Application of GRASP in Project Desperado

Group 4
Alex Sears, Drew Hearing, Judson August, Olga Maneylova, Tyler Blatt

Principle 1: Controller

The purpose of the controller pattern is to assign the responsibility of handling system events to a non-UI class, which is purpose built as required by the use case. We have used separate controllers for nearly all use cases in this project.

Examples:

MeController	Used to retrieve info of the logged-in user
ProjectsController	Used to perform all CRUD actions related to projects
UsersController	Used to perform all CRUD actions for users, as well as administrators
ProjectApplicationsController	Used to create membership applications for projects
ProjectUsersController	Used to add user membership to a project, retrieve users on a project, and remove users from a project

Principle 2: Creator

Creators do just that, create objects. In Project Desperado, we chose to use the controllers as creators because they have access to all of the initializing information for what is being created. Creation of users is handled by the UsersController while project creation is handled by the ProjectsController. The membership that couples users and projects is created by ProjectUsersController when a user is added to a project.

Principle 3: High Cohesion

In Project Desperado we achieve high cohesion by keeping the classes relevant and focused, as exemplified by the controllers. Designing the classes in this way allows for easier maintenance, as well as the ability to make changes without requiring large amounts of refactoring. Also, by keeping the classes small and focused, the overall readability of the code is greatly increased.

Principle 4: Indirection

The indirection principle focuses on low coupling between two objects by introducing a mediating object between them. An example of how Project Desperado achieves this is through the use of controllers. By having the controller mediate the data flow between the view and the model, the reusability of classes is improved, and the addition of any new functionality would not require major overhauls to the current code.

Principle 5: Information Expert

The idea behind the Information Expert principle is that the class with the most access to the information needed to perform a task will become responsible for handling said task. The controllers are (yet again) an excellent example of this. For something like creating a user object, the UsersController seems like the obvious choice, since it has access to the data from the view, it makes sense to use it as the creator of the user object.

Principle 6: Low Coupling

The low coupling design pattern specifies that dependencies between classes should be minimal. In doing so, any change to one class will not result in major changes needing to be made to other classes as well. This aids in code reusability, as well as the overall comprehensibility of the code. Project Desperado embraces low coupling by having very focused classes, and by using controllers to alleviate the dependencies between classes.

Principle 7: Polymorphism

The principle of polymorphism is the idea of allowing the operations of one type to be applied to other types. Examples of polymorphism include function overloading and overriding, subtyping, and the use of generics. In Project Desperado we apply the concepts of polymorphism by having the User, Project, and Membership classes extend ActiveRecord::Base, whose objects don't specify their attributes directly, but rather infer them from the table definition with which they're linked.

Principle 8: Protected Variations

Project Desperado does not make use of any interfaces, and as such does not make use of the Protected Variations design principle.

Principle 9: Pure Fabrication

Project Desperado does not make use of any Pure Fabrication classes.