ISTM 415

Semester Project – Jasper Green

Possible Points	300 points (three phases each worth 100 points)
Due Date	Phase I – April 2, 2025
	Phase II – April 16, 2025
	Phase III – April 30, 2025
Due Time	11:55 p.m.

What is the Purpose of This Project?

You will develop a fully functional web application based on the DTC Dental case used in ISTM 320, ISTM 315, and ISTM 410. This project gives you the opportunity to apply the skills you have acquired in HTML, CSS, C# programming, Bootstrap, databases, SQL, MVC, EF Core, and ASP.NET to demonstrate your mastery of these technologies.

Be creative in developing the solution to this project. Your final web application must not only meet the minimum requirements outlined below but also reflect a thoughtful and innovative approach. Ensure that your application is fully functional by thoroughly testing every aspect.

Use the principles from your previous MIS courses to guide the planning and design of your web application. Investing time in proper analysis and design will lead to a more robust and efficient final product during the development phase.

What is the Application?

Jasper Green (JG) is a mowing and lawn care service based in College Station, Texas, owned and operated by Jasper Jones and his four brothers. Jasper launched the business as a teenager, initially mowing lawns in his neighborhood to save money for his first iPhone. His first customer paid him just $5 per lawn!

Using his family's mower and other lawn equipment, Jasper offered three essential services: mowing, edging along landscaping, sidewalks, and driveways, and blowing grass clippings from walkways and driveways. In the fall, he expanded his offerings to include raking and bagging leaves.

Jasper quickly realized he enjoyed working outdoors and took pride in keeping his customers' lawns pristine. Once he reached his savings goal, he decided to grow the business rather than scale back. He eventually brought his younger brothers on board, turning a small neighborhood operation into a thriving enterprise. Today, JG serves approximately 250 regular customers—both residential and commercial—with a team of 21 employees.

With rapid growth, Jasper recognized the need for an information system to help manage customers, track services, and streamline payments. Across ISTM 320, ISTM 315, and ISTM 410, you have analyzed Jasper’s business needs, designed a fully normalized database, and developed an RFP. After reviewing feedback from project teams in these courses, Jasper has decided to scale back some initial ambitions and begin with a web application that meets core business needs while allowing for future expansion.

As part of refining the system’s scope, Jasper made key decisions about how customer and crew information would be managed in the database. He decided not to track customer contracts or itemize individual services. Instead, the contracted service fee is now recorded as an attribute of each property, representing the charge for JG’s basic mowing package (mowing, edging, and blowing clippings). Additionally, Jasper has discontinued offering services beyond mowing.

To better manage operations, the database administrator introduced a new entity, CREW, to track JG’s mowing teams. This entity includes CrewForeman, CrewMember1, and CrewMember2, each serving as a foreign key referencing EmployeeID from the EMPLOYEE table. This change allows the system to efficiently manage crew assignments while maintaining a structured approach to employee data.

The revised Entity-Relationship Diagram (ERD) in Figure 1 reflects these updates, providing the foundation for Jasper Green’s initial web application.

