Basing on analysis, Model-View-Controller architecture pattern has been chosen. The system is divided into 4 parts: model, view, controller and file manager.

1. Model (show class diagram of the model)- contains all classes describing the logics of the system. As the class diagram shows, these are 17 classes. The classes are connected either by a simple association or by inheritance. Association has been chosen over composition, because when a change is made on an object in one class, it is expected to change in all of them (give an example). Concerning the use of inheritance we have two super classes: the event class, which is an abstract class and the Participant class. The event class is a parent to 4 classes: Lecture, Seminar, Workshop and Trip. They differ to take an example by the amount of lecturers. As it has been mentioned before, the Event class is an abstract class. The reason of that is, that it shouldn’t be possible to create an event on it’s own, but at the same time, the four mentioned classes share most of the fields and available methods. Another approach was used with the Participant class. It’s subclass is a member. Both a member and a participant can exist separately and the difference between them is that a member is an upgraded participant, with more fields and methods.  
   Furthermore, this part of the system contains 4 singletons, which are: eventList, memberList, lecturerList and VIAManager. Another design pattern used is the Façade, which is the VIAManager class. All of the remaining 16 classes can be reached from VIAManager. This pattern has been chosen due to the choice of MVC architecture pattern.
2. File Manager- strictly connected to the model via VIAManager. It is responsible for secondary storage of information. More about how it is done, will be said by Michał in implementation.
3. View (Show class diagram of the view)- an user interface dedicated part. It was developed using the Swing library for GUI, due to the fact, that every member was familiar with it and it is more beginner friendly comparing to JavaFx. In order to achieve full control of what is displayed, Widow Builder hasn’t been used. The thought behind the layout was simplicity and clarity. The application is divided into decision panels, form panels and lists. In decision panels user decides where to go next. Form panels are for creating events and signing up members and lecturers. List panels have informative function. They display content of the system. Specific instructions about how to use the application are concluded in the user guide.
4. Last but not least is the controller- this is a façade and singleton that connects all parts of the system together. It is illustrated by the sequence diagram (example of looking for lecturers- how is called from gui, showing what happens, displays back in gui).

To make the system organized and clear, the code has been divided into 4 packages, each dedicated to a specific part of the system.