HTBLuVA Salzburg

Höhere Lehranstalt für
Elektronik und Technische Informatik

DIPLOMARBEIT

Gesamtprojekt

KeyS

Entwicklung eines sicheren Daten Management Systems

Andreas Schlager 5AHEL Betreuer: Dipl.-Ing. Roman Schragl

Maximilian Irran 5AHEL

Benjamin Bayer 5AHEL

Kooperationspartner: ITSP Services GmbH

ausgeführt im Schuljahr 2019/20

Abgabevermerk:

Datum: 03.04.2020 übernommen von:

htl bildung mit zukunft	**HÖHERE TECHNISCHE BUNDESLEHR- UND VERSUCHSANSTALT Salzburg**
	Elektronik und Technische Informatik

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Salzburg, am 03.04.2020

Verfasserinnen / Verfasser:

Andreas Schlager

Maximilian Irran

Benjamin Bayer

htl bildung mit zukunft	**HÖHERE TECHNISCHE BUNDESLEHR- UND VERSUCHSANSTALT Salzburg**
	Elektronik und Technische Informatik

DIPLOMARBEIT
DOKUMENTATION

Namen der Verfasserinnen / Verfasser	Andreas Schlager Maximilian Irran Benjamin Bayer
Jahrgang Schuljahr	5AHEL 2019/20
Thema der Diplomarbeit	KeyS – Entwicklung eines sicheren Daten Management Systems
Kooperationspartner	ITSP Services GmbH

Aufgabenstellung	In Unternehmen ist es gängig viele verschiedene Kennwörter für arbeitsrelevante Services zu erhalten. Der Verlust eines Kennworts ist ein Sicherheitsrisiko, denn meist ist die Wiederherstellung oder Zurücksetzung mit großem Aufwand verbunden. KeyS ist ein unternehmeninternes Verwaltungsprogramm für sensible Daten, wie Passwörter und vertrauliche Dokumente. Die Daten werden verschlüsselt in einer Datenbank verwaltet und können bei Bedarf mit ausgewählten Benutzern geteilt werden.

Realisierung	Über eine Webanwendung, basierend auf dem ASP.NET Core Framework, kann KeyS gesteuert und konfiguriert werden, eine Client-Anwendung ermöglicht die Verschlüsselung der Daten. Sämtliche Daten werden über eine SQLite Datenbank verwaltet. Die Software ist flexibel und erweiterbar aufgrund der Anwendung von modernen Design-Prinzipien und Entwurfsmustern. KeyS und ASP.NET Core sind in der Programmiersprache C# entwickelt.

Ergebnisse	KeyS ist durch eine sichere Verschlüsselung und modularem Design ein nützliches Programm für den unternehmenweiten Einsatz. Über die Client Anwendung können Daten sicher verschlüsselt und verwaltet werden. Die Konfiguration über die Client- oder Webanwendung ermöglicht einen flexiblen und komfortablen Umgang mit der Software.

htl bildung mit zukunft	**HÖHERE TECHNISCHE BUNDESLEHR- UND VERSUCHSANSTALT Salzburg**
	Elektronik und Technische Informatik

Typische Grafik, Foto etc. (mit Erläuterung)	

Teilnahme an Wettbewerben, Auszeichnungen	

Möglichkeiten der Einsichtnahme in die Arbeit	Bibliothek der HTBLuVA-Salzburg

Approbation (Datum / Unterschrift)	Prüferin / Prüfer	Direktorin / Direktor Abteilungsvorständin / Abteilungsvorstand

htl bildung mit zukunft	**COLLEGE OF ENGINEERING** **Salzburg**
	Electronics and Computer Engineering

DIPLOMA THESIS
Documentation

Author(s)	Andreas Schlager Maximilian Irran Benjamin Bayer
Form Academic year	5AHEL 2019/20
Topic	KeyS – development of a secure data management system
Co-operation Partners	ITSP Services GmbH

Assignment of Tasks	In companies it is common to have many different passwords for work-related services. The loss of a password is a danger and the recovery or reset requires great effort. KeyS is an in-house management tool for sensitive data, such as passwords and confidential documents. The data is encrypted in a database and can be shared with selected users as needed.

Realisation	KeyS can be controlled and configured via a web application based on the ASP.NET Core Framework, a client application enables the encryption of the data. All data is managed via an SQLite database. The software is flexible and expandable due to the application of modern design principles and design patterns. KeyS and ASP.NET Core are developed in the C# programming language.

Results	With secure encryption and modular design, KeyS is a useful program for company-wide use. Data can be securely encrypted and managed using the client application. Configuration via the client or web application enables flexible and convenient use of the software.

htl bildung mit zukunft	**COLLEGE OF ENGINEERING** **Salzburg**
	Electronics and Computer Engineering

Illustrative Graph, Photo (incl. explanation)	

Participation in Competitions Awards	

Accessibility of Diploma Thesis	Library of the HTBLuVA-Salzburg

Approval (Date / Sign)	Examiner	Head of College Head of Department

Vorwort

Im Rahmen dieser Arbeit beschäftigen wir uns mit der Entwicklung eines sicheren Daten Management Systems. In Unternehmen kommt es vor, dass ein Mitarbeiter viele verschiedene Kennwörter für arbeitsrelevante Services besitzt. Der Verlust eines Kennworts ist ein Sicherheitsrisiko, denn meist ist die Wiederherstellung oder Zurücksetzung mit großem Aufwand verbunden. KeyS ist ein unternehmeninternes Verwaltungsprogramm für sensible Daten, wie Passwörter und vertrauliche Dokumente. Die Daten werden verschlüsselt in einer Datenbank verwaltet und können bei Bedarf mit ausgewählten Benutzern geteilt werden.

Danksagung

An dieser Stelle möchten wir uns ganz herzlich bei allen Mitwirkenden bedanken, die uns über die Dauer des Projekts mit Rat zur Seite gestanden sind.
Wir bedanken uns ganz herzlich bei Herr Prof. Mag. phil. Mario Wegscheider für das gemeinsame Vorbereiten der Präsentation und die Unterstützung bei der schriftlichen Arbeit.
Einen besonders großen Dank möchten wir an unseren Projektbetreuer, Herr Prof. Dipl.-Ing. Roman Schragl, ausrichten. Stets als beratende Stimme an unserer Seite, war Herr Schragl eine zentrale Figur für den Erfolg unserer Arbeit.
Vielen Dank an alle Mitwirkenden!

Inhaltsverzeichnis

1 Einleitung

Um Passwörter und Daten zu verwalten, können Unternehmen die Software von KeyS verwenden. Das Unternehmen richtet einen KeyS-Server im internen Netzwerk ein, auf dem die Benutzer verschlüsselt ihre Daten ablegen können. Daher bildet eine ASP.NET Core Webanwendung die zentrale Einheit der Software. Für die Bedienung der Anwendung stehen eine, über den Browser erreichbare, Weboberfläche und eine Programmschnittstelle, in Form einer REST API, zur Verfügung. Der Server kann über die Weboberfläche konfiguriert werden. Der KeyS-Client, eine Anwendung in Form einer Konsolen Applikation, kapselt die Verschlüsselung der Daten und die Kommunikation mit der REST API. Die Webanwendung verwaltet die Daten des Clients in einer SQLite Datenbank.

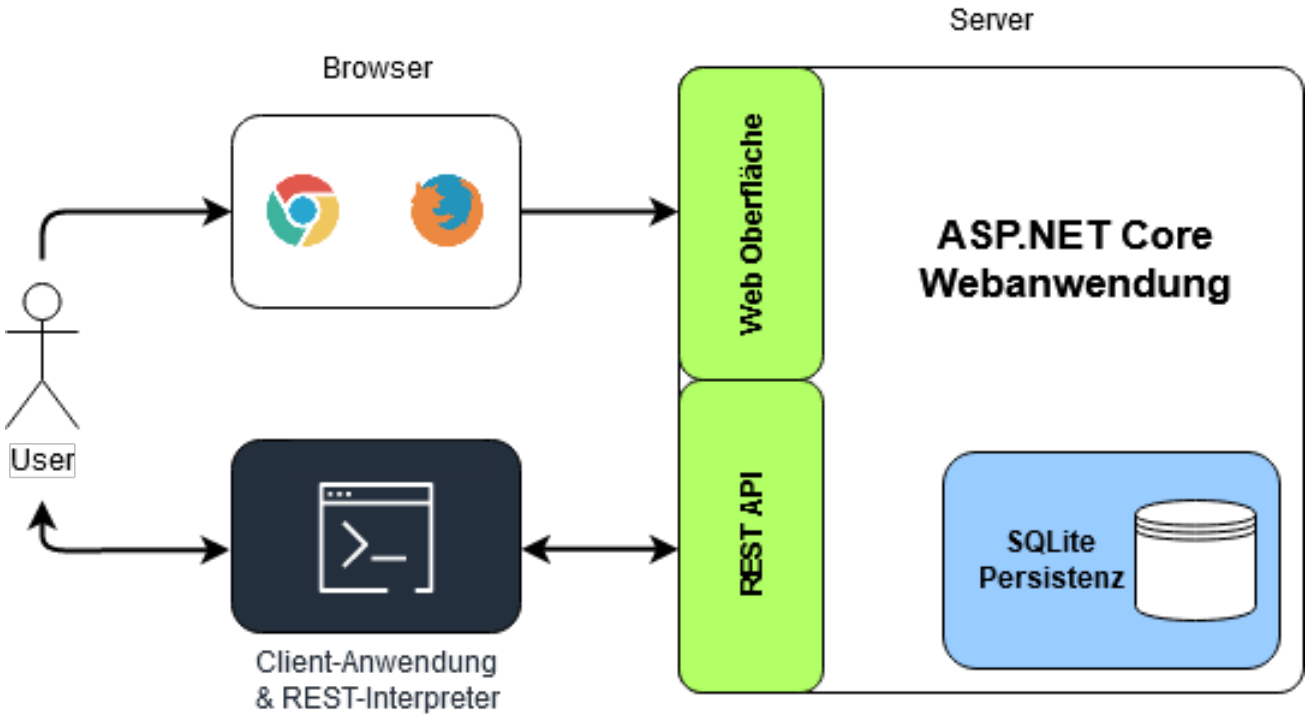


Abbildung 1.1: Aufbau von KeyS

2 Software Architektur

Die Architektur einer Software beschreibt den Aufbau von Programmelementen. Sie definiert Programmgruppen (Module) und ihre Interaktionen untereinander. Das Ziel einer Architektur ist es, die Module voneinander unabhängig zu gestalten und dadurch den einfachen Austausch einzelner zu gewährleisten.

2.1 SOLID Prinzipien

Die SOLID Design Prinzipien sind Regeln für die Entwicklung von flexibler, wartbarer (objektorientierter) Software. Sie vermeiden stark gekoppelten Code und führen zu robusten Lösungen, die flexible auf Änderungen reagieren. Die fünf namensgebenden Prinzipien sind:[1]

- **S**ingle-Responsibility-Prinzip
- **O**pen-Closed-Prinzip
- **L**iskovsches Substitutionsprinzip
- **I**nterface-Segregation-Prinzip
- **D**ependency-Inversion-Prinzip

Software die nach den SOLID Prinzipien entwickelt wird verspricht eine besonders hohe Wartbarkeit und somit eine längere Lebensdauer. Die Entwicklung und das Design von KeyS ist besonders stark durch drei dieser Prinzipien geprägt, auf die im Folgenden näher eingegangen wird.

[1] Robert C. Martin. *Clean Architecture*. Robert C. Martin Series. 2017. ISBN: 978-0-13-449416-6, S. 47-78.

2.1.1 Single-Responsibility-Prinzip

Das Single-Responsibility-Prinzip besagt, dass eine Klasse, oder ein Modul sich nur gegenüber den Anforderungen *eines* Akteurs ändern darf.[2] Ein Akteur ist eine Quelle, die Gründe bietet, weshalb die Software angepasst werden muss, zum Beispiel eine Änderung des Datenbank Schemas, oder die Änderung der Protokollausgaben. Bindet eine Klasse die Verantwortung gegenüber mehreren Akteuren, so koppelt sie diese auch aneinander. Eine Änderungen im Sinne einer Verantwortung kann dadurch Einfluss auf einen komplett anderen Aspekt des Systems haben. Klassen, die nur einem Akteur verantwortlich sind, ändern sich seltener und bleiben in ihrer Funktionalität überschaubar. Oft kann eine Verteilung der Verantwortung über ein Decorator Pattern erreicht werden.

2.1.2 Interface-Segregation-Prinzip

In der objektorientierten Software Entwicklung sind die Programmierer dazu angehalten gegen Schnittstellen, fortführend als Interfaces, zu programmieren, d.h. konkrete Klassen zu vermeiden. Entwickler können dazu verleitet werden viele Methoden in einem Interface zu definieren. Dadurch entsteht eine Abhängigkeit der implementierenden Klassen auf alle Funktionen des Interfaces. Klassen, welche das Interface verwenden haben Abhängigkeiten zu Funktionen, die sie möglicherweise nicht brauchen. Um dies zu vermeiden, werden die Interfaces aufgeteilt. Ein entscheidendes Kriterium in der Aufteilung ist außerdem das Single-Responsibility-Prinzip, da die Funktionen nach der Teilung so gruppiert sein müssen, dass sie nur einem Akteur gegenüber verantwortlich sind.

[2] Martin, *Clean Architecture*, S. 50.

2.1.3 Dependency-Inversion-Prinzip

Ein typischer Ansatz in der Software Entwicklung ist es Module in Schichten zu unterteilen. Je niedriger die Schicht, in welcher ein Modul arbeitet, desto spezifischer sind die Implementation an periphere Details gebunden. Man spricht oft von High-, Mid- und Low-Level Modulen. Ein Low-Level Modul kann beispielsweise die Kommunikation mit der unterliegenen Hardware kapseln. High-Level Module beinhalten Vorgänge, die auf dieser Abstraktion arbeiten. Die Änderung der Hardware sollte keinen direkten Einfluss auf die Vorgänge in High-Level Komponenten haben. Die folgende Grafik illustriert diesen Aufbau.

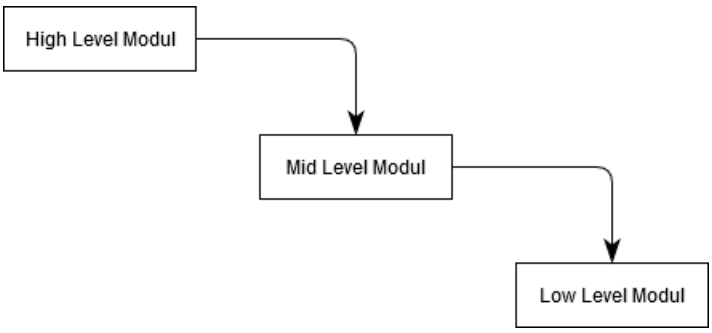


Abbildung 2.1: Struktur in Schichten

In Abbildung 2.1 bildet ein Pfeil eine Abhängigkeit ab. Ein Modul ist von einem anderen Modul in Richtung des Pfeiles abhängig. In dieser Struktur ist jede Abhängigkeit eine Quellcode-Abhängigkeit, d. h. die konkrete Klasse wird in dem Quelltext der anderen erwähnt. In C/C++ kann solch eine Abhängigkeit beispielsweise über eine Include-Direktive entstehen.

```
// HighLevelModul.cpp
#include "MidLevelModul.h"
```

Listing 2.1: Quellcode-Abhängigkeit durch C/C++-Include-Direktive

Ändert sich ein Low-Level Modul, so muss dieses erneut kompiliert werden. Durch die transitive Eigenschaft der Quellcode-Abhängigkeit müssen auch alle Mid-Level Komponenten, die auf dem entsprechenden Low-Level Modul beruhen, neu kompilieren. Diese Verhalten setzt sich bis zu den High-Level Komponenten fort. Dieser Einfluss der niedrigsten Module auf die Höchsten lässt auf ein Problem schließen. Ein bessere Struktur definiert auf jeder Schicht eines oder mehrere Interfaces, welche durch die Elemente der niedrigeren Schichten implementiert werden müssen (Abb. 2.2).

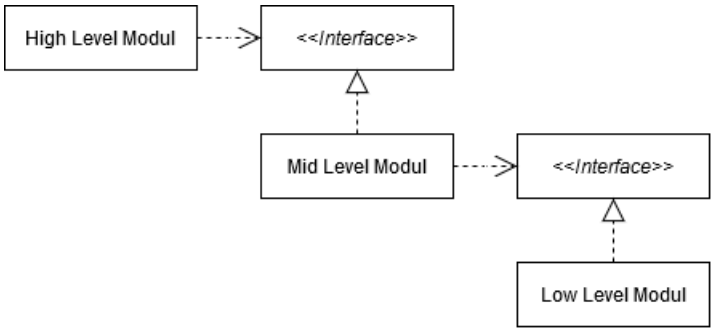


Abbildung 2.2: Dependency-Inversion durch Interfaces

Die Interfaces bilden einen Vertrag an den sich Module niedrigerer Schicht halten müssen. Die Abhängigkeiten invertieren sich im Diagramm und zeigen in Richtung der höheren Schichten. Dieser Aufbau garantiert robusteren und wiederverwendbaren Code, da die zentrale Logik der Anwendung in den höheren Schichten abstrakt ist und sich von peripheren Details löst.

2.2 Testen

Mit der wachsenden Komplexität eines Projektes ist es unmögliche eine klare Aussagen über die Funktionalität der Software zu einem gegeben Zeitpunkt zu treffen, ohne sie laufend zu testen. Hierbei ist es entscheidend, dass die Tests nur dann bestanden werden können, wenn die Software den Anforderungen entspricht. Ein Softwaretest soll nicht die konkrete Implementation des zu testenden Moduls überprüfen, sondern dessen Funktionsweise.

