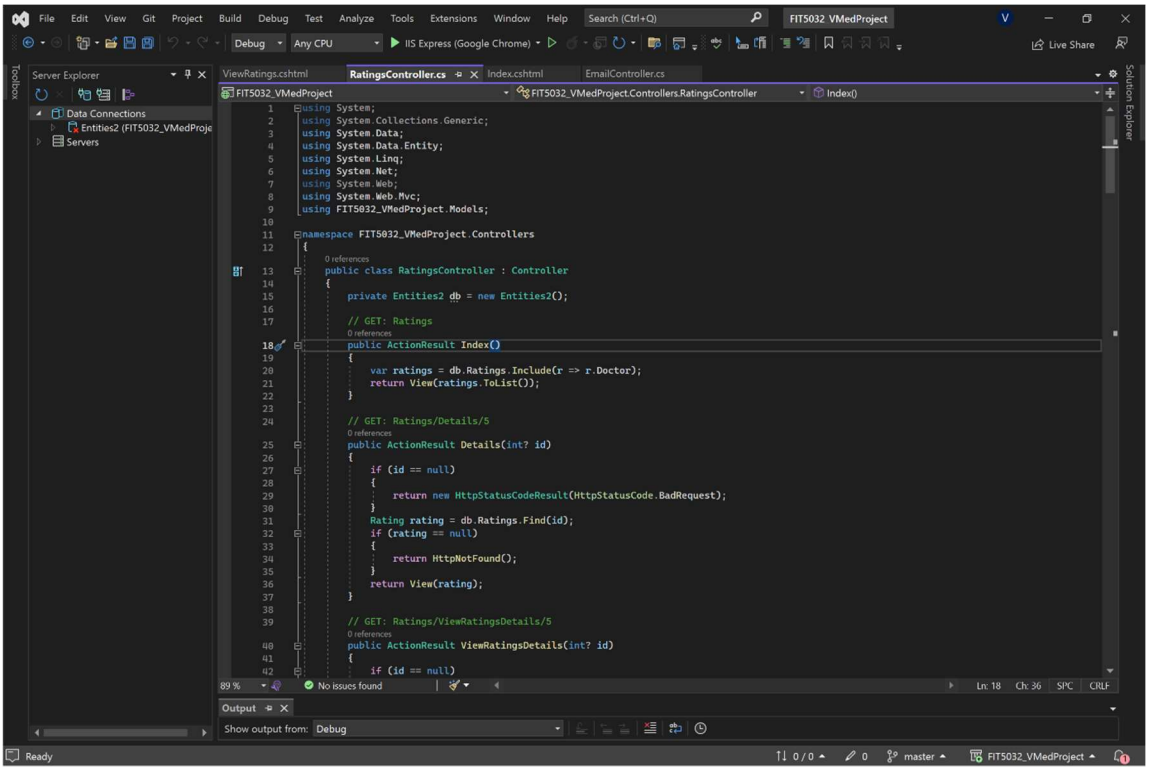
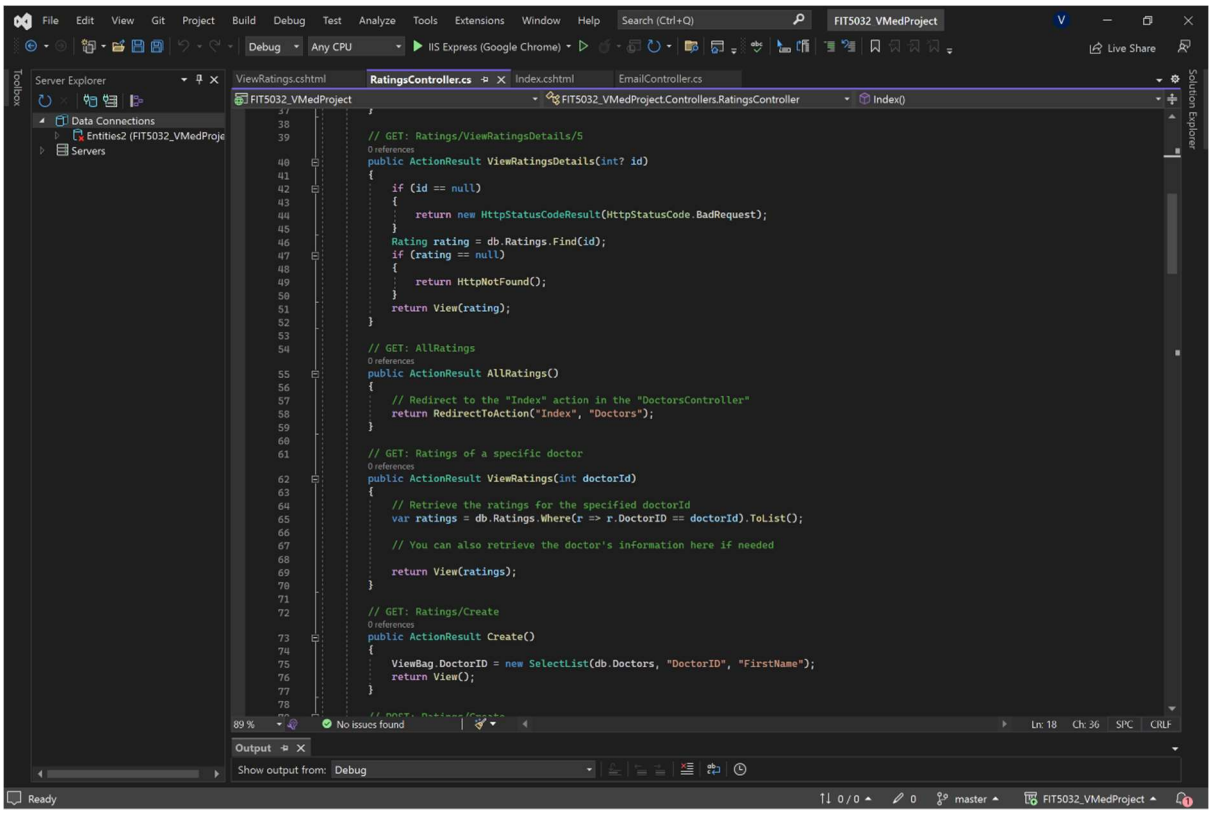
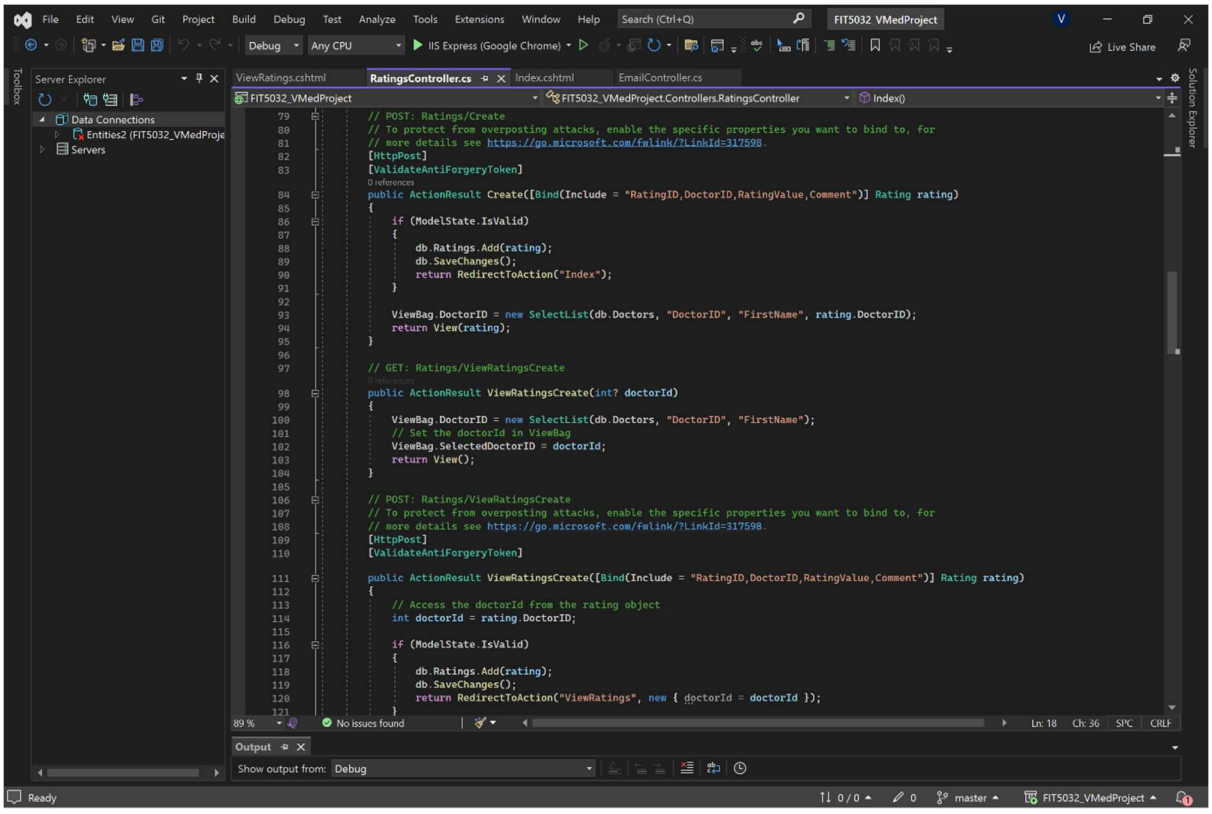


