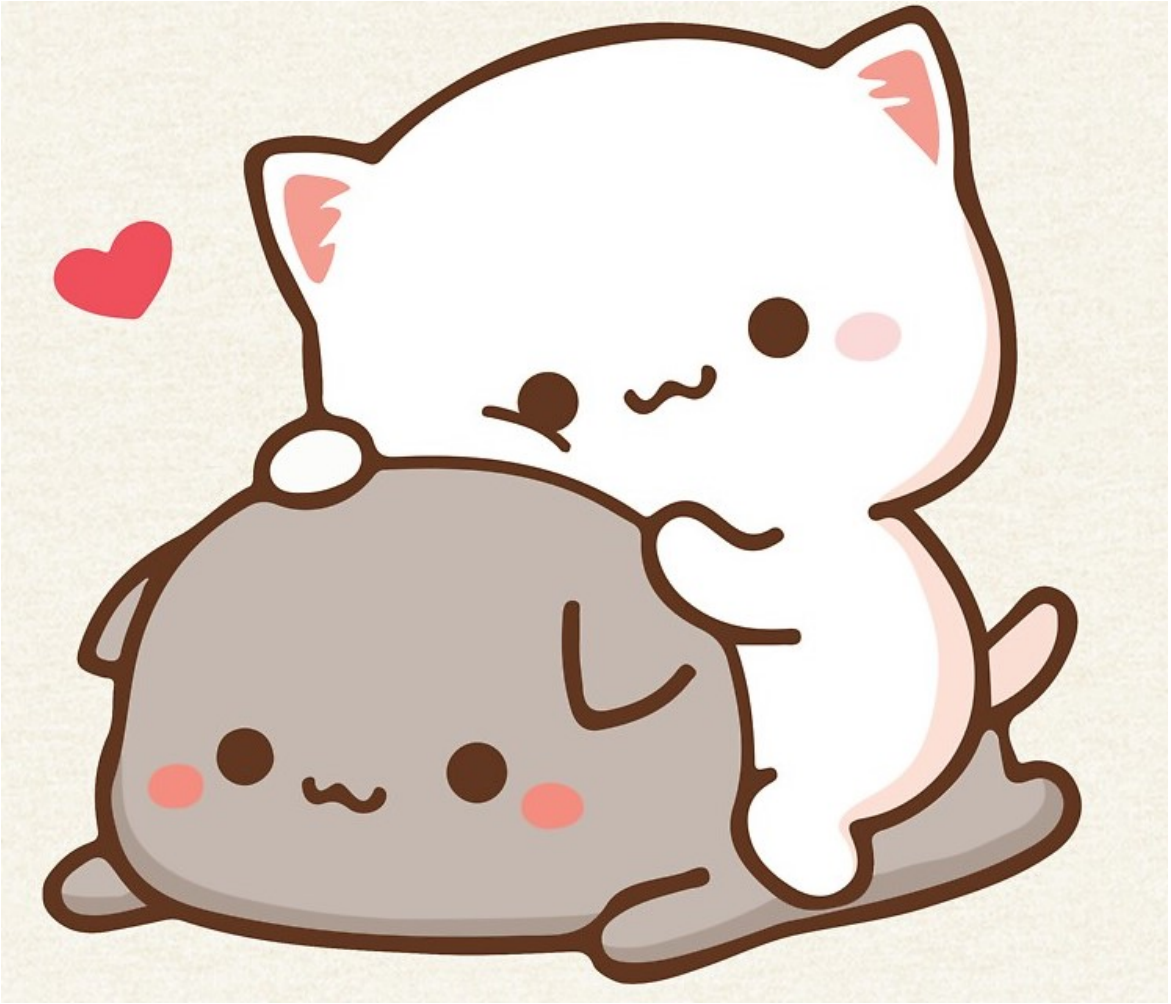
PetStudyBuddies

Software Entwicklung 2, Sommersemester 2021

https://gitlab.mi.hdm-stuttgart.de/kb136/petstudybuddies



Kristin Behringer (kb136)

Bastian Bodenhöfer (bb100)

Felix Schindler (fs146)

Kurzbeschreibung des Projekts

Bei unserem Projekt handelt es sich um einen Studienplaner mit der Funktion Accounts, Pets, eigene Notizen und ToDo-Listen anzulegen und zu verwalten. Notizen und ToDo-Listen können außerdem freigegeben werden. Die Speicherung und Verwaltung der persistenten Daten findet über eine SQLite Datenbank statt, die beim ersten Mal ausführen automatisch erstellt wird.

Jeder Benutzer legt seinen eigenen Account an. Hierfür benötigt er einen Benutzername, ein Passwort und eine E-Mail-Adresse. Passwort und E-Mail kann der Benutzer bei Bedarf ändern.

Nach dem Login befindet sich der Benutzer auf der Startseite. Von hier aus kann er zwischen den verschiedenen Menüpunkten navigieren.

Unter dem Menüpunkt “To Do” kann man eigene ToDo-Listen anlegen und löschen. Wenn man eine auswählt kann man sie außerdem flaggen, umbenennen und freigeben für andere Benutzer. Außerdem Tasks erstellen, verwalten und zuteilen.

Je nachdem wie schnell der User diese Listen dann bearbeitet wirkt sich das auf die Stimmung seines Pet's aus. Sollte ein User noch sehr viele unerledigte Aufgaben offen haben, so wird sein Pet traurig. Erledigt dieser jedoch seine Aufgaben sehr schnell, so wird sein Pet glücklich sein und sich über den Fortschritt freuen. Der Gemütszustand des Pet's kann jederzeit im Menüpunkt Pet über Text und dynamisch wechselnde Bilder nachgeschaut werden.

Unter dem Menüpunkt “Notes” kann der Benutzer seine Notizen erstellen, löschen, bearbeiten und diese an andere freigeben.

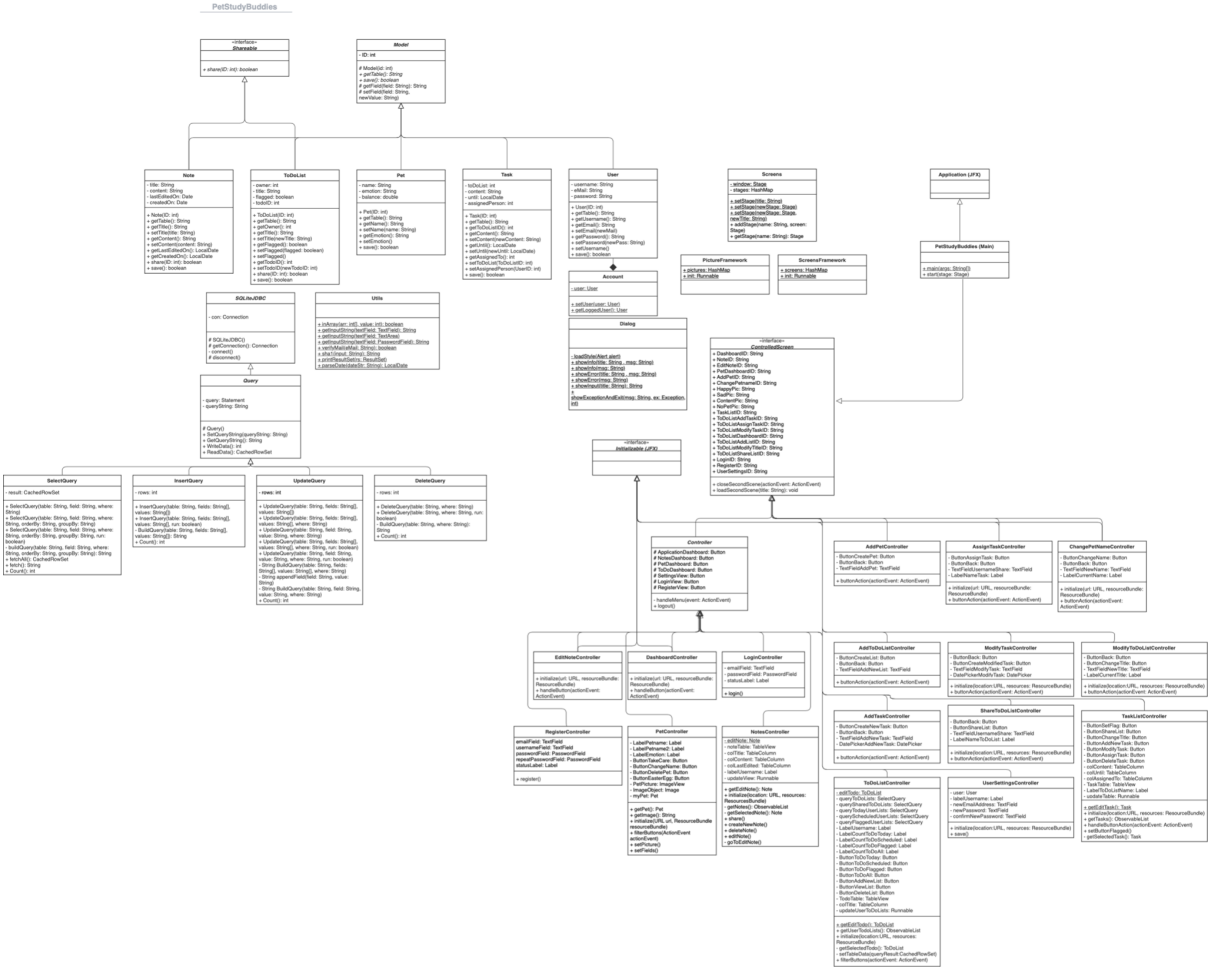
Startklasse

Die Main – Methode befindet sich in der Klasse “PetStudyBuddies“ im Package “de.hdm_stuttgart.mi.PetStudyBuddies“.