2.2.1 Komponententests

Ein Komponententest überprüft kleinstmögliche Komponenten des Softwaresystems. Eine Komponente kann aus einer einzelnen Klasse bestehen und wird über ihre öffentlichen Funktionen getestet. Dabei wird versucht die Komponente möglichst isoliert von dem System zu überprüfen, weshalb es wichtig ist sie von externen Abhängigkeiten zu entkoppeln. Die nötigen Abhängigkeiten werden während den Tests durch sogenannte Mock-Objekte ersetzt. Ein Mock ist eine Art Stellvertreter, welcher eine vordefinierte Rückgabe liefert und an den jeweilige Test angepasst wird. Ein Komponententest folgt in der Regel einem Arrange-Act-Assert-Aufbau. In der Arrange-Phase muss eine Testumgebung vorbereitet werden, in diesem Schritt erstellt der Entwickler Testdaten und initialisiert Mock-Objekte, welche für die Komponente notwendig sind. Anschließend wird die zu testende Operation in der Act-Phase durchgeführt. Mittels Assert-Anweisungen wird die Ausgabe des Ablaufes überprüft und festgestellt, ob der Test erfolgreich ist.[3]

```
[TestMethod]
public void IsOddNumberTest() {
        // arrange
        MyClass instance = new MyClass();
        int[] numbers = {1, 2};
        // act & assert
        Assert.IsTrue(instance.IsOdd(numbers[0]));
        Assert.IsFalse(instance.IsOdd(numbers[1]));
}
```

Listing 2.2: Beispiel C# Komponententest mittels MSTest-Framework

Ein Test inspiziert nicht den Algorithmus, welcher zu der Ausgabe führt. In Listing 2.2 wird nicht überprüft, wie eine ungerade Zahl ermittelt wird, sondern lediglich anhand der Lösung, ob die Rückgabe richtig ist.

[3] *MSTest-Framework*. Microsoft. URL: https://docs.microsoft.com/en-us/dotnet/api/microsoft.visualstudio.testtools.unittesting (besucht am 26. 03. 2020).

2.2.1.1 Moq-Framework

Um ein Mock-Objekt zur Testzeit instanziieren zu können ist es nötig eine extra Klasse zu erstellen, die an das gewünschte Verhalten angepasst werden muss. Bei vielen kleinen Tests und mehreren Abhängigkeiten ist der Aufwand in der Erstellung geeigneter Klassen beachtlich. Eine Anpassung der Mock-Klasse für einen gegebenen Testfall kann außerdem einen unerwünschten Einfluss auf andere Tests haben. Damit das Testen mittels Mock-Objekten schnell und sicher funktioniert werden Mocking-Frameworks verwendet. Das Moq-Framework bietet eine flexible, minimalistische und streng typisierte Möglichkeit Mock-Objekte zu erzeugen und für jeden Test anzupassen.[4] Um einen neuen Mock zu erzeugen reicht es ein Objekt des Typs *Mock<T>* zu instanziieren, wobei *T* der Typ der gewünschten Mock-Klasse ist. Das Verhalten des Mock-Objekts kann über die Funktionen der *Mock* Instanz bestimmt und an den Test angepasst werden. Dieser Vorgang ist einfacher anhand des folgenden Beispiels illustriert. Abbildung 2.3 stellt ein Interface mit zwei Methoden zum Ein- und Ausloggen eines Benutzers

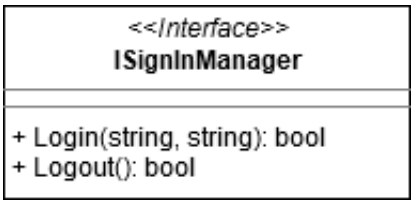


Abbildung 2.3: ISignInManager Interface

dar. Zur Testzeit ist nicht bekannt, ob die Benutzer beispielsweise lokal, oder über einen API-Endpoint eines anderen Servers eingeloggt werden. Um Klassen, die auf *ISignInManager* beruhen trotzdem testen zu können, wird ein Mock der Klasse benötigt.

```
Mock<ISignInManager> mock = new Mock<ISignInManager>();

mock.Setup(x => x.Login(It.Is<string>(s => s == "fooUser"),
                        It.Is<string>(s => s == "barPassword"))
          ).Returns(true);

ISignInManager manager = mock.Object;
```

Listing 2.3: Instanziierung und Konfiguration eines Mocks

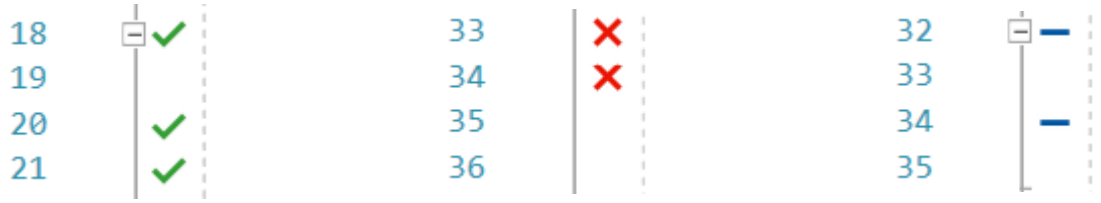
Nach der Instanziierung des Objekts *Mock<ISignInManager>* wird das Verhalten der benötigten Methode, in diesem Fall *Login*, angepasst. Über die statische Klasse *It* des Moq-Frameworks werden die erwarteten Aufrufparameter festgelegt. Über die Funktion *Returns* kann der gewünschte Rückgabewert bestimmt werden. Wird die *Login* Methode mit dem entsprechenden Benutzernamen und Passwort auf-

[4] *Moq-Framework*. Clarius Consulting, Manas und InSTEDD. URL: https://github.com/moq/moq4 (besucht am 26. 03. 2020).

gerufen, liefert sie als Rückgabewert *true*, anderfalls *false*. Der Zugriff auf das konfigurierte *ISignInManager* Objekt ist über das *Object* Property der Klasse *Mock* möglich.

2.2.1.2 *Live Unit Testing mit Visual Studio*

Live Unit Testing ist ein Feature der Visual Studio IDE seit Visual Studio 2017 und erlaubt es dem Entwickler Tests automatisiert auszuführen. Bei jeder Änderung des Codes kompiliert Visual Studio die nötigen Dateien erneut und führt jeden relevanten Test aus. Der Entwickler erhält somit sofort Feedback über die derzeitige Funktionalität der Software und kann frühzeitig auf neu eingeführte Fehler reagieren und diese beheben. Außerdem bietet das Live Unit Testing eine Echtzeit-Testabdeckungsvisualisierung anhand von Markierungen neben dem Code.



(a) Abgedeckt und bestanden (b) Abgedeckt aber nicht bestanden (c) Nicht von einem Test abgedeckt

Abbildung 2.4: Live Unit Testing Markierungen

2.2.2 Testgetriebene Entwicklung

Die Testgetriebene Entwicklung (englisch *test-driven-development*, TDD) ist ein Prozess, in dem Tests vor dem Produktionscode geschrieben werden.[5] Die Test treiben daher die Entwicklung der Software. Der Prozess wird in drei Schritte, in der Literatur oft unter den folgenden Namen, unterteilt, welche wiederholt durchlaufen werden:

1. Red
2. Green
3. Refactoring

In der erste Phase wird ein Test verfasst, welcher sofort fehlschlägt, da der zu testende Produktionscode noch nicht vorhanden ist. Ein nicht kompilierbarer Test zählt ebenfalls als fehlgeschlagen. Der Entwickler befindet sich in der zweiten Phase. Der Produktionscode muss um die nötigsten Komponenten erweitert werden, damit der derzeit fehlschlagenden Test besteht. Im letzten Schritte wird der Produktionscode

[5] Robert C. Martin. *Professionalism and Test-Driven Development*. 2007. URL: `http://fpl.cs.depaul.edu/jriely/450/extras/prof-tdd.pdf` (besucht am 26. 03. 2020).

entsprechend den geltenden Code-Standards aufgeräumt, hier werden etwaige Leistungsoptimierungen, Abstraktionen oder Verbesserungen der Lesbarkeit vorgenommen. Über die Schritte wird iteriert, bis das Feature vollständig implementiert und getestet ist. Eine Iteration dauert nur wenige Minuten.

2.2.2.1 Vorteile

Aufgrund der kurzen Iterationsdauer und den strikten Regeln der testgetriebenen Entwicklung ist die Arbeit auf das Wesentliche fokussiert, da nicht mehr Code als unbedingt nötig geschrieben werden kann. Dadurch wird keine Zeit an Funktionalitäten verschwendet, die nicht gebraucht werden und der Code bleibt überschaubar. Ein weiterer Vorteil ist die große und aussagekräftige Testsammlung, die eine Evaluation der Funktionalität zu jedem Zeitpunkt ermöglicht. Das Risiko einen Fehler bei der Erweiterung und der Verbesserung des Produktionscodes einzuschleichen ist somit wesentlich niedriger, die Tests sind also indirekt für die Flexibilität des Codes verantwortlich. Über das laufende Testen kann auch Zeit, welche zum Debuggen der Anwendungen verschwendet wird, verringert werden, da Fehler sofort auffallen. Die große Testsammlung dient außerdem als eine Art Dokumentation, da die kompakten Tests jede mögliche Verwendung der Software demonstrieren.

3 Server Persistenz

KeyS ist in zwei wichtige Bestandteile aufgeteilt. Die Client-Seite, welche für Verschlüsselungen und Server Anfragen verantwortlich ist und die Server-Seite, welche die Anfragen entgegen nimmt und sämtliche Daten verwaltet. Die Verwaltung der Daten beinhaltet das Einfügen, Updaten und Löschen von Benutzerdaten, Passwörtern, Schlüsseln, geheimen Daten und Ordnern. Da eine Datenbank allerdings ein peripheres Detail ist, muss die Anwendung eine Abstraktion um diese bilden. Die Entwicklung einer Persistenz Schicht ist notwendig. Der Vorteil einer Persistenz-Schicht ist, dass die zentrale Logik und Regeln der Anwendung vor Details der Datenbank, unabhängig davon ob diese nun beispielsweise eine SQLite Datenbank ist, oder ob die Daten in einer einfachen Text Datei verwaltet werden, geschützt sind. Außerdem kann ein Wechsel auf ein anderes Datenbank Management System jederzeit und ohne großen Aufwand erfolgen.

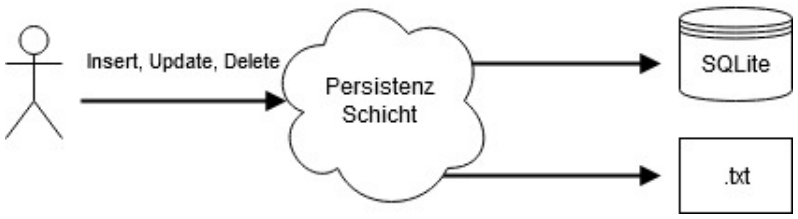


Abbildung 3.1: Abstraktion über Persistenz-Schicht

3.1 DB-Schema

Als Datenbank Management System verwendet KeyS eine SQLite Datenbank.[1] SQLite ist zum Testen und während der Entwicklungsphase besonders von Vorteil, da kein weiterer Webserver benötigt wird. Die Datenbank wird lokal auf dem Rechner in Form einer .db-Datei erstellt. Um mit der Datenbank arbeiten zu können muss zuerst ein Schema erstellt werden, welches die verschiedenen Daten in sinngemäße Tabellen unterteilt und korrekt widerspiegelt.

[1] *SQLite*. SQLite-Team. URL: `https://www.sqlite.org` (besucht am 30. 03. 2020).

3.1.1 Entity-Relationship-Diagramm

Eine Form der Entwicklung eines Datenbankschemas ist das Entity-Relationship-Diagramm (ERD). Ein ER-Diagramm besteht aus Entitäten, logische Objekte im Kontext der Anwendung, deren Eigenschaften und Beziehungen zu anderen Entitäten. Es gibt drei Arten der Beziehungen: 1:1, 1:N und N:M. Das Modell kann anschließend in Relationen umgewandelt werden. Jede Entität führt zu einer Tabelle, die Eigenschaften bilden die Attribute. Je nach Beziehung entsteht entweder eine eigene Tabelle, wenn es sich um N:M handelt, oder ein Fremdschlüssel als Attribut. Einige Vorteile des ER-Diagramms sind die intuitive Vorgehensweise, bekannt aus der objektorientierten Programmierung, und die automatisch hohe Normalform des resultierenden Schemas, wenn das Diagramm gut aufgebaut ist.

3.1.1.1 User

Der User ist eine der wichtigsten Entitäten der Anwendung. Er kann sich mit Benutzernamen und Passwort über den Servern anmelden und so auf seine Daten zugreifen. Der Benutzername kann einfach als Eigenschaft der Entität gesehen werden, das Passwort allerdings nicht. Passwörter können nicht als einfacher Text abgespeichert werden, da dies die Sicherheit massiv verringern würde. Stattdessen wird das Passwort durch eine Hashfunktion in andere Zeichenfolge konvertiert. Der Hash alleine würde weiterhin Angriffe über Rainbow Tables ermöglichen, weshalb dem Passwort zusätzlich eine zufällig gewählte Zeichenkette hinzugefügt wird, auch Salt genannt. Das gehashte Passwort und der Salt können nun zusammen als Attribut der Tabelle gespeichert werden. Außerdem erhält der Nutzer eine eindeutige ID, um die Identifikation und Suche zu erleichtern. Somit ergibt sich die Entität aus Abbildung 3.2.

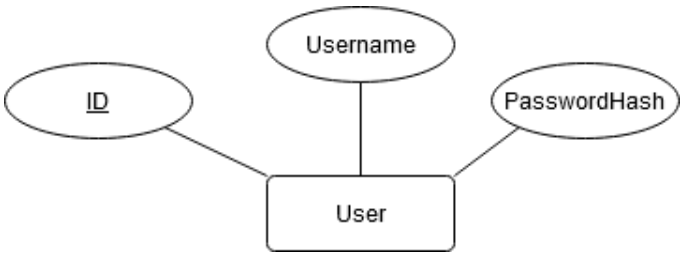


Abbildung 3.2: User Entität

Für die verschiedenen Berechtigungen eines Nutzers auf dem Server werden Rollen eingeführt. Je nach Rolle können Einstellungen am Server vorgenommen werden. Da mehrere Benutzer einer Rolle zugewiesen sein können, es aber keinen Nutzer ohne Rolle geben kann, entsteht eine 1:N Beziehung. Genauer betrachtet kann eine Rolle auch ohne Benutzer existieren, da nicht zu jeder Zeit ein Benutzer vorhanden sein muss, weshalb die Beziehung 1:0..N ist.

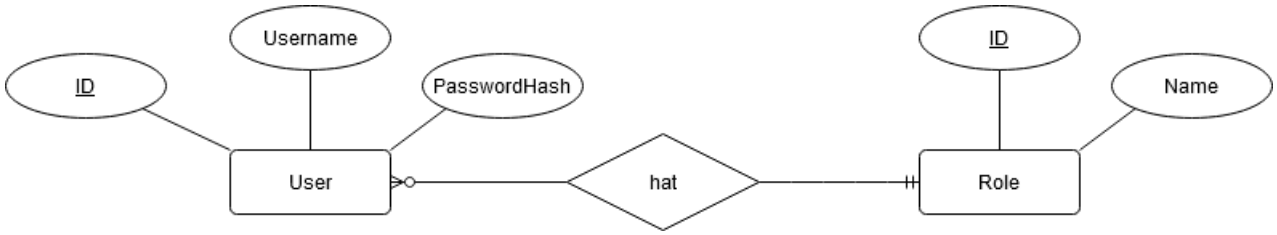


Abbildung 3.3: User & Roles Beziehung

3.1.1.2 Öffentliche Schlüssel

Damit ein Benutzer eine neue Zugriffsstelle, beispielsweise den Firmen-PC, oder den Heimlaptop, registrieren kann, erstellt er ein neues asymmetrisches Schlüsselpaar. Der öffentliche Schlüssel, in Grafiken als *Public Key* vermerkt, wird in der Datenbank, in Form eines Base64-Strings, des Servers abgelegt und muss dem Benutzer genau zugeordnet werden können. Außerdem können die Schlüssel aktiviert und deaktiviert werden, um den Zugriff einer Maschine auf neue geschützte Daten zu begrenzen. Es entsteht eine 1:0..N Beziehung, da ein Nutzer mehrere Schlüssel registrieren kann, aber nicht muss, und jeder Schlüssel nur einem Benutzer zugeordnet werden kann.

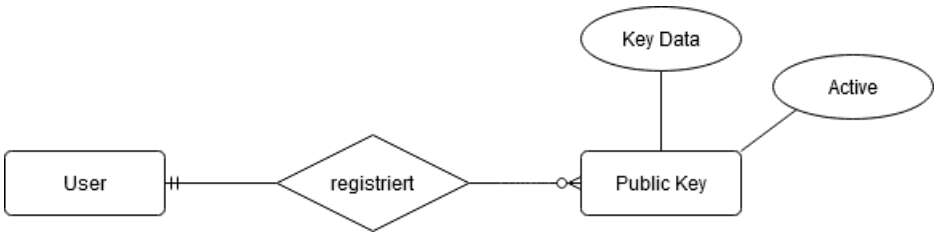


Abbildung 3.4: User & Öffentliche Schlüssel Beziehung

3.1.1.3 Geschützte Daten

Die geschützten Daten, in Grafiken als *Protected Data* vermerkt, bestehen aus den verschlüsselten Daten an sich, in Form eines Base64-Strings, und einer ID zur einfachen Identifikation. Um in dem UI eine aussagekräftige Information anzeigen zu können und dem Nutzer die Zuordnung zu erleichtern, erhalten alle einen Namen.

