VENUEFLOW

Minh Triet Vu

Elizabeth Mokrusa

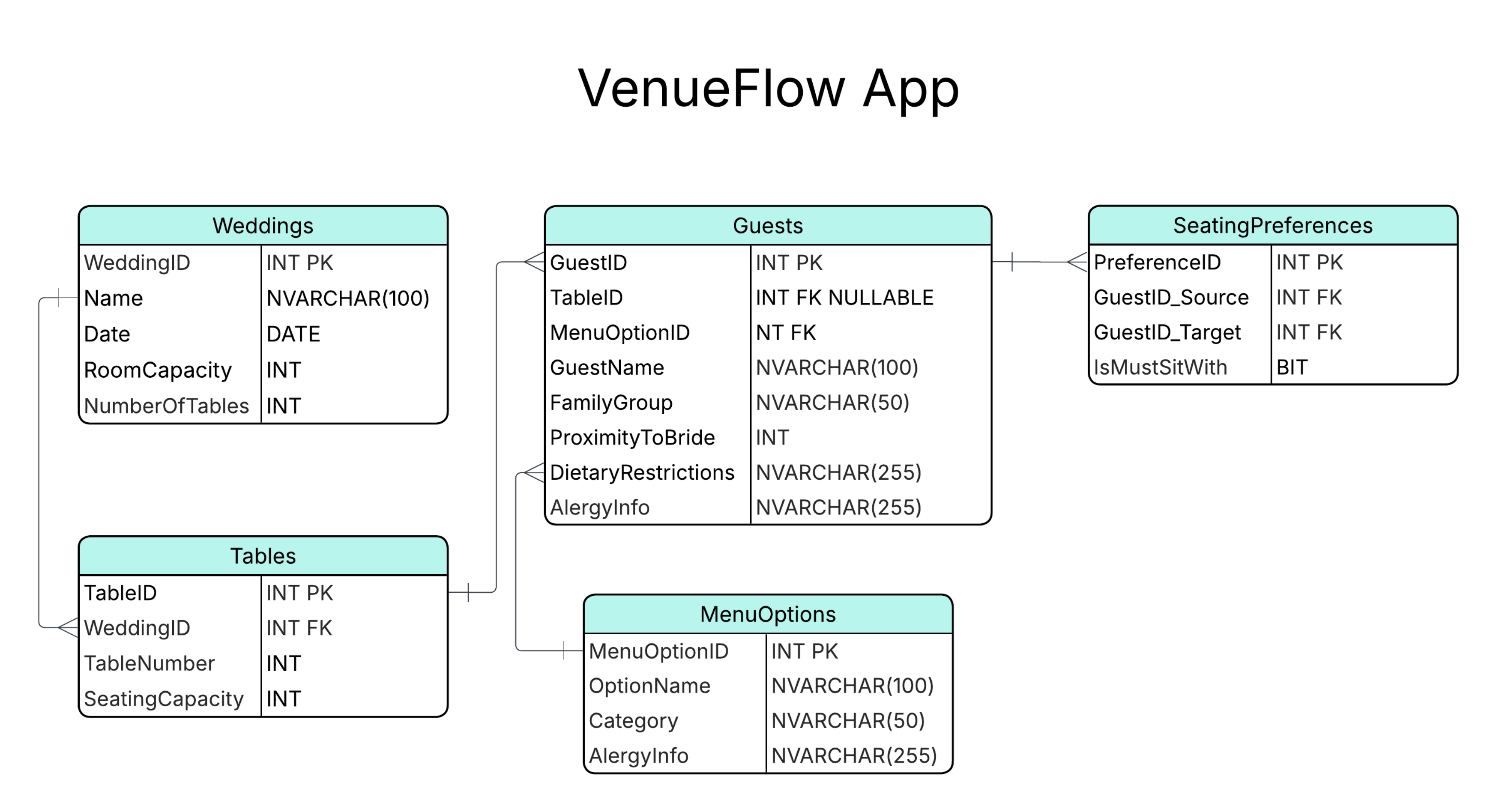
BACKGROUND

- Event venues get requests to host weddings and need to manage multiple rooms and have other challenges like:
    - Guests with food allergies
    - Guests with special dietary preferences
    - Managing distribution of food on the day of the event with multiple menu options

OUR SOLUTION

- A desktop application which can be used to:
  - Assign rooms to weddings on a particular date
  - Guest lists can be imported from an Excel spreadsheet to avoid typing each guests information manually
  - Seating is initially automatically assigned based on proximity to bride
  - Ability to print the seating plan with guests who have allergies highlighted in pink and chosen menu option under their name
- Benefits:
  - Avoids accidentally double booking rooms
  - Automatic seating algorithm saves time on assigning seats
  - Printable floor plan helps waiting staff distributing meals and ensuring safety for people with allergies

OUR SOLUTION



VenueFlow App

Weddings	
WeddingID	INT PK
Name	NVARCHAR(100)
Date	DATE
RoomCapacity	INT
NumberOfTables	INT

Tables	
TableID	INT PK
WeddingID	INT FK
TableNumber	INT
SeatingCapacity	INT

Guests	
GuestID	INT PK
TableID	INT FK NULLABLE
MenuOptionID	NT FK
GuestName	NVARCHAR(100)
FamilyGroup	NVARCHAR(50)
ProximityToBride	INT
DietaryRestrictions	NVARCHAR(255)
AlergyInfo	NVARCHAR(255)

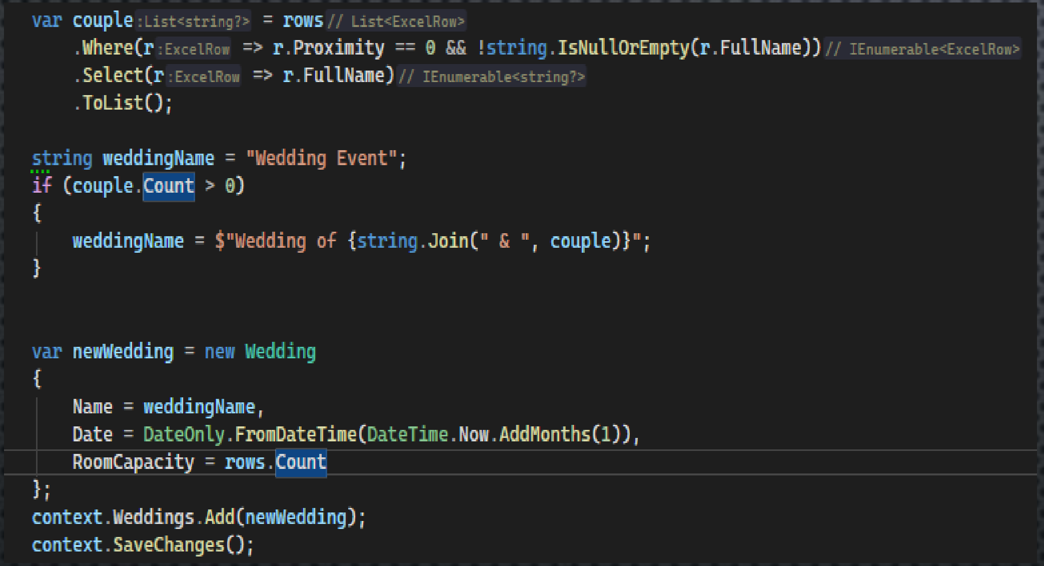
MenuOptions	
MenuOptionID	INT PK
OptionName	NVARCHAR(100)
Category	NVARCHAR(50)
AlergyInfo	NVARCHAR(255)

SeatingPreferences	
PreferenceID	INT PK
GuestID_Source	INT FK
GuestID_Target	INT FK
IsMustSitWith	BIT