Figure 31. RatingsController.cs Screenshot [Part 1]



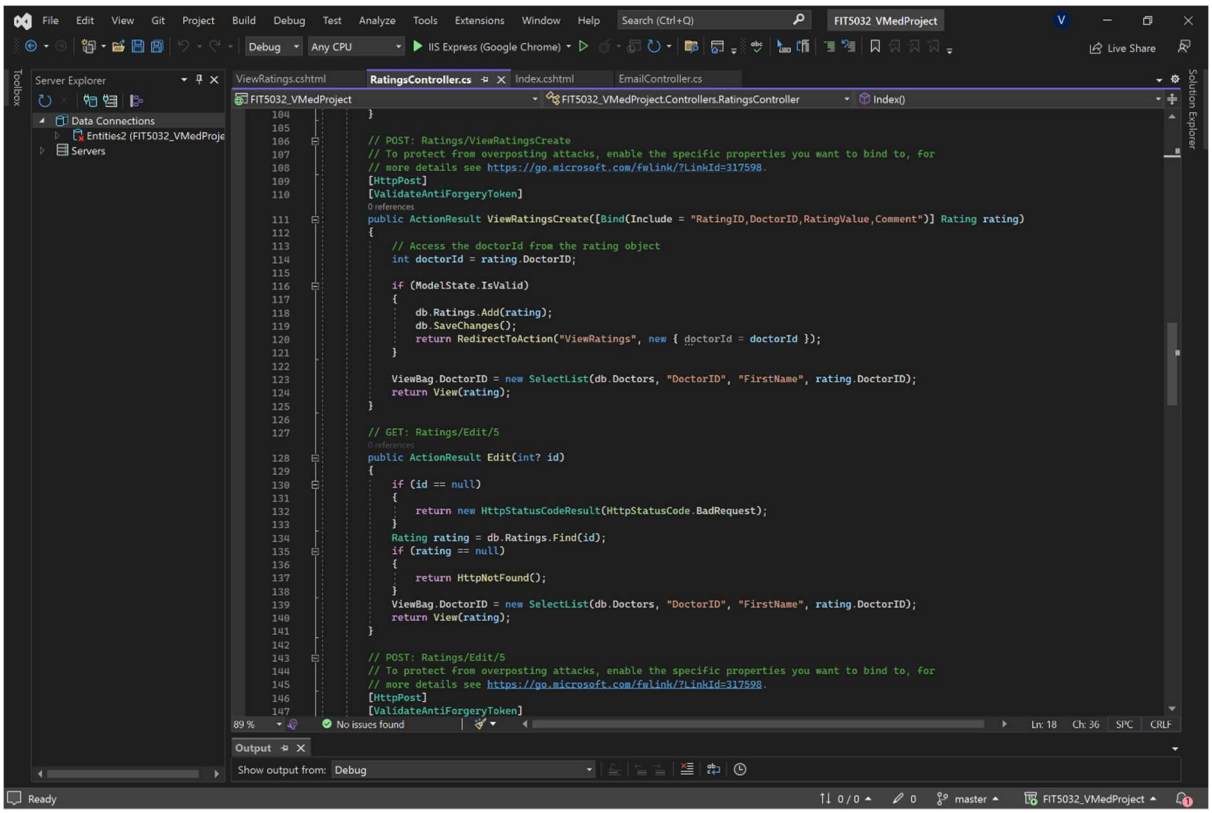
```
// GET: Ratings/ViewRatingsDetails/5
public ActionResult ViewRatingsDetails(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Rating rating = db.Ratings.Find(id);
    if (rating == null)
    {
        return HttpNotFound();
    }
    return View(rating);
}

// GET: AllRatings
public ActionResult AllRatings()
{
    // Redirect to the "Index" action in the "DoctorsController"
    return RedirectToAction("Index", "Doctors");
}

// GET: Ratings of a specific doctor
public ActionResult ViewRatings(int doctorId)
{
    // Retrieve the ratings for the specified doctorId
    var ratings = db.Ratings.Where(r => r.DoctorID == doctorId).ToList();

    // You can also retrieve the doctor's information here if needed

    return View(ratings);
}

// GET: Ratings/Create
public ActionResult Create()
{
    ViewBag.DoctorID = new SelectList(db.Doctors, "DoctorID", "FirstName");
    return View();
}
```

Figure 32. RatingsController.cs Screenshot [Part 2]