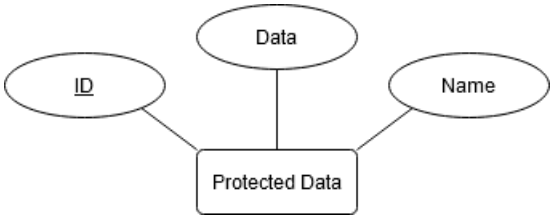


Abbildung 3.5: Geschützte Daten Entität

Mehrere Benutzer können Zugriff auf mehrere Daten haben und diese manipulieren, daher ergibt sich eine N:M Beziehung. Da eine N:M Beziehung immer zu einer eigenen Tabelle führt, wird in dieser im weiteren Verlauf auch die Art der Berechtigung des Benutzer festgehalten. Möglichen Berechtigungen sind: Read, Write und View.

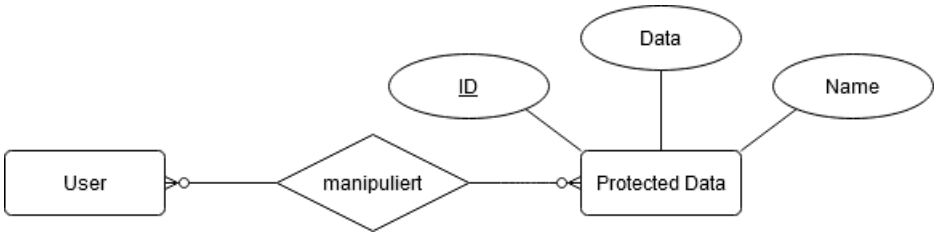


Abbildung 3.6: User & Geschützte Daten Beziehung

3.1.1.4 Symmetrische Schlüssel

Alle Daten auf die ein Nutzer zugreifen möchte sind mit einem symmetrischen Schlüssel verschlüsselt. Die symmetrischen Schlüssel an sich sind mit dem öffentlichen Schlüssel des Nutzer verschlüsselt. Da diese allerdings nicht durch den Nutzer selbst verschlüsselt wurden, sondern durch jemanden mit Zugriff auf die Daten, und bei einer Änderungen der Daten aktualisiert werden, müssen sie in der Datenbank gespeichert und direkt einem öffentlichen Schlüssel und den geschützten Daten zuordenbar sein. Da die Daten allerdings für mehr als einen Nutzer zugänglich gemacht werden können, existieren auch mehrere Verschlüsselungen des symmetrischen Schlüssels auf dem Server. Deshalb ist die Beziehung der Schlüssel und der Daten 1:N. Der symmetrischer Schlüssel kann auch nur mit einem öffentlichen

Schlüssel verknüpft sein, somit ist diese Beziehung ebenfalls 1:N. Der Schlüssel besteht lediglich aus den Schlüsseldaten, in Form eines Base64-Strings.

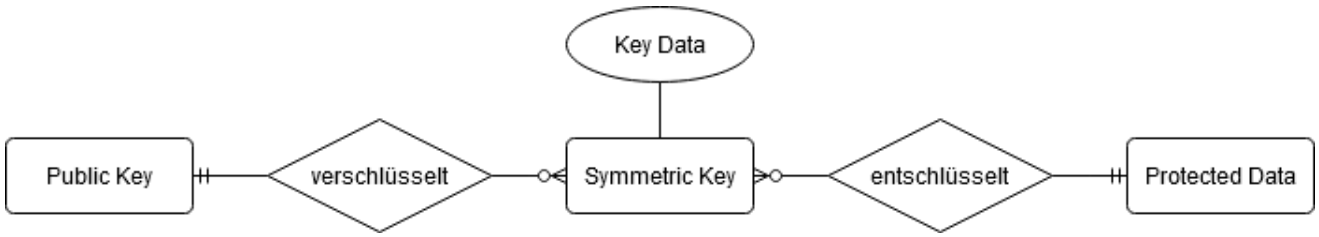


Abbildung 3.7: Öffentliche & Symmetrische Schlüssel mit Geschützten Daten Beziehungen

3.1.1.5 Ordner

Da es KeyS dem Benutzer außerdem ermöglicht, um für einen besseren Überblick zu sorgen, seine Daten in Ordnerstrukturen abzulegen müssen die einzelnen Ordner ebenfalls gespeichert werden. In einem Ordner können sich zu jeder Zeit mehrere Unterordner befinden, so wie es auch von modernen Betriebssystemen bekannt ist. Die Ordner benötigen eine Eigenschaft um diese Hierarchie abzubilden. Überordner werden in diesem Kontext als Parents und die Unterordner als Children bezeichnet. In einem ersten Ansatz scheint es eine gute Idee zu sein, dass jeder Parent weiß, wer seine Children sind, d.h der Überordner weiß über seine Unterordner bescheid. Ein Problem entsteht allerdings sobald zwei Parents das selbe Child referenzieren und somit der selbe Ordner an zwei Orten liegen könnte. Dieses Verhalten ist unerwünscht und soll vermieden werden. Stattdessen referenzieren die Children ihren Parent, so kann kein Ordner an zwei verschiedenen Orten auftauchen, da nur bekannt ist in welchem Überordner er selbst liegt. An der höchste der Stelle der Hierarchie befindet sich der *root* Ordner, dieser besitzt keinen Parent. Dieser Aufbau erleichtert außerdem das Löschen von Ordnern und Dateien, da einfach alle mit Referenzen auf den gelöschten Ordner ebenfalls entfernt werden. Die Beziehung der Ordner ist somit 1:0..N. Ähnlich den geschützten Daten erhalten Ordner eine ID und einen Namen zur einfachen Identifikation.

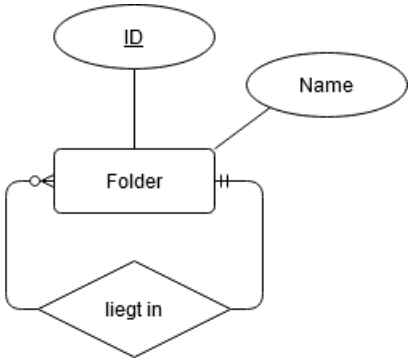


Abbildung 3.8: Folder Entität mit Beziehung zu sich selbst

Die Daten sind ebenfalls Unterelemente eines Ordners und folgen der gleichen Beziehung.

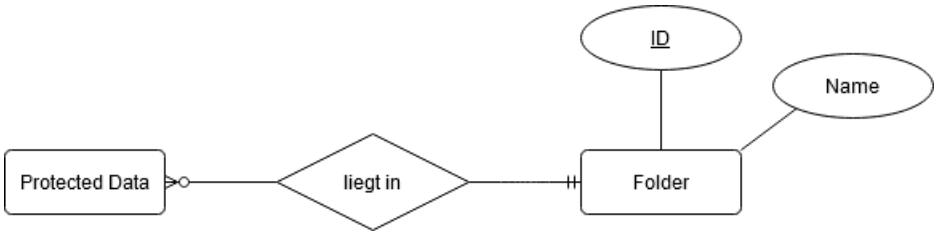


Abbildung 3.9: Ordner & Geschützte Daten Beziehung

3.2 Implementierung

Um die Kommunikation mit der SQLite Datenbank zu ermöglichen, bedarf es einer entsprechenden Bibliothek. Für diese Implementation wurde die *Microsoft.Data.Sqlite* Bibliothek verwendet, da sie der ADO.NET Architektur folgt.[2][3] ADO.NET ermöglicht es Entwicklern einfach und einheitlich mit Datenquellen umzugehen und bietet ein umfangreiches Gerüst an Klassen und Funktion für die Manipulation der Daten. Über die Bibliothek kann mittels einer Singleton-Factory ein Objekt erstellt werden, um eine Verbindung zur SQLite Datenbank zu öffnen.

```
DbConnection con = SqliteFactory.Instance.CreateConnection();
con.ConnectionString = "DataSource=keys.db;Cache=Shared";
con.Open();
// ...
con.Close();
```

Listing 3.1: Öffnen der Datenbank Verbindung über Microsoft.Data.Sqlite & ADO.NET Klassen

Für die Verbindung ist ein *ConnectionString* notwendig. Der *ConnectionString* gibt an, wie sich das Programm zur Datenbank verbinden soll. Er besteht aus einer, durch Semikolon getrennten, Liste mit verschiedenen Parametern. Der Parameter *DataSource* enthält den Pfad zur lokalen .db-Datei. Optional kann der Zugriff auch auf Leseoperationen beschränkt werden. Die möglichen Einstellungen sind in der offiziellen Dokumentation von Microsoft festgehalten.[4]

[2] *Microsoft.Data.Sqlite*. Microsoft. URL: https://docs.microsoft.com/en-us/dotnet/standard/data/sqlite (besucht am 30. 03. 2020).

[3] *ADO.NET*. Microsoft. URL: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet (besucht am 30. 03. 2020).

[4] *Microsoft.Data.Sqlite*.

Über die Verbindungen können nun SQL Befehle ausgeführt werden. ADO.NET bietet für diesen Vorgang eine abstrakte *DbCommand* Klasse an.[5]

```
DbCommand command = con.CreateCommand();
command.CommandText = "SELECT * FROM Users";
IDataReader reader = command.ExecuteReader();
```

Listing 3.2: Ausführen eines SQL Befehls über DbCommand

Falls die Query erfolgreich war, ist das Ergebnis über das Objekt *IDataReader* verfügbar. *IDataReader* bietet eine *Read* Methode an, über die eine Zeile des Ergebnisses eingelesen wird und abgerufen werden kann.

3.2.1 Repository/Unit-of-Work Pattern

Für die Abstraktion über die Persistenz Schicht verwendet KeyS ein Repository Design Pattern in Kombination mit dem Unit-of-Work Pattern. Das Repository bietet einen einheitlichen Zugriff auf eine Datenquelle und kapselt die Logik, um die Daten zu bearbeiten. Entsprechend dem Pattern entsteht ein eigenes Repository, in Form einer Klasse, für jedes Datenkonzept. Ein Datenkonzept ist in dieser Anwendungen beispielsweise der Benutzer, oder die geschützten Daten. Wichtig ist, dass nicht für jede Tabelle ein Repository erstellt werden muss, da dies die Anwendung wieder an das Schema koppeln würde, sondern nur für abstraktere Konzepte. In Abbildung 3.10 bildet das jeweilige Repository eine Schnittstelle zwischen der Anwendung und der Datenbank für dessen Datenkonzept.

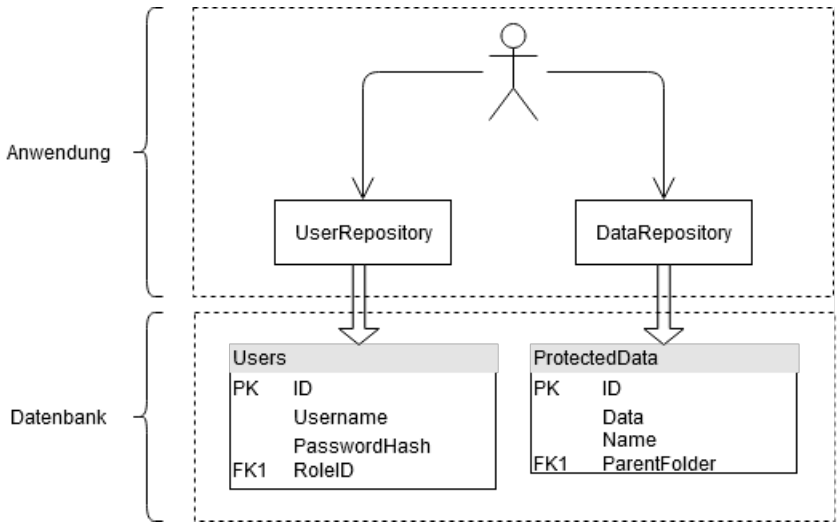


Abbildung 3.10: Repository Pattern

[5] *.NET API Browser*. Microsoft. URL: https://docs.microsoft.com/en-us/dotnet/api/ (besucht am 01.04.2020).

Die Erweiterung der Repositorys um das Unit-of-Work Pattern dient der Trennung von Read- und Write-Operationen. Während ein Repository ausschließlich die Datenbank abfragen kann, bewirkt die Unit-of-Work eine Änderung der Daten. Die Unit-of-Work kapselt eine Transaktion (Insert, Update oder Delete) in Form einer Klasse. Sobald eine Transaktion vollständig ist, können über einen Aufruf der *Complete* Methode alle Änderungen in die Datenbank übernommen werden. Sollte währenddessen ein Fehler passieren, führt die Unit-of-Work ein Rollback durch, d.h. die Daten bleiben unverändert. Die gewünschten Änderungen werden in dem *Entity* Property (Abb. 3.11) der Klasse vorgenommen.

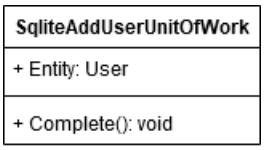


Abbildung 3.11: Unit-of-Work zum Hinzufügen eines neuen Users

Um zu verhindern, dass die Anwendung von den konkreten SQLite Implementation abhängt sind, gemäß dem Dependency-Inversion-Prinzip, Interfaces definiert, die von den High Level Modulen verwendet werden. Gemeinsame Operation sind in einem generischen *IRepository* Interface zusammengefasst und durch davon erbende Klassen um Kontext-relevante Funktionen erweitert und genauer spezifiziert. Eine ähnliche Hierarchie liegt den Unit-of-Work Klassen zugrunde.

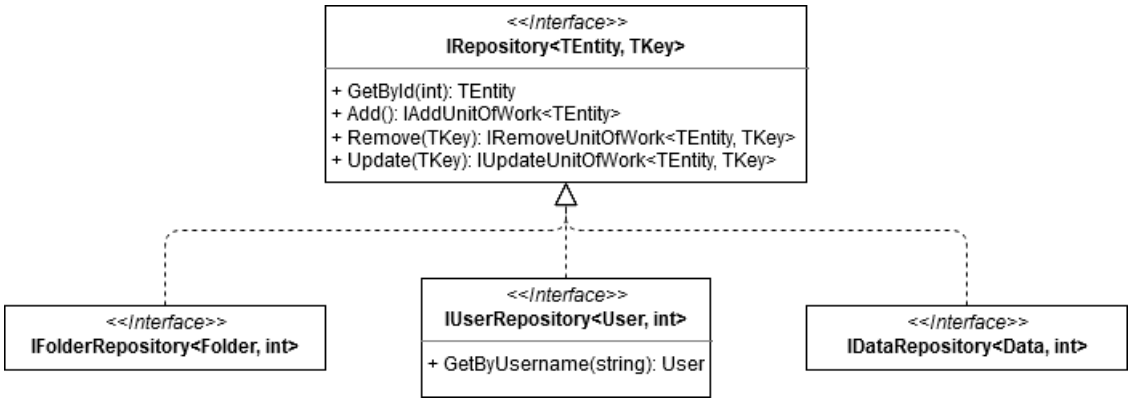


Abbildung 3.12: Generisches IRepository mit spezifischen Unterklassen

3.2.1.1 Automatische Ressourcenfreigabe

Da für jede Transaktion mit der Datenbank eine Verbindung geöffnet und somit Ressourcen belegt werden, muss diese auch wieder geschlossen werden (siehe Listing 3.1). Es besteht die Gefahr, dass auf das Schließen der Verbindung vergessen wird. Um dem entgegen zu wirken und mögliche Fehler zu vermeiden, automatisiert man die Ressourcenfreigabe. Dafür wird die Ressourcenbelegung und Freigabe direkt

an die Initialisierung gebunden. Eine abstrakte Klasse *SqliteUnitOfWork* implementiert die *Complete* Methode des Interfaces *IUnitOfWork*, welches durch die High Level Module definiert ist. Außerdem definiert die Klasse eine eigene abstrakte *Complete* Methode. In der *Complete* Methode des Interfaces wird nun eine Verbindung zur Datenbank geöffnet, der Aufruf an die eigene abstrakte *Complete* Methode weitergeleitet und anschließend die Verbindung wieder geschlossen. Somit ist garantiert, dass nach der Ausführung der eigene *Complete* Methode die Ressourcen automatisch wieder freigegeben werden.

```
// SqliteUnitOfWork
void IUnitOfWork.Complete()
{
        DbConnection con = SqliteFactory.Instance.CreateConnection();
        con.ConnectionString = _connectionString;
        con.Open();
        this.Complete(con);
        con.Close();
}
protected abstract void Complete(DbConnection con);
```

Listing 3.3: Complete Methoden der SqliteUnitOfWork Klasse

Die unterschiedlichen SQLite Implementation der Unit-of-Work Klassen können nun von *SqliteUnitofWork* erben und müssen nur die abstrakte Methode implementieren, ohne dabei selbst die Ressourcen verwalten zu müssen.

