Software Entwicklung 2
Dokumentation

Wintersemester 2021/22

MI7

Moodtracker

Jannika Seybold, js443@hdm-stuttgart.de
Alessa Wiesner, aw184@hdm-stuttgart.de
Peter Gutjahr, pg055@hdm-stuttgart.de

https://gitlab.mi.hdm-stuttgart.de/js443/habittracker/

Inhaltsverzeichnis

1. Kurzbeschreibung

Unser Moodtracker ist ein Programm, mit dem man die Stimmungen und Gefühle in einer Datenbank speichern kann, abfragen und verändern kann. So kann man einen Überblick über den Verlauf der eigenen Stimmung bekommen. Diese Daten können später mit anderen Faktoren korreliert werden, zum Beispiel Periode, Gewohnheiten. Diese sind jedoch noch nicht implementiert. Ursprünglich wollten wir einen Mood- und Habittracker machen, haben es aber auch einen Moodtracker reduziert.

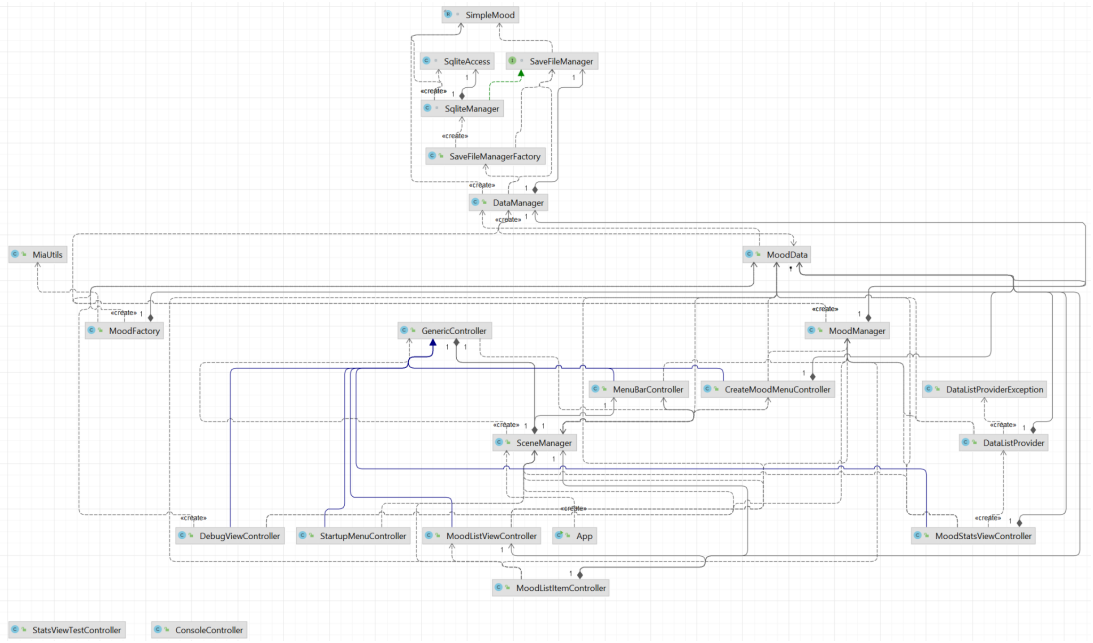
2. Startklasse

“master” Branch: moodtracker/src/main/java/org/jap/view/App.java

3. Besonderheiten

- Die MoodStatsView, eine Ansicht der Daten in einem Liniendiagramm ist noch nicht fertig implementiert. Von dort wird DataListProvider als Callable aufgerufen, es funktionieren bisher jedoch nur die Tests für DataListProvider.
- Wir verwenden eine Java Record Klasse namens “SimpleMood” innerhalb des “datahandler” package. Diese Klasse ist dafür da, die Daten, die eines MoodData Objekts, in einer Form zu speichern, mit der die Datenmanager umgehen können.
- Unser Standard DataManager ist der SQLiteManager, mit dem wir die persistenten Nutzerdaten, die Moods, in einer SQLite Datenbank speichern.
- Wir nutzen Java 17, da es die aktuelle LTS Version ist

