CS 752 Software Architecture and Design Practices

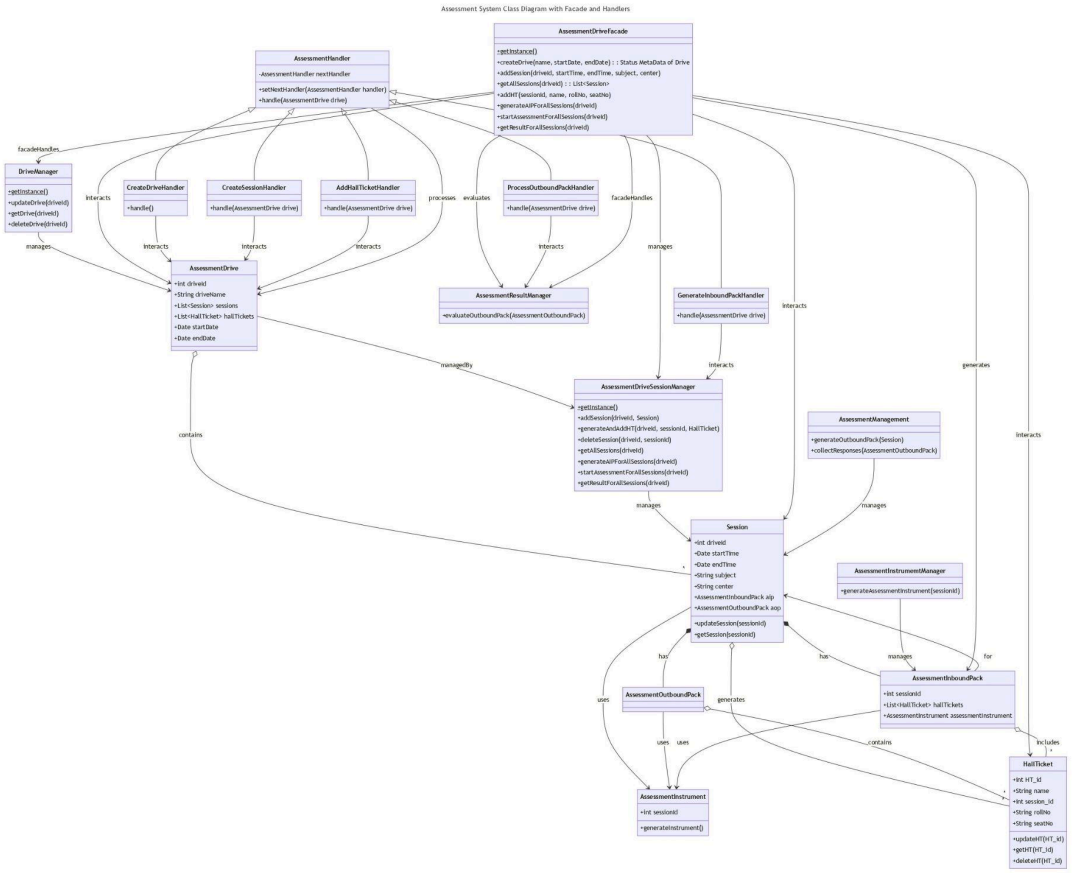
Course Project – PART A

Software Design

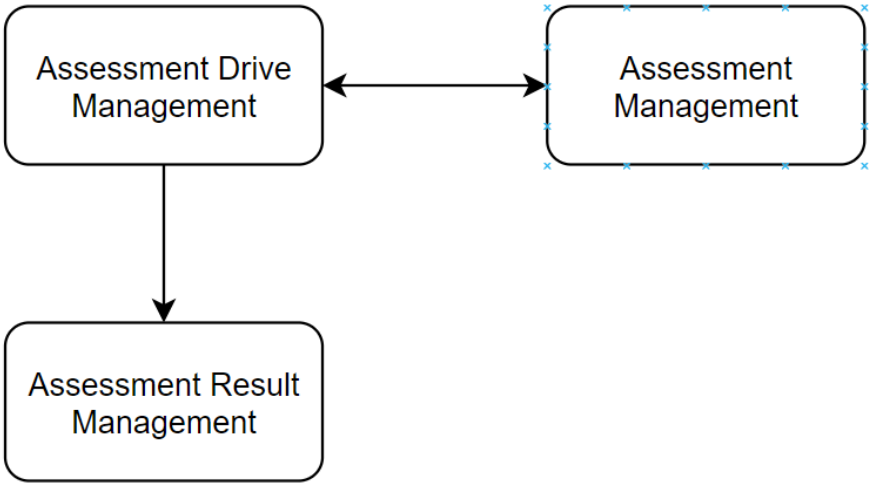
Group Number	6
BRD Module Code	7
BRD Module Name	Assessment Drive Management
Group Members	
Roll Number	**Student Name**
MT2023046	Abhishek Manilal Gupta
MT2023071	Nikhil Nagesh Singh
MT2023091	Abhishek Sharma

Contents:

1. Class Diagram

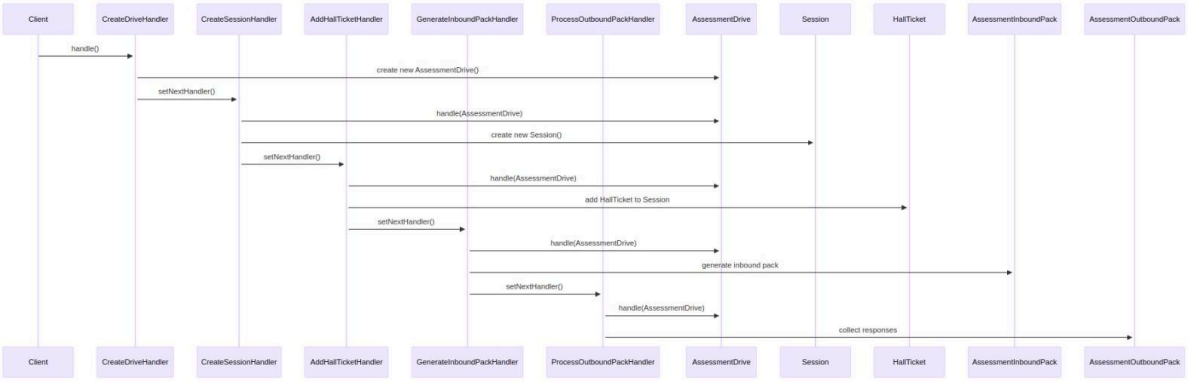


2. Module Dependency

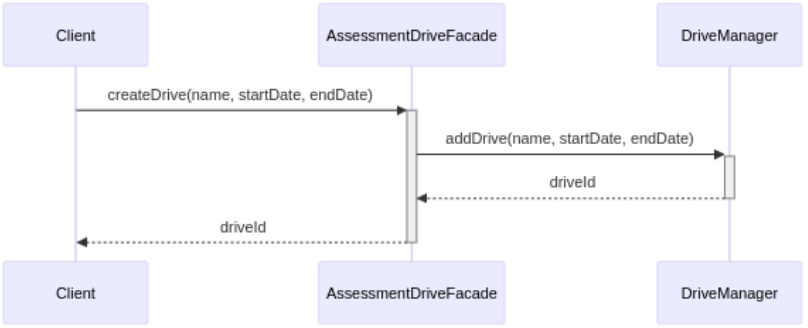


3. Sequence Diagrams:

a. Handler Sequence Diagram

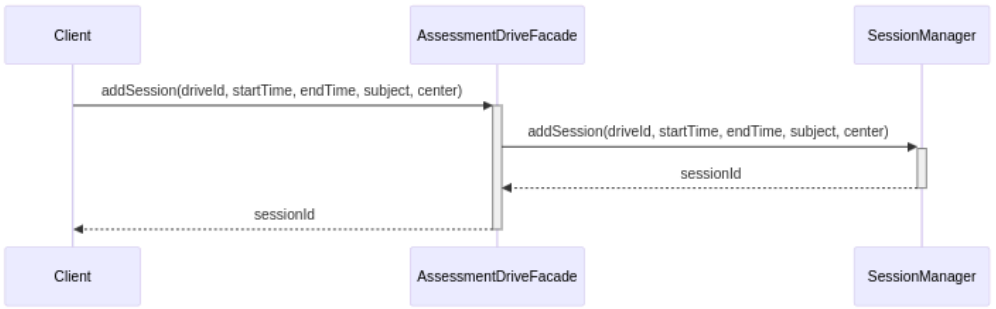


b. Create Drive Sequence Diagram

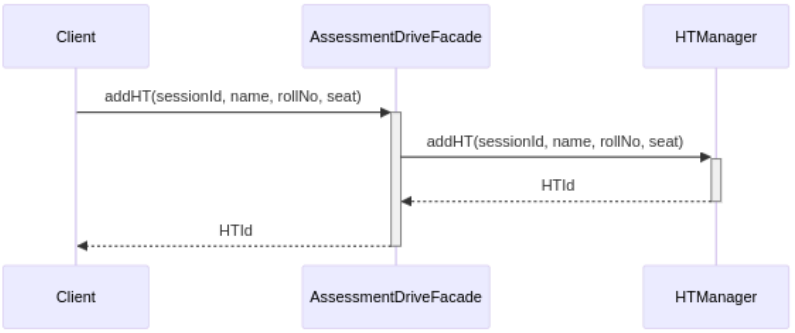


c

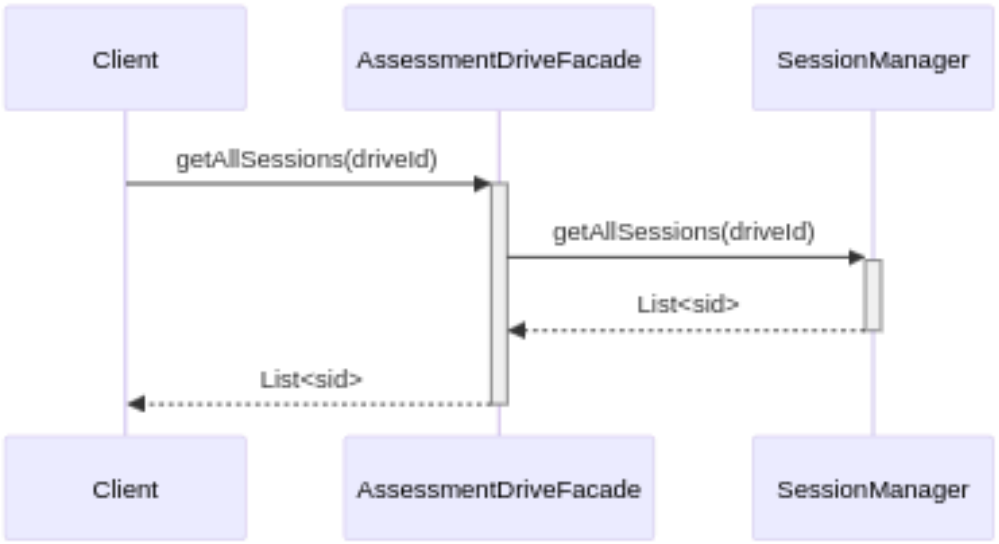
c. . Add Session Sequence Diagram



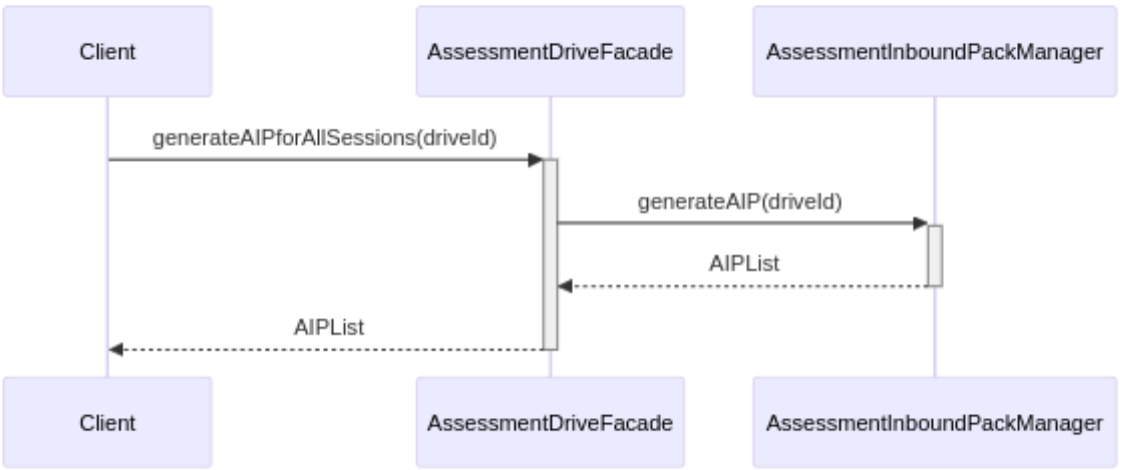
d. Add Hall Ticket Sequence Diagram



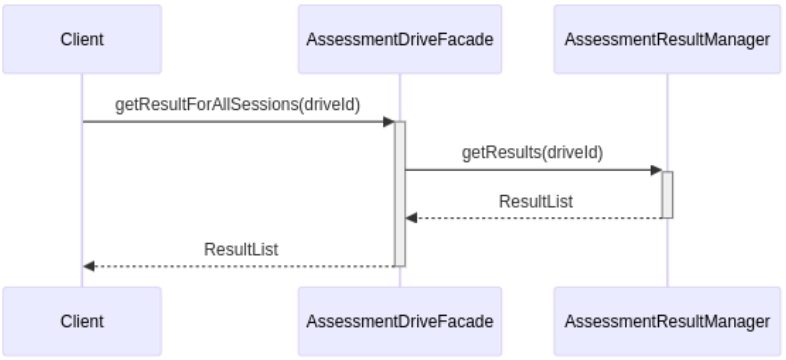
e. Get All Sessions Sequence Diagram



f. Generate Assessment Instrument For Sessions Sequence Diagram



g. Get Results for All Sessions Sequence Diagram



4. Key Design Principles

4.1 Chain of Responsibility:

a. In the **CoR** pattern, requests (like creating a drive, adding sessions, generating hall tickets) are passed along a chain of handlers. Each handler performs a specific task and then forwards the request to the next handler in the chain until the task is fully processed.

b. For example:
   i. **CreateDriveHandler** handles creating a drive.
   ii. **CreateSessionHandler** handles session creation.
   iii. **AddHallTicketHandler** handles hall ticket creation.

c. If you want to add functionality, like generating reports or logging errors, you can easily insert a new handler in the chain without modifying the others.

d. The Major Benefit we got is:
   i. Decoupling and Flexibility
   ii. Conditional Handling
   iii. Separation of Concerns
   iv. Ease of Modification

4.2 Facade:

e. The Facade provides a simplified interface to interact with various subsystems like DriveManager, SessionManager, HallTicketManager, etc. Instead of calling multiple managers or classes, you just call methods on the AssessmentDriveFacade, and it handles the complexity behind the scenes.

f. For example:
   i. createDrive() in the facade might internally call the DriveManager to create a drive and then call the SessionManager to initialise sessions.

4.3 Comparison between CoR and Facade design pattern

Aspect	Chain of Responsibility	Facade
Modularity	High modularity; each handler focuses on a specific task.	Lower modularity; the facade centralises interactions.
Flexibility	Can easily add or remove handlers for different tasks.	Provides a fixed interface, harder to change for custom tasks.
Simplicity	More complex, as it involves a sequence of handlers.	Simplifies the interface by hiding internal complexities.
Customization	Each request can be processed differently based on the handler.	Less customizable; the facade defines the whole workflow.
Separation of Concerns	Each handler has a single responsibility, making it easy to maintain.	Combines multiple tasks in one interface.

4.3 Singleton: Manager classes will consist of business logic and should be only created once as we do not require multiple objects of the same service. Hence, we used the Singleton Design Pattern for this.

4.4 Composite: We created many entities that'll contain the List of other objects (dependent objects). So, here a composite pattern is used.