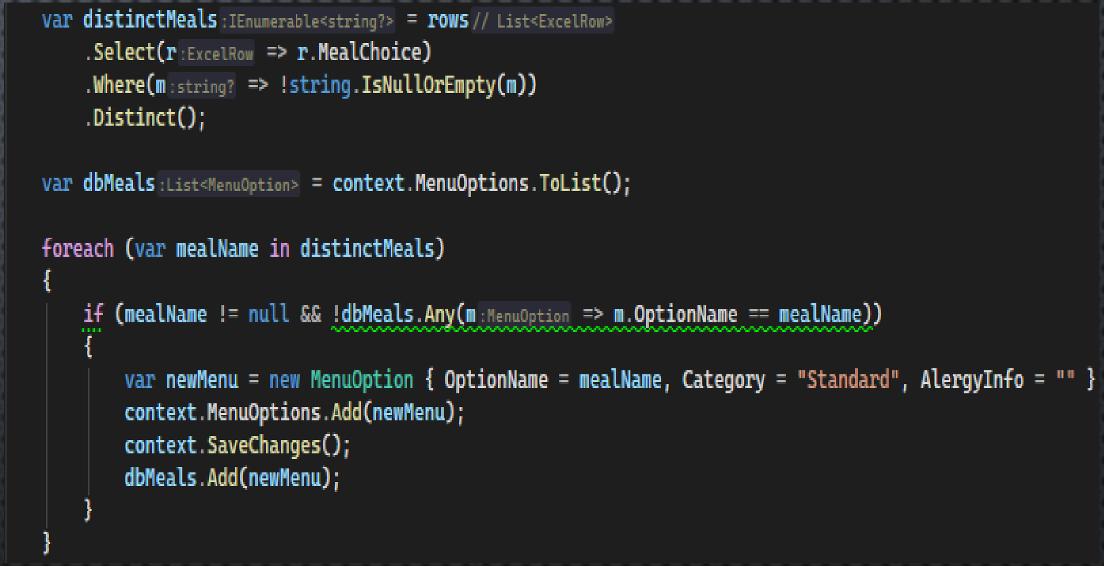
CHALLENGE: DATA IMPORT

- The user need: Wedding planners live in Excel. They don't want to manually type 150 guest names into our app
- The Technical Problem:
  - Excel is "Flat" (just rows and columns)
  - Our Database is "Relational" (Schema with Foreign Keys)
  - The Risk: If we just copy-pasted data, we would have duplicates (e.g., 50 rows saying "Beef") or "Orphaned Records" (Guests with no Wedding ID)

CHALLENGE: DATA IMPORT



```
var couple :List<string?> = rows // List<ExcelRow>
    .Where(r :ExcelRow => r.Proximity == 0 && !string.IsNullOrEmpty(r.FullName)) // IEnumerable<ExcelRow>
    .Select(r :ExcelRow => r.FullName) // IEnumerable<string?>
    .ToList();

string weddingName = "Wedding Event";
if (couple.Count > 0)
{
    weddingName = $"Wedding of {string.Join(" & ", couple)}";
}


var newWedding = new Wedding
{
    Name = weddingName,
    Date = DateOnly.FromDateTime(DateTime.Now.AddMonths(1)),
    RoomCapacity = rows.Count
};
context.Weddings.Add(newWedding);
context.SaveChanges();
```



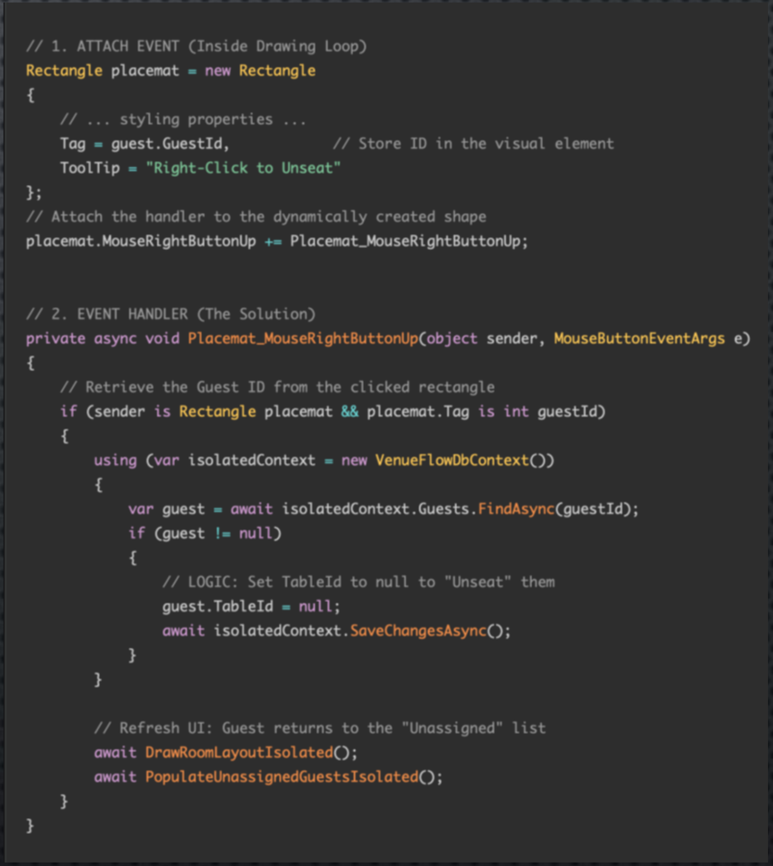
```
var distinctMeals :IEnumerable<string?> = rows // List<ExcelRow>
    .Select(r :ExcelRow => r.MealChoice)
    .Where(m :string? => !string.IsNullOrEmpty(m))
    .Distinct();

var dbMeals :List<MenuOption> = context.MenuOptions.ToList();

foreach (var mealName in distinctMeals)
{
    if (mealName != null && !dbMeals.Any(m :MenuOption => m.OptionName == mealName))
    {
        var newMenu = new MenuOption { OptionName = mealName, Category = "Standard", AlergyInfo = "" }
        context.MenuOptions.Add(newMenu);
        context.SaveChanges();
        dbMeals.Add(newMenu);
    }
}
```