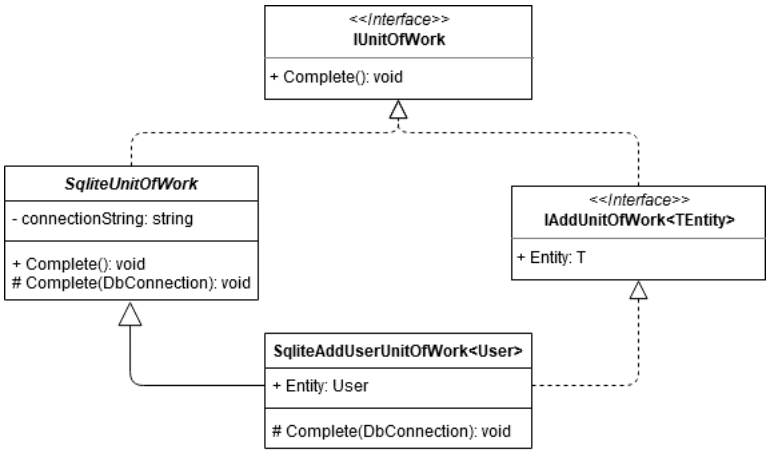


Abbildung 3.13: Hierarchie der SqliteAddUserUnitOfWork Klasse

3.2.2 Testen der Persistenz Schicht

Um sicherzustellen, dass die Persistenz Schicht funktioniert werden parallel zur Entwicklung Tests verfasst. Das Erstellen eines isolierten Komponententests ist nicht möglich, da eine Datenbank benötigt wird und damit auch die entsprechenden Bibliotheken. Außerdem müsste speziell bei SQLite eine lokale Datei erstellt werden, wodurch die Verfügbarkeit eines Speichermediums ebenfalls erforderlich wäre.

SQLite bietet allerdings die Möglichkeit eine In-Memory Datenbank, d.h. im Arbeitsspeicher, zu erstellen. Dadurch fällt die Anforderung eines Speichermediums weg und der Test ist davon entkoppelt. Um eine Datenbank In-Memory zu erstellen, muss dem Connection String das *Mode=Memory* Schlüsselwort hinzugefügt werden. So wird die Datenbank bei der Öffnung der Verbindung angelegt und bei der Schließung wieder gelöscht. Das bedeutet allerdings auch, dass die Verbindung während dem Test dauerhaft geöffnet sein muss, da sonst die Daten wieder freigegeben werden. Das MSTest-Framework bietet eine Funktion an, um Operation vor und nach einem Test auszuführen.

```
[TestInitialize]
public void Init()
{
    string constring = $"DataSource={_testContext.TestName};Mode=Memory;";
    memoryDbConnection = SqliteFactory.Instance.CreateConnection();
    memoryDbConnection.ConnectionString = constring;
    memoryDbConnection.Open();
}

[TestCleanup]
public void Cleanup()
{
    memoryDbConnection.Close();
}
```

Listing 3.4: Test Cleanup und Test Initialize Methoden

Indem die zu testende Klasse den gleichen Connection String verwendet, werden die Ressourcen nicht frühzeitig freigegeben und es kann leicht gegen die selbe In-Memory Datenbank getestet werden. Um mehrere Tests parallel auszuführen braucht es eine eindeutige *DataSource*. Der Name kann beliebig sein, beispielsweise der Testname (siehe Listing 3.4).

4 Webanwendung

Damit der Benutzer mit dem Server arbeiten kann gibt es in KeyS eine Webanwendung, über die öffentliche Schlüssel und gespeicherte Daten verwaltet werden können. Zu dieser gibt es eine RESTful API, über die der Datenaustausch zwischen Webanwendung und Client stattfindet.
Webanwendungen werden, anders als Desktopanwendungen, auf einem Webserver installiert. Die Anwendung ist anhand eines Client-Server-Modell aufgebaut, wobei die Datenverarbeitung auf dem Server stattfindet und diese Daten über das HTTP-Protokoll an den Client, also den Benutzer, welcher mit der Anwendung über einen Browser arbeitet, gesendet werden.
Aus diesen Eigenschaften ergeben sich gewisse Vorteile: Webanwendungen sind Plattformübergreifend.

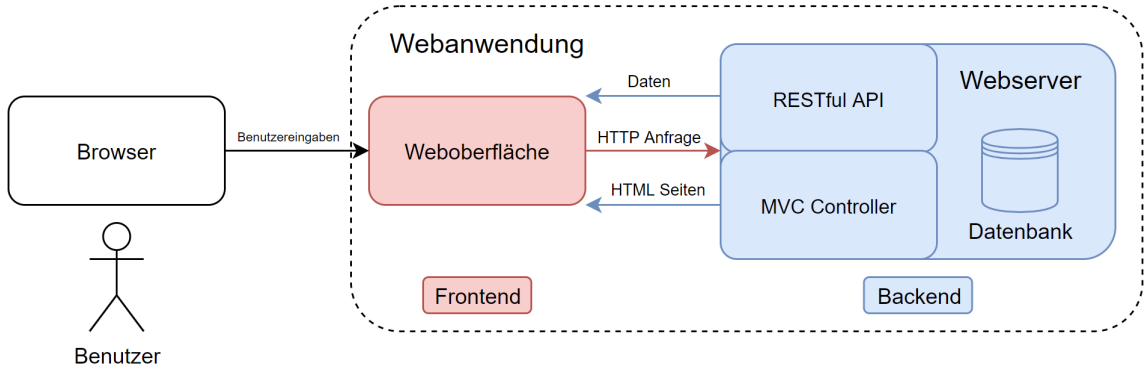


Abbildung 4.1: Aufbau einer Webanwendung

Die Daten werden auf dem Webserver gespeichert, wodurch ein Benutzer von verschiedenen Geräten aus, mit seinen Daten in der Anwendung arbeiten kann.
Ein Nachteil der dadurch entsteht, dass die Anwendung nicht wie bei Desktopanwendungen durch dass Betriebssystem ausgeführt wird, ist dass nur beschränkt auf Systemressourcen wie CPU oder Arbeitsspeicher zugegriffen werden kann. Dadurch lassen sich leistungsbelastende Programme nur schwer als Webanwendung umsetzen. Da dies für KeyS kein Problem darstellt, wird diese Art der Entwicklung verwendet.[1]

[1] *Webapplikationen und Webanwendungen*. oneclick. 15. Jan. 2019. URL: `https://oneclick-cloud.com/de/blog/trends/webapplikationen-webanwendungen/` (besucht am 31. 03. 2020).

4.1 Backend

Zum Backend zählt alles, mit dem der Benutzer nicht in Kontakt kommt. Dazu zählt in erster Linie die Implementation Logik, mit der der Ablauf einer Anwendung geregelt wird, sowie die Datenverwaltung, in Form von Kommunikation mit der Datenbank.

4.1.1 Webframeworks

In der Entwicklung von Webanwendungen wird in den meisten Fällen auf die Unterstützung eines serverseitigen Webframeworks zurückgegriffen. Diese stellen Bibliotheken und Werkzeuge bereit, mit denen Funktionalitäten, wie URL-Routing, Interaktionen mit Datenbanken, Autorisierung oder Caching umgesetzt werden können. Einige der am aktuell meistverwendetsten Webframeworks sind:[2]

- React – eine modulare Grundlage für wiederverwendbare Oberflächenkomponenten basierend auf JavaScript.
- Angular – ein umfangreiches Framework, ausgerichtet auf Typescript, mit Unterstützung für Javascript und Dart.
- Django – ein auf Python basiertes Webframework, mit dem viele Funktionalitäten aufgrund von modularem Design und Verfügbarkeit von Komponenten einfach umgesetzt werden können.
- Spring – ein Framework für Java und dadurch oft in Backend Webanwendungen verwendet.
- ASP.NET – ein Framework dass für jede .NET Sprache entwickelt wurde. Durch die Verwendung von kompilierten Sprachen ist es schnell und gut skalierbar.

[2] Ryan Donovan. *The Top 10 Frameworks and What Tech Recruiters Need to Know About Them*. Stack Overflow. 17. Dez. 2019. URL: `https://stackoverflow.blog/2019/12/17/the-top-10-frameworks-and-what-tech-recruiters-need-to-know-about-them/` (besucht am 31. 03. 2020).

4.1.1.1 Entscheidungskriterien für Webframeworks

Die große Menge an verfügbaren Webframeworks für diverse Programmiersprachen erschwert die Entscheidung für das richtige Framework. Eine gute Entscheidung wirkt sich fördernd für den Verlauf eines Projektes aus und steigert die Produktivität der Entwickler. Die Wahl sollte anhand verschiedener Faktoren, abhängig von der Art des Projektes und der Erfahrungen der Entwickler, getroffen werden.

Aufwand Je nach Erfahrungen und Wissensstand der Entwickler bringt jedes neue Framework potentiell viel Aufwand mit, bevor produktiv damit gearbeitet werden kann. Dazu gehört in erster Linie die Beherrschung der bereitgestellten Funktionen und Prinzipien. Es werden Frameworks verwenden, die bereits erlernte Programmiersprachen unterstützen, da der Aufwand einer neuen Sprache potentielle Vorteile, hinsichtlich Produktivität, überwiegt.
Durch eine hochwertige Dokumentation kann der auftretende Lernaufwand reduziert werden.

Produktivität Die Steigerung in Produktivität muss den entstandenen Aufwand überwiegen, um eine getroffene Entscheidung zu rechtfertigen. Produktivität bezieht sich hierbei auf die Effizienz, mit der neue Funktionen entwickelt und bereits bestehende gewartet werden können.
Viele Frameworks wurden gezielt für gewisse Aufgaben und Anwendungsbereiche entwickelt und eignen sich besser für diese als andere.

4.1.1.2 ASP.NET Core

ASP.NET Core ist ein Open Source Framework zur Entwicklung von Web Anwendungen, basierend auf der .NET Core Plattform. Mit dem ASP.NET Core Framework werden Projektvorlagen mitgeliefert, wie beispielsweise eine MVC Vorlage. Diese Vorlage bietet die Möglichkeit Anwendungen einfach und übersichtlich anhand des MVC Design Patterns aufzubauen, wodurch sie leichter zu testen und zu warten sind. ASP.NET Core enthält noch weitere integrierte Features zur Entwicklung von Anwendungen wie zum Beispiel die Folgenden.[3]

Routing Damit der Client die spezifische Methoden der Webanwendung aufrufen kann, müssen an die Server URL zugehörige Endpunkte angehängt werden. Über diese zusammengefügte URL kann der Client gezielt Anfragen stellen.

```
[Route("/api/[controller]")]
[ApiController]
public class KeyController : ControllerBase
{
        [HttpPost("add")]
        public IActionResult AddNewKey([FromBody]PublicKeyModel model)
        {
        }
}
```

Listing 4.1: Routing des KeyControllers und der AddNewKey Methode

In Listing 4.1 wird der KeyController durch den *Route* Tag der URL */api/key* zugeordnet. Die Methode *AddNewKey* erhält durch den *HttpPost* Tag den URL Anhang */add*. Die Methode kann über die zusammengesetzte URL */api/key/add*.

Web APIs Um die Entwicklung von Web APIs zu vereinfachen enthält ASP.NET Core integrierte Unterstützung für typische Datenformate von HTTP Requests wie JSON oder XML. Dadurch lassen sich Umformatierungen der Ausgangsdaten vermeiden, die dann aus diesem Format in ein Model umgewandelt werden können.

```
data:   {
                KeyData: data,
                KeyNumber: 0,
                Active: "true"
        }
```

Listing 4.2: data Attribut eines API Aufrufes mit Ajax

[3] Steve Smith. *Overview of ASP.NET Core MVC*. Microsoft. 12. Feb. 2020. URL: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-3.1 (besucht am 26. 03. 2020).

In Listing 4.3 können die Daten aus der Anfrage von Listing 4.2 in einem Controller Endpunkt ohne weitere Umformatierungen zu einem *Key* Objekt gefügt werden.

```
public IActionResult AddNewKey([FromBody]PublicKeyModel model)
{
        Key key = new Key()
        {
                Flag = Key.KeyFlag.ACTIVE,
                KeyData = Encoding.UTF8.GetBytes(model.KeyData)
        };
}
```

Listing 4.3: Ausschnitt des AddNewKey Endpoints

Razor View Engine Mit Razor können dynamisch Inhalte in Views mit C# Logik generiert werden. Dabei wird in cshtml Dateien HTML gemeinsam mit C# verwendet.

```
<tbody>
        @for (int i = 0; i < Model.PublicKeys.Count; i++)
        {
                <tr>
                <td>@i</td>
                <td>@Model.PublicKeys[i].KeyData</td>
                <td>@Model.PublicKeys[i].Flag</td>
                <td> ... </td>
                </tr>
        }
</tbody>
```

Listing 4.4: Tabelle mit Razor

In Listing 4.4 wird dynamisch eine Tabelle generiert, die ein Model, das eine Liste von Key-Objekten enthält, mithilfe einer for-Schleife unabhängig ihrer Größe vollständig darstellt.
Da das ganze Team Erfahrungen mit C# oder ähnlichen Programmiersprachen hat und Asp .NET Core alle benötigten Funktionalitäten enthält, wird es zur Entwicklung der Anwendung verwendet.

4.1.2 MVC Design Pattern

Mit dem Model-View-Controller Design Pattern werden Anwendungen übersichtlich auf drei Komponenten aufgeteilt. Die drei namensgebenden Komponenten sind:

Model Repräsentiert die zentrale, dynamische Datenstruktur einer Anwendung und damit direkt ihren derzeitigen Zustand.

View Enthält jegliche Grafische Darstellung von Informationen, bezogen von den Daten des Models.

Controller Verarbeitet jegliche Art von Benutzereingaben und führt dementsprechende Aktionen am Model aus.

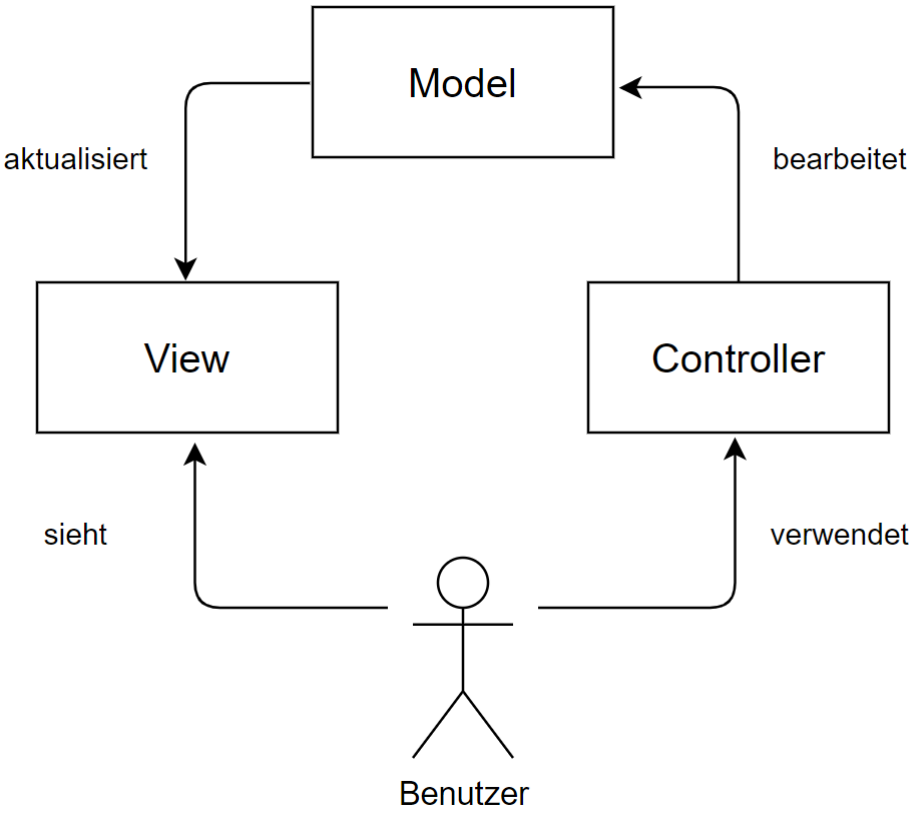


Abbildung 4.2: Beziehungen der MVC Komponenten

MVC ist ein architektonisches Design Pattern und ermöglicht weitere Abstraktion in Anwendungen. Da jede Komponente auf seine wesentlichen Aufgaben beschränkt wird: Model auf Datenstrukturen, View auf Formatierung und Darstellung der Daten und Controller auf Anwendungslogik und Benutzereingaben, kann bei auftretenden Problemen schnell auf die richtige Ursache von Fehlern getestet, sowie an einer Lösung dieser Probleme gearbeitet werden. Die zusätzliche Abstraktion fördert die Produktivität der Entwicklung, da für neue Funktionen bestehende Programmteile wiederverwertet, und zu wartende Funktionen, vom Rest der Anwendung getrennt, leicht ausgetauscht werden können. MVC ermöglicht dass parallele Arbeiten eines ganzen Teams, da ein Backend Team an den Models und den Controllern arbeiteten kann, während das Frontend Team an den Views arbeitet.[4]

4.1.2.1 Model

Models repräsentieren anwendungsbezogene Daten und spiegeln damit den aktuellen Zustand einer Anwendung wieder. In Form von Model Objekten kann der aktuelle Zustand eines Models in einer Persistenz abgelegt oder wieder herausgeholt werden. Solche Model Klassen sind oft sehr simpel anhand des DTO Patterns aufgebaut.

DTO Pattern - Data Transfer Object Ein Pattern mit der Absicht Kommunikation zwischen Prozessen zu vereinfachen. Würde man in einer Webanwendung verschiedene Daten, wie einen Benutzernamen, ein Passwort oder eine Adresse, an den Server senden wollen, müsste man für jede Information eine eigene Anfrage senden. Um dies zu vermeiden werden die Daten zu einem Benutzer Objekt zusammengefügt. Dieses Objekt wird für den Datenaustausch zusammengehörender Daten genutzt, und kann über eine Anfrage alle Daten übermitteln. DTO Klassen limitieren ihr eingebautes Verhalten auf Getter und Setter, sowie optional Parser und Serializer, enthalten also keine Anwendungslogik.[5]

[4]Florian Siebler. *Design Patterns mit Java*. 2014. ISBN: 978-3-446-43616-9, S. 48.
[5]Robert C. Martin. *Clean Code*. Robert C. Martin Series. 2008. ISBN: 978-0-13-235088-4.