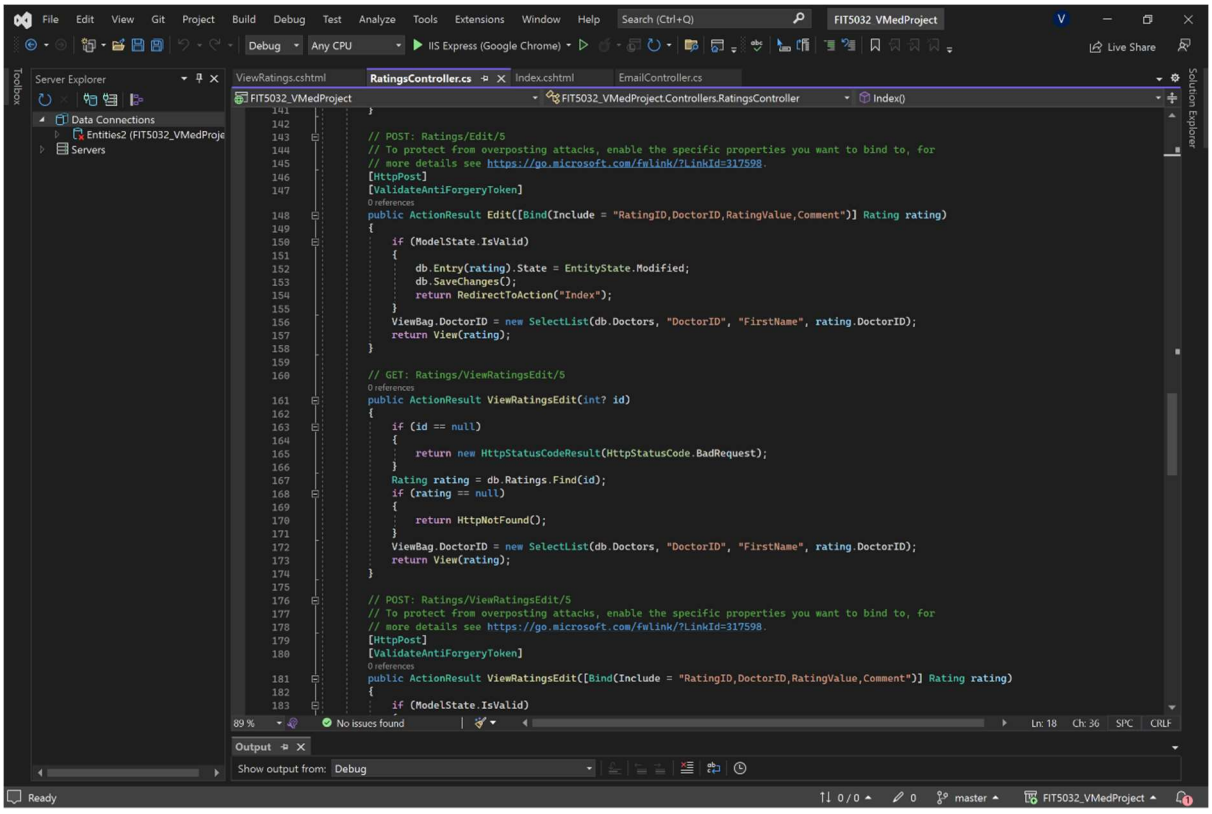
```
// POST: Ratings/Create
// To protect from overposting attacks, enable the specific properties you want to bind to, for
// more details see https://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include = "RatingID,DoctorID,RatingValue,Comment")] Rating rating)
{
    if (ModelState.IsValid)
    {
        db.Ratings.Add(rating);
        db.SaveChanges();
        return RedirectToAction("Index");
    }

    ViewBag.DoctorID = new SelectList(db.Doctors, "DoctorID", "FirstName", rating.DoctorID);
    return View(rating);
}

// GET: Ratings/ViewRatingsCreate
public ActionResult ViewRatingsCreate(int? doctorId)
{
    ViewBag.DoctorID = new SelectList(db.Doctors, "DoctorID", "FirstName");
    // Set the doctorId in ViewBag
    ViewBag.SelectedDoctorID = doctorId;
    return View();
}

// POST: Ratings/ViewRatingsCreate
// To protect from overposting attacks, enable the specific properties you want to bind to, for
// more details see https://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]

public ActionResult ViewRatingsCreate([Bind(Include = "RatingID,DoctorID,RatingValue,Comment")] Rating rating)
{
    // Access the doctorId from the rating object
    int doctorId = rating.DoctorID;

    if (ModelState.IsValid)
    {
        db.Ratings.Add(rating);
        db.SaveChanges();
        return RedirectToAction("ViewRatings", new { doctorId = doctorId });
    }
```

Figure 33. RatingsController.cs Screenshot [Part 3]



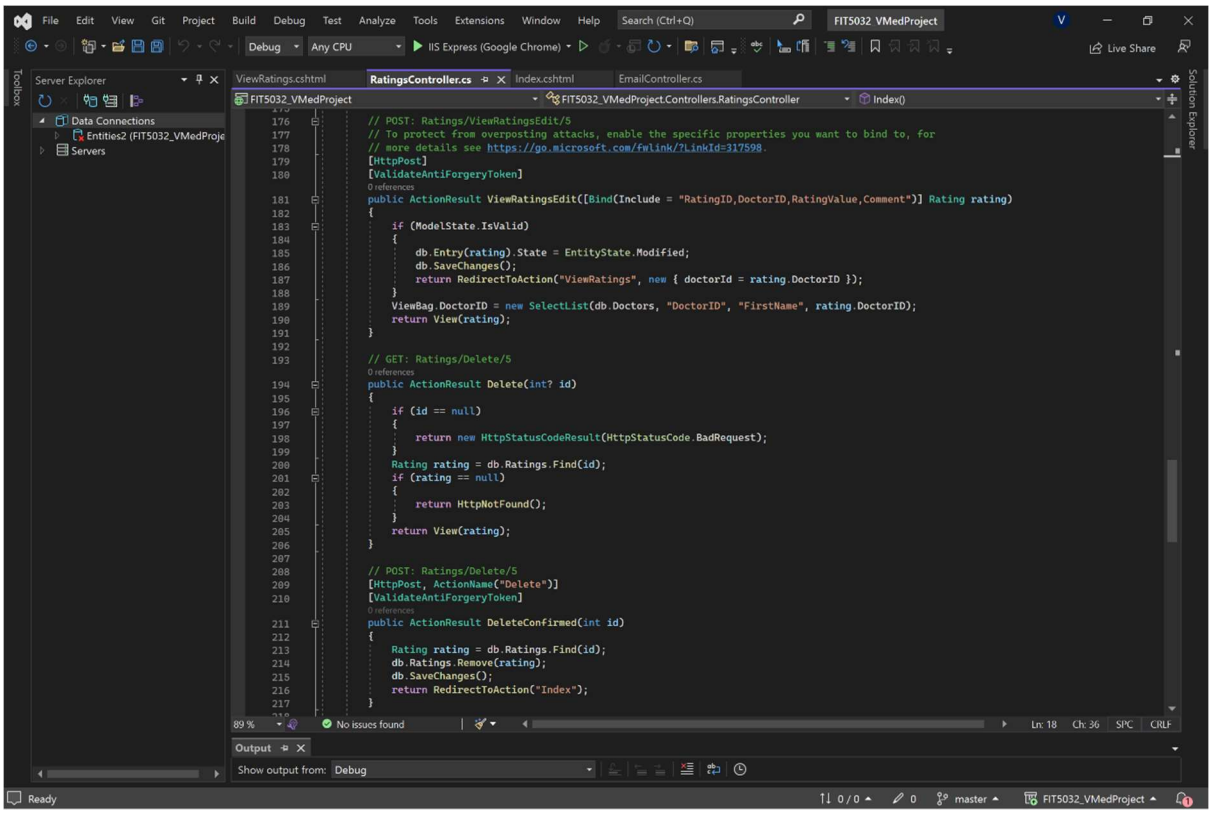
```
        }

        // POST: Ratings/ViewRatingsCreate
        // To protect from overposting attacks, enable the specific properties you want to bind to, for
        // more details see https://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        0 references
        public ActionResult ViewRatingsCreate([Bind(Include = "RatingID,DoctorID,RatingValue,Comment")] Rating rating)
        {
            // Access the doctorId from the rating object
            int doctorId = rating.DoctorID;

            if (ModelState.IsValid)
            {
                db.Ratings.Add(rating);
                db.SaveChanges();
                return RedirectToAction("ViewRatings", new { doctorId = doctorId });
            }

            ViewBag.DoctorID = new SelectList(db.Doctors, "DoctorID", "FirstName", rating.DoctorID);
            return View(rating);
        }

        // GET: Ratings/Edit/5
        0 references
        public ActionResult Edit(int? id)
        {
            if (id == null)
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Rating rating = db.Ratings.Find(id);
            if (rating == null)
            {
                return HttpNotFound();
            }
            ViewBag.DoctorID = new SelectList(db.Doctors, "DoctorID", "FirstName", rating.DoctorID);
            return View(rating);
        }

        // POST: Ratings/Edit/5
        // To protect from overposting attacks, enable the specific properties you want to bind to, for
        // more details see https://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
```

Figure 34. RatingsController.cs Screenshot [Part 4]



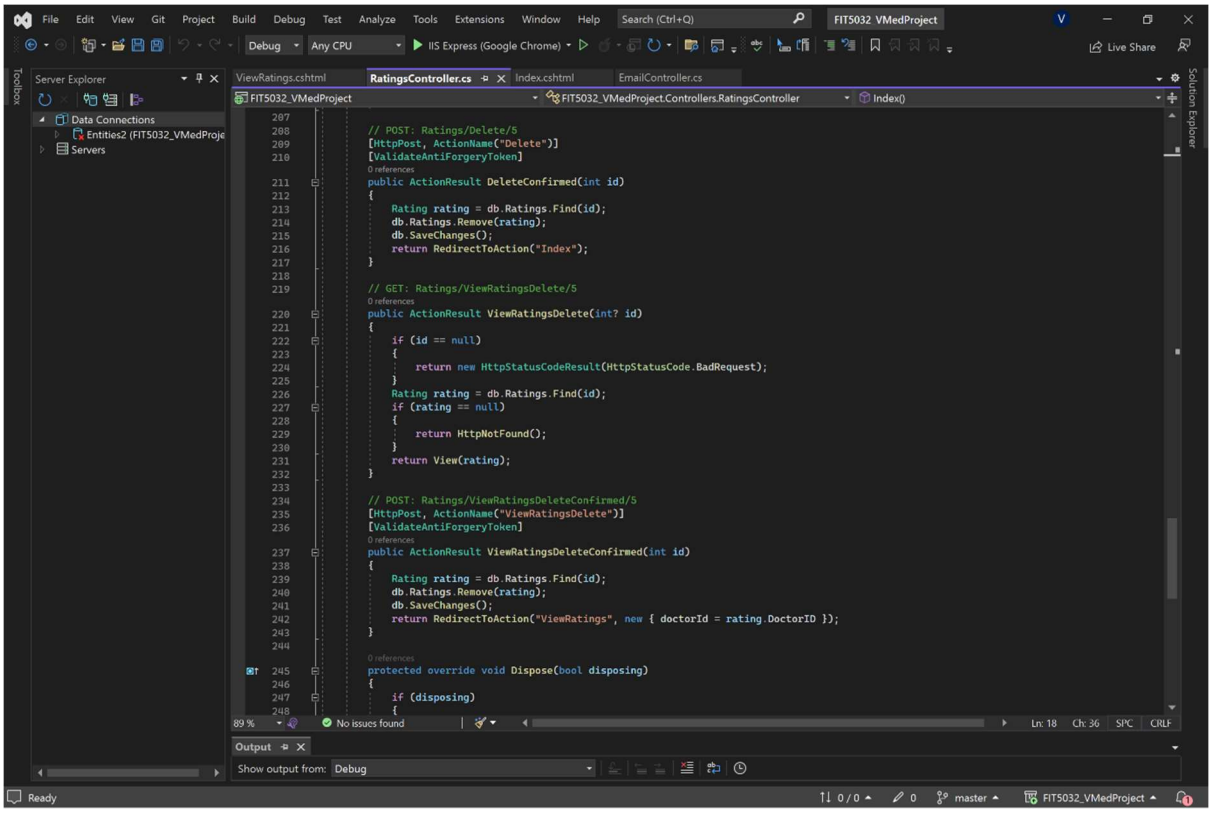
```
        // POST: Ratings/Edit/5
        // To protect from overposting attacks, enable the specific properties you want to bind to, for
        // more details see https://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        0 references
        public ActionResult Edit([Bind(Include = "RatingID,DoctorID,RatingValue,Comment")] Rating rating)
        {
            if (ModelState.IsValid)
            {
                db.Entry(rating).State = EntityState.Modified;
                db.SaveChanges();
                return RedirectToAction("Index");
            }
            ViewBag.DoctorID = new SelectList(db.Doctors, "DoctorID", "FirstName", rating.DoctorID);
            return View(rating);
        }

        // GET: Ratings/ViewRatingsEdit/5
        0 references
        public ActionResult ViewRatingsEdit(int? id)
        {
            if (id == null)
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Rating rating = db.Ratings.Find(id);
            if (rating == null)
            {
                return HttpNotFound();
            }
            ViewBag.DoctorID = new SelectList(db.Doctors, "DoctorID", "FirstName", rating.DoctorID);
            return View(rating);
        }

        // POST: Ratings/ViewRatingsEdit/5
        // To protect from overposting attacks, enable the specific properties you want to bind to, for
        // more details see https://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        0 references
        public ActionResult ViewRatingsEdit([Bind(Include = "RatingID,DoctorID,RatingValue,Comment")] Rating rating)
        {
            if (ModelState.IsValid)
```

Figure 35. RatingsController.cs Screenshot [Part 5]



```
// POST: Ratings/ViewRatingsEdit/5
// To protect from overposting attacks, enable the specific properties you want to bind to, for
// more details see https://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public ActionResult ViewRatingsEdit([Bind(Include = "RatingID,DoctorID,RatingValue,Comment")] Rating rating)
{
    if (ModelState.IsValid)
    {
        db.Entry(rating).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("ViewRatings", new { doctorId = rating.DoctorID });
    }
    ViewBag.DoctorID = new SelectList(db.Doctors, "DoctorID", "FirstName", rating.DoctorID);
    return View(rating);
}

// GET: Ratings/Delete/5
0 references
public ActionResult Delete(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Rating rating = db.Ratings.Find(id);
    if (rating == null)
    {
        return HttpNotFound();
    }
    return View(rating);
}

// POST: Ratings/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 references
public ActionResult DeleteConfirmed(int id)
{
    Rating rating = db.Ratings.Find(id);
    db.Ratings.Remove(rating);
    db.SaveChanges();
    return RedirectToAction("Index");
}
```

Figure 36. RatingsController.cs Screenshot [Part 6]



```
// POST: Ratings/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 references
public ActionResult DeleteConfirmed(int id)
{
    Rating rating = db.Ratings.Find(id);
    db.Ratings.Remove(rating);
    db.SaveChanges();
    return RedirectToAction("Index");
}

// GET: Ratings/ViewRatingsDelete/5
0 references
public ActionResult ViewRatingsDelete(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Rating rating = db.Ratings.Find(id);
    if (rating == null)
    {
        return HttpNotFound();
    }
    return View(rating);
}

// POST: Ratings/ViewRatingsDeleteConfirmed/5
[HttpPost, ActionName("ViewRatingsDelete")]
[ValidateAntiForgeryToken]
0 references
public ActionResult ViewRatingsDeleteConfirmed(int id)
{
    Rating rating = db.Ratings.Find(id);
    db.Ratings.Remove(rating);
    db.SaveChanges();
    return RedirectToAction("ViewRatings", new { doctorId = rating.DoctorID });
}

0 references
protected override void Dispose(bool disposing)
{
    if (disposing)
    {
```

Figure 37. RatingsController.cs Screenshot [Part 7]

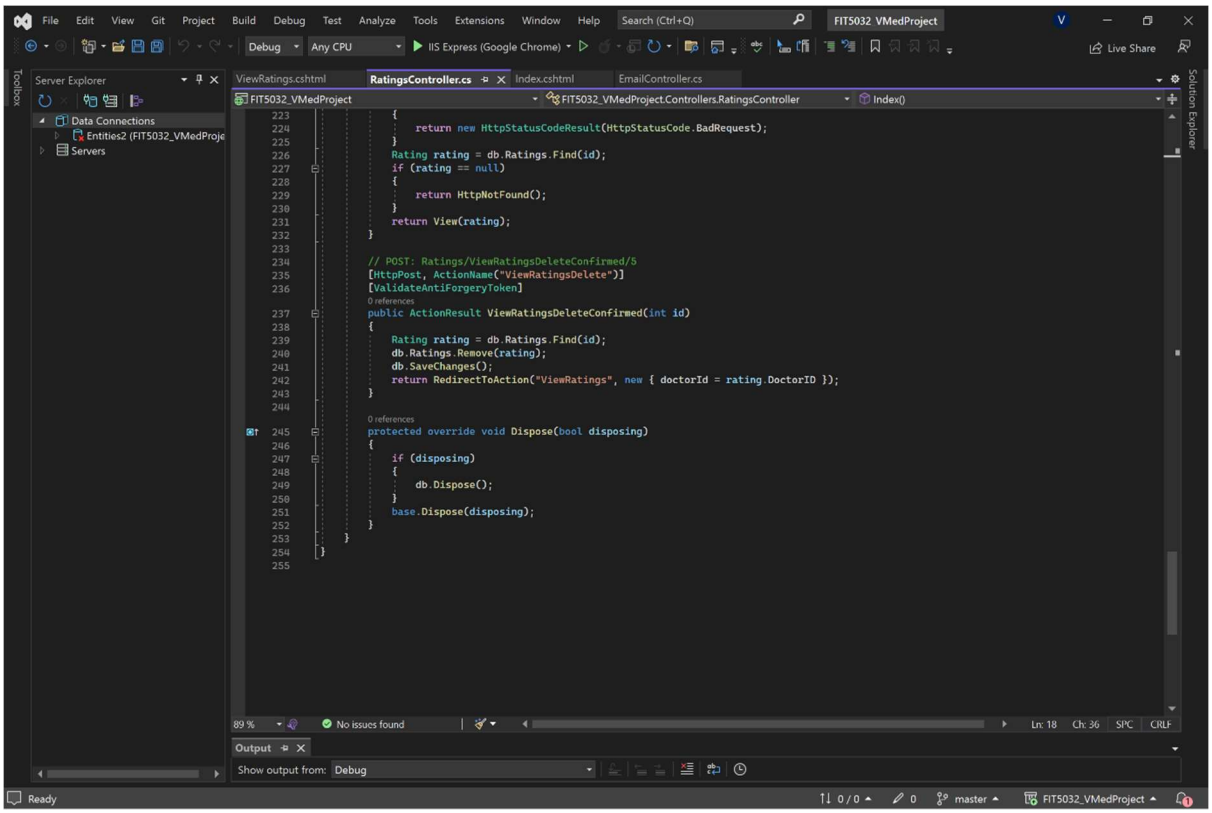


Figure 38. RatingsController.cs Screenshot [Part 8]

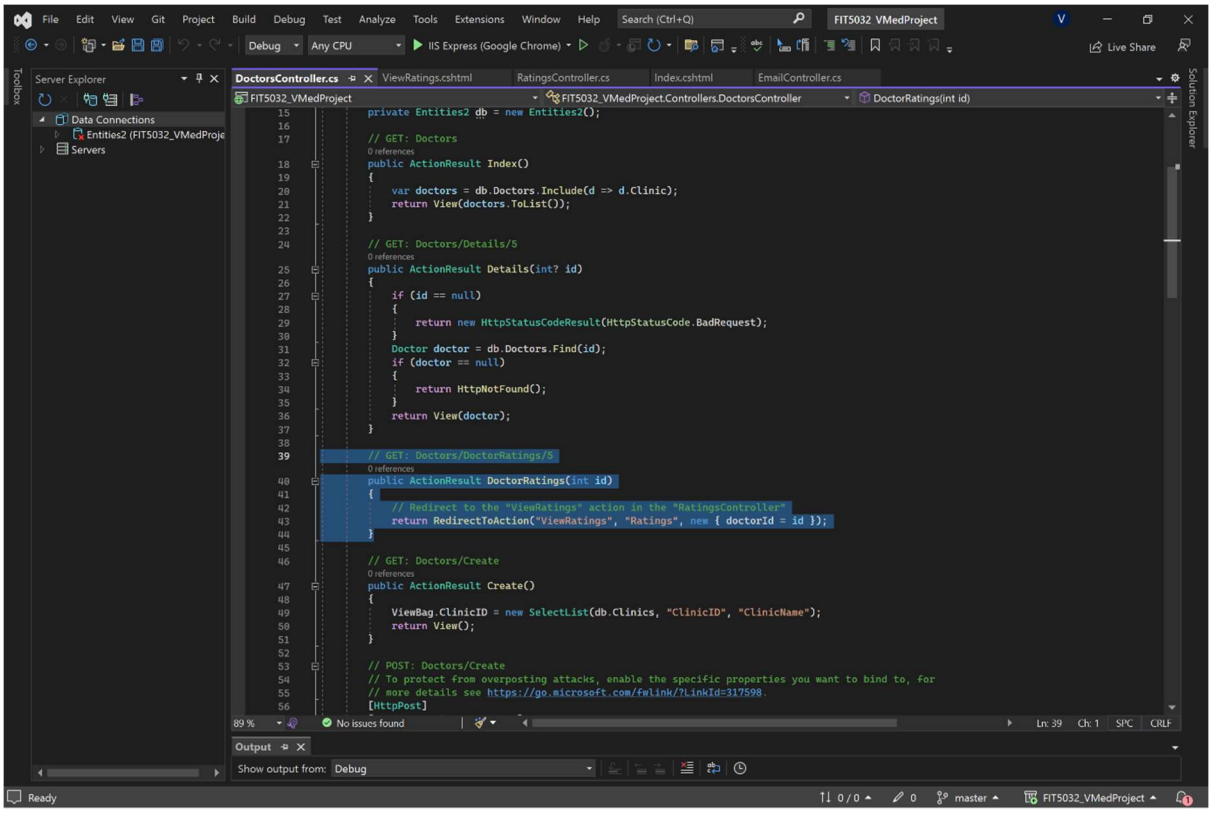


Figure 39. Redirect Action Method from Doctors/Index.cshtml to ViewRatings.cshtml in DoctorsController.cs Screenshot

View: Full Implementation Refer to Views/Doctors/Index.cshtml and Views/Ratings/ViewRatings[Create/Delete/Details/Edit].cshtml in the Git Repository.

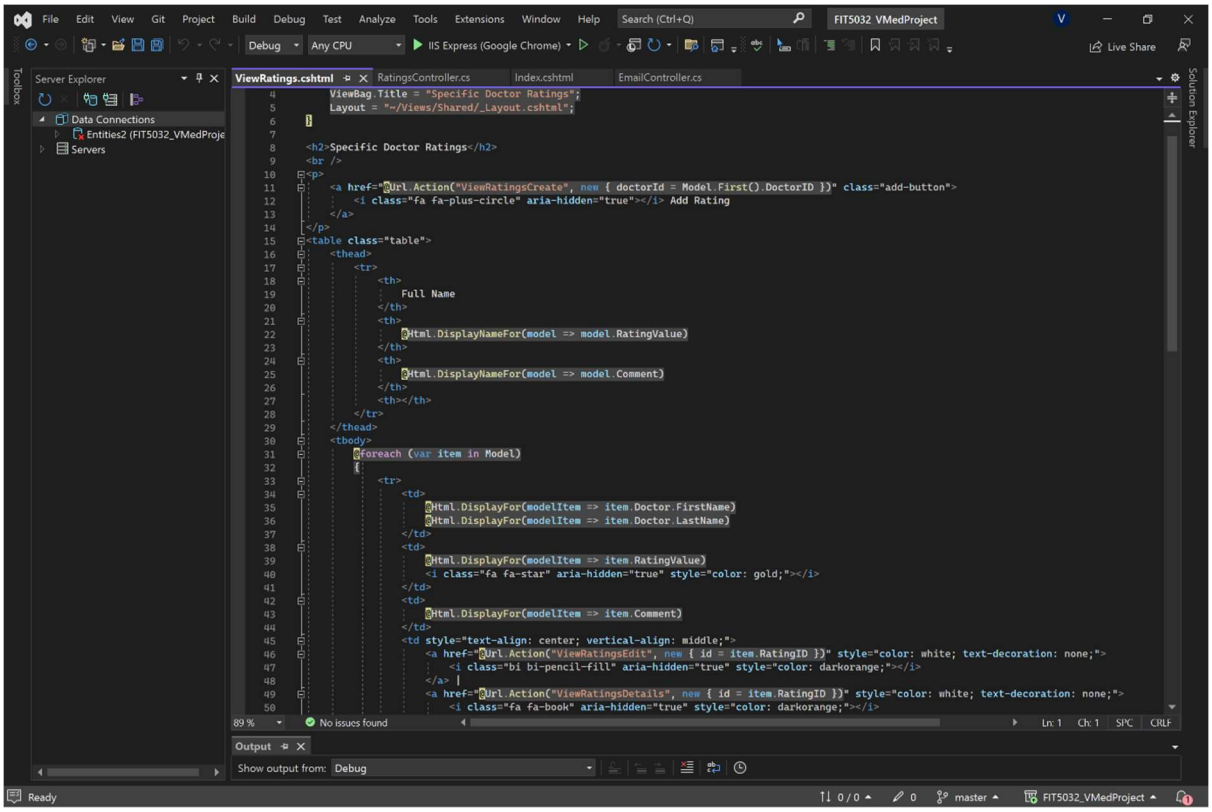


Figure 40. ViewRatings.cshtml Screenshot [Part 1]

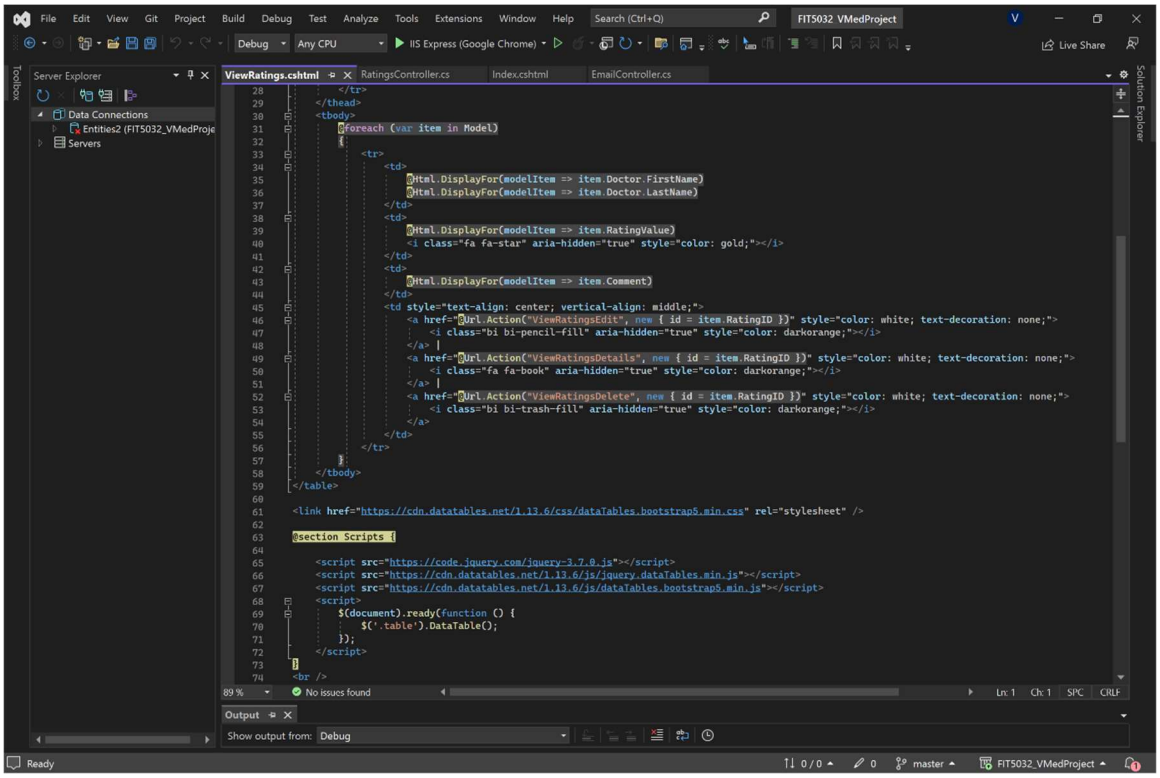


Figure 41. ViewRatings.cshtml Screenshot [Part 2]

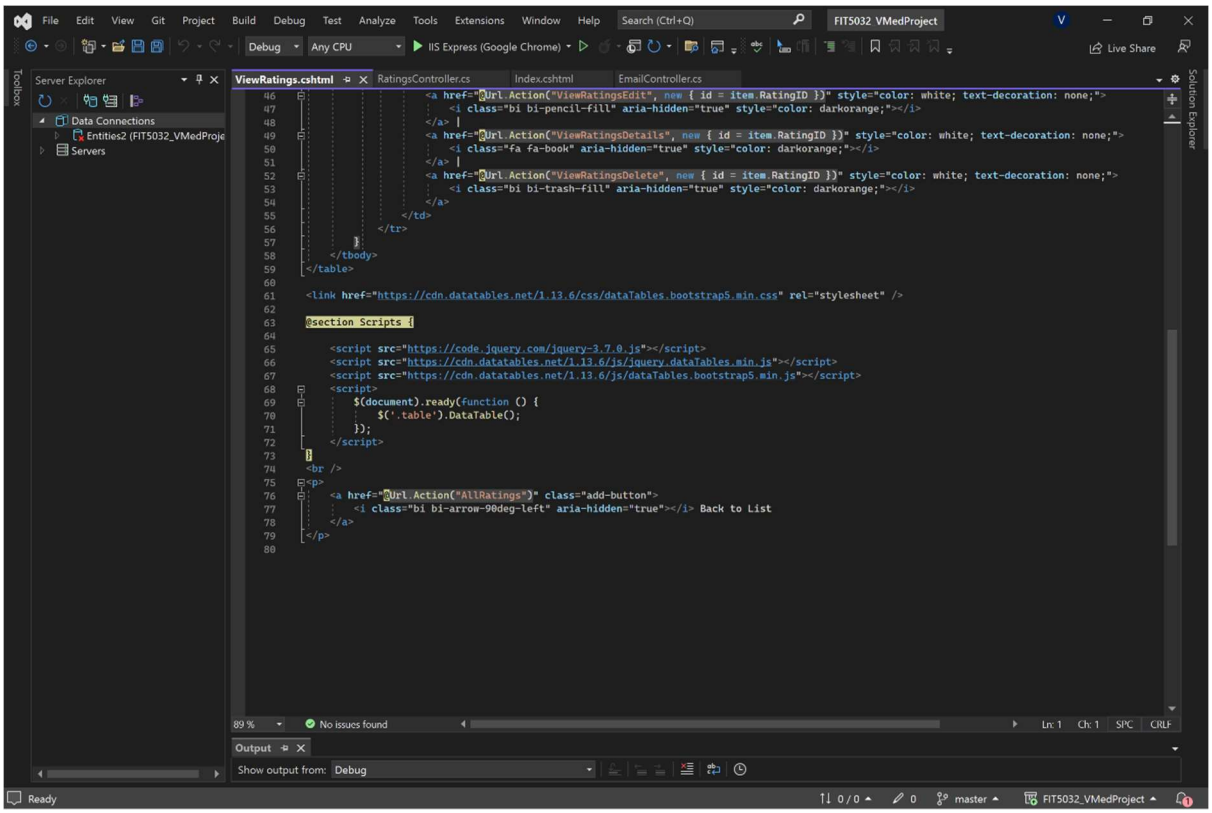


Figure 42. ViewRatings.cshtml Screenshot [Part 3]

12. Checklist of Site Functionality

1. (Layout Page)	TICK if complete
Good Design	V
Stylesheet	V
JavaScript	V
Menu	V
2. (Home page)	
Design and content	V
Banner Image	V
3. (User Log in)	
Web form and validation controls	V
Formatted data entry display	V
Overall page design	V
4. (Customised Views and Controllers)	
Customised Views	V
Customised Controllers	V
Other customisations	V
5. (Documentation)	
Code Comments	V
Attribution of Source of any code used	V
6 Business Requirements	
BR(A1): for P	V
BR(A2): for P	V
BR(B1): for C to C+	V
BR(B2): for C to C+	V
BR(C1): for C+ to C++	V
BR(C2): for C+ to C++	V
BR(C3): for C+ to C++	V
BR(C4): for C+ to C++	V
BR(D1): for D to D++	V
BR(D2): for D to D++	V
BR(D3): for D to D++	V
BR(D4): for D to D++	V
BR(E1): for HD to HD+	V
BR(E2): for HD to HD+	V
BR(F1): for HD+ to HD++	V
Audit	
No breaking of copyright	

Note: For the computer, OS, and Visual Studio specifications, refer to Appendix B.

References

Norman, D. A. (1988). *The psychology of everyday things*. Basic books.

Educative. (2023). *What are Norman's design principles?* https://www.educative.io/answers/what-are-normans-design-principles

