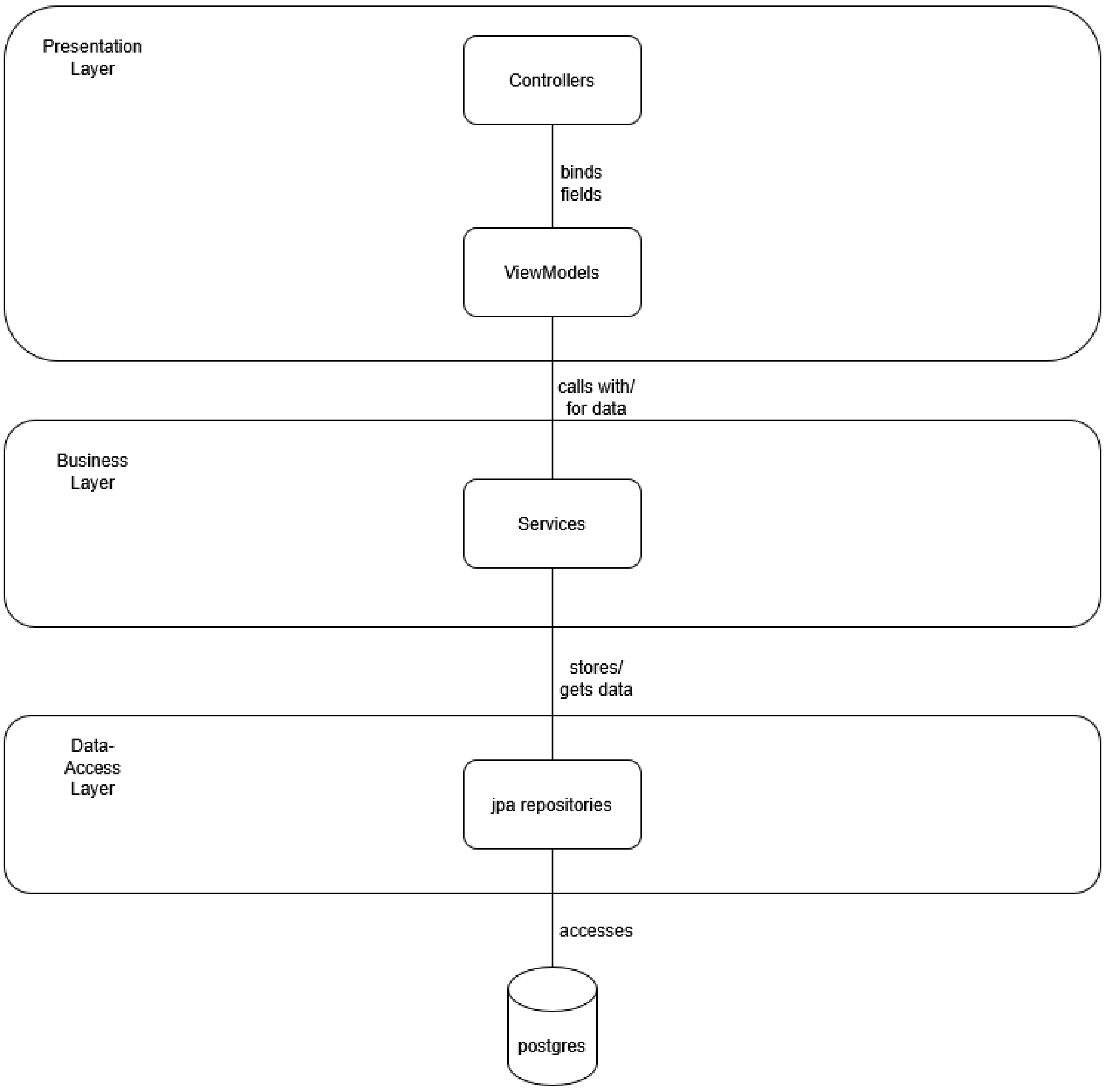
Meeting Planner

Layered Architecture



Presentation Layer

There are 6 Controllers + ViewModels in the Presentation Layer, including the MainViewController.java Class which instantiates the AddMeetingController immediately, the MeetingCardViewControllers immediately and fills them with the Meeting-data of all the meetings and the MeetingDetailsViewController after a Button-Call. The MeetingDetailsController then instantiates the addNoteController immediately, the NoteCardViewController immediately and fills them with the Notes. Both the NoteCardViewController and the MeetingCardViewController are annotated with @Scope(“prototype”) to ensure they get instatiated as a new object for every Meeting/Note. The

Editing fields are in the views themselves, but are bound to a visibleProperty in the ViewModels that gets toggled after the editButton Call.

