Design Patterns Used

MVVM

We used the MVVM design pattern in our presentation layer.

We used bindings to link between our views and their corresponding viewmodels, keeping the actual logic of the presentation in the viewmodels. The viewmodels manipulate the model objects, which in turn triggers functions in the BackendController, which links to the Backend. We generally tried to keep a one to one relation between views and viewmodels.

We noticed the advantage of MVVM when we needed to make a big change in the SpecificBoardView, and it was seamless to change the UI without worrying about behaviour changing (since it is contained within the viewmodel).

Data Access Object

We used the Data Access Object design pattern in our data access layer.

By making an abstract class which handles retrieval of data for a general abstract data transfer object, we could build up from it to handle each of the objects we needed (user, board, column, task) and use their corresponding DAOs to interface with the database, without worrying about the implementation details when in the business layer.

Didn’t make the cut

Builder

We considered using the Builder design pattern for construction of Board objects, when adding new ones through the business layer, and when loading from data. This could help encapsulate handling of an unfinished board (while its columns and tasks are being loaded). We decided against implementing it due to it bringing too much complexity to the code with the additional classes it would require; the problem of handling unfinished boards is not actually dangerous at this current state so we decided it is better to be less complex. If the project were to continue expanding and boards would become more complex, however, the builder pattern may be a good candidate.