4.1.2.2 Controller

Controller sind für den Ablauf einer Anwendung zuständig. Wenn der Benutzer eine bestimmte Seite anfragt, ist der Controller dafür verantwortlich, die richtigen Daten, bei normalen Controllern eine View, zurückzuliefern. Zusätzlich regeln Controller auch die Anwendungslogik. Wenn der Benutzer Anfragen zum Eingeben, Bearbeiten oder Löschen von Daten stellt, werden diese vom Controller erhalten und mithilfe des Models die Persistenz aktualisiert. Anschließend liefert er das aktualisierte Model wieder zurück zur View.

```
public class UserController : Controller
{
        private readonly IUserRepository _userRepository;

        public UserController(IRepositoryManager repositoryManager)
        {
                _userRepository = repositoryManager.UserRepository;
        }
}
```

Listing 4.5: Quellcode Ausschnitt des UserControllers

In Listing 4.5 ist der typischen Aufbau eines einfachen Controllers zu sehen. In ASP.NET Core müssen sie immer die Endung *Controller* haben. Der Vorsatz bestimmt dann den URL Anteil, über den ein Controller aufgerufen werden kann. In diesem Beispiel */user*.

```
public IActionResult Index()
{
        User user = _userRepository.GetUserByName(User.Identity.Name);
        return View(user);
}
```

Listing 4.6: Index Methode des UserControllers

Um die verschiedenen Methoden aufzurufen, welche auch als Action Methoden bezeichnet werden, muss der Name des Endpunkts an die URL angehängt werden. Um in Listing 4.6 die Methode *Index* im *UserController* aufzurufen ergibt sich die URL */user/index*.
In der Action Methode *Index* wird ein Objekt der Model Klasse *User* mithilfe des richtigen Benutzers aus der Datenbank angelegt, welcher anhand des Benutzernamens des eingeloggten Benutzers gesucht und zurückgeliefert wird.
Anschließend sendet der Controller die View, die gleich der Action Methode benannt wird, sowie dass Model Objekt *user*, dass als Parameter angeführt wird, zurück.

API Controller Der Unterschied zu normalen Controllern in ASP.NET Core ist, dass API Controller Daten und keine Views zurückgeben. Action Methoden eines API Controllers werden auch als API Endpunkte bezeichnet.
Sie verwenden auch ein anderes Routing Schema, womit sie eine RESTful API bereitstellen.

4.1.3 Application Programming Interface

Ein Application Programming Interface, kurz API, ist eine Schnittstelle eines Programms, einer Bibliothek oder eines Internet Services, um einem externem Programm Zugriff auf bestimmte Funktionalitäten zu ermöglichen.
Durch eine API können tiefgängige Implementationen abstrahiert werden und der Entwickler erhält nur Informationen über Objekte und Aktionen die er tatsächlich benötigt. Dadurch muss der Entwickler sich nicht selbst damit befassen, und kann effizienter arbeiten.[6]

4.1.3.1 Betriebssysteme

APIs die auf diesem Level arbeiten, werden dafür verwendet, um Anwendungen bei der Kommunikation mit unterliegenden Schichten zu unterstützen. Dies funktioniert durch das realisieren verschiedener Protokolle und Spezifikationen wie z.B.: POSIX, kurz für Portable Operating System Interface.

4.1.3.2 Bibliotheken

Die API spezifiziert das zu erwartende Verhalten während die Bibliothek dieses implementiert. Aus dieser Trennung zwischen API und Implementation erfolgen mehrere Vorteile, wie die Verwendung von Bibliotheken in anderen Programmiersprachen.

4.1.3.3 Web API

Eine serverseitige Web API ist eine Schnittstelle, bestehend aus mehreren, öffentlich zugänglichen Endpunkten, welche über das Web, typisch in Form eines HTTP basiertem Web Servers, zugänglich ist. Hierbei wird mit einem definiertem Anfrage-Antwort System gearbeitet, bei dem Daten im JSON oder XML Format übertragen werden.

[6] Jonathan Freeman. *What is an API? Application programming interfaces explained*. InfoWorld. 8. Aug. 2019. URL: `https://www.infoworld.com/article/3269878/what-is-an-api-application-programming-interfaces-explained.html` (besucht am 31. 03. 2020).

Enpunkte Endpunkte bilden die Ansprechstellen einer Web API. In Endpunkten werden die Ressourcen definiert, auf die außenstehende Zugreifen können. Sie sind über eine Anfrage an eine bestimmte URL erreichbar, über die auch eine Antwort zurück kommt. Dabei ist es wichtig dass diese statisch sind, also nicht verändert werden. Ist dies nicht der Fall, ist die Funktionalität von Webanwendungen, die diese APIs verwenden, nicht mehr garantiert. Um Endpunkte trotzdem zu überarbeiten werden Versionsnummern in URLs eingebaut.

4.1.4 RESTful API

Representational state transfer, kurz REST, ist eine Art der Softwarearchitektur, die gewisse Regeln zur Entwicklung von Web Services definiert. Web Services, die sich an die architektonischen Richtlinien von REST halten, ermöglichen die Zusammenarbeit mehrerer Systeme über das Internet, in Hinsicht auf den Austausch und das Arbeiten mit Web Ressourcen.
Der Begriff Web Ressourcen bezieht sich hierbei auf Alles, das identifiziert, benannt, adressiert oder in jeglicher Form im Web behandelt werden kann. Bei einem RESTful Web Service wird über eine Anfrage auf solche Ressourcen, in textueller Form, zugegriffen oder diese bearbeitet. Der Service liefert dann je nach Anfrage eine bestimmte Antwort zurück.[7][][S.76-92]

[7] Roy Thomas Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. Doctoral dissertation. University of California, Irvine, 2000.

4.1.4.1 Richtlinien

REST definiert folgende Richtlinien, die den Server dabei einschränken, wie er mit Anfragen des Clients umgehen soll, so dass dieser bestimmte Eigenschaften wie Performance, Skalierbarkeit, Sichtbarkeit und Portabilität erhält.

Client-Server Architektur Diese Richtlinie basiert auf dem Design Prinzip Seperation of Concerns, oder auch Single Responsability, genauer beschrieben in Kapitel 2.1.1. Dabei wird die Benutzeroberfläche von der Datenablagerung getrennt. Durch die daraus folgende Vereinfachung des Servers wird die Skalierbarkeit und die Portabilität gefördert.

Zustandslosigkeit Die Client-Server Kommunikation wird so eingeschränkt, dass der Server zwischen verschiedenen Anfragen keine Kontextinformationen zum Client speichert. Jede neue Anfrage des Clients soll ausreichende Informationen über die Sitzung enthalten, damit der Server die Anfrage richtig verarbeiten kann. Der Sitzungszustand befindet sich also beim Client. Zur Authentifizierung oder anderen Funktionen kann dieser allerdings auch vom Server an einen anderen Service, wie einer Datenbank, weitergegeben werden.

Mehrstufige Systeme Der Client kann nicht feststellen ob er direkt mit dem Endserver verbunden ist oder ob in der Verbindung noch weitere, vermittelnde Server zwischengeschaltet sind. Dadurch lassen sich Systeme größer skalieren und andere Service, wie ein Sicherheitssystem, können eigenständig implementiert werden.

Einheitliche Schnittstellen Diese Bedingung ist elementar beim Design eines RESTful Services. Die Architektur wird anhand mehrerer Prinzipien entworfen.
Ressourcen müssen in Anfragen identifizierbar sein, dies kann durch URIs umgesetzt werden, da der Server nicht die Ressourcen selbst sondern XML oder JSON Repräsentationen an den Client vermittelt. Der Client erhält mit diesen Repräsentationen genug Informationen um diese Ressourcen zu bearbeiten beziehungsweise sie zu löschen.
Es muss immer genug Information enthalten sein, um Nachrichten richtig zu bearbeiten. Wird zum Beispiel ein Bild im PNG Format mitgegeben muss dies durch den richtigen Medien Typen *image/png* gekennzeichnet werden.

4.1.4.2 HTTP Methoden

HTTP definiert eine Auswahl an Methoden, mit denen festgestellt werden soll, welchen Aktionen eine Anfrage durchführen soll.

GET Mit dieser Methode wird eine Repräsentation einer Ressource angefordert. Solche Anfragen sollen ausschließlich Daten erhalten.

POST Zum erstellen einer Ressource aus den Daten die der Anfrage mitgegeben werden. Die URI der erstellten Ressource wird mit der Antwort zurückgegeben.

PUT Methode um die Ressource zu ersetzen oder eine neue zu erstellen, falls diese noch nicht existiert.

PATCH Methode um die Ressource zu aktualisieren oder zur Erstellung einer neuen, falls diese noch nicht existiert.

DELETE Methode um die Ressource zu löschen.

4.2 Frontend

Frontend bezieht sich auf den Systemanteil, mit dem der Benutzer interagiert. Dazu zählen einerseits die Ansichten und Benutzeroberflächen mit denen der Benutzer direkt in Kontakt kommt, andererseits auch die Verarbeitung von Benutzereingaben, beispielsweise über API Aufrufe.

4.2.1 API Aufrufe

Wenn bei einer Anfrage an den Server die richtige URL mit einem Endpunkt anhängt, ruft man damit die zugehörige API auf, es wird also ein API Call getätigt.
API Calls werden getätigt, um auf bestimmte Ressourcen der API zuzugreifen oder diese zu bearbeiten. Die Art der Anfrage wird durch die verwendete HTTP Methode festgelegt. Die zu übertragenden Daten werden im Körper der Anfrage mitgesendet und die zu erhaltenden im Körper der Antwort.

4.2.1.1 Ajax

Mithilfe von Ajax, kurz für Asynchronous Javascript and XML, können Web Anwendungen asynchron Daten mit dem Server austauschen. Es können also asynchron API Calls im Hintergrund ausgeführt werden, ohne das Verhalten und damit auch die Darstellung einer Webseite zu stören. Es ist keine eigene Technologie sondern eine Kombination aus diversen anderen Technologien. Die wichtigsten sind folgende:

- XML und Json zum Datenaustausch
- XMLHttpRequest eine API zur asynchronen Kommunikation
- JavaScript zur Kombination dieser Technologien

Für den Fall dass ein Web Browser diese Technologien nicht unterstützt, können Web Anwendungen und Web Seiten die Ajax verwenden nicht korrekt verwendet werden.
Obwohl XML im Namen enthalten ist, wird fast ausschließlich JSON zur Repräsentation von Daten verwendet.

4.2.1.2 Implementation

Über die Bibliothek jQuery können Ajax API Calls simpel und einfach implementiert werden.

```
$.ajax({
        type: "POST",
        url: "/api/Key/add",
        data: JSON.stringify({
                KeyData: keyData,
                KeyNumber: "-1",
                Active: "true"
        }),
        contentType: 'application/json; charset=utf-8',
        success: function () {
                closeAddKeyPopup();
                location.reload();
        }
});
```

Listing 4.7: API Call mit Ajax

Dies erfolgt über die Funktion *ajax*, der wichtige Informationen, wie die HTTP Methode, die URL mit dem anzusprechendem Endpunkt, die zu sendenden Daten in Form von Schlüssel-Wert-Paaren im JSON Format, den Medien Typen und optional noch *success* oder *error* Funktionen, die gegebenenfalls ausgeführt werden. Auf die selbe Art wie in diesem Beispiel eine POST Anfrage implementiert wurde, können alle anderen HTTP Methoden realisiert werden. Der Aufbau der Funktion muss je nach Methode dementsprechend angepasst werden. Bei einer GET Methode würden keine Daten mitgegeben, sondern die angeforderten Daten über die *success* Funktion ausgelesen werden.

4.2.2 Markup Sprache Razor

Razor ist eine Markup Sprache, die es ermöglicht, C# oder VB.Net Quellcode in Web Seiten einzubauen. ASP.NET Core stellt eine View engine zur Verfügung die es ermöglicht, Razor in Webanwendungen zu verwenden und damit HTML Seiten zu generieren.
Razor eignet sich gut zur Verwendung in MVC Views. Dazu wird in Kombination mit HTML Komponenten eine Vorlage von Strukturen zusammengesetzt, welche mit den Daten des Models befüllt wird. Aus dieser wird dann eine dynamisch generierte View erzeugt.

Das Arbeiten mit Models wird durch die verschiedenen Werkzeuge, die Razor in Form von C# Logik mitbringt, stark vereinfacht und ermöglicht verschiedene Wege, mit simplen Programmiergrundlagen, wie Verzweigungen oder Schleifen, dynamisch Inhalte abhängig vom momentanen Zustand des Models zu erstellen.

```
@{
        int number = 1;
        string message = "Die Zahl ist: " + number;
        <p>@message</p>
}

<p>@message</p>
```

Listing 4.8: Verwendung von Variablen in Razor

Um Quellcode zu schreiben, der von Razor erkannt werden kann, muss davor ein @ gesetzt werden. Größere Codeblöcke müssen in geschwungene Klammern gehüllt werden. Wie in Listing 4.8 dargestellt, kann man, wie von C# gewohnt, Variablen über mehrere Zeilen verwenden und diese auch in HTML Tags verwenden. Dafür kann sich der Tag außerhalb des Code Blocks befinden. Dazu muss ein @ vorgesetzt werden, auch wenn sich der Tag schon in einem Code Block befindet

```
@{
        int c = Model.a + Model.b
}

<p>@Model.a + @Model.b = @c</p>
```

Listing 4.9: Verwendung von Model Objekten

Mit Model Objekten kann in Razor Views wie mit normalen Objekte gearbeitet werden. In Listing 4.9 erhält liefert der Controller ein Model mit den Attribute *a* und *b* aus denen die Variable *c* berechnet wird. Wenn das Model beispielsweise die Werte *a = 1* und *b = 2* hätte, würde der abschließende *paragraph* den Text *1 + 2 = 3* enthalten.

Verzweigungen Logische Verzweigungen bieten eine Vielzahl an Möglichkeiten dynamisch Inhalte zu generieren.

```
@if (SignInManager.IsSignedIn(User))
        {
        <li class="nav-item float-right">
                <form asp-controller="Account" asp-action="Logout"
                asp-route-returnUrl="@Url.Action("Index","Home",)">
                <button type="submit">
                Logout</button>
                </form>
        </li>
        }
else
        {
        <li class="nav-item">
                <a class="nav-link text-dark" asp-controller="Account"
                asp-action="Register">Register</a>
        </li>
        <li class="nav-item">
                <a class="nav-link text-dark" asp-controller="Account"
                asp-action="Login">Login</a>
        </li>
        }
```

Listing 4.10: if Verzweigung mit Razor

In Listing 4.10 hängt die Verzweigung davon, ob der Benutzer eingeloggt ist. Zu Beginn wird geprüft, ob der Benutzer eingeloggt ist. Wenn dass der Fall ist, wird ein Knopf erzeugt, um den Benutzer auszuloggen. Dazu wird ein *Button* generiert, der über eine *Form* mit einer Action Methode verknüpft wird die, sobald aufgerufen, im Controller den Benutzer ausloggt. Sollte kein Benutzer eingeloggt sein, werden *Buttons* für *Login* und *Register* generiert, die mit den jeweils zugehörigen *action methods* verlinkt werden.

Schleifen Um ein Model unabhängig des Inhaltes auszulesen und dynamisch aufzulisten, werden Schleifen benötigt.

```
@for (int i = 0; i < Model.PublicKeys.Count; i++)
{
        <tr>
        <td>@i</td>
        <td>@Encoding.UTF8.GetString(Model.PublicKeys[i].KeyData)</td>
        <td>@Model.PublicKeys[i].Flag</td>
        <td>
                <button class="btn deactivate" id="deactivate-key-button"
                onclick="deactivateKey('@i')">
                Deactivate<button>
                <button class="btn edit" onclick="openEditKeyPopup('@i')">
                Edit</button>
        </td>
        </tr>
}
```

Listing 4.11: for Schleife mit Razor

In Listing 4.11 wird das Model dynamisch in einer Tabelle ausgegeben. Dies kann je nach Präferenz mit einer *for* – oder mit einer *foreach* Schleife realisiert werden. Sobald einem Model eine Liste an beliebigen Objekten mitgegeben wird, gibt man der *for* Schleife die Anzahl der Objekte oder die Länge der Liste mit und definiert, wie die einzelnen Elemente dargestellt werden sollen. In diesem Beispiel wird eine Tabelle erstellt, in der mehrere *PublicKey* Objekte dargestellt werden sollen. Jedes Objekt nimmt dabei eine Zeile ein, in der dann Daten, Status und zugehörige Benutzereingaben in jeweils eigenen Spalten platziert werden.

4.2.3 Layout der Weboberfläche

Webseiten werden in mehrere grundlegende Segmente eingeteilt. Ein einfaches Layout das in KeyS angewandt wird besteht aus Kopfzeile, Navigation, Inhalt und abschließend eine Fußzeile.

Kopfzeile Befindet sich im am Anfang, am oberen Rand einer Seite, und beinhaltet grundlegende Informationen wie den Namen oder das Logo der Seite.