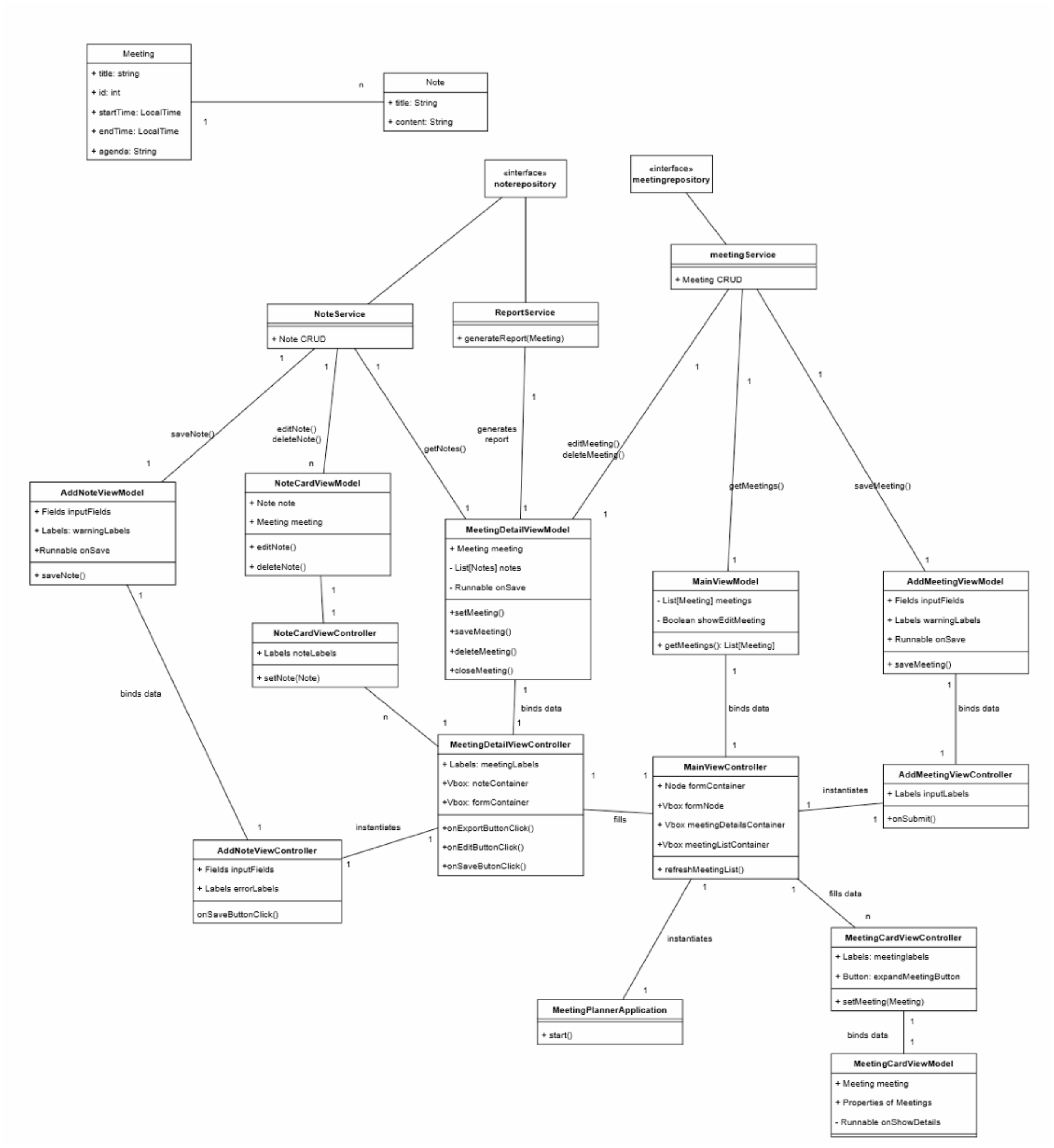
Business Layer

The Business Layer consists of a MeetingService, a NoteService and a ReportService. The MeetingService and NoteService are responsible for CRUD of their respective models and the ReportService for the generating of the report. The ReportService takes a Meeting for the report as input and gets the respective notes from the NoteService. It probably would be cleaner to get the meeting/notes from the repos directly or the notes as input as well but getting the meeting from the repository directly would create duplicated code, and time constraints didn't allow for the notes to be passed as a parameter.

Data Access Layer

The data access layer consists of two jpa repositories, noterepository and meetingrepository. They handle the models meeting and note, which are created after the template in the specification.

Class Diagram



Unit Testing Decisions

I decided to test the business layer for the crud functionality of meetings and notes. Secondly I tested the UI for creating notes and meetings with the junit5 Framework. I integrated the ui tests last due to the giant disadvantage of the framework being very time-consuming when running the tests. Every ui unit test takes at least 5-10 seconds when the junit5 robot enters the input into the ui. The reason I decided to focus on the crud-testing was because these code-sections are the critical point of data transfer between my ui-Layer and my dal, and are the fundamental features of the application that everything else get built around.

Design Pattern

I integrated a Presentation-Based architecture although because of my usage of ViewModels and Controllers its probably closer to a MVVM design pattern, the difference/definition of a presentation based design architecture is not cleary formulated anywhere. Anyway this design approach allowed for a cleaner differentiation of data manipulation and data presentation. For example, my controllers call the sub-controllers/sub-views since these are ui aspects but my viewmodels are responsible for passing data/ filling the data of these sub controllers or at least for providing the data the subcontrollers get filled with. Also these subcontrollers do not store this passed data but pass them on again to their respective viewmodels. A big design-decision that I have made was to have some of the controller-viewmodel combos store the meetings as new objects and not just have one list of Meetings in my main viewmodel and only binding the data of the contained meetings in that list to the sub-viewmodels. On the one hand this made it more difficult to keep all of the data up to date when its manipulated elsewhere and made the code a little less clean but on the other hand it allowed for a simpler manipulation of data itself since I didn’t have to worry about access-rights that saved a lot of time which is why I went for this approach.

Git-Link

https://github.com/acelaender/meetingPlanner