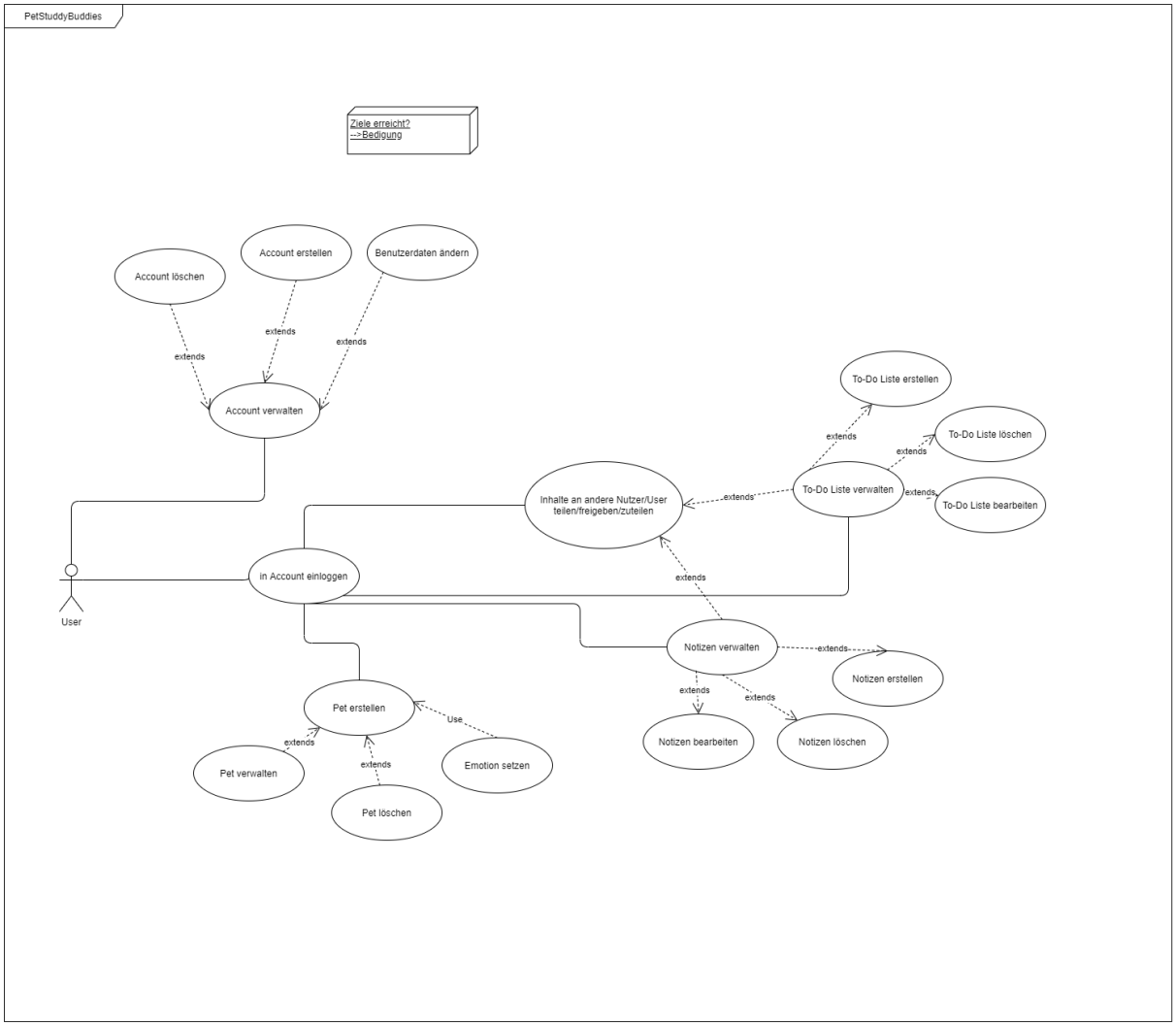
Besonderheiten

Die Funktionen Studium, Vorlesungen und Kalender wurden nicht umgesetzt, sind aber als Erweiterungen des Programms gedacht. Diese befinden sich im root-Ordner des Projekts unter “Erweiterungsmöglichkeiten”. Unser Wunsch war es, diese auch in unser Projekt einzubauen. Zeitliche, persönliche/familiäre Umstände bzw. fehlende Gruppenbeteiligung verhinderten dies leider.

UML – Klassendiagramm



UML Use-Case-Diagramm



Stellungnahmen:

Architektur:

Ordner Struktur:

In dem Ordner “de.hdm_stuttgart.mi.PetStudyBuddies“ befinden sich, neben der Startklasse PetStudyBuddies, noch die Unterordner:

controllers: Dieser Ordner beinhaltet alle Controller Klassen, die die fxml Dateien steuern

core: In diesem Ordner befinden sich Klassen, die von überall aufgerufen werden, darunter die Frameworks für Pictures und Screens, aber auch u. a. die Utils-Klasse. Außerdem sich zwei Unterordner:

- -db : In diesem Ordner befinden sich die Query Klassen, die mit der Datenbank kommunizieren
- -user: In diesem Ordner befindet sich die Account-Klasse, die den aktuell angemeldeten User speichert

models: In diesem Ordner befinden sich die Model Klassen für die verschiedenen Objekte (z.B. Pet, User, Note)

views: In diesem Ordner befindet sich die Dialog Klasse, die einfache Dialogfenster definiert

Clean Code:

Hierbei wurde darauf geachtet, dass soweit möglich auf Public deklarierte Objekt Variablen verzichtet wurde. Es wurde auf eine einheitliche Naming Convention geachtet und es wurden verständliche Kommentare hinzugefügt, um schnell verstehen zu können wo was wann warum passiert.

Tests:

Um die Funktionen der wichtigsten Methoden zu überprüfen, haben wir in dem Ordner (src > test > java > de.hdm_stuttgart.mi.Pet StudyBuddies > Core Test) verschiedene Testklassen erstellt. Diese beinhalten sowohl Positiv- als auch Negativtests.

GUI:

Die GUI haben wir in JavaFX mit Hilfe des Scene Builders erstellt. In dem Ordner (src > main > resources > fxml) befinden sich die verschiedenen fxml Dateien.

In dem Ordner (src > main > java > de.hdm_stuttgart.mi.Pet StudyBuddies > controllers) befinden sich die jeweiligen Controller Klassen, die die fxml Dateien steuern.