Abbildung 4.3: Kopfzeile der Weboberfläche

Navigation Möglichkeiten um durch die Seite zu navigieren. Wird hauptsächlich, durch verlinkte Eingabemöglichkeiten wie Knöpfe, auf zwei verschiedene Weisen umgesetzt. In Kombination mit der Kopfzeile oder linksbündig am äußeren Rand. Im ersten Fall werden wichtige Links entweder direkt in die Kopfzeile eingebaut oder separat in einer Navigationsleiste direkt darunter. Im zweiten Fall werden eher spezifischere Links in einer Spalte am linken Rand aufgelistet. Die zweite Art der Navigation wird hauptsächlich in Kombination mit der ersten verwendet. In KeyS wird die erste Option genutzt, da sich die wichtigsten Funktionalitäten auf den beiden Seiten Public Keys und Secrets befinden, weshalb die Kopfzeile ausreicht um diese übersichtlich zu verlinken.

Abbildung 4.4: Navigation der Weboberfläche

Inhalt Der tatsächliche Inhalt kann je nach Art der Seite auf mehrere verschiedene Arten dargestellt werden. In den meisten Fällen wird ein Spalten bezogenes Layout verwendet. Dabei wird der Inhalt für eine übersichtliche Ansicht auf bis zu drei Spalten aufgeteilt. Je nach Zielgerät und daraus folgender Bildschirmgröße eignen sich für Smartphones eine, etwas größere Geräte wie Laptops oder Tabletts zwei und für Desktops drei Spalten. Es können auch mehr als drei verwendet werden, allerdings kann dies schnell zu unübersichtlichen Inhalten führen.

In KeyS werden Inhalte tabellarisch dargestellt. Damit werden alle relevanten Informationen geordnet und übersichtlich angezeigt.

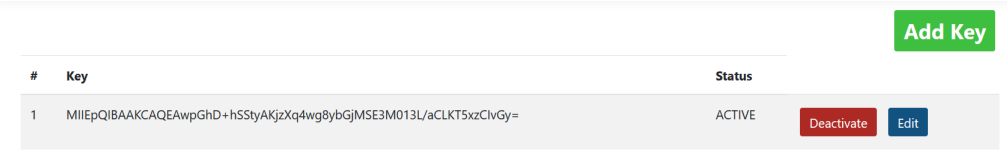


Abbildung 4.5: Inhalt der Weboberfläche

Fußzeile Als Gegenstück zur Kopfzeile befindet sich die Fußzeile am Ende der Seite, am unteren Rand. Sie enthält Kontaktinformationen und Informationen zum Copyright.

Abbildung 4.6: Fußzeile der Weboberfläche

4.2.4 Design der Weboberfläche

Beim Design der Benutzeroberfläche der Web Seite liegt der Fokus auf einer effizienten Arbeitsumgebung. Das Ziel ist also viel mehr eine einfache, leicht zu bedienende und übersichtliche als eine unglaublich ansprechend gestaltete Oberfläche zu entwerfen.
Der Aufbau richtet sich nach einem simplen Layout, bei dem über die Kopfzeile einfach durch die ganze Anwendung navigiert werden kann. Im Inhalt werden arbeitsrelevante Daten tabellarisch dargestellt, um eine gute Übersicht zu bieten. Potentielle Eingaben durch Knöpfe oder ähnlichem werden direkt bei den zugehörigen Daten angeordnet.

4.2.4.1 Cascading Style Sheets

In der Web Entwicklung wird Design hauptsächlich durch Cascading Style Sheets, kurz Css, umgesetzt. Durch Css lässt sich genau definieren, wie HTML Elemente vom Browser angezeigt werden sollen. Dabei können zum einen optische Eigenschaften, wie Farben, Größen oder Schriftarten, sowie auch architektonische Verhaltensweisen, wie die genau festgelegte oder auch dynamisch anpassbare Anordnung aller Elemente, angegeben werden.
Css kann sowohl innerhalb einer HTML Datei, als auch extern in Css Dateien implementiert werden. Der große Vorteil den externer Implementation mitbringt, ist, dass mit einer einzigen Datei mehrere Seiten auf einmal gestaltet werden können.

4.2.4.2 Design Framework - Bootstrap

Ein Großteil des Designs der Web Seite basiert auf Bootstrap, eines der beliebtesten open-source Css Frameworks. Bootstrap stellt simple Designs aller HTML Elemente, zahlreiche Werkzeuge zur Layout Entwicklung sowie diversen JavaScript Komponenten in Form von jQuery Plugins.
Durch die Verwendung von Bootsrap wurde der Arbeitsaufwand des Web Seiten Designs reduziert, indem größtenteils nur noch kleinere Anpassungen an bereits bestehenden Css Klassen vorgenommen wurden.

5 Client

Der Client ist eine Desktop Anwendung die dem Benutzer eine Möglichkeit bietet sowohl einfach mit dem Server zu interagieren, als auch zuverlässig Daten mit anderen Nutzern der Software auszutauschen.

5.1 Client Bibliothek

Die im Client beinhaltete Bibliothek ermöglicht sowohl eine Verschlüsselung von Daten als auch eine Verwaltung der Client-Server Kommunikation. Sie soll Befehle und Argumente entgegennehmen und dementsprechend handeln. Möglich soll das Erstellen, Aktualisieren und Herunterladen von Daten sein.

5.1.1 Verschlüsselung

Eine spezielle kryptographische Methode ist nötig, damit versichert werden kann, dass verschickte Informationen ausschließlich für den Empfänger entschlüsselt sichtbar sind. Auch sollen geschickte und gespeicherte Datenmengen vermindert werden. Verwendet werden zwei Verfahren: eine asymmetrische und symmetrische Verschlüsselung.

5.1.1.1 Symmetrische Verschlüsselung

Symmetrisch werden Daten mit einem symmetrischen Schlüssel gesichert. Dieser sperrt und entsperrt den gleichen Datensatz.

Abbildung 5.1: Symmetrische Verschlüsselung

Verwendet wird der AES[1] (Advanced Encryption Standard) Algorithmus. Aufgrund des gleichen verwendeten Schlüssels, und damit im Vergleich zur asymmetrischen Verschlüsselung einfachen Rechenverfahren, muss die Schlüsselgröße aktuell im Bereich von 128-256 Bit liegen, um die gewollte Sicherheit zu gewährleisten. Der Datensatz wird in 128 Bit große Blöcke aufgeteilt und jeder Block wird über 10-14 (abhängig von der Schlüsselgröße) Runden von Rechenvorgängen in den Ciphertext gebracht. Pro Runde wird der Datensatz mit einer Abfolge von Rechnungen, wie zum Beispiel der Addition von Daten und Schlüssel, in den Ciphertext gerechnet. Der Rechenweg selbst ist öffentlich. Die Sicherheit des Verfahrens entsteht dadurch, dass immer andere Teile des symmetrischen Schlüssels für die Berechnung der Verschlüsselung genutzt werden. Mit aktuellen technischen Mitteln ist es schwer vom Ciphertext auf den Schlüssel oder den Klartext zurückzurechnen. Für einen Angreifer ist es allerdings möglich, jeden Schlüssel für die Entschlüsselung auszuprobieren. So ein Vorgehen dauert aber lange. Um auch diese Art von Angriff zu verhindern müssen symmetrische Schlüssel mindestens jedes Jahr ausgewechselt werden. Entschlüsselt kann der Ciphertext durch den invertierten Rechenweg mit dem symmetrischen Schlüssel werden. Bei der Ent- und Verschlüsselung liegt ein im Vergleich zu asymmetrischen Verschlüsselungsverfahren niedriger Rechenaufwand vor.

[1] Abdullah Ako Muhammad. *Advanced Encryption Standard (AES) Algorithm to Encrypt and Decrypt Data*. 2017-06. URL: https://www.researchgate.net/profile/Ako_Abdullah/publication/317615794_Advanced_Encryption_Standard_AES_Algorithm_to_Encrypt_and_Decrypt_Data/links/59437cd8a6fdccb93ab28a48/Advanced-Encryption-Standard-AES-Algorithm-to-Encrypt-and-Decrypt-Data.pdf (besucht am 30. 03. 2020).

5.1.1.2 Asymmetrische Verschlüsselung

Asymmetrische Verschlüsselung, in der Literatur auch als Public Key Cryptography bekannt, zeichnet sich darin aus, dass sie zwei Schlüssel braucht, ein sogenanntes Schlüsselpaar. Es besteht aus einem öffentlichen und privaten Schlüssel.

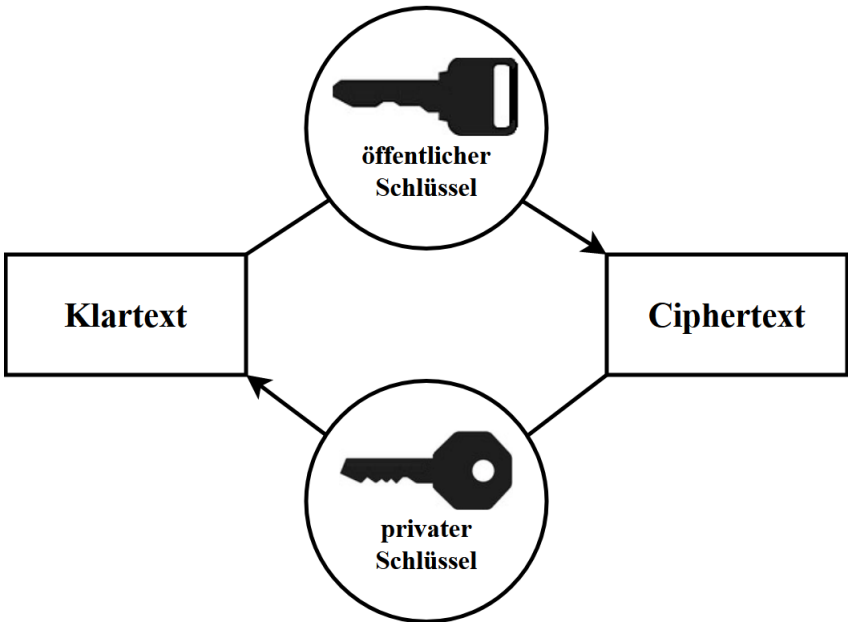


Abbildung 5.2: Asymmetrische Verschlüsselung

Bei solchen Verfahren wird einer der beiden verwendet um einen Datensatz zu verschlüsseln und es kann nur der jeweils andere zum Entschlüsseln genutzt werden. Um solch einen Vorgang zu realisieren, muss ein „Trapdoor" Verhalten kreiert werden. Dafür muss es eine mathematische Funktion geben, welche die Daten einfach in den Ciphertext rechnet, jedoch es erschwert, diese ohne den benötigten Schlüssel wieder in den Klartext zu rechnen. Diese „Trapdoor" soll bei der Verschlüsselung mit einem öffentlichen Schlüssel vorhanden sein. Bei der Zurückrechnung soll die „Trapdoor" mit Benutzung des privaten Schlüssels umgangen werden und somit mit dem gleichem Aufwand wie bei der Verschlüsselung entschlüsselt werden. Ein weit verbreiteter und in diesem System verwendeter Algorithmus ist RSA[2], benannt nach den drei Erfindern. Er beinhaltet die Verschlüsselungsverfahren als auch die Schlüsselpaar Erzeugung und verwendet bei der Verschlüsselung eine Multiplikation zweier großer Primzahlen (p und q), bei der eine Rückführung auf die beiden Faktoren äußerst aufwendig ist.

[2]Evgeny Milanov. *The RSA Algorithm*. 2009-06-03. URL: `https://sites.math.washington.edu/~morrow/336_09/papers/Yevgeny.pdf` (besucht am 30. 03. 2020).

Dieses Produkt (n) ist bereits einer von zwei Teilen des öffentlichen und privaten Schlüssels.

$$n = p * q$$

Das kleinste gemeinsame Vielfache der Primzahlen weniger Eins bildet λ(n):

$$\lambda(n) = kgV(p-1, q-1)$$

Für den Exponenten e (encrypt), der dem öffentlichen Schlüssel angehängt wird, wird eine Zahl gewählt, welche folgende Voraussetzungen erfüllt:

$$1 < e < \lambda(n)$$

$$ggT(e, \lambda(n)) = 1$$

Dem privaten Schlüssel wird ein aus dem Exponenten e mittels dem mittels erweiterten Euklidischen Algorithmus[3] berechnetem Exponent d (decrypt) hinzugefügt.

$$d = (e \mod \lambda(n))^{-1}$$

Ein solches Schlüsselpaar kann schlussendlich immer nur im Ganzen erstellt werden. Die Ver- und Entschlüsselung beruhen auf der gleichen Formel, jedoch anderer Verwendeter Exponenten der Schlüssel, wodurch die Funktion der Entschlüsselung immer nur mit dem jeweils anderen Schlüssel sichergestellt wird. m (message) ist hier die Nachricht und c (cipher) der Ciphertext.

$$c = (m^e \mod n)$$

$$m = (c^d \mod n)$$

Zum heutigen Stand werden bei einem RSA Schlüsselpaar Schlüssel in Größen von 1024-4096 Bit verwendet. Bei asymmetrischen Verschlüsselungsverfahren kommt es im Vergleich zu symmetrischen zu höheren Rechendauern bei Ver- und Entschlüsselung.

[3] Juan Álvaro Muñoz Naranjo. *Applications of the Extended Euclidean Algorithm to Privacy and SecureCommunications*. 2010-06. URL: https://www.researchgate.net/profile/Juan_Alvaro_Naranjo/publication/244477217_Applications_of_the_Extended_Euclidean_Algorithm_to_Privacy_and_Secure_Communications/links/54e707e50cf277664ff77562/Applications-of-the-Extended-Euclidean-Algorithm-to-Privacy-and-Secure-Communications.pdf (besucht am 30. 03. 2020).

5.1.1.3 Vor- und Nachteile

Angenommen die geheimen Daten werden symmetrisch verschlüsselt und an den Server übermittelt. Kein Dritter kann somit auf die Daten zugreifen. Allerdings ist es nun ein Problem den symmetrischen Schlüssel zugehörig der Daten an den Empfänger zu schicken, da dieser von einem „man in the middle" abgegriffen und so auf die ursprünglichen geheimen Daten zurückgeführt werden kann. Es gibt Möglichkeiten sich einen Schlüssel für längere Zeit auszumachen oder physisch zu übergeben, doch diese Methoden sind zeitaufwendig und bieten immer noch nicht die verlangte Sicherheit des Systems.
Wird statt dem symmetrischen Verschlüsselungsverfahren ein asymmetrisches verwendet, so erstellt sich jeder Teilnehmer in einem System ein eigenes Schlüsselpaar und stellt den öffentlichen Schlüssel für jeden sichtbar dar. Der Sender bildet dann einen asymmetrisch verschlüssleten Datensatz mit dem öffentlichen Schlüssel des Empfängers und gibt diesen weiter. Hierbei wäre auch ein „man in the middle attack" möglich indem dieser seinen eigenen öffentlichen Schlüssel verschickt und sich als den echten Empfänger tarnt. Da bei RSA allerdings eine digitale Signatur enthalten ist und so garantiert werden kann, dass Daten oder Schlüssel von dem gewollten Empfänger stammen, ist dieser Angriff kein Problem. Zu einem Problem werden aber die Datenmengen die verschickt und auf dem Server abgelegt werden müssen. Der Grund dafür ist, dass für jede Person, jene Zugriff auf bestimmte geheime Daten haben, eine individuelle verschlüsselte Version des Datensatzes verschickt und abgelegt werden muss. Bei einer Datei dessen Größe beispielsweise im Gigabyte Bereich liegt, weist sich dieser Vorgang als nicht optimal aus und entspricht ebenfalls nicht den Anforderungen an das System.

5.1.1.4 Hybridverfahren

Die verwendete Lösung ist eine Kombination der beiden oben angeführten Verfahren. Erst werden die geheimen Daten symmetrisch verschlüsselt. Anschließend muss der symmetrische Schlüssel asymmetrisch gesichert werden. Dafür muss der öffentlichen Schlüssel des Empfängers genutzt werden. Beide Dateien werden auf den Server gespeichert und sind nun bereit zur Abfrage. So werden alle übermittelten Datensätze sicher verschickt und abgelegt. Wenn ein anderer Nutzer Zugriff auf bestimmte geheime Daten erhalten soll, muss von einem der Berechtigten nur der symmetrischen Schlüssel mit dem öffentlichen Schlüssel des neuen Nutzers verschlüsselt werden und es müssen keine verschiedene Versionen der ursprünglichen geheimen Daten existieren. Alle Anforderungen sind somit erfüllt und eine sichere Kommunikation ist gewährleistet.

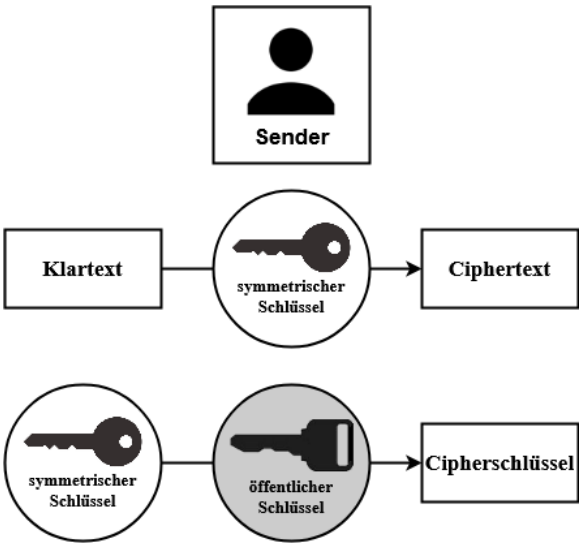


Abbildung 5.3: Hybrid Verfahren

In Abbildung 5.3 werden die Aufgaben des Senders dargestellt. Der Sender will seinen Klartext mit einem Empfänger teilen. Weiß hinterlegte Objekte stellen dar, dass sie vom Sender stammen, wobei grau hinterlegte von Empfänger stammen. Nach Durchführen dieser Vorgänge schickt der Sender die beiden rechts dargestellten Dateien an den Empfänger.