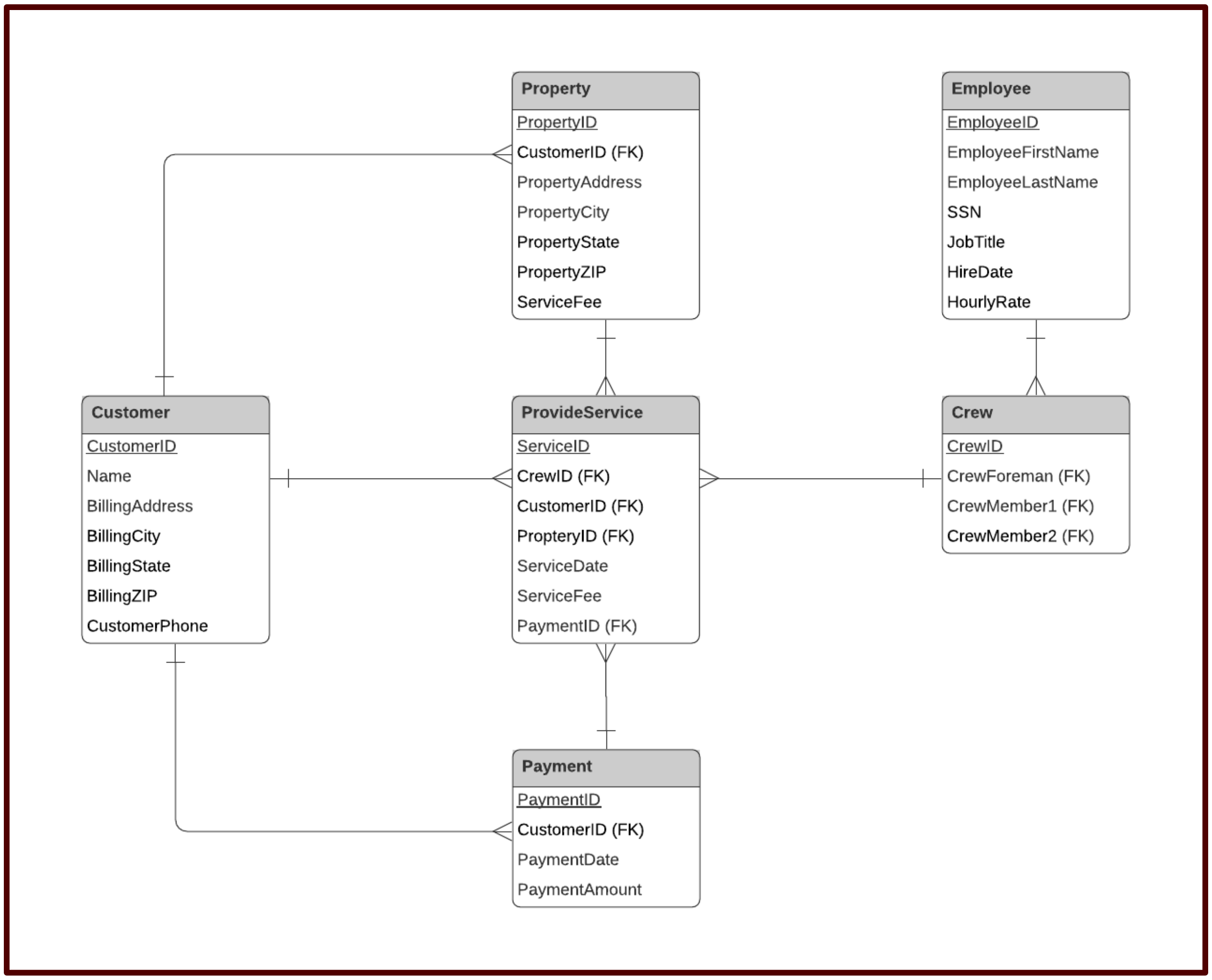

Figure 1 - Revised ERD for Jasper Green

Jasper Green Operations

The following sections outline the key operational aspects of Jasper Green (JG) that Jasper has identified for inclusion in his initial web application system. This system will support essential business functions, including managing customers and properties, tracking lawn care services, and recording customer payments. By streamlining these processes, Jasper aims to improve efficiency, maintain accurate records, and position JG for continued growth.

Managing Customers and Properties

Jasper requires a customer management system to track customers and their properties where Jasper Green provides lawn care services. For each customer, the system should store essential contact details, including name, phone number, and email address. A single customer may own one or multiple properties, each associated with an ongoing lawn care agreement.

When a potential customer reaches out, Jasper visits their property to discuss service options and frequency. During the visit, he evaluates factors such as lot size, number of trees and shrubs, landscaping obstacles, and the length of sidewalks, curbs, and driveways to determine an appropriate service fee.

Once a customer agrees to the services, they sign a contract outlining the property details, service package, frequency, cost, and initial service appointment. (Note: Jasper will continue managing contracts through a paper-based system, and this information will not be stored in the web application.) However, the system should store key service-related details for each property, such as agreed-upon service frequency and pricing, to assist in scheduling and invoicing.

Additionally, Jasper has discontinued customer referral promotions, so the web application does not need to support referral tracking.

Once a customer agreement is in place, Jasper assigns a crew to handle the property's lawn care needs.

Providing Lawn Care Services

Jasper employs 21 Service Professionals, organized into seven crews with three members per crew. Each crew is led by a crew foreman, who is responsible for overseeing daily operations and ensuring quality service.

At the start of each workday, Jasper provides each crew foreman with a list of properties scheduled for service. Depending on the size of the properties, each crew is assigned three to five lawns per day. Whenever possible, Jasper aims to have the same crew service the same properties to maintain consistency and familiarity with customer preferences.

Upon completing a job, the crew foreman handles invoicing based on the customer type:

- For residential customers, the foreman leaves an invoice at the property to confirm that services were provided.
- For commercial customers, no physical invoice is left on-site; instead, Jasper sends a monthly statement summarizing all services rendered.

To enhance business operations, Jasper's new information system must track each service event, capturing customer details, property information, date and time of service, and the crew responsible for completing the job.

After tracking service completion, the system must also integrate with the payment process to ensure accurate customer billing.

Collecting Customer Payments

Customers can pay for their lawn care services in one of three ways: at the time of service, by mailing a payment for each service, or by mailing a monthly payment for the statement balance. Although some customers have requested online payment options, Jasper has not yet implemented this method. However, the system should be designed to accommodate online payments in the future.

