Zürcher Hochschule
für Angewandte Wissenschaften

Pretty Secure Cloud

Zentrale Verschlüsselung für Cloud-Dienste

Lösungsarchitektur

Autoren: Guerotto Sandro, Kindle Tristan, Sevimli Ridvan, Sieber Lorenz, Waldburger Safiyya und Walser Christoph Marjan Markus

Modul: Software Projekt 3, Frühlingssemester 2022

Dozierende: Liebhart Daniel, Schnatz Nina Isabelle

18. Mai 2022

Abstract

Various file hosting services, also called cloud storage, such as Microsoft's OneDrive or Apple's iCloud, offer flexible online file storage options. Those services, however, do not encrypted the stored files automatically and, thus, if required, the user must encrypt the files before uploading them. Accordingly, when the file needs to be downloaded afterwards, the file must be decrypted as well to be edited. Providing a convenient alternative for the described manual encryption and decryption process, Pretty Secure Cloud (hereinafter PSC) was developed. PSC is an automatic encryption and decryption service for files that are to be stored in a third-party cloud service. However, the encryption and decryption functionality of PSC may also be used for documents stored on any other local storage device. Furthermore, PSC's interface provides an overview of all files currently stored on a linked storage location and their encryption status. In other words, PSC acts as single point of contact for the user to access documents, while automatically handling the encryption and decryption in the background.

Inhalt

1 Projektidee

1.1 Hintergrund

Verschiedene File-Hosting-Dienste, auch Cloud-Speicher genannt, wie OneDrive von Microsoft oder iCloud von Apple, ermöglichen eine online Dateispeicherung [1]. Diese bieten sich insbesondere dann an, wenn die zu speichernden, digitalen Dateien orts- und geräteunabhängig verfügbar sein sollen. Ein Bedürfnis, das insbesondere durch die vergangene COVID-Pandemie an Geltung gewonnen hat, da Arbeitnehmer_innen aus diversen Branchen von zu Hause ausarbeiten mussten *(home office)* [2].

Die eingangs erwähnten Dienste konzentrieren sich allerdings nicht auf die Verschlüsselung der gespeicherten Dateien, so dass der/die Nutzer_in bei Bedarf die Dateien vor dem Hochladen verschlüsseln muss. Wenn die Dateien anschließend heruntergeladen werden sollen, müssen diese entsprechend auch entschlüsselt werden, damit sie wieder bearbeitbar werden. Wird auf eine Verschlüsselung verzichtet, so sind diese im Falle eines unerlaubten Zugriffs einseh- und insbesondere verwendbar [3]. Um die digitalen Dateien besser zu schützen und als automatische Alternative für den beschriebenen manuellen Ver- und Entschlüsselungsprozess wurde Pretty Secure Cloud (im Folgenden PSC) entwickelt. PSC ist ein automatischer Verschlüsselungs- und Entschlüsselungsdienst für Dateien, die in einem Cloud-Dienst eines Drittanbieters (oder einem anderen lokalen Speicher) gespeichert werden. Wie in Abbildung 1 ersichtlich ist, interagiert der/die Benutzer_in nicht mehr mit seinen/ihren Online-Dateispeicherdiensten («Cloud»), sondern direkt mit PSC, welche sich automatisch um das Verschlüsseln und Hochladen sowie das Herunterladen und Entschlüsseln kümmert.

Abbildung 1 Funktionsweise der PSC

1.2 Kundennutzen

Die durch die Verschlüsselung gewonnene Datensicherheit («Datenschutz und Privatsphäre») kann mit PSC in bestehende Arbeitsprozesse integriert werden («Integrierbarkeit»). Wichtig ist, dass der Verschlüsselