I. Design explanation

This solution is a console application designed following OOP, SOLID and TDD approach:

1. Include 3 following classes which have different responsibilities:

* **Seat Repository**: act like a physical storage of seats and bookings.
  + Maintain in memory the number of rows, seats per row, seats' status, bookings.
  + Calculate available seat count, create draft booking, save booking, retrieve booking, print booking.
* **Booking Service**: act like a booking assistant between user and the seat repository.
  + Maintain movie title.
  + Control the UI flow and input/output for booking creation, booking confirmation, booking check, by working directly with Seat Repository.
* **User Interface**: act like a reception, facing end users/customers.
  + Display main menu and control UI flow for data initiation, booking creation and checking, work directly with Booking Service.

1. Apply dependency injection to manage the dependencies between these classes, for better code maintenance, enhancement and scaling up.
2. Include unit test cases to cover most of the code, which help identify bugs early in the development process and ensuring code quality.

The following things can be done later to enhance the solution:

* Add more unit test cases with mock objects of SeatRepository or BookingService.
* Add more unit test cases for seat selection logic of SeatRepository.
* Logic for default seat selection algorithm may be enhanced further in real life. The following example shows that it tried to pick middle-most seats, but there is one seat far apart from the others. In such cases, we can enhance it to pick the next seat instead.

A screen shot of a computer

AI-generated content may be incorrect.

II. Deployment instruction

As this solution uses .NET 8 Console Application, it can be deployed and run on Windows, Linux and MacOS with .NET 8 runtime installed as production environment.

But for only code compiling, running and evaluation purpose, you can use Visual Studio or Visual Studio Code with the necessary .NET 8 SDK installed, allowing you to run the application directly from the development environment on your local machine.

In this instruction, I recommend you to use a Windows machine, with .NET 8 SDK and runtime installed. When you have these already installed, please follow below steps:

* **Using Visual Studio 2022 (or higher version):**

1. Open GICCinemas.sln in Visual Studio. Then you can read and evaluate the code.

A screenshot of a computer

Description automatically generated

1. On the top menu, click Build > Build Solution. It should download dependencies from Nuget and show successful messages in the Output.

A screenshot of a computer

Description automatically generated

1. On the top menu, click Debug > Start Without Debugging (or press Ctrl+F5). The console application will start and allow you to input data, select menus and display outputs.

A screenshot of a computer

Description automatically generated

1. On the top menu, click Test > Test Explorer. It will show all the written test cases. Then click Run All Tests In View, it will run and show results for each test cases (successful / failed).

A screenshot of a computer

Description automatically generated

* **Using Command Prompt:**

1. Open Command Prompt. Cd to solution folder/GICCinemas
2. Enter "dotnet build". It should restore dependencies and build successfully into \bin\Debug\net8.0\GICCinemas.dll

A screenshot of a computer program

Description automatically generated

1. Enter "dotnet run". It should run the console application.

A screenshot of a computer

Description automatically generated

1. Cd to solution folder/GICCinemas.Test
2. Enter "dotnet test". It should build the test project and execute all test cases successfully.

A screenshot of a computer program

Description automatically generated