Challenge: Drag and Drop for Seating Assignment

- The seating plan is drawn dynamically on a **Canvas**. Implementing a full 'drag-and-drop' system from one vector shape (a seat) to another is mathematically complex.
- It requires:
  - Attaching drag event handlers to every dynamically generated rectangle.
  - Calculating 'hit testing' in real-time to see if the mouse is hovering over a different table.
  - Managing the state of the 'dragged' visual so it follows the mouse cursor."

SOLUTION: UNSEAT TO REASSIGN

- To solve this within our time constraints without compromising functionality, we simplified the user flow. Instead of dragging directly from seat A to seat B, we implemented an 'Unseat' feature.
    - Logic: Users simply right-click a seated guest to return them to the 'Unassigned' list.
    - Benefit: It turned a complex interaction problem (any seat to any seat) into a simpler state change (Seated → Unassigned → Seated).



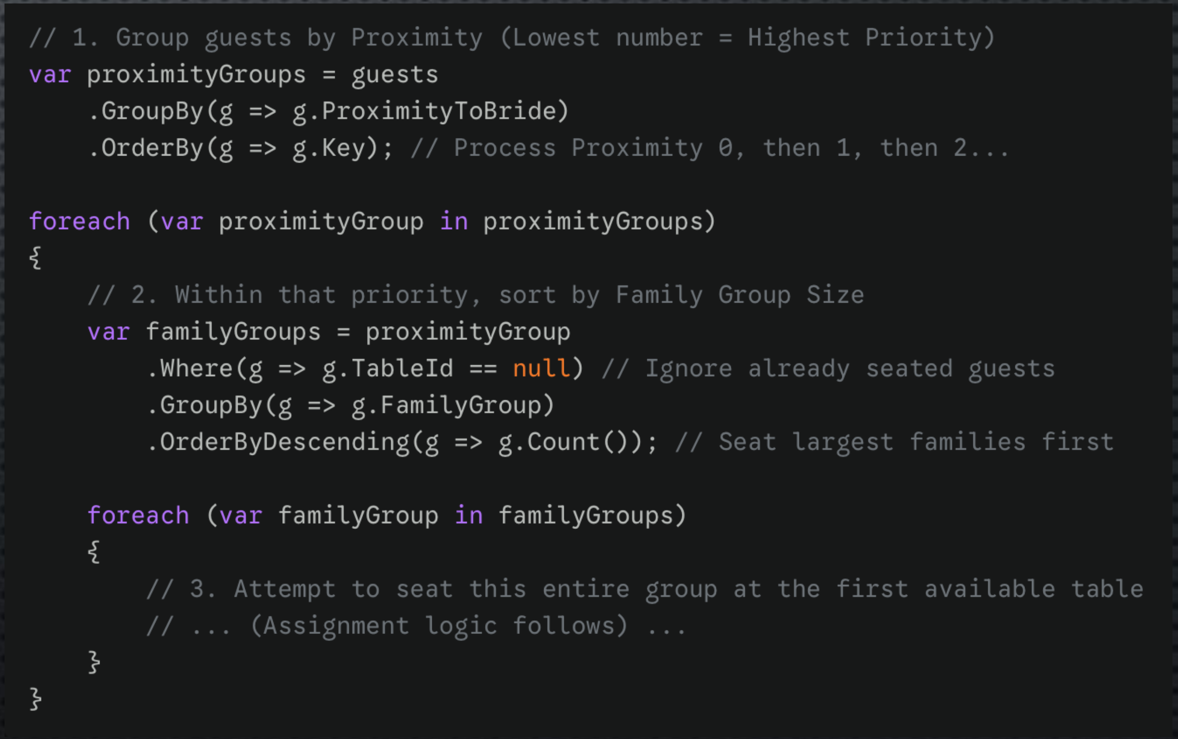
```
// 1. ATTACH EVENT (Inside Drawing Loop)
Rectangle placemat = new Rectangle
{
    // ... styling properties ...
    Tag = guest.GuestId,            // Store ID in the visual element
    ToolTip = "Right-Click to Unseat"
};
// Attach the handler to the dynamically created shape
placemat.MouseRightButtonUp += Placemat_MouseRightButtonUp;


// 2. EVENT HANDLER (The Solution)
private async void Placemat_MouseRightButtonUp(object sender, MouseButtonEventArgs e)
{
    // Retrieve the Guest ID from the clicked rectangle
    if (sender is Rectangle placemat && placemat.Tag is int guestId)
    {
        using (var isolatedContext = new VenueFlowDbContext())
        {
            var guest = await isolatedContext.Guests.FindAsync(guestId);
            if (guest != null)
            {
                // LOGIC: Set TableId to null to "Unseat" them
                guest.TableId = null;
                await isolatedContext.SaveChangesAsync();
            }
        }

        // Refresh UI: Guest returns to the "Unassigned" list
        await DrawRoomLayoutIsolated();
        await PopulateUnassignedGuestsIsolated();
    }
}
```

What we learned: Seating Algorythm

- The paper by Kresten Lindorff-Larsen proposes treating the seating arrangement not just as a logic puzzle, but as a physics optimization problem, similar to finding the lowest energy state in a system of interacting spins.
  - This is very complex and the number of options is N!
- The paper uses simulated annealing (a Monte Carlo method) to randomly swap guests and slowly "cool down" the system to find a "good enough" solution rather than a perfect one.
- Coding this requires:
  - Building adjacency matrices for the room geometry.
  - Quantifying "compatibility" as a numerical vector for every guest pair.
  - Implementing an iterative physics simulation loop that runs thousands of times.

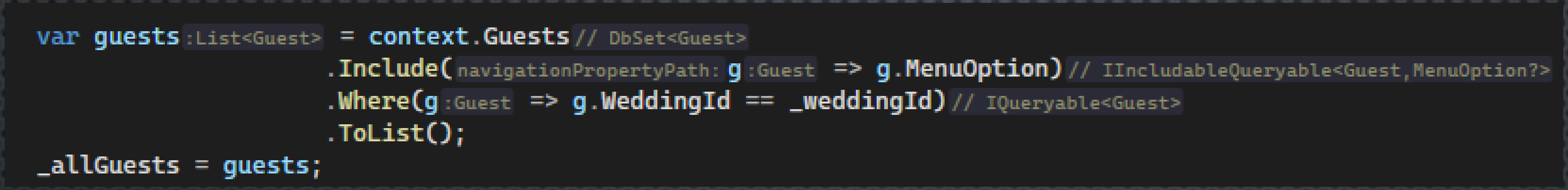
What We Learned: Seating Algorythm

- Our solution: A logical "Greedy Algorithm" powered by LINQ sorting.
- We prioritized specific rules (Family & Proximity) and satisfied them one by one. This is much faster and easier to debug than a probabilistic simulation.
- We used LINQ to sort guests into a strict priority queue.

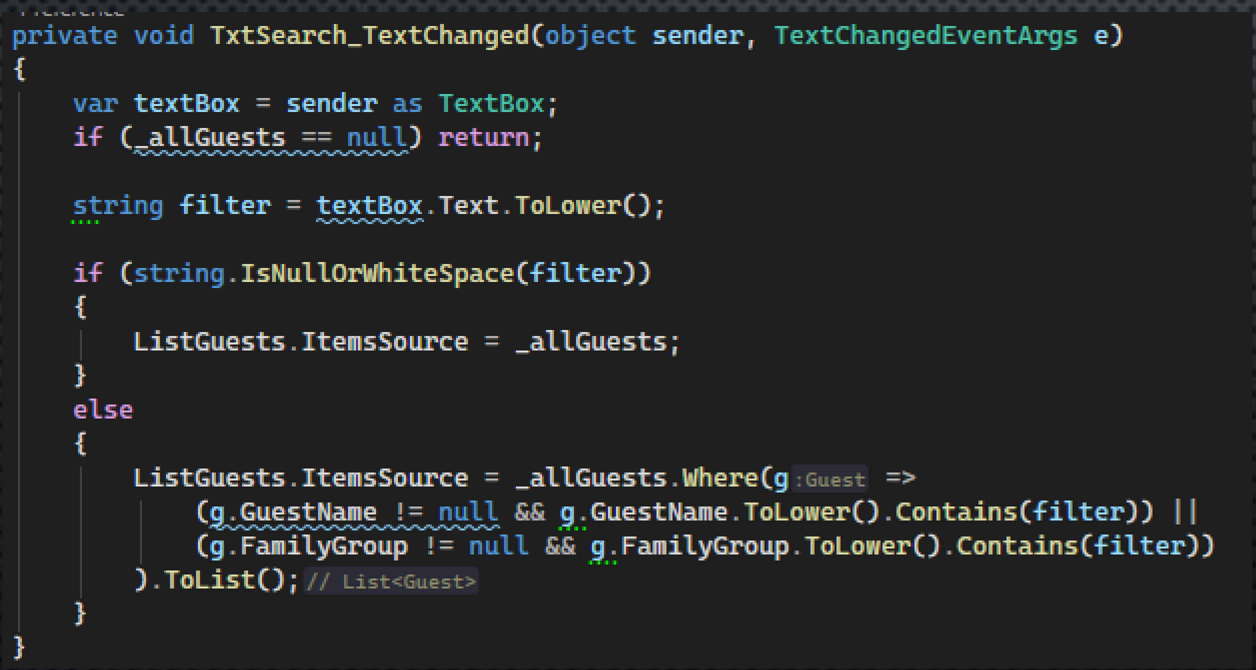


```csharp
// 1. Group guests by Proximity (Lowest number = Highest Priority)
var proximityGroups = guests
    .GroupBy(g => g.ProximityToBride)
    .OrderBy(g => g.Key); // Process Proximity 0, then 1, then 2...

foreach (var proximityGroup in proximityGroups)
{
    // 2. Within that priority, sort by Family Group Size
    var familyGroups = proximityGroup
        .Where(g => g.TableId == null) // Ignore already seated guests
        .GroupBy(g => g.FamilyGroup)
        .OrderByDescending(g => g.Count()); // Seat largest families first

    foreach (var familyGroup in familyGroups)
    {
        // 3. Attempt to seat this entire group at the first available table
        // ... (Assignment logic follows) ...
    }
}
```

WHAT WE LEARNED: SEARCH BAR(REAL-TIME FILTERING)

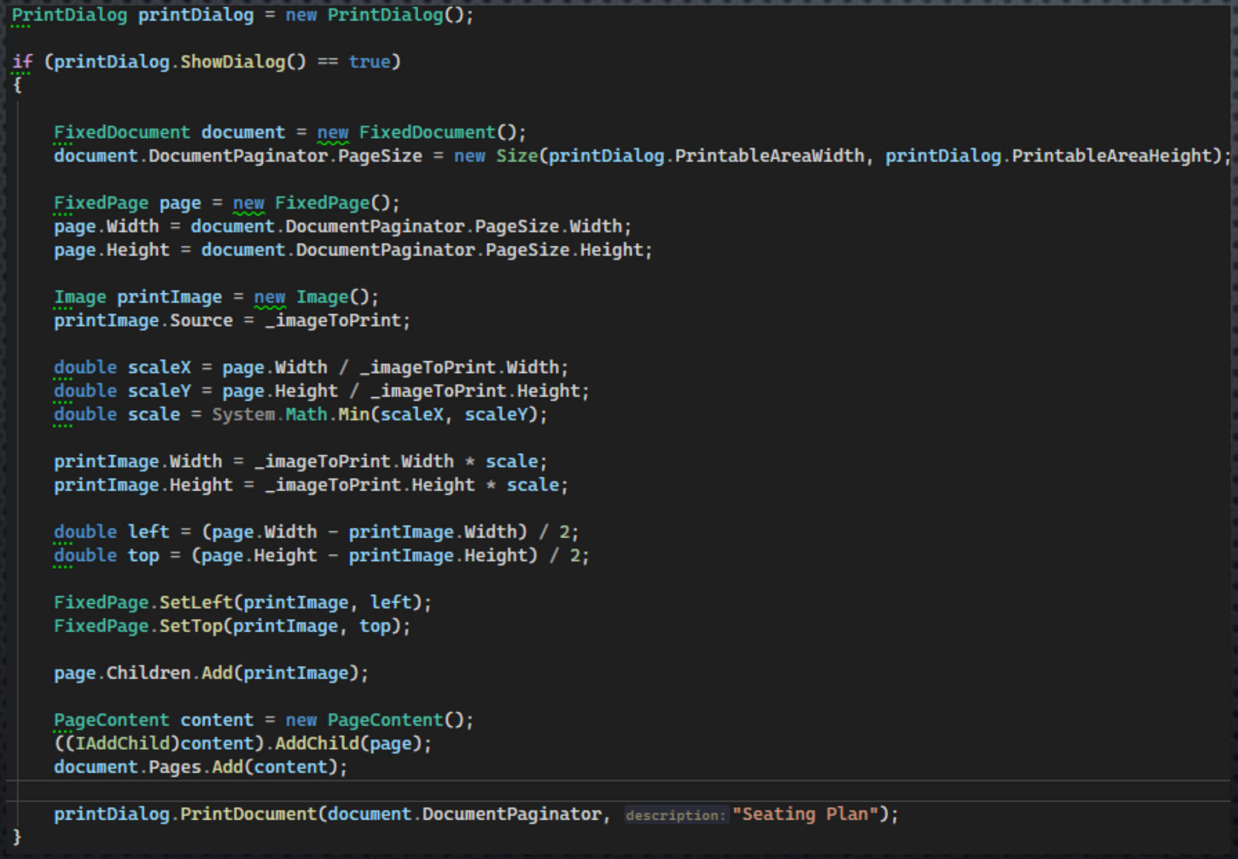


```
var guests:List<Guest> = context.Guests// DbSet<Guest>
                    .Include(navigationPropertyPath: g:Guest => g.MenuOption)// IIncludableQueryable<Guest,MenuOption?>
                    .Where(g:Guest => g.WeddingId == _weddingId)// IQueryable<Guest>
                    .ToList();
_allGuests = guests;
```



```
private void TxtSearch_TextChanged(object sender, TextChangedEventArgs e)
{
    var textBox = sender as TextBox;
    if (_allGuests == null) return;

    string filter = textBox.Text.ToLower();

    if (string.IsNullOrWhiteSpace(filter))
    {
        ListGuests.ItemsSource = _allGuests;
    }
    else
    {
        ListGuests.ItemsSource = _allGuests.Where(g:Guest =>
            (g.GuestName != null && g.GuestName.ToLower().Contains(filter)) ||
            (g.FamilyGroup != null && g.FamilyGroup.ToLower().Contains(filter))
        ).ToList();// List<Guest>
    }
}
```

WHAT WE LEARNED: PRINT TO PDF



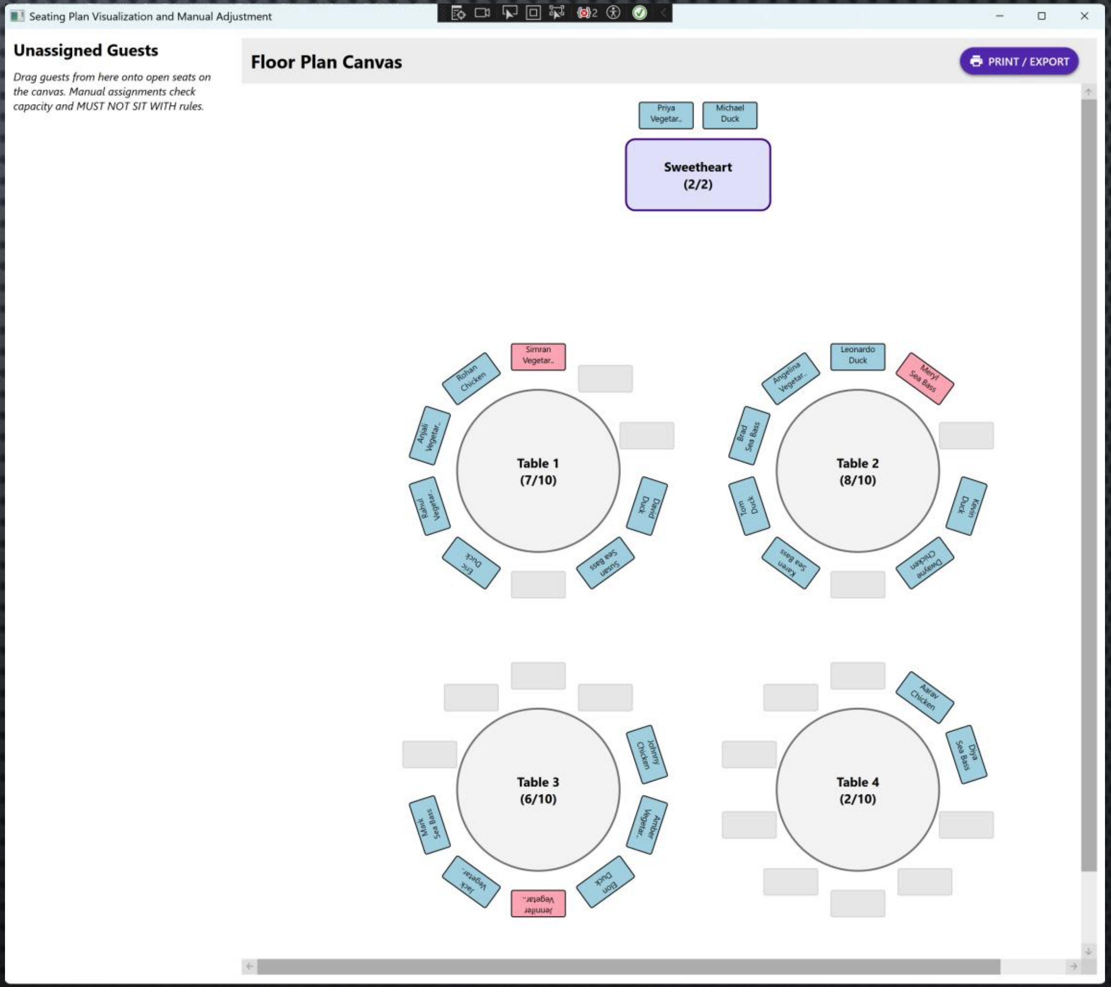
```
PrintDialog printDialog = new PrintDialog();

if (printDialog.ShowDialog() == true)
{
    FixedDocument document = new FixedDocument();
    document.DocumentPaginator.PageSize = new Size(printDialog.PrintableAreaWidth, printDialog.PrintableAreaHeight);

    FixedPage page = new FixedPage();
    page.Width = document.DocumentPaginator.PageSize.Width;
    page.Height = document.DocumentPaginator.PageSize.Height;

    Image printImage = new Image();
    printImage.Source = _imageToPrint;

    double scaleX = page.Width / _imageToPrint.Width;
    double scaleY = page.Height / _imageToPrint.Height;
    double scale = System.Math.Min(scaleX, scaleY);

    printImage.Width = _imageToPrint.Width * scale;
    printImage.Height = _imageToPrint.Height * scale;

    double left = (page.Width - printImage.Width) / 2;
    double top = (page.Height - printImage.Height) / 2;

    FixedPage.SetLeft(printImage, left);
    FixedPage.SetTop(printImage, top);

    page.Children.Add(printImage);

    PageContent content = new PageContent();
    ((IAddChild)content).AddChild(page);
    document.Pages.Add(content);

    printDialog.PrintDocument(document.DocumentPaginator, description: "Seating Plan");
}
```

Future of Work

- Nicer UI
- Ability to manually create weddings
- Ability to change the layout of the tables
- Option to seat guests manually instead of them being automatically assigned

Summary

- Delivered a functional WPF Desktop Application integrated with Azure SQL Database for robust, cloud-based data persistence.
- A priority-based greedy algorithm that successfully automates guest placement based on "Proximity to Bride" and "Family Group" logic.
- Developed a custom graphical engine that draws accurate room layouts, tables, and seats with rotated text for readability.
- Overcame UI challenges to implement a drag-and-drop system combined with a "right-click to unseat" workflow for easy manual adjustments.
- Successfully implemented visual cues (color-coded placemats) to flag allergies, directly aiding event staff.



Seating Plan Visualization and Manual Adjustment

Unassigned Guests

Drag guests from here onto open seats on the canvas. Manual assignments check capacity and MUST NOT SIT WITH rules.

Floor Plan Canvas

Distribution of Work

- Minh
  - Setup and models
  - Main window
  - Wedding view window
  - Import from Excel and export to pdf

- Elizabeth
  - Seating algorythm
  - Seating view window
  - Unit tests