4. UML Diagramme



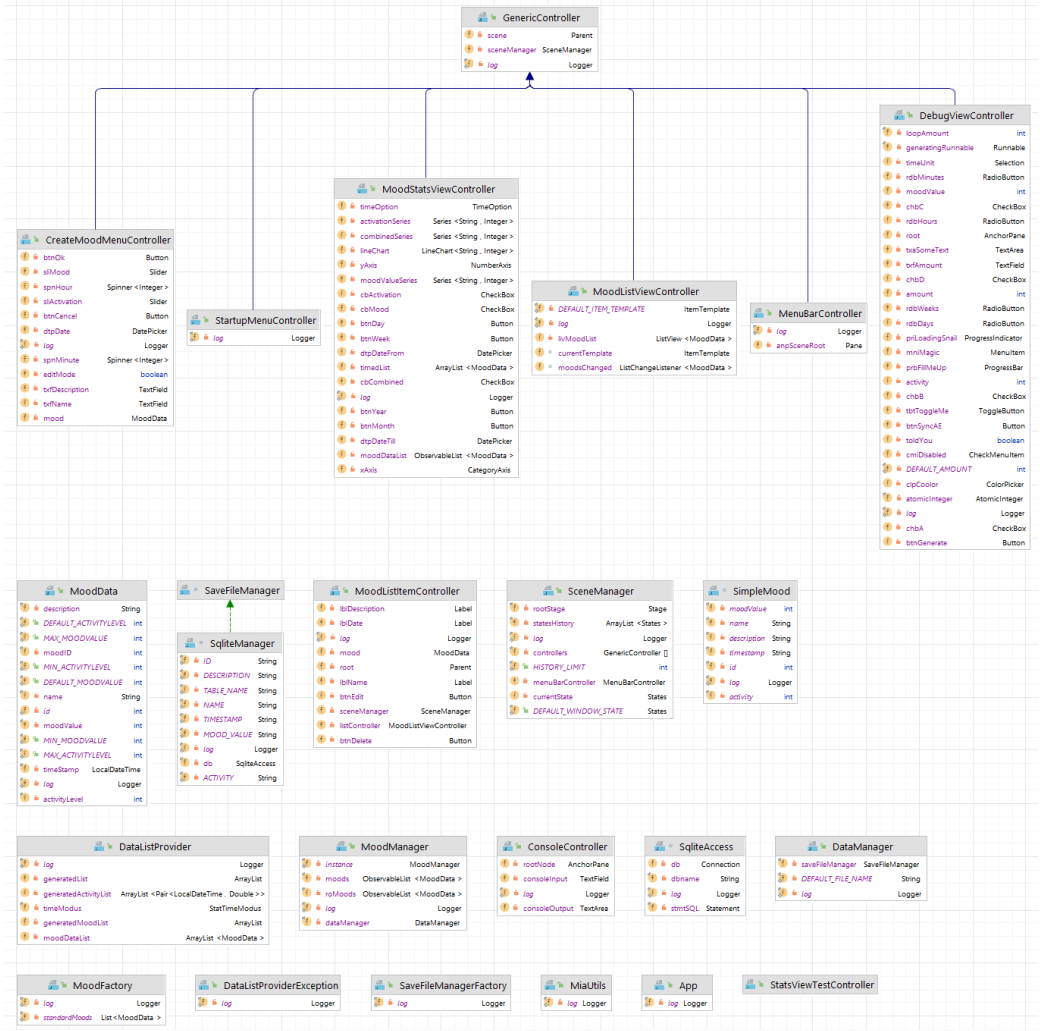


Diagramme sind auch im Repository!

