Morik Password Manager

Programmentwurf

von

Moritz Gutfleisch

und

Erik Zimmermann

Abgabedatum:	01. Februar 2018
Bearbeitungszeitraum:	01.10.2017 - 31.01.2018
Matrikelnummer, Student:	0000000, Moritz Gutfleisch
Matrikelnummer, Student:	0000000, Erik Zimmermann
Kurs:	TINF19B1
Gutachter der Dualen Hochschule:	Daniel Lindner

Abstract

- English -

This is the starting point of the Abstract. For the final bachelor thesis, there must be an abstract included in your document. So, start now writing it in German and English. The abstract is a short summary with around 200 to 250 words.

Try to include in this abstract the main question of your work, the methods you used or the main results of your work.

Abstract

- Deutsch -

Dies ist der Beginn des Abstracts. Für die finale Bachelorarbeit musst du ein Abstract in deinem Dokument mit einbauen. So, schreibe es am besten jetzt in Deutsch und Englisch. Das Abstract ist eine kurze Zusammenfassung mit ca. 200 bis 250 Wörtern.

Versuche in das Abstract folgende Punkte aufzunehmen: Fragestellung der Arbeit, methodische Vorgehensweise oder die Hauptergebnisse deiner Arbeit.

Inhaltsverzeichnis

Abkürzungsverzeichnis

SQL Structured Query Language

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

1 Einleitung

2 Clean Architecture

2.1 Plugins

2.1.1 Datenbank

Die Datenbank stellt eines der Plugins dar. Als Technologie wurde hier eine SQLite3 Datenbank verwendet. Die konkrete Implementierung der Schnittstelle zur SQLite-Datenbank befindet sich in der Klasse *SQLiteDatabase*. Diese Klasse erbt von der abstrakten Klasse *AbstractDatabase*, die sich in der Applikationsschicht befindet. Hierdurch wird eine Dependency Inversion erreicht, da nun eine *AbstractDatabase* von anderen Klassen verwendet werden kann, um SQL-Befehle auf der Datenbank auszuführen, statt eine konkrete *SQLiteDatabase* zu verwenden, was die Abhängigkeit von innen nach außen laufen lassen würde.

2.1.2 Verschlüsselung

Ein weiteres Plugin ist die Verschlüsselungsbibliothek. Als solche wurde Cryptopp[1] verwendet. Die abstrakte Klasse *Cipher* definiert die Schnittstelle in der Applikationsschicht, die das Plugin implementieren muss. Eine konkrete Implementierung dieser Schnittstelle befindet sich in der Klasse *CBC_Cipher*. Dadurch findet auch hier eine Dependency Inversion statt, da *Cipher* verwendet werden kann, um Verschlüsselung zu verwenden, statt eine Dependency auf eine konkrete Klasse der Pluginschicht zu brauchen. Die Schnittstelle muss lediglich einen String ver- und entschlüsseln können unter Angabe des Klar- bzw. Geheimtextes und eines Schlüssels. *CBC_Cipher* implementiert dies für die Chiffren, die im *BLOCK* enum aufgezählt sind. Momentan handelt es sich dabei um AES und Serpent, neue Chiffren können allerdings leicht hinzugefügt werden, solange sie von Cryptopp unterstützt werden. Diese werden als Block-Chiffren mit Cipher Block Chaining (CBC) als Betriebsmodus verwendet. Um andere Betriebsmodi, oder Nicht-Blockchiffren zu

[1] https://cryptopp.com/

verwenden, müssten weitere Implementierungen des *Cipher* Interfaces hinzugefügt werden. Dies ist ohne Weiteres möglich, solange ein *String* zür Schlüsselherleitung hinreichend ist.

2.2 Adapter

2.2.1 Datenbank

Die Klasse *DbInterface* implementiert die eigentliche Funktionalität in Form der SQL-Anweisungen und führt diese über eine konkrete Implementierung der *AbstractDatabase* in der Datenbank aus. Somit geht keine Funktionalität verloren, wenn das Plugin durch eine andere Datenbanktechnologie ausgetauscht wird. Die Umsetzung der Funktionalität als SQL-Anweisungen bedeutet jedoch, dass ein Austausch des Plugins nur ohne Weiteres möglich ist, wenn die neue Datenbank ebenfalls eine SQL-Datenbank ist. Handelt es sich bei der neuen Datenbank jedoch beispielsweise um eine noSQL-Datenbank, so muss die Funktionalität, also die Abfragen, angepasst werden. Der Klasse *DbInterface* wird im Konstruktor eine *AbstractDatabase* übergeben, was die Dependency Injection umsetzt.
Auf die Benutzung von Prepared Statements wurde innerhalb des Adapters verzichtet, da die Datenbank lokal ist und nur der Benutzer Befehle auf ihr ausführt. Würde die Datenbank über eine öffentliche Schnittstelle angesteuert werden, so wäre dies nicht zu vernachlässigen.