1. Project Title: Event-Driven Library Management

2. Project Description

This project implements a Library Management System (LMS) using **Event-Driven Architecture (EDA)** in **C# and ASP.NET Core**. The system manages:

- Book borrowing
- Book returning
- Overdue notifications

The main goal is to ensure **loose coupling**, **scalability**, and **asynchronous processing**. The event-driven approach allows the system to react to events like BookCheckedOutEvent, BookReturnedEvent, or OverdueNoticeEvent in real time without tight dependencies.

II. Event-Driven Architecture Overview

1. Core Entities

Entity	Attributes	Description
Book	BookId, Title, Author, IsAvailable	Represents a book in the library.
Borrower	BorrowerId, Name, Email	Represents a library user.
Transaction	TransactionId, BookId, BorrowerId, BorrowDate, DueDate, ReturnDate	Tracks book borrowing and returning transactions.

2. Events Implemented

Event-driven systems rely on events to trigger responses. The LMS uses the following events:

Event Name	Trigger
BookCheckedOutEvent	Fired when a borrower checks out a book.
BookReturnedEvent	Fired when a borrower returns a book.
OverdueNoticeEvent	Fired when a book is overdue for return.

3. Event Flow Diagram

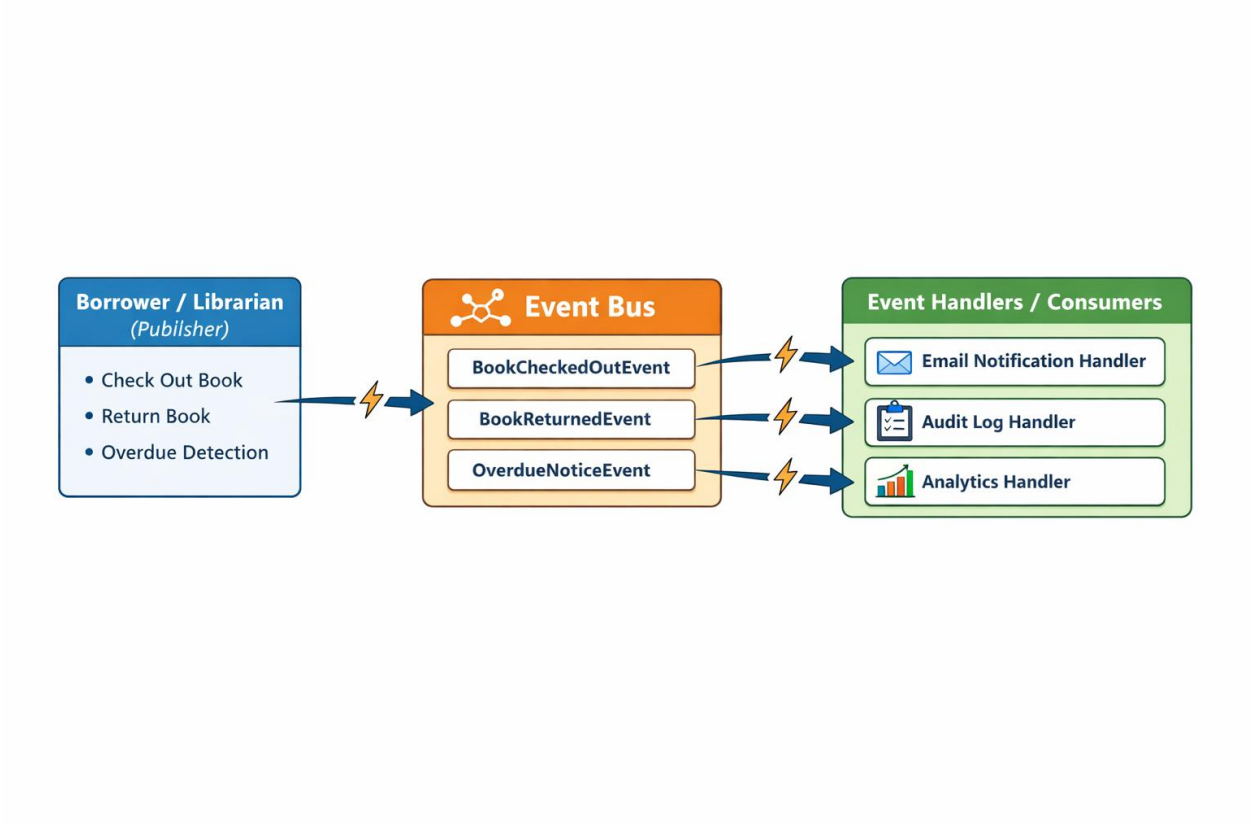
Event Flow Diagram (Text-Based Version)

Borrower Event Bus Event Handlers

Librarian Publish Events Consumers

(Publisher)

Check Out Book BookCheckedOutEvent EmailNotificationHandler | Return Book | BookReturnedEvent | AuditLogHandle | Overdue Detection | OverdueNoticeEvent



III. Step-by-Step Project Implementation

Phase 1: Setup and Planning

Step 1: Group Formation

- A group of **7 students** was formed to work collaboratively.
- Roles assigned: Project Manager, Lead Developer, Frontend Developer, Backend Developer, QA Tester, Documentation Lead, and Git Manager.

Step 2: Project Selection

- Library Management System was selected for its relevance and ability to demonstrate EDA principles.

Step 3: Event Definition

- Defined the core events:
    - BookCheckedOutEvent
    - BookReturnedEvent
    - OverdueNoticeEvent

Step 4: GitLab / GitHub Accounts (Example)

Student Name	GitLab Username	Account Type
Elias Yadeta	elias.yadeta	Personal
Kayof Negaro	kayof.negaro	Personal
Girma Gadisa	girma.gadisa	Personal
Kasahun Getu	kasahun.getu	Personal
Tolesa Taresa	tolesa.taresa	Personal

Each student created and managed their **personal GitLab account** using personal emails. No shared accounts were used.

Step 5: Initial Submission

- Submitted **project title**, **group members**, and **selected events** to the instructor.

Phase 2: Technical Setup & Collaboration

Step 6: Repository Creation

Repository URL:

https://gitlab.com/<your-gitlab-username>/event-driven-library-management

- One member created a **private repository** on GitLab.

Example:

- Repository Owner: Elias Yadeta
- Repository Name: event-driven-library-management
- Visibility: Private

  Repository URL: https://gitlab.com/<your-gitlab-username>/event-driven-library-management

Step 7: Add Team Members (Example)

GitLab Username	Role
Elias.yadeta	maintainer
kayof.nagaro	maintainer
girma.gadisa	maintainer
tolesa.taresa	maintainer
kasahun.getu	maintainer
Yaikob.kuse	Maintainer
ababo.degaga	maintainer

- Maintainer role allows **branch creation**, **commits**, and **merge requests**.

Step 8: Add Instructor

- Instructor added with **Reporter role** for monitoring and evaluation.

Step 9: Local Repository Setup

- All team members cloned the repository to their machines using **Git CLI or GUI**.

Step 10: Initialize ASP.NET Core Project

- Created a **ASP.NET Core MVC project** with folders for Models, Views, Controllers, Services, Events, and Handlers.

Step 11: Branching Strategy

- **Feature branches** created for each module.
- No direct commits to main branch.

Phase 3: Core EDA Implementation

Step 12: Base Event Interface

```
public interface IEvent
{
    Guid Id { get; }
    DateTime OccurredOn { get; }
}
```

Step 13: Event Bus Implementation

```
public interface IEventBus
{
    void Publish<TEvent>(TEvent @event) where TEvent : IEvent;
    void Subscribe<TEvent, THandler>()
        where TEvent : IEvent
        where THandler : IEventHandler<TEvent>;
}
```

Step 14: Event Handler Interface

```
public interface IEventHandler<TEvent> where TEvent : IEvent
{
  Task HandleAsync(TEvent @event);
}
```

- Ensures **all event handlers** implement a standard method.

Step 15: Publisher Services

```
public class LibraryService
{
  private readonly IEventBus _eventBus;
```

```
    public LibraryService(IEventBus eventBus) { _eventBus = eventBus; }
    public void CheckOutBook(Book book, Borrower borrower)
    {
        book.IsAvailable = false;
        var @event = new BookCheckedOutEvent(book.BookId, borrower.BorrowerId);
        _eventBus.Publish(@event);
    }
}
```

- Business operations **publish events** whenever a significant action occurs.

Step 16: Event Consumers

- **EmailNotificationHandler**: Sends emails to borrowers.
- **AuditLogHandler**: Logs events for monitoring and debugging.

```
public class EmailNotificationHandler : IEventHandler<BookCheckedOutEvent>
{
    public Task HandleAsync(BookCheckedOutEvent @event)
    {
        // Send email logic
    }
}
```

Phase 4: Review and Finalization

Step 17: Peer Review

- Merge requests reviewed and approved by at least one team member.
- Ensures **code quality and collaboration**.

Step 18: Documentation

- README.md created with:
    - Setup instructions
    - Architecture overview
    - Event flow diagram

Step 19: Final Submission

GitLab Repository Link:

https://gitlab.com/<your-gitlab-username>/event-driven-library-management

- Submitted final project with:
    - Stable main branch
    - Complete documentation
    - All technical and functional requirements met

      GitLab Repository Link:
      https://gitlab.com/<your-gitlab-username>/event-driven-library-management

IV. Conclusion

The **Event-Driven Library Management System** demonstrates:

- Event-Driven Architecture principles
- Loose coupling between components
- Asynchronous processing of events
- Scalable and maintainable code

It provides a real-world example of how modern library systems can benefit from event-driven design using **C# and ASP.NET Core**.

V. Additional Features for Expansion (Optional)

1. **Overdue Notifications**: Automatically triggers reminders.
2. **Analytics Dashboard**: Tracks popular books, overdue trends.
3. **Role-Based Access Control**: Admin, Librarian, Borrower roles.
4. **Unit Tests**: For event handlers and services.
5. **CI/CD Integration**: Using GitLab pipelines for automated builds.

VI. References

1. Microsoft Docs – ASP.NET Core MVC
2. Martin Fowler – Event-Driven Architecture
3. GitLab Docs – Managing Repositories

1. **User Action / Publisher**
    - The borrower or librarian performs an action:
        - **Check out a book**
        - **Return a book**
        - **Overdue detection**
    - This triggers an **event** in the system.
2. **Event Bus**
    - Central hub that receives events and dispatches them to all registered **event handlers**.
    - Ensures **loose coupling**: Publishers don't need to know who handles the events.
3. **Event Handlers / Consumers**
    - **EmailNotificationHandler** → Sends email notifications to borrowers.
    - **AuditLogHandler** → Logs transactions in the system for monitoring and debugging.
    - **Other possible handlers** → UI updates, analytics, overdue alerts, etc.

Sample Event Flow

Scenario: Borrower checks out a book

1. Borrower clicks "Check Out Book".
2. LibraryService.CheckOutBook() is called.
3. BookCheckedOutEvent is **published** to the Event Bus.
4. Event Bus forwards event to all subscribers:
    - EmailNotificationHandler → sends email to borrower.

- AuditLogHandler → records transaction in audit logs.

Professional Diagram (Visual Description)

1. **Left Side: Publisher**
   - Box labeled **"Borrower / Librarian"**
   - Actions listed inside: Check Out Book, Return Book, Overdue Detection
2. **Center: Event Bus**
   - Box labeled **"Event Bus"**
   - Arrows pointing **from Publisher → Event Bus**
3. **Right Side: Event Handlers**
   - Box labeled **"Event Handlers / Consumers"**
   - Inside: EmailNotificationHandler, AuditLogHandler, AnalyticsHandler
   - Arrows from Event Bus → each handler
4. **Optional:**
   - Add **asynchronous symbol** (like small lightning or dashed lines) to show events are processed asynchronously.