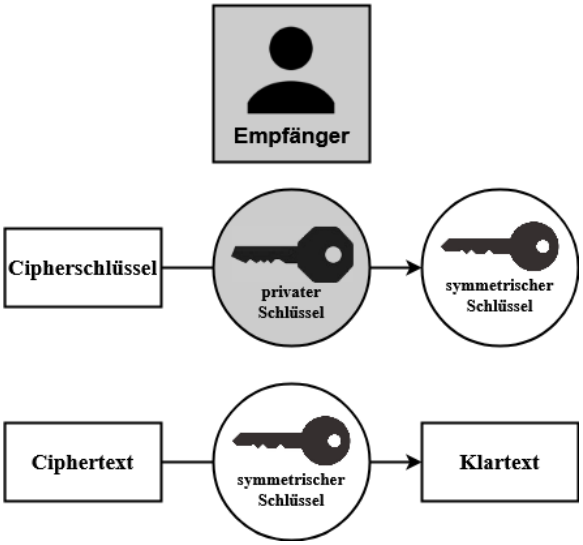


Abbildung 5.4: Hybrid Verfahren

In Abbildung 5.4 benutzt der Empfänger dieser Übertragung seinen privaten Schlüssel um den symmetrischen Schlüssel zu errechnen. Damit kann er den Ciphertext des Senders entschlüsseln.

5.1.2 REST API Client

Um die Client-Server Kommunikation über REST zu Realisieren, muss die Client Bibliothek dem Server Anfragen an dessen spezifischen Endpunkt schicken und die Antwort dessen empfangen und bearbeiten. Für diese Funktionalität ist die *RestApiClient* Klasse zuständig. Diese erweitert das *IApiClient* Interface und initialisiert im Konstruktor eine *IHttpClientFactory*, welche in der *RequestFolderById* Funktion für die *HttpClient* Kreation genutzt wird:

```
class RestApiClient : IApiClient
{
    private IHttpClientFactory _provider;

    public RestApiClient(IHttpClientFactory provider)
    {
        _provider = provider;
    }
}
```

Listing 5.1: RestApiClient

```
public async Task<FolderModel> RequestFolderById(int? id)
{
    using (HttpClient client = _provider.CreateClient())
    {
        using (HttpResponseMessage response = await client.GetAsync(
            $"/api/protecteddata/folder/{id}"))
        {
            return await JsonSerializer.DeserializeAsync<FolderModel>(
                await response.Content.ReadAsStreamAsync());
        }
    }
}
```

Listing 5.2: RequestFolderById Funktion

Mithilfe des neu kreierten *HTTPClients* kann nun eine Anfrage, mit dem gewollten Pfad des Endpunktes, mithilfe der *GetAsync* Funktion gestellt werden. Bei einer Ausführung mit dem Argument id = 1 wird der Endpunkt */api/protecteddata/folder/{1}* angesprochen und der Server soll entsprechend auf diese Anfrage reagieren. Aus der zurückkommenden *HttpResponseMessage* wird das in diesem Fall benötigte *FolderModel* extrahiert und zurückgegeben. Davor muss der Inhalt der Antwort vom Json Format in ein *FolderModel* Objekt konvertiert werden. Die Methode ist durch das *async* Schlüsselwort nebenläufig, also läuft sie in einem seperatem Thread. Diese Nebenläufigkeit ist hier sinnvoll, da der Client immernoch seine Funktionalitäten ablaufen kann auch wenn Laufzeiten auf dem Server beziehungsweise der Verbindung beider ansteigen. *id* ist ein Nullable Integer und kann somit auch null sein. In diesem Fall führt eine Anfrage an */api/protecteddata/folder/{null}* zu einer Rückgabe des Stammverzeichnisses.

5.1.3 Base64 Strings

Bei der Datenübertragung an den Server werden Variablen wie etwa Namen, Inhalte der Dateien sowie ein öffentlich-symmetrisch Schlüsselpaar gesendet. Der Name einer Datei kann problemlos als Zeichenfolge übertragen werden. Die Inhalte und Schlüssel sind allerdings als Bytefolgen. Dies stellt allerdings ein Problem da, weil HTTP, das der RESTful Kommunikation zugrundliegende Protokoll, keine Möglichkeit bietet, Byte Arrays zu versenden. Gelöst wird dies durch eine Base64 Kodierung. So wird aus einer Binärdatei eine Textdatei. Genauer werden jeweils 6 Bit in einen Buchstaben oder Symbol kodiert. Wenn die Anzahl der Bit im Ausgangsarray nicht genau durch 6 teilbar ist, wird mit 0 auf einen durch 6 teilbaren Wert aufgerundet. Um eine Dekodierung auf ein Bytearray zu ermöglichen muss die Bitlänge der Zeichenfolge durch 8 teilbar sein. Falls die Zeichenfolge nach der Kodierung zu kurz ist, wird er mit dem Fullzeichen = erweitert, bis die Länge durch 8 teilbar ist.

```
byte[] byteData = { 0x1A, 0x2B, 0x3C, 0x4D, 0x5E, 0x6F };
string data = System.Convert.ToBase64String(byteData);
```

Listing 5.3: Beispiel zu Kodierung Bytearray zu Base64 String

In Listing 5.3 wird das Bytearray *byteData* initialisiert. Dann wird er mit der *System.Convert.FromBase64String* Funktion in einen Base64 String Kodiert. Somit sind alle Variablen als Zeichenfolgen problemlos übertragbar.

5.1.4 Testen des Clients

Wie im Kapitel 2.2.2 über Test-Driven-Development angeführt, muss jede Klasse, auch hier am Client, von einem Test behandelt werden. Als Beispiel wird die *RestApiClient* Klasse, welche für die in 5.1.2 erklärte RESTful Kommunikation zuständig ist, durch Verwendung des Moq Frameworks im *ProtectedDataClient* Test ersetzt. Der *ProtectedDataClient* ist in der Client Bibliothek jene Klasse, welche die jeweiligen Eingaben des Nutzers behandelt. Der *ProtectedDataClient* wird folgendermaßen modular, also ohne Abhängigkeit auf die Funktionalität des *RestApiClient*, getestet:

```
bool wasCalled = false;

Mock<IApiClient> restClient = new Mock<IApiClient>();

int CheckSendCreateData()
{
    wasCalled = true;
    return 1;
}

restClient.Setup(x => x.SendCreateData(
    It.Is<string>(s => s == "MyPasswords/MailAccount/"),
    It.Is<DataModel>(s => s.Name == "MailAccount/" &&
                          s.Data == "SecretPassword")))
                ).Returns(CheckSendCreateData);

ProtectedDataClient client = new ProtectedDataClient(restClient.Object);
client.Set("MyPasswords/MailAccount/", "SecretPassword");

Assert.IsTrue(wasCalled);
```

Listing 5.4: Mocking des RestApiClients

Somit wird getestet, ob bei einem Aufruf der *Set* Methode des *ProtectedDataClients SendCreateData* richtig aufruft. In Zeile 1 wird eine Prüfvariable erstellt, welche am Ende des Tests wieder mit *Assert.IsTrue* abgeprüft wird. Der *RestApiClient* wird dann über sein Interface gemockt und die Funktion *SendCreateData* wird mit *restClient.Setup* für das Testumfeld überschrieben. *SendCreateData* liefert nur mehr einen Rückgabewert wenn sie mit genau den Argumenten, wie hier angegeben, aufgerufen wird. Für eine Prüfung des Aufrufens von *SendCreateData* muss allerdings die Methode *CheckSendCreateData* bei *Returns* anstatt eines fixen Rückgabewertes eingetragen werden. Anschließend muss der *ProtectedDataClient* mit Mitgabe des gemocktem *RestApiClients* instanziiert und dessen *Set* Funktion aufgerufen werden. Wird *SendCreateData* nicht oder falsch verwendet, so schlägt das *Assert* Statement fehl.

Die in 5.1.2 beschriebene *RestApiClient* Klasse kann jedoch nur getestet werden, wenn der angesprochene Server gemockt wird, da dieser zum Testen nicht existiert. Bei dem Mocken eines Servers muss

anders als bei vorgegangen werden. In Listing 5.5 wird die bereits erläuterte Methode *RequestFolderById* darauf geprüft, ob sie an den Server korrekte Anfragen schickt.

```
Mock<IHttpClientFactory> clientFactory = new Mock<IHttpClientFactory>();
FolderModel rootFolder = new FolderModel()
{
    ParentId = null,
    Name = "root",
    ProtectedDataIds = new List<int>(),
    SubfolderIds = new List<int>()
};

HttpResponseMessage Callback (HttpRequestMessage request)
{
    Assert.AreEqual("/api/protecteddata/folder/",
        request.RequestUri.LocalPath);
    string json = JsonSerializer.Serialize(rootFolder);
    return new HttpResponseMessage(System.Net.HttpStatusCode.OK)
        { Content = new StringContent(json) };
}
clientFactory.Setup(x => x.CreateClient(It.IsAny<string>())
                    ).Returns(
                        new HttpClient(new MockHttpMessageHandler(Callback))
                        {
                            BaseAddress = new Uri("http://test.com")
                        });

IApiClient restClient = new RestApiClient(clientFactory.Object);
FolderModel responseFolder = await restClient.RequestFolderById(null);

Assert.AreEqual(rootFolder.ParentId, responseFolder.ParentId);
Assert.AreEqual(rootFolder.Name, responseFolder.Name);
CollectionAssert.AreEquivalent(rootFolder.ProtectedDataIds.ToArray(),
    responseFolder.ProtectedDataIds.ToArray());
CollectionAssert.AreEquivalent(rootFolder.SubfolderIds.ToArray(),
    responseFolder.SubfolderIds.ToArray());
```

Listing 5.5: Server Mocking

In diesem Beispiel wird die geschickte Anfrage auf seine Inhalte und Richtigkeit geprüft. Im Gegensatz zu dem in Listing 5.4 angeführtem Beispiel muss eine *IHttpClientFactory* dem *RestApiClient* bei der Initialisierung als Parameter übergeben werden. Zu Beginn des Tests wird ein *FolderModel* Objekt mit den erwarteten Werten vorbereitet um am Ende des Tests einen Vergleich zur Überprüfung der Richtigkeit zu bieten. *CreateClient* wird über die *Setup* Methode des Mock Objekts konfiguriert um einen definierten *HttpClient* zurückzugeben. Diesem wird ein Mock eines *HttpMessageHandlers* bei der Initialisierung übergeben. Es wird also eine Verbindung mit einem Server der Adresse *http://test.com* und dem Verhalten der vorher erstellten *Callback* Funktion vorgetäuscht. So können vorgefertigte Antworten zu spezifischen Anfragen des *RestApiClient* zurückgegeben werden. In diesem *Callback* wird auf die Richtigkeit des Pfades des angesprochenen Endpunkte geprüft und anschließend eine *HttpResponseMessage* zurückgegeben mit dem *FolderModel* und dem *HttpStatusCode* 200 (OK). Durch dieses Vorgehen

ist der *RestApiClient* mit der einzigen Abhängigkeit, dass der spezifische Pfad des *FolderModel* auf dem Server mit der echten verwendeten Serverstruktur übereinstimmt, vollständig getestet. Bei Änderung der Dateistruktur am Server muss der Test und die *RestApiClient* Klasse entsprechend angepasst werden.

5.2 CLI-Tool

Ein Command Line Interface - Tool (CLI-Tool) ist eine Applikation, welche Befehle als Eingaben in Textform in jeglicher Art von Command Line oder Terminal entgegennimmt. Das Interface ist nur das sichtbare Ein- und Rückgabe Textfeld. Bei bekannten Beispielen wie Windows CMD oder Git Bash kann diese folgendermaßen aussehen:

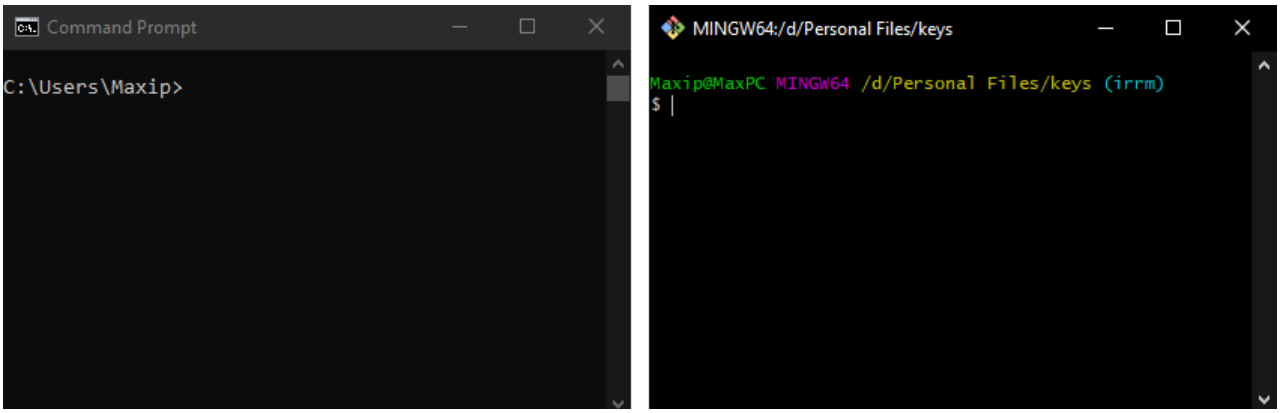


Abbildung 5.5: Command Line Interface Beispiele

Die zugrundliegende Logik des CLI-Tools wird Interpreter genannt. Möglich soll eine Ausführung der in Kapitel 5.1 angeführten Funktionen wie *Set* und *Get* sein. Bei einer Eingabe wird das Programm an erster Stelle angeschrieben, gefolgt von dem gewünschten Befehl und dessen Argumente. Gewählt

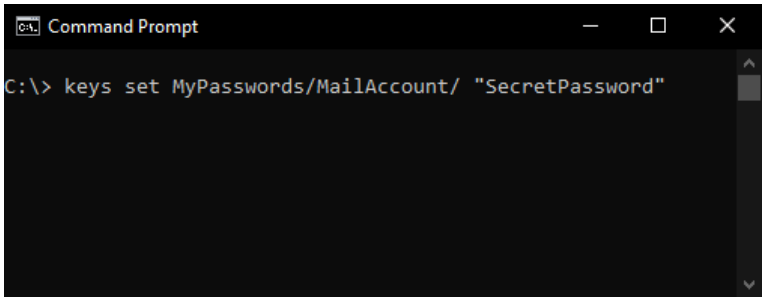


Abbildung 5.6: Möglicher Aufbau eines Set Befehls

wurde diese Art von Interface anstelle einer graphischen Oberfläche aufgrund einer einfachen Nutzung und Realisierung.

6 Anhang

Literatur

.NET API Browser. Microsoft. URL: https://docs.microsoft.com/en-us/dotnet/api/ (besucht am 01. 04. 2020).

ADO.NET. Microsoft. URL: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet (besucht am 30. 03. 2020).

Donovan, Ryan. *The Top 10 Frameworks and What Tech Recruiters Need to Know About Them*. Stack Overflow. 17. Dez. 2019. URL: https://stackoverflow.blog/2019/12/17/the-top-10-frameworks-and-what-tech-recruiters-need-to-know-about-them/ (besucht am 31. 03. 2020).

Fielding, Roy Thomas. „Architectural Styles and the Design of Network-based Software Architectures“. Doctoral dissertation. University of California, Irvine, 2000.

Freeman, Jonathan. *What is an API? Application programming interfaces explained*. InfoWorld. 8. Aug. 2019. URL: https://www.infoworld.com/article/3269878/what-is-an-api-application-programming-interfaces-explained.html (besucht am 31. 03. 2020).

Martin, Robert C. *Clean Architecture*. Robert C. Martin Series. 2017. ISBN: 978-0-13-449416-6.

— *Clean Code*. Robert C. Martin Series. 2008. ISBN: 978-0-13-235088-4.

— *Professionalism and Test-Driven Development*. 2007. URL: http://fpl.cs.depaul.edu/jriely/450/extras/prof-tdd.pdf (besucht am 26. 03. 2020).

Microsoft.Data.Sqlite. Microsoft. URL: https://docs.microsoft.com/en-us/dotnet/standard/data/sqlite (besucht am 30. 03. 2020).

Milanov, Evgeny. *The RSA Algorithm*. 2009-06-03. URL: https://sites.math.washington.edu/~morrow/336_09/papers/Yevgeny.pdf (besucht am 30. 03. 2020).

Moq-Framework. Clarius Consulting, Manas und InSTEDD. URL: https://github.com/moq/moq4 (besucht am 26. 03. 2020).

MSTest-Framework. Microsoft. URL: https://docs.microsoft.com/en-us/dotnet/api/microsoft.visualstudio.testtools.unittesting (besucht am 26. 03. 2020).

Muhammad, Abdullah Ako. *Advanced Encryption Standard (AES) Algorithm to Encrypt and Decrypt Data*. 2017-06. URL: https://www.researchgate.net/profile/Ako_Abdullah/publication/317615794_Advanced_Encryption_Standard_AES_Algorithm_to_Encrypt_and_Decrypt_Data/links/59437cd8a6fdccb93ab28a48/Advanced-Encryption-Standard-AES-Algorithm-to-Encrypt-and-Decrypt-Data.pdf (besucht am 30. 03. 2020).

