Design patterns implemented

Adapter Design pattern: This design pattern was used to allow the RequestFacade controller to delegate the work of printing menus and taking user input to the presenters. The adapter design pattern allows the RequestFacade to interact with the MenuPresenter. The RequestFacade class contains a MenuAdapter object which takes in the request controllers in the RequestFacade and turns it into a form usable by the MenuPresenter class.

Memento Design pattern: The event manager use case makes use of the Memento pattern to keep track of which events have been approved by Admin. Since use cases can only access entities and Interfaces of gateways, most design patterns could not make the most out of themselves since they couldn't access the datas without breaking Clean Architecture principles. The memento pattern allows the states of files to be saved and restored when needed. In this application, whenever a new Event is added, the new events are stored in Memento classes. The moment the events get approved by the admin, the events are restored from the memento. Then they're added to the original event manager in the HashMap which stores all the events. Another pattern, that could've been used would've been the state pattern (changing event states from Drafts to Approved). However, implementing a state pattern would void the Clean Architecture principles since the Entities would require a State class to be instantiated which cannot be done. A strategy pattern might also be useful to change states but strategy patterns work better for algorithm than states.

Strategy Design pattern: The strategy design pattern was used in the search functionality to provide flexibility for the algorithm used in the search functions. The SearchByTitle, SearchByUsernameAdmin and

SearchByUsernameRegular are classes that use this pattern. Instead of instantiating individual SimilarityScore classes inside these classes, these classes depend on the ISimilarityScore interface which every SimilarityScore algorithm class implements. This allows us to easily add new algorithms of type ISimilarityScore and use them in the Search classes without modifying the Search classes. The SearchControllers also use this pattern as they do not know which Search classes they are using and they do not depend on any particular Search class and hence new Searchers can be added to the program without having to modify any of the Search Controllers.

Façade: This design pattern was used to tie together all the independent controllers that represent a menu option for users without violating the Single Responsibility Principle. The RequestFacade class uses this design pattern. It takes in a list of RequestControllers and calls their handleRequest methods to perform the task requested by the user. This avoids the problem of having one class that stores all these RequestControllers which would violate Single Responsibility Principle.