MVC

The model-view-controller was applied by having a view and a controller associated with each model. In my application the controller is the mediator between the model and the view and it is in charge of keeping the view and the model in sync. For that reason, each controller in my application has a model object and a view object.

The models contain the business logic and structure of the application, for that reason they are interconnected (a list of tasks in a column, a list of columns in a project, etc.).

The views where implemented usin FXML and Scenebuilder and represent only a template for the information contained in the models.

SOLID

Single responsibility

The single responsibility principle was applied throughout the project. For instance, at the project level, all database operations were extracted into a single class in charge only of CRUD operations. Also, at the class level, operations were divided into small manageable methods that are in charge of single operations within the class which make them easily maintainable.

Open/Closed Principle

The open/closed principle was applied by keeping all instance variables private and providing getters and setters for their manipulation, also, by making methods public I ensured that the classes could be extendable.

Liskov's Substitution Principle

Since I could not see a clear usecase for inheritance, the Liskov substitution principle was not implemented. However, I could see how having different kind of tasks in the program could benefit from having a Task superclass and a number of different task subclasses if the program was going to be extended further.

Interface Segregation Principle

The interface segregation principle was implemented by creating interfaces updateable and delectable and identifiable interfaces and implementing them in the models for their database interactions, allowing the classes to only implement those interfaces necessary to them. This also became useful when deleting from the database as those clases which imkplemented the Identifiable interface could be treated the same, since only the id is needed when deleting a record from a table.

Dependency Inversion Principle

The dependency inversion principle was applied by extracting the interfaces from the models and using the interfaces types when possible. That way the applications becomes free of implementation details and it is easy to test other model implementations.

Design Patterns

Repository Design Pattern

The repository design pattern was used to group all database operations into one centralised class in charge of interacting with the SQLite database. This class and methods will then be used by the models when performing their CRUD operations.

I decided to implement this design pattern here because it allowed me to develop my application independent from my choice of database and if in the future I decided to migrate from SQLite to another database provider I would only need to update the code in the DBManager class without having to change anything in the models.