Regardless of the payment method, Jasper does not accept partial payments.

If a customer pays at the time of service, the crew supervisor marks the invoice as paid and submits both the payment and a copy of the invoice to the business office. Customers who prefer to pay later can mail individual payments after each service or settle their account with a single monthly payment after receiving their statement.

Each month, Jasper sends an invoice statement to all customers, detailing:

- The dates of service for the previous month
- Any payments received during that period
- The current account balance

Customers are required to pay any outstanding balance within 10 days of receiving their invoice statement.

Jasper’s new information system must capture and store all customer payment transactions, providing a reliable record of payments and outstanding balances.

Project Requirements

Your team is responsible for developing a web application that aligns with Jasper Green’s business operations as outlined above. Having worked with this case for multiple semesters, you should already be familiar with the system’s core requirements.

The project will be completed in three phases, allowing for a structured and iterative development process. Each phase will focus on implementing key system functionalities to ensure a comprehensive and scalable solution.

Phase I – Configure Database and Home Views (100 points)

Phase I focuses on laying the foundation for the Jasper Green web application by completing two key tasks:

1. Configuring the Database – Using Entity Framework (EF) Core, you will define the database structure, establish entity relationships, and implement the "Code First" development approach to create and populate the database.
2. Designing the Layout and Home Views – You will develop a consistent user interface by implementing a layout view, navigation menu, and key home pages, ensuring a structured and professional appearance for the application.

This phase establishes the core database schema and initial user interface, providing the necessary groundwork for future development. By completing these tasks, your team will create a solid technical framework that supports the full functionality of the system in later phases.

Configure Jasper Green Database

In this phase, you will apply your knowledge of Entity Framework (EF) Core to define the database structure and relationships for the Jasper Green web application. You will follow the "Code First" development approach, creating entity classes, a database context class, and configuration files to establish and manage the database schema.

Database Design Considerations

Your development team is responsible for making key database design decisions, including:

- Selecting appropriate data types for each field.
- Defining required fields and enforcing constraints.
- Using integer primary keys configured as identity keys for automatic value generation.

For relationship configurations, follow these best practices:

- Use convention-based and attribute-based approaches whenever possible.

- Use the Fluent API approach only when convention-based and attribute-based configurations are insufficient (see Chapter 12 of the textbook). For example, you will need to apply the Fluent API to define the three one-to-many (1:M) relationships between Employee and Crew to ensure proper relationship mapping.

To populate the database, refer to the Bookstore web application example (Chapter 13) for guidance on creating "seed" classes. These classes will allow you to insert sample data into the database during initialization.

Database Connection Configuration

You should define your connection string in the appsettings.json file under the "ConnectionStrings" section using the following example as a model. Ensure the entire connection string remains on a single line.

```
"Server=(localdb)\\mssqllocaldb;Database=JasperGreenTeam##;Trusted_Con
nection=True;MultipleActiveResultSets=true"
```

- Replace ## with your two-digit team number. For example, if your team number is 03, your database name should be JasperGreenTeam03.
- To find your team number, navigate to "People" in Canvas and select the "Semester Project" tab.

Seeding the Database

Your database should include seed data to facilitate testing and initial functionality. Populate your database with the following records:

- 5 Customer records
- 10 Property records
- 15 Employee records
- 5 Crew records
- 10 ProvideService records
- 5 Payment records

This structured approach ensures that your database is properly configured, relationships are established, and sample data is available for testing before moving to the next phase of development.

You might find the following websites useful for generating data for your database.

- Random email address generator

- Random name generator
- Random phone number generator
- Random address generator

Create the Layout and Home Views

To ensure a consistent look and feel across your web application, you must implement a layout view that defines the overall structure of the site. Use external CSS files to style the layout and maintain a clean, organized codebase.

Styling and Layout Requirements

Your web application should use Bootstrap for responsive design and layout structuring. Additionally, you must apply styling to the following elements:

- Header – Create a logo image for Jasper Green and insert it into the header section of each page.
- Footer – The footer should appear consistently across all pages and include the text: "© Jasper Green - Team {name}" (Replace {name} with your actual team name.) Ensure the footer text is centered and properly styled.
- Navigation – Implement a horizontal navigation menu to facilitate user navigation across the application.

CSS File Management

To maintain clean and manageable styling:

- Place all custom CSS rules in external stylesheets rather than using inline styles.
- Store all CSS files in the `/wwwroot/css/` folder.
- Create multiple CSS files if necessary to organize styles effectively.
- Ensure all pages in the application link to the appropriate CSS files for consistent styling.

Home Views

In Phase 1, you will not develop all views for the web application. As a result, some navigation links may not function until their corresponding views are built. This approach is like the empty shell provided in the SportsPro web application used in the homework assignments.

Your Home Views folder must include the following views:

- Index View – Serves as the main landing page.
- Contact Us View – Displays contact information for Jasper Green.
- About View – Provides background information about the business.

Create a Home Controller and add these three views to your web application. Your team may create appropriate content for each page.

Phase I Deliverable

Before submitting, ensure your project meets the following requirements:

Database Configuration

- EF Core entity classes and relationships established
- Code First approach implemented
- Connection string properly defined in appsettings.json
- Seed data added to the database

Layout and Styling

- Bootstrap used for structuring the layout
- External CSS file(s) applied
- Header, footer, and navigation menu styled

Home Views

- Index, Contact Us, and About views created
- Home Controller added
- Content included on the Index, Contact Us, and About views

Submission Instructions

Compress all files related to your web application into a single .zip file. Submit this .zip file in Canvas as your Phase I semester project deliverable. Ensure that all required components are included before submission.

Phase II – Controllers and Views to Manage Data (100 points)

Phase II builds on the foundation established in Phase I by developing the core functionality needed to manage business data. In this phase, you will implement controllers and views that allow Jasper to efficiently manage customers, properties, employees, and crews within the web application.

Your application must support full CRUD (Create, Read, Update, Delete) operations for these entities, enabling Jasper to:

- View lists of customers, properties, employees, and crews
- Add new records with validated user input
- Edit existing records while maintaining data integrity
- Delete records with confirmation prompts

To streamline development, your application should follow best practices by:

- Using shared Razor view files for adding and editing records
- Implementing dropdown lists for selecting related entities (e.g., assigning customers to properties, employees to crews)
- Validating required fields to ensure data consistency

You can use the SportsPro case study assignments 4-1, 4-2, 4-3, and 4-4 as a guide for implementing these features.

By completing this phase, you will enhance the interactivity and usability of the Jasper Green web application, making it a functional tool for managing business operations.

Manage Customers

To enable Jasper to manage customer records efficiently, your application should provide full CRUD (Create, Read, Update, Delete) functionality for customers. Implement the necessary controller and associated views to support the following operations:

- Display Customer List – When Jasper selects the Customer option from the navigation menu, the application should display a list of all customers. This view must include:
    - An Edit button for each customer
    - A Delete button for each customer
    - An Add Customer button to create a new record

- Add a Customer – Clicking the Add Customer button should open the Add/Edit Customer page with blank fields for data entry.
- Edit a Customer – Clicking the Edit button should open the Add/Edit Customer page, pre-populated with the selected customer's data.
- Delete a Customer – Clicking the Delete button should open the Delete Customer page, prompting Jasper to confirm before removing the record.
- Return to Customer List – After adding, editing, or deleting a customer, the application should return to the customer list page. If Jasper cancels an operation, they should also be redirected to this list.
- Data Validation – Ensure that all required fields are validated when adding or editing a customer to maintain data integrity.
- Reuse Views – Use a single Razor view file for both adding and editing customers to reduce redundancy and maintain consistency.

This functionality will allow Jasper to efficiently manage customer records while ensuring a smooth and user-friendly experience.

Manage Properties

To enable Jasper to efficiently manage property records, your application should implement full CRUD (Create, Read, Update, Delete) functionality for properties. Develop the necessary controller and associated views to support the following operations:

- Display Property List – When Jasper selects the Property option from the navigation menu, the application should display a list of all properties. This view must include:
    - An Edit button for each property
    - A Delete button for each property
    - An Add Property button to create a new record
- Add a Property – Clicking the Add Property button should open the Add/Edit Property page with blank fields for data entry.
- Edit a Property – Clicking the Edit button should open the Add/Edit Property page, pre-populated with the selected property's data.
- Delete a Property – Clicking the Delete button should open the Delete Property page, prompting Jasper to confirm before removing the record.
- Return to Property List – After adding, editing, or deleting a property, the application should return to the property list page. If Jasper cancels an operation, they should also be redirected to this list.

Customer Selection for Properties

- The Add/Edit Property page should include a dropdown list that allows Jasper to assign a customer to the property.
- The dropdown list should display the full name of each customer (implement a method in your Customer model to derive the full name).
- You may use either ViewBag or ViewModel to pass the customer list from the controller to the view.
- When editing a property, ensure that the currently assigned customer is preselected in the dropdown list.

Validation and View Management

- Validate required fields when adding or editing a property to ensure data integrity.
- Use a single Razor view file for both adding and editing properties to maintain consistency and reduce redundancy.

This functionality will provide Jasper with a user-friendly and efficient way to manage property records while ensuring proper customer associations.

Manage Employees

To allow Jasper to efficiently manage employee records, your application should implement full CRUD (Create, Read, Update, Delete) functionality for employees. Develop the necessary controller and associated views to support the following operations:

- Display Employee List – When Jasper selects the Employee option from the navigation menu, the application should display a list of all employees. This view must include:
  - An Edit button for each employee
  - A Delete button for each employee
  - An Add Employee button to create a new record
- Add an Employee – Clicking the Add Employee button should open the Add/Edit Employee page with blank fields for data entry.
- Edit an Employee – Clicking the Edit button should open the Add/Edit Employee page, pre-populated with the selected employee’s data.
- Delete an Employee – Clicking the Delete button should open the Delete Employee page, prompting Jasper to confirm before removing the record.

- Return to Employee List – After adding, editing, or deleting an employee, the application should return to the employee list page. If Jasper cancels an operation, they should also be redirected to this list.

Validation and View Management

  - Validate required fields when adding or editing an employee to ensure data accuracy.
  - Use a single Razor view file for both adding and editing employees to maintain consistency and streamline development.

By implementing these features, your application will provide Jasper with a simple and efficient way to manage employee records, ensuring accurate workforce tracking.

Manage Crews

To enable Jasper to effectively manage crew assignments, your application should implement full CRUD (Create, Read, Update, Delete) functionality for crews. Develop the necessary controller and associated views to support the following operations:

- Display Crew List – When Jasper selects the Crew option from the navigation menu, the application should display a list of all crews. This view must include:
  - An Edit button for each crew
  - A Delete button for each crew
  - An Add Crew button to create a new record
- Add a Crew – Clicking the Add Crew button should open the Add/Edit Crew page with blank fields for data entry.
- Edit a Crew – Clicking the Edit button should open the Add/Edit Crew page, pre-populated with the selected crew's data.
- Delete a Crew – Clicking the Delete button should open the Delete Crew page, prompting Jasper to confirm before removing the record.
- Return to Crew List – After adding, editing, or deleting a crew, the application should return to the crew list page. If Jasper cancels an operation, they should also be redirected to this list.

Defining Multiple 1:M Relationships Between Crew and Employee

When defining the Crew and Employee entities, you must establish three separate one-to-many (1:M) relationships between Crew and Employee: one for the Crew Foreman, one for Crew Member 1, and one for Crew Member 2.

To implement this correctly, follow these steps:

1. Define Navigation Properties in the Crew Model

   Modify the Crew class to include three separate foreign key references to the Employee class, ensuring each crew member is properly linked as shown in the following code.

```
public class Crew
{
    public int CrewID { get; set; }
    public string CrewName { get; set; } = string.Empty;

    public int CrewForemanID { get; set; }
    [ValidateNever]
    [ForeignKey(nameof(CrewForemanID))]
    public Employee CrewForeman { get; set; } = null!;

    public int CrewMember1ID { get; set; }
    [ValidateNever]
    [ForeignKey(nameof(CrewMember1ID))]
    public Employee CrewMember1 { get; set; } = null!;

    public int CrewMember2ID { get; set; }
    [ValidateNever]
    [ForeignKey(nameof(CrewMember2ID))]
    public Employee CrewMember2 { get; set; } = null!;
}
```

2. Add Collections to the Employee Model

Since each Employee can be assigned to multiple crews, modify the Employee model to establish the reverse relationship.

```
public class Employee

{

    public int EmployeeID { get; set; }

    public string FullName { get; set; } = string.Empty;


    // Foreman role

    public ICollection<Crew> Crews { get; set; } = null!;

    // Crew Member 1 role

    public ICollection<Crew> Member1 { get; set; } = null!;

    // Crew Member 2 role

    public ICollection<Crew> Member2 { get; set; } = null!;

}
```

3. Use Fluent API to Define Relationships in JasperGreenContext.cs

   Inside the `OnModelCreating()` method, explicitly define the three 1:M relationships using the Fluent API to avoid issues with cascading deletes.

```
modelBuilder.Entity<Crew>()

    .HasOne(c => c.CrewForeman)

    .WithMany(e => e.Crews)

    .HasForeignKey(cf => cf.CrewForemanID)

    .OnDelete(DeleteBehavior.Restrict);

modelBuilder.Entity<Crew>()

    .HasOne(c => c.CrewMember1)

    .WithMany(e => e.Member1)

    .HasForeignKey(cf => cf.CrewMember1ID)

    .OnDelete(DeleteBehavior.Restrict);
```

```
modelBuilder.Entity<Crew>()
    .HasOne(c => c.CrewMember2)
    .WithMany(e => e.Member2)
    .HasForeignKey(cf => cf.CrewMember2ID)
    .OnDelete(DeleteBehavior.Restrict);
```

Assigning Employees to Crews

- The Add/Edit Crew page should include dropdown lists to allow Jasper to assign employees to crew positions, including:
  - Crew Foreman
  - Crew Member 1
  - Crew Member 2
- The dropdown lists should display the full name of each employee (implement a method in your Employee model to derive the full name).
- You may use either ViewBag or ViewModel to pass the employee list from the controller to the view.
- When editing a crew, ensure that the currently assigned employees are preselected in the dropdown lists.

Validation and View Management

- Validate required fields when adding or editing a crew to maintain data accuracy.
- Use a single Razor view file for both adding and editing crews to ensure consistency and minimize redundancy.

By implementing these features, your application will provide Jasper with a structured and efficient way to manage crew assignments, ensuring proper workforce organization.

Final Testing and Validation

- Before completing this phase, thoroughly test your application to ensure the following:
- Required field validation functions correctly for all add and edit operations.
- Jasper can navigate seamlessly to the Customer, Property, Employee, and Crew management views from the user interface.
- CRUD operations (Create, Read, Update, Delete) work as expected for each entity.

Proper testing ensures that your web application provides a functional and user-friendly experience for managing Jasper Green's business data.

Handling Deletion Errors for Related Records

When deleting a Customer, Property, Service Record, or Payment, you may encounter a DbUpdateException due to foreign key constraints. This happens because these entities have one-to-many (1:M) relationships with other records in the database.

To ensure your application handles deletions properly, follow these best practices:

1. Define Relationships with Navigation Properties

   Ensure that all related entities have correct navigation properties in their model classes. For example,

   In the Customer.cs domain model:

   ```
   // Navigation property for related properties
   public ICollection<Property> Properties { get; set; }
   ```

   In the Property.cs domain model:

   ```
   // Foreign key and navigation property to Customer
   [Required(ErrorMessage = "A customer is required")]
   public int CustomerID { get; set; }
   public Customer Customer { get; set; }
   ```

   Repeat this process for ProvideService and Payment to ensure all relationships are properly linked.

2. Use Fluent API to Restrict Deletions on Related Records

   Explicitly define relationships in JasperGreenContext.cs using Fluent API, setting DeleteBehavior.Restrict for foreign key constraints. For example,

   ```
   // Prevent cascading deletes between Customer and Property
   modelBuilder.Entity<Property>()
       .HasOne(p => p.Customer)
       .WithMany(c => c.Properties)
       .HasForeignKey(p => p.CustomerID)
       .OnDelete(DeleteBehavior.Restrict);
   ```

   Apply similar Fluent API constraints for all foreign key relationships in the system, including:

   - Customer → Property
   - Property → Provide Service
   - Customer → Payment
   - Payment → Provide Service

3. Handle Deletion Errors with Try/Catch in the Delete Action Method

   Modify the Delete() action method to catch errors gracefully when a record cannot be deleted.

```
[HttpPost]
public IActionResult Delete(Customer customer)
{
    var c = Context.Customer.Find(customer.CustomerID);
    try
    {
        Context.Customer.Remove(c);
        Context.SaveChanges();
    }
    catch (DbUpdateException)
    {
        // Provide user-friendly feedback
        TempData["message"] = $"{c.FullName} cannot be deleted
because there are related records in the system.";
    }
    return RedirectToAction("List");
}
```

   Apply similar exception handling for Properties, Provide Service, and Payments to ensure all deletions are handled properly.

4. Display User-Friendly Messages in the UI

Modify _Layout.cshtml to display TempData messages, helping users understand why deletions may fail.

@if (TempData.Keys.Contains("message"))

{

    <h2 class="bg-info text-white text-center p-2 mt-2
rounded">@TempData["message"]</h2>

}

Key Takeaways

- DbUpdateExceptions occur when deleting records with foreign key dependencies.
- All foreign key relationships should be explicitly defined using navigation properties and Fluent API.
- Wrap deletion logic in a try/catch block to prevent system crashes and provide clear feedback to users.
- Display meaningful error messages so users understand why a deletion may fail.

Phase II Deliverable

Before submitting, ensure your project meets the following requirements:

Controllers and Views Implemented for CRUD Operations

- Customers
- Properties
- Employees
- Crews

Navigation and User Interface

- Jasper can access all management views from the navigation menu.
- The application properly redirects to the list views after adding, editing, or deleting records.

Data Entry and Validation

- Required field validation works correctly for all add and edit operations.
- Dropdown lists are correctly implemented for:
- Customer selection when adding/editing properties.
- Employee selection when assigning crew members.

Code Organization and Best Practices

- Single Razor view files used for adding and editing each entity.
- ViewBag or ViewModel approach used appropriately for dropdown lists.