Naranjo, Juan Álvaro Muñoz. *Applications of the Extended Euclidean Algorithm to Privacy and SecureCommunications*. 2010-06. URL: https://www.researchgate.net/profile/Juan_Alvaro_Naranjo/publication/244477217_Applications_of_the_Extended_Euclidean_Algorithm_to_Privacy_and_Secure_Communications/links/54e707e50cf277664ff77562/Applications-of-the-Extended-Euclidean-Algorithm-to-Privacy-and-Secure-Communications.pdf (besucht am 30. 03. 2020).

Siebler, Florian. *Design Patterns mit Java*. 2014. ISBN: 978-3-446-43616-9.

Smith, Steve. *Overview of ASP.NET Core MVC*. Microsoft. 12. Feb. 2020. URL: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-3.1 (besucht am 26. 03. 2020).

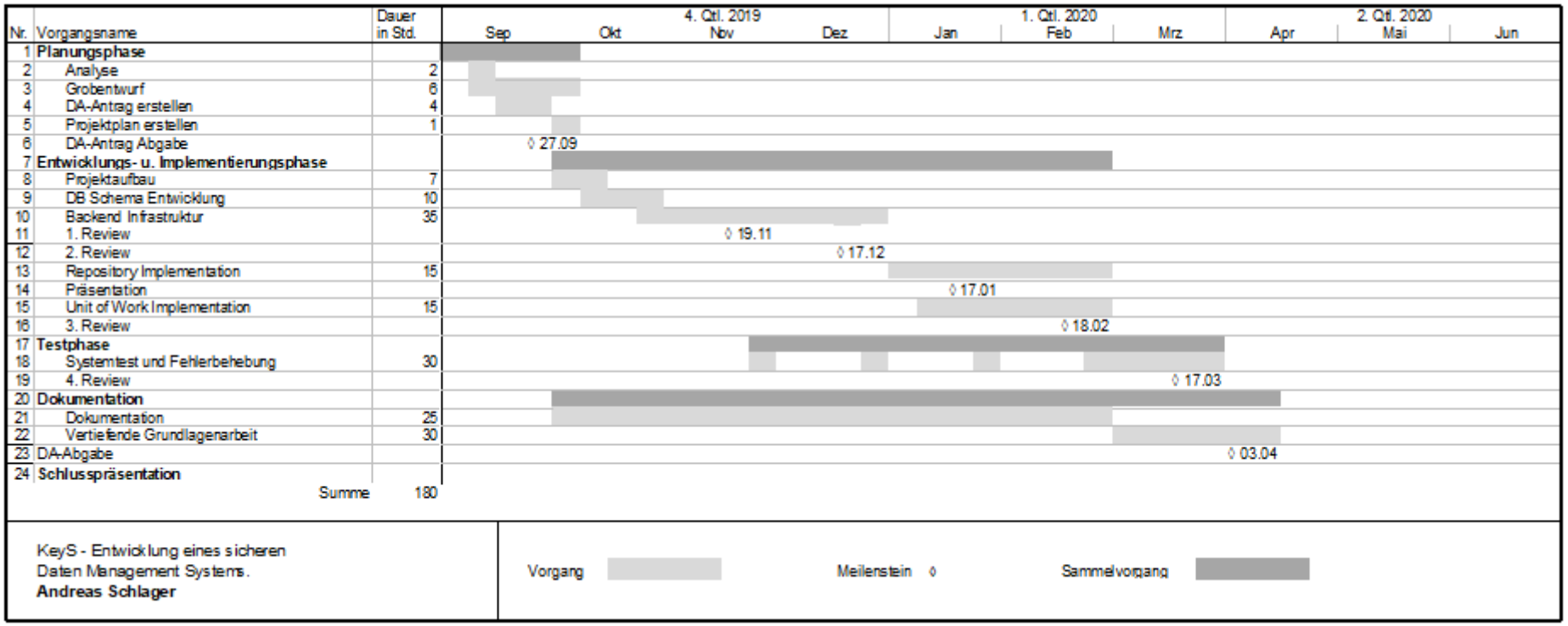
SQLite. SQLite-Team. URL: https://www.sqlite.org (besucht am 30. 03. 2020).

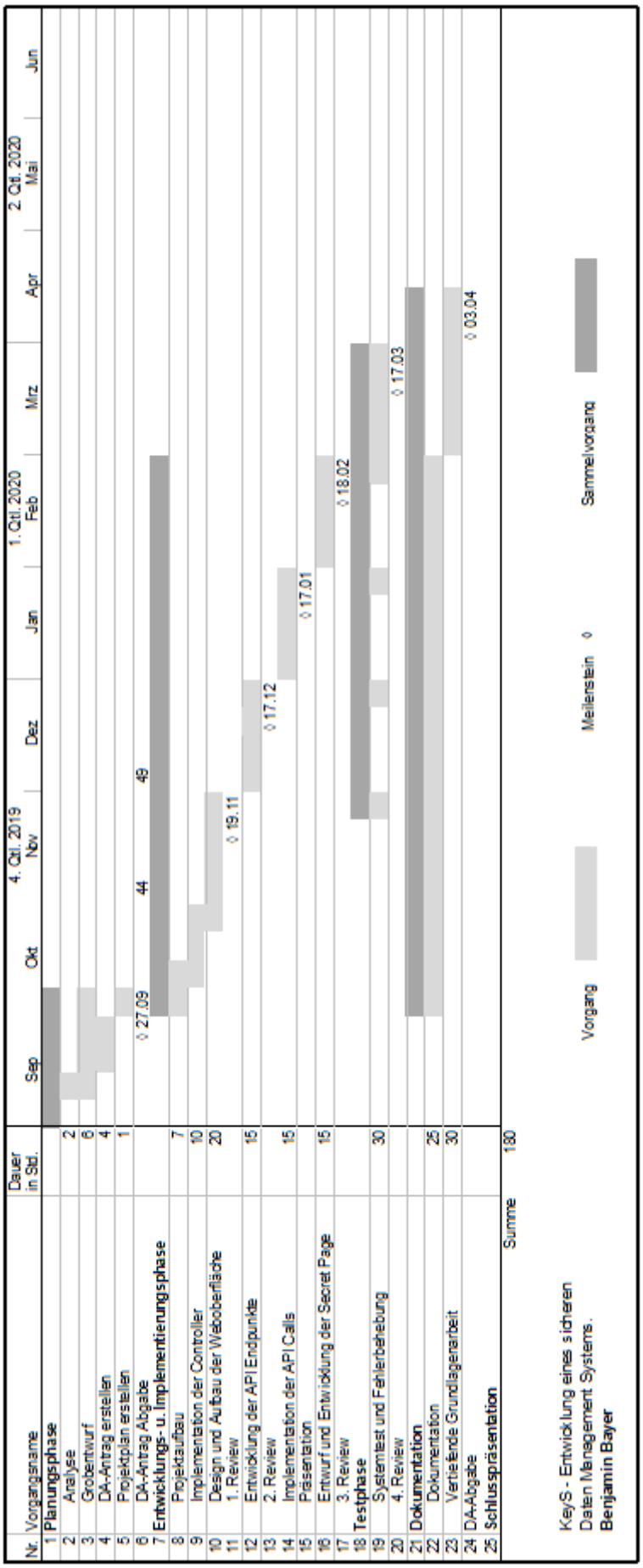
Webapplikationen und Webanwendungen. oneclick. 15. Jan. 2019. URL: https://oneclick-cloud.com/de/blog/trends/webapplikationen-webanwendungen/ (besucht am 31. 03. 2020).

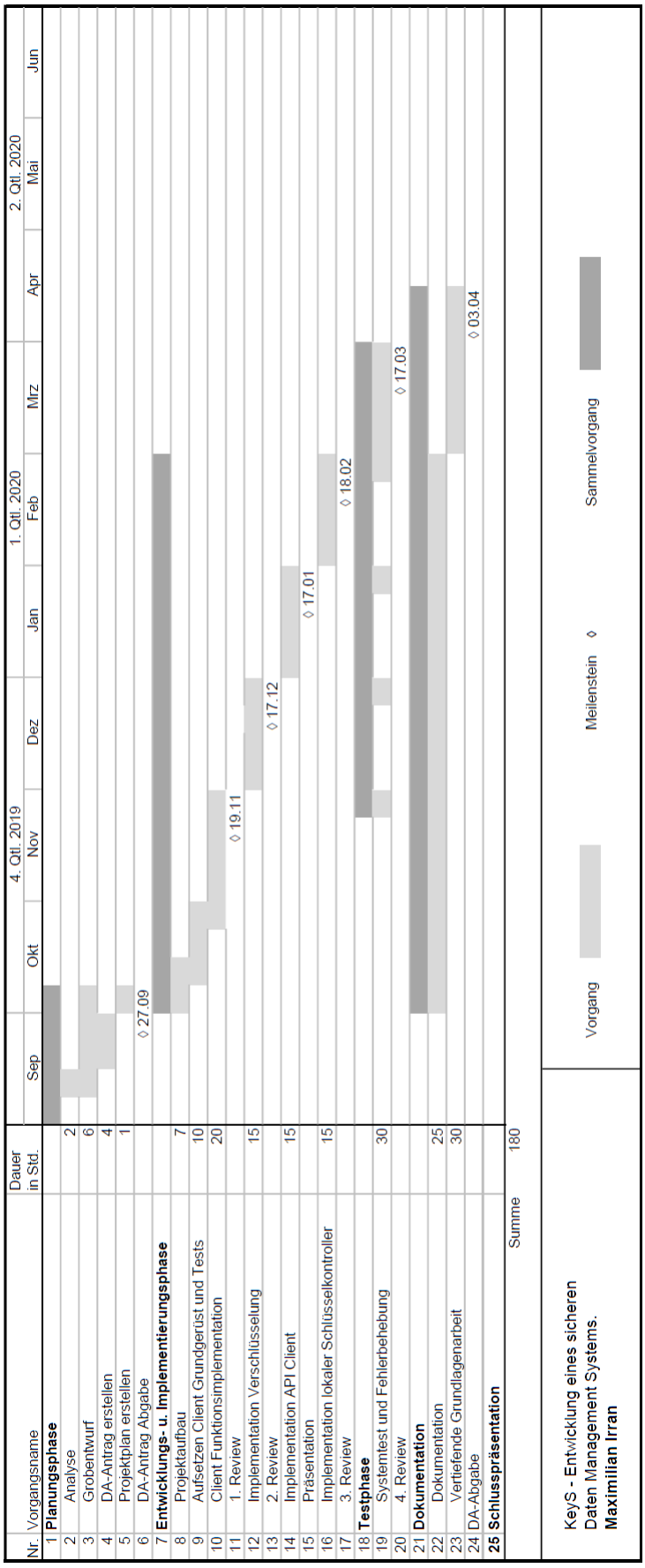
Abbildungsverzeichnis

Listings

GANTT-Diagramm




Nr. Vorgangsname Dauer in Std.
1 Planungsphase
2 Analyse 2
3 Grobentwurf 6
4 DA-Antrag erstellen 4
5 Projektplan erstellen 1
6 DA-Antrag Abgabe
7 Entwicklungs- u. Implementierungsphase
8 Projektaufbau 7
9 Implementation der Controller 10
10 Design und Aufbau der Weboberfläche 20
11 1. Review
12 Entwicklung der API Endpunkte 15
13 2. Review
14 Implementation der API Calls 15
15 Präsentation
16 Entwurf und Entwicklung der Secret Page 15
17 3. Review
18 Testphase
19 Systemtest und Fehlerbehebung 30
20 4. Review
21 Dokumentation
22 Dokumentation 25
23 Vertiefende Grundlagenarbeit 30
24 DA-Abgabe
25 Schlusspräsentation
Summe 180
4. Qtl. 2019
Sep Okt Nov Dez
1. Qtl. 2020
Jan Feb Mrz
2. Qtl. 2020
Apr Mai Jun
◊ 27.09
44
49
◊ 19.11
◊ 17.12
◊ 17.01
◊ 18.02
◊ 17.03
◊ 03.04
Vorgang
Meilenstein
Sammelvorgang
KeyS - Entwicklung eines sicheren Daten Management Systems.
Benjamin Bayer


Nr. Vorgangsname
Dauer in Std.
4. Qtl. 2019
Sep
Okt
Nov
Dez
1. Qtl. 2020
Jan
Feb
Mrz
2. Qtl. 2020
Apr
Mai
Jun
1 Planungsphase
2 Analyse 2
3 Grobentwurf 6
4 DA-Antrag erstellen 4
5 Projektplan erstellen 1
6 DA-Antrag Abgabe ◊ 27.09
7 Entwicklungs- u. Implementierungsphase
8 Projektaufbau 7
9 Aufsetzen Client Grundgerüst und Tests 10
10 Client Funktionsimplementation 20
11 1. Review ◊ 19.11
12 Implementation Verschlüsselung 15
13 2. Review ◊ 17.12
14 Implementation API Client 15
15 Präsentation ◊ 17.01
16 Implementation lokaler Schlüsselkontroller 15
17 3. Review ◊ 18.02
18 Testphase
19 Systemtest und Fehlerbehebung 30
20 4. Review ◊ 17.03
21 Dokumentation
22 Dokumentation 25
23 Vertiefende Grundlagenarbeit 30
24 DA-Abgabe ◊ 03.04
25 Schlusspräsentation
Summe 180
KeyS - Entwicklung eines sicheren Daten Management Systems.
Maximilian Irran
Vorgang
Meilenstein ◊
Sammelvorgang

Begleitprotokoll

Andreas Schlager

KW	Beschreibung	Zeitaufwand
38	Diplomarbeitsantrag erstellen	4h
39	Diplomarbeitsantrag einreichen	1h
40	Konfiguration der Versions Kontrolle	7h
41	Erstellen des Projektes	8h
42	Konfiguration des Projektes	7h
43	Entwicklung des Datenbank Schemas (ERD)	8h
44	Entwicklung des Datenbank Schemas (ERD)	8h
45	SQL Initialisierungs Script	8h
46	SQL Trigger und Views	8h
47	Definition der Repositories	5h
48	Abstraktionen der Repositories für Backend erstellt	6h
49	Abstraktionen der Unit of Work für Backend erstellt	8h
50	Vorbereitung für erste Präsentation	6h
51	Implementation des User Repositories	8h
52	-	
1	-	
2	Implementation der User Unit of Work	7h
3	Vorbereitung Tag der offeen Tür	6h
4	Testen des User Repositories mit SQLite Datenbank	4h
5	Testem der User Unit Of Work mit SQLite Datenbank	8h
6	Fehlerbehebung	7h
7	-	
8	Optimierung und Fehlerbehebung	8h
9	Einführung von Use-Case Struktur für Frontend	8h
10	Implementation des Data Repositories	8h
11	Implementation der Data Unit Of Work	8h
12	Verfassen der Diplomarbeit	8h
13	Verfassen der Diplomarbeit	8h
14	Verfassen der Diplomarbeit	8h

Benjamin Bayer

KW	Beschreibung	Zeitaufwand
38	Diplomarbeitsantrag erstellen	4h
39	Diplomarbeitsantrag einreichen	1h
40	Projektplanung, Aufgabenferfeinerung, Ganttdiagramm	2h
41	Projektaufbau	7h
42	Controller Entwurf	9h
43	Implementation der Controller	10h
44	Entwurf der Weboberfläche	10h
45	Entwurf der Weboberfläche	10h
46	Implementation der Weboberfläche	10h
47	Implementation der Weboberfläche	10h
48	Fehlerbehebung und Optimierung	8h
49	Fehlerbehebung und Optimierung	9h
50	Vorbereitung für die 1. Präsentation	6h
51	Implementation der API Endupunkte	10h
52	-	
1	-	
2	Fehlerbehebung und Optimierung	7h
3	Vorbereitung für Tag der offenen Tür	5h
4	Implementatation der API Calls	6h
5	Fehlerbehebung und Optimierung	6h
6	Entwurf der Secret Page	6h
7	-	
8	Implementation und der Secret Page	5h
9	Fehlerbehebung und Optimierung	5h
10	Überarbeitung des Designs	5h
11	Finalisierung des Designs	5h
12	Verfassen der Diplomarbeit	8h
13	Verfassen der Diplomarbeit	8h
14	Verfassen der Diplomarbeit	8h

Maximilian Irran

KW	Beschreibung	Zeitaufwand
38	Diplomarbeitsantrag erstellen	4h
39	Diplomarbeitsantrag einreichen	1h
40	Projektplanung, Aufgabenverfeinerung, Ganttdiagramm	2h
41	Client Bibliothek Gundgerüst	8h
42	Anlegen der Testumgebung	6h
43	Anlegen der Testumgebung	6h
44	Client Testung	8h
45	Client Funktionsimplementation	8h
46	Client Funktionsimplementation	8h
47	Client Funktionsimplementation	8h
48	Testung der Verschlüsselung	8h
49	Implementation Verschlüsselung	8h
50	Vorbereitung für erste Präsentation	6h
51	Implementation Verschlüsselung	8h
52	-	
1	-	
2	Implementation Verschlüsselung	7h
3	Vorbereitung Tag der öffenen Tür	6h
4	Testung API Client	8h
5	Implementation API Client	8h
6	Implementation API Client	8h
7	-	
8	Implementation localer Schlüsselcontroller	8h
9	Implementation localer Schlüsselcontroller	8h
10	Implementation localer Schlüsselcontroller	6h
11	Funktionalitäts Verbesserung	8h
12	Verfassen der Diplomarbeit	8h
13	Verfassen der Diplomarbeit	8h
14	Verfassen der Diplomarbeit	8h