Logging / Exceptions:

log.debug	> Allgemeinen Informationen, um nachvollziehen zu können, was in diesem Moment passiert.
log.error	> Wenn ein Vorgang nicht funktioniert hat (z.B. ein Fehler ist bei der Registrierung aufgetreten)
log.catching	> Um Exceptions zu loggen
log.info	> Zusätzliche Informationen zu z. B. einem Fehler

Außerdem werfen einige Funktionen Exceptions, wenn man z. B. versucht korrupte Daten zu speichern.

UML:

Zu Beginn des Projekts haben wir uns Erstmal ein Use Case Diagramm erstellt. Darin haben wir die Grundlegenden Funktionen und Abläufe skizziert und definiert.

Dies hat sehr geholfen um einen ersten Überblick über das Projekt zu bekommen.

Anschließend haben wir alles in einem Klassendiagramm spezifiziert und im Detail angepasst.

Threads:

Zum Initialisieren des Screen Frameworks, Picture Frameworks sowie zum Aktualisieren der Screens “Notes”, “ToDoListDashboard” und “TaskList” werden Threads verwendet. Diese werden als Runnable initialisiert und arbeiten so unabhängig vom Application-Thread.

Streams und Lambda-Funktionen auf Collections:

In der Model-Klasse “Pet” wird in der Methode “setEmotion()” eine ArrayList zur Berechnung der Emotion des Pets genutzt. Die Liste speichert alle IDs der ToDo Lists, die ein Benutzer besitzt. Die IDs enthält die Liste aus einer Datenbankabfrage (SelectQuery). Diese werden später genutzt, um die Anzahl der Tasks, die Zahl der offenen und die Zahl der erledigten Tasks in diesen ToDo Listen des jeweiligen Benutzers festzustellen mit einer weiteren Datenbankabfrage. Dazu wird im Weiteren mit zwei Streams auf die Liste zugegriffen. Diese Streams rufen für jede ToDoList ID in der Liste die Datenbank nach der Zahl der enthaltenen Tasks auf. Das Ergebnis wird nach Umwandlung in einen Integer auf den Betrag der Variablen “nTasksOpen/Closed” addiert, welche die Gesamtzahl erfassen.

Einsatz von Interfaces bzw. Vererbung

Für das bessere Managen der Screenwechsel zwischen den einzelnen Scenes, haben wir das Interface “ControlledScreen” implementiert. Dieses enthält notwendige Variablen zum Scene-Wechsel. Diese werden den Frameworks zum Picture- und fxml-Management übergeben. Die Frameworks enthalten beide jeweils eine HashMap, welche das einfache Auffinden der Ressourcen mit searchKeys möglich macht.

Außerdem enthält das Interface ControlledScreen die Methoden “closeSecondScene” und “loadScecondscene”, welche bei Nutzung von zwei Fenstern auf dem Screen erforderlich sind.

Das Package “models” enthält ein weiteres Interface “Shareable”, welches die Models implementieren, die man sharen kann (z. B. ToDoList oder Note).

Wir nutzen in unserem Projekt komplexe Vererbungen über mehrere Packages. Daher verzichten wir auf nähere Erläuterungen und weisen auf unser ausführliches UML-Klassendiagramm hin.

Dokumentation:

Wichtige Methoden und Klassen mittels javadoc dokumentiert, außerdem viele Abläufe innerhalb Funktionen kommentiert.

Ausgefüllter Bewertungsbogen (Excel) für das Projekt

Vorname	Vorname	Nachname	Kürzel	MatrikelnummProjekt	Arc.	Clean Code	Doku	Tests	GUI	Logging/Except.	UML	Threads	Streams	Profiling	Summe - Projekt	Kommentar	Projekt-Note
	Felix	Schindler	fs146	40892 PetStuddyBuddies	2		1	3	3	3	3	3	3	2	23,00		2,30
	Kristin	Behringer	kb136	40898 PetStuddyBuddies	2		1	3	3	3	3	3	3	2	23,00		2,30
	Bastian	Bodenhofer													0,00		5,00