5. Stellungnahmen

5.1 Architektur

Wir haben uns für den MVC entschieden, wobei im DataHandler zusätzlich das Layer-Modell eingesetzt wird, um die Austauschbarkeit der Datenbank zu erleichtern. Wir haben uns außerdem darauf geeinigt, alle Klasse möglichst atomar zu halten, um eine Erweiterung und Wiederverwendung zu erleichtern.

5.2 Clean Code

Wir verwenden keinen public-Zugriff in unseren Kernklassen und auch möglichst wenig in anderen Klassen.
Die getter-Methoden geben immer Kopien der Objekte zurück.
Statische Methoden waren nötig in unserer Singleton-Class MoodManager.

5.3 Tests

Wir haben Tests für unsere Model-Klassen MoodManager und DataManager. Sowie einen Test für DataListProvider, der überprüft, dass das Ergebnis der Liste der Form entspricht, wie es nachher sein soll und die Werte korrekt zusammengeführt wurden.

5.4 GUI (JavaFX)

Für das GUI haben wir mit JavaFX gearbeitet und unsere fxml-Files mit dem Scenebuilder erzeugt. Außerdem haben wir ein css-fiel für die Farben, in dem die Farben in Variablen gespeichert sind. So soll ein späteres Customising vereinfacht werden.
Die Files sind im Ordner resources.

5.5 Logging/Exceptions

Wir loggen mit log4j und in allen Klassen.

Als Custom Exception haben wir eine DataListProvider Exception, wenn die generierte Liste null ist.

5.6 UML

Wir haben zu Anfang ein Usecase-Szenario gemacht, dass wir aber nicht mehr angepasst haben. Es zeigt daher noch das Usecase-Szenario von dem Habittracker, den wir anfangs geplant hatten. Ansonsten haben wir über das Semester hinweg mit draw.io gearbeitet, um dort die Änderungen in den Klassen festzuhalten. Für die Abgabe haben wir uns aber entschieden, das UML aus IntellJ zu exportieren.

5.7 Threads

Threads haben wir zwei: einen Debug-Thread, der vollständig implementiert ist und eine DataListProvider-Klasse, deren Aufruf noch nicht fertig implementiert ist.
Debug: generiert so viele MoodDaten, wie gefordert, um die Funktionalität zu testen.
DataListProvider: Callable returned eine zusammengefasste Liste mit den Moodwerten. Die DataListProvider-Klasse soll im StatsView aufgerufen werden und als Basis für den Liniengraph dienen.
Anmerkung: DataListProvider-Threads sollen als ExecutorPool in MoodStatsViewController, dieser Teil ist jedoch noch nicht fertig gestellt.

5.8 Streams und Lambda-Funktionen

DataListProvider: ParallelStream und Lamda zur Bearbeitung der moodDataList. Außerdem wird die Sicherheitsüberprüfung bei delete all im MenuBarController mit einer lamda-Funktion ausgewertet.
Wir verwenden auch sonst, an ein paar Stellen, Lambda-Ausdrücke, weil diese den Code übersichtlicher und kürzer machen.

5.9 Factories

Um verschiedene Speicher-Formate zu unterstützen, haben wir uns entschieden ein SaveFileManager-Interface zu verwenden. Wird dieses

Interface in einer neuen Klasse z.B. “JsonManager” implementiert, so braucht man diese nur in der SaveFileManagerFactory hinzufügen und im DataManager auswählen, um die Daten im neuen, gewünschten Format zu speichern.
Eine mögliche Erweiterung wäre, die Auswahl des Datei-Formats dem Nutzer zu überlassen z.B. über eine ComboBox.

6. Ausgefüllter Bewertungsbogen

Vorname	Nachname	Kürzel	Mat	Projekt	Arc.	Clea	Doku	Tests	GUI	Logg	UML	Threads	Streams	Profiling	Summe	Kom	Projekt-Note
Alessa	Wiesner	aw184		Moodtracker	3	3	3	3	3	3	2	3	3	3	29,00		1,00
Peter	Gutjahr	pg055													0,00		5,00
Jannika	Seybold	js443													0,00		5,00

Bewertungsbogen ist auch im Repository!