**System Design and Decomposition**

**Due on Monday, October 30 before 11:59 pm**

1. **System Decomposition ()**
   * Provide a diagram showing system decomposition into subsystems according to the MVC. (Rae)
   * Provide a clear explanation of the Model-View-Controller architectural style and how coupling and cohesion are addressed in MVC. (Gilad)
   * Provide a clear explanation of each subsystem responsibility (Gilad)
   * remember that answers will be different based on implementation scope (Console vs. GUI) (Gilad)
2. **Provide a detailed class diagram**
   * This class diagram will contain only entity objects (Although you can include GUI classes)
   * This class diagram will be a refined version of the class diagram you provided in the requirement analysis deliverable
   * Show aggregation, composition, associations, multiplicity, inheritance etc. between participating classes.
   * a detailed explanation of the class responsibility.
   * All classes must have their properties and methods identifies and listed.
   * All methods must have their return types.

**Answers:**

MVC is a product design - the structure of the framework - that isolates space/application/business rationale from whatever remains of the UI. It does this by isolating the application into three sections: the model, the view, and the controller.

Coupling in MVC:

Models ought not be coupled to some other get together. Output views are coupled to models, on the grounds that the motivation behind a yield see is to show/beautiful print data installed in a model. No part ought to be coupled to a output view. Controllers are composed to register answers and to control/authorize the best possible utilization of models, so controllers are ordinarily coupled to the models they control. On the off chance that they are coupled to some other part, an input view/form is coupled to the controller that does the calculation asked for by the info occasion. No segment ought to be coupled to an info see.

Cohesion in MVC:

In the original design of the Model-View-Controller design, the Model's part was to speak to true procedures and substances; the View's part was to give the perception of the model; the Controller's part was to fill in as an interface between the end client and the model. The Controller's duties were accomplished by catching equipment motions as console and mouse occasions and making an interpretation of them to operations upon the Model. As operations upon the Model changed its express, the View would be refreshed using the Observer Pattern. The Controller's part in interfacing with the View was essentially introduction particular concerns. As the MVC design was adjusted for use with Web applications, the equipment signals were supplanted by HTTP asks for and the Controller played a more dynamic part in imparting updates to the View because of the stateless idea of the Web. In spite of the fact that the Controller needed to venture up its obligation a bit in this new appearance of the MVC design, its motivation was as yet that of Model interface. While the Controller's essential obligation is to decipher flags, and adjust those signs to significant operations upon the Model, the cohesiveness of the operations upon the Controller ought to be guided by the Single Responsibly Principle inside the setting of its part generalization. In other words, the part of the Controller inside the Model-View-Controller design isn't to reflect the Model's obligation as far as HTTP solicitations and reactions, yet to fill in as a connector whose duties are sorted out in light of the relatedness of the specialized needs to serve in this adjusting part.

Model - The model speaks to information and the guidelines that represent access to and updates of this information. In big business programming, a model frequently fills in as a product estimate of a genuine procedure.

View - The view renders the substance of a model. It determines precisely how the model information ought to be displayed. On the off chance that the model information changes, the view must refresh its introduction as required. This can be accomplished by utilizing a push display, in which the view registers itself with the model for change warnings, or a draw demonstrate, in which the view is in charge of calling the model when it needs to recover the most current information.

Controller - The controller deciphers the client's connections with the view into activities that the model will perform. In a remain solitary GUI client, client cooperation’s could be catch snaps or menu choices, though in a venture web application, they show up as GET and POST HTTP asks. Contingent upon the specific situation, a controller may likewise choose another view - for instance, a site page of results - to display back to the client.