Rails applications are built off the model-view-controller software architecture. The model contains the database information, the schema and the functions, such as all member information, or the ability to search for a certain member's profile. The view is essentially anything the user will see, such as the login screen when trying to login as a member. Finally, there is the controller that takes input from the browser and calls on either model or view accordingly.

The use of the model-view-controller method creates a consistent workflow throughout the project. Our code consists of five models; admin, course, enrollment, parq, and participant. Each model has a set schema (validations for its attributes), and functions to query, or manipulate the data. For example in the participant.rb file there is the search method that would query through all members and find ones that match the criteria in which the user inputted. Scrolling down to the bottom of that file you will find validation statements for the various attributes that are necessary for each "participant". Such as, every participants ID must be different than every other participants ID, or that various attributes, such as a participants name, is required in order to exist as a member.

The view files all correspond to the models, as in, each model has some sort of information needed from the user, and through the user interface (the view) the program is able to gather this information. For example, in any view folder there is a new.html.erb; this file holds the code for the interface a user will interact with to insert information necessary to create this new type of this model.

Finally there is the controller which facilitates the contact between the model, all saved information, and the view, all inserting and exported of data through the users. An example of how the controller is used would be if the administrator asked the program for all members information (such as when wanting the export all information to a csv file). The user sees the link clicks on it, the controller then calls on the model for all the information by calling a function of that particular model (in participant model calling the to_csv function), and displays the file to be downloaded on the view side, which is what the user would see.

The model-view-controller is extremely flexible, and adaptable in the sense that if at any time the admin wants different kinds of information from the members it would be easy to change the database input and also what the user would be asked for. It is also easily adaptable to other organizations who may want a membership system, everything would be essentially the same other than the user interface look, and database attributes.

Refer to the manifesto for a more detailed outline of the design.