Appendix

Appendix A

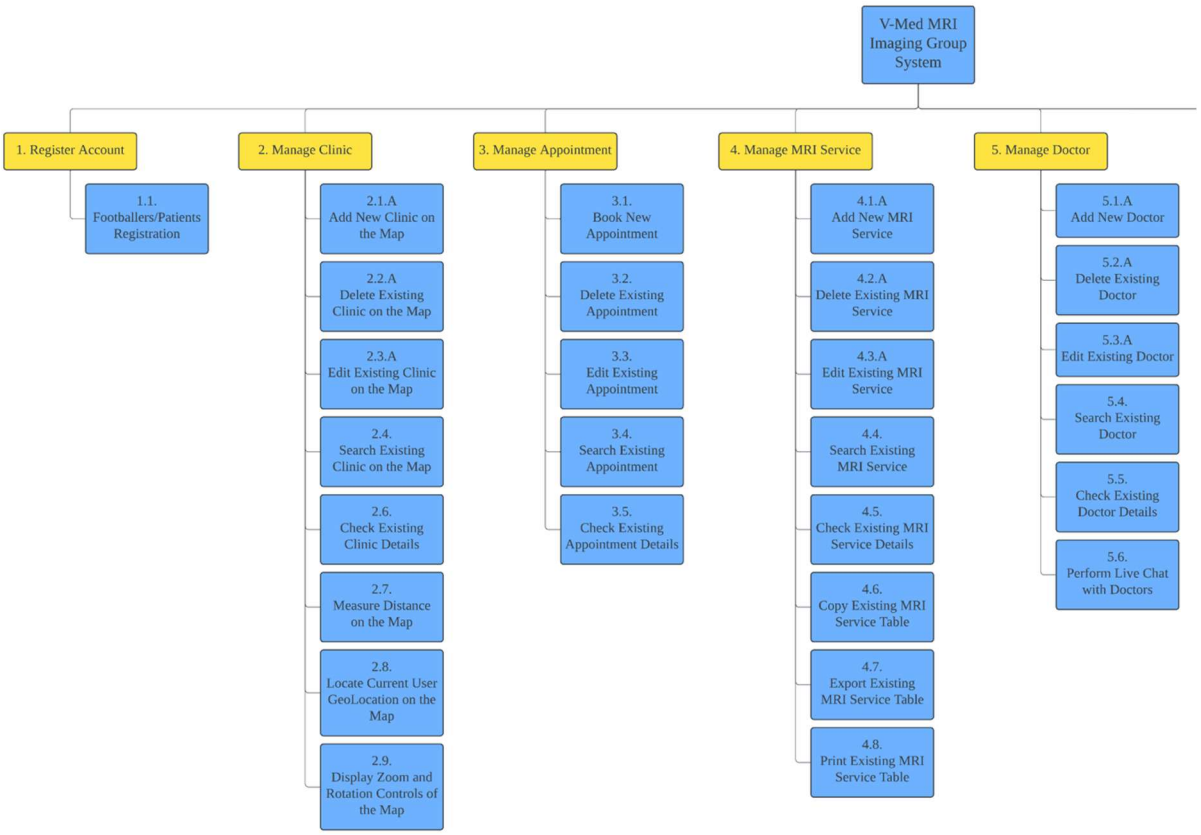


Figure 43. Block/Function Diagram Part 1

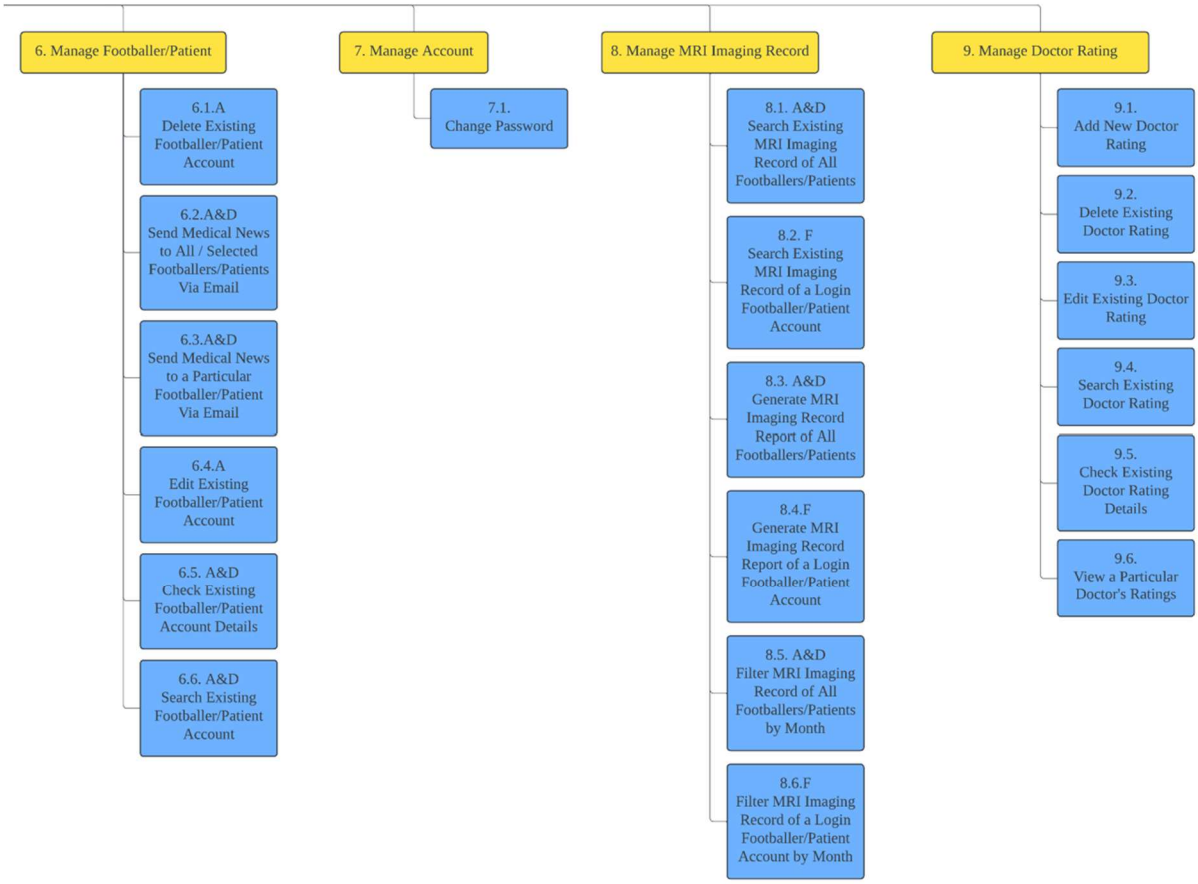


Figure 44. Block/Functional Diagram Part 2

Appendix B

Laptop: Surface Pro 7

OS: Windows 11

Visual Studio:

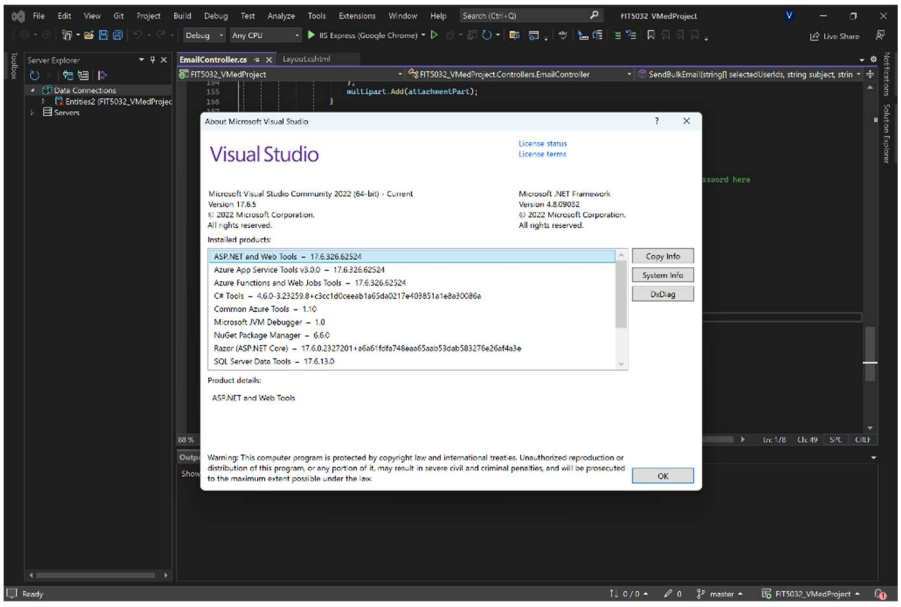


Figure 45. Visual Studio Version Screenshot