Final Testing

- All CRUD operations (Create, Read, Update, Delete) function as expected.
- User input validation prevents incomplete or incorrect data entry.
- Application runs smoothly without errors.

Submission Instructions

Compress all files related to your web application into a single .zip file. Submit this .zip file in Canvas as your Phase II semester project deliverable. Ensure that all required components are included before submission.

Phase III – Manage Mowing Service, Validation, and View Models (100 points)

Phase III builds upon the work completed in Phase I (database and UI setup) and Phase II (CRUD functionality for core entities) by introducing advanced functionality and data validation. In this phase, you will implement additional business logic, validation rules, and improved data handling to enhance the reliability and usability of the Jasper Green web application.

Your work in this phase will focus on five key areas:

1. Managing Provide Service Events – Implement the ability to view, add, edit, delete, and filter service records while ensuring that service fees meet minimum property-based requirements.
2. Managing Payment Events – Develop a payment tracking system that allows Jasper to log, update, and validate customer payments.
3. Enhancing Crew Management with ViewModels – Modify the Crew views to utilize ViewModels, ensuring efficient data handling between controllers and views.
4. Improving Data Validation – Strengthen input validation for customers and crews, ensuring data integrity for key fields such as ZIP codes, phone numbers, and employee assignments.
5. Project Report – Document your approach, design decisions, and any key assumptions in a concise five-page report.

By completing Phase III, your team will enhance the business logic, user experience, and data integrity of the Jasper Green application, transforming it into a robust and efficient system for managing mowing services, processing payments, and organizing employee assignments.

Manage Provide Service Events

To enable Jasper to efficiently track and manage service events, your application should implement full CRUD (Create, Read, Update, Delete) functionality for provide service events. Develop the necessary controller and associated views to support the following operations:

- Display Service Event List – When Jasper selects the Provide Service Events option from the navigation menu, the application should display a list of all service events. This view must include:
  - An Edit button for each event
  - A Delete button for each event

    - A Payment button for each event to record a payment
    - An Add Provide Service Event button to create a new record
- Add a Service Event – Clicking the Add Provide Service Event button should open the Add/Edit Provide Service Event page with blank fields. The service date field should be pre-filled with the current date.
- Edit a Service Event – Clicking the Edit button should open the Add/Edit Provide Service Event page, pre-populated with the selected event's data.
- Delete a Service Event – Clicking the Delete button should open the Delete Provide Service Event page, prompting Jasper to confirm before removing the record.
- Record a Payment – Clicking the Payment button should open the Add/Edit Payment page with blank fields. The payment date field should be pre-filled with the current date.
- Return to Service Event List – After adding, editing, or deleting a service event, the application should return to the service event list page. If Jasper cancels an operation, they should also be redirected to this list.

Data Validation

- Ensure all required fields are validated when adding or editing a service event.
- Implement custom validation to check that the service fee entered is greater than or equal to the service fee stored with the associated property. This validation should be implemented on the SERVER side only – client-side validation is not required.
- The PaymentID field does not need to be included on the Add/Edit Provide Service Event page.

Filtering Service Events

To enhance usability, add filtering options to the service event list view, allowing Jasper to filter records by Customer, Property, or Crew. Each filter should operate independently, meaning Jasper can choose one filter at a time.

Filter by Customer

- Clicking the "By Customer" button should display a Get Customer view with a dropdown list of all customers and a Select button (similar to the "Get Technician" view in SportsPro).
- After Jasper selects a customer and clicks the Select button, the Provide Service Events list view should reload, displaying only the events for the selected customer.

  - Implement two GetCustomer action methods in the provide service controller (one for HttpGet and one for HttpPost).

Filter by Property

  - Clicking the "By Property" button should display a Get Property view with a dropdown list of all properties and a Select button.
  - After Jasper selects a property and clicks Select, the Provide Service Events list view should reload, displaying only the events for the selected property.
  - Implement two GetProperty action methods in the provide service controller (one for HttpGet and one for HttpPost).

Filter by Crew

  - Clicking the "By Crew" button should display a Get Crew view with a dropdown list of all crews and a Select button.
  - After Jasper selects a crew and clicks Select, the Provide Service Events list view should reload, displaying only the events for the selected crew.
  - Implement two GetCrew action methods in the provide service controller (one for HttpGet and one for HttpPost).

By implementing these features, your application will provide Jasper with an efficient, user-friendly way to manage service records, ensuring accurate tracking and streamlined payment processing.

Manage Payment Records

To enable Jasper to efficiently track and manage customer payments, your application should implement full CRUD (Create, Read, Update, Delete) functionality for payment records. Develop the necessary controller and associated views to support the following operations:

- Display Payment List – When Jasper selects the Record Payment option from the navigation menu, the application should display a list of all recorded payments. This view must include:
  - An Edit button for each payment record
  - A Delete button for each payment record
  - An Add Payment button to create a new payment entry

- Add a Payment – Clicking the Add Payment button should open the Add/Edit Payment page with blank fields. The payment date field should be pre-filled with the current date.
- Edit a Payment – Clicking the Edit button should open the Add/Edit Payment page, pre-populated with the selected payment's data.
- Delete a Payment – Clicking the Delete button should open the Delete Payment page, prompting Jasper to confirm before removing the record.
- Return to Payment List – After adding, editing, or deleting a payment, the application should return to the payment list page. If Jasper cancels an operation, they should also be redirected to this list.

Data Validation

- Ensure all required fields are validated when adding or editing a payment record to prevent incomplete or incorrect entries.
- Use a single Razor view file for both adding and editing payments to maintain consistency and streamline development.

By implementing these features, your application will provide Jasper with a structured and reliable way to manage customer payments, ensuring accurate financial tracking and record-keeping.

Manage Customer Data Validation

To ensure data accuracy and consistency, enhance the validation for the Add/Edit Customer view. Your web application should implement the following data validation checks:

- ZIP Code Validation – Must be either five (5) or nine (9) numeric characters. Use a regular expression attribute to enforce this format.
- Phone Number Validation – Must contain exactly ten (10) numeric characters. Use a regular expression attribute to ensure correct formatting.
- Required Fields Validation – All required fields must contain valid input. Clearly define which fields are mandatory and ensure validation prevents missing entries.
- State Abbreviation Validation – Must be a valid two-character state abbreviation. This can be enforced using:
    - A dropdown list containing predefined state abbreviations.
    - A lookup table that validates state input.

User Feedback & Messaging

- If the data is valid, the system should add or update the customer record. Use TempData to display a confirmation message on the customer list view, informing Jasper that the record was successfully added or updated.
- If the data is invalid, the system should:
    - Display error messages prompting the user to correct invalid fields.
    - Highlight the fields containing errors to improve user clarity.

User Guidance & Form Instructions

- Provide clear on-screen instructions on the Add/Edit Customer form. This should include:
    - Indicating which fields are required.
    - Explaining the correct format for entering ZIP codes and phone numbers.

By implementing these validation rules and usability improvements, your application will ensure data integrity, prevent input errors, and enhance the user experience when managing customer records.

Manage Crew Data Validation

Enhance the validation for the Add/Edit Crew view by implementing a custom validation method that ensures proper crew assignments.

- The validation method should verify that three distinct employees are assigned to each crew by checking that:
    - CrewForemanID, CrewMember1, and CrewMember2 contain unique EmployeeID values.
    - No employee is assigned to multiple roles within the same crew.
- This validation should be implemented on the SERVER side only – client-side validation is not required.

If the validation fails, display an error message prompting Jasper to select different employees for each role. Highlight the invalid fields to guide corrections.

By implementing this validation, your application will prevent duplicate employee assignments, ensuring each crew is properly structured and operational.

Project Report

Prepare a comprehensive project report detailing your team's approach to designing and developing the Jasper Green web application. Your report should:

- Explain how you conducted the project, including key design and implementation decisions.
- Document any assumptions made during the development process.
- Provide insights into challenges faced and how they were addressed.

Formatting & Submission Guidelines

- The report should be concise and well-structured, with a maximum length of five (5) pages (excluding the cover sheet).
- Include a cover sheet listing the names of all team members.
- Use clear headings and sections to organize your content effectively.

This report will serve as a reflection of your design process and decision-making, demonstrating your understanding of the project requirements and implementation.

Phase III Deliverable

Before submitting, ensure your project meets the following requirements:

Application Files

- Add all files related to your web application to a single, compressed .zip file.
- Ensure that all controllers, views, models, and database configurations are included.
- Verify that the application runs correctly and meets all Phase III specifications before submission.

Project Report

- Submit a PDF version of your Project Report in Canvas.
- Ensure the report follows the formatting guidelines, includes a cover sheet with team member names, and does not exceed five (5) pages.

Submission Instructions

- Upload the .zip file containing your web application files to Canvas.
- Upload the .pdf file of your Project Report separately in Canvas.

By following this checklist, you will ensure your submission is complete, well organized, and meets all Phase III requirements.

Other Considerations

The instructions for Phases I, II, and III provide a structured framework for developing the Jasper Green web application but do not specify every detail. This intentional flexibility encourages you to apply creativity and problem-solving in designing your solution.

As you work on the project:

- Make reasonable assumptions where needed to ensure a functional and complete web application.
- Document any assumptions in your Project Report, explaining how they influenced your design and implementation decisions.

This approach encourages critical thinking and innovation, giving your team the opportunity to tailor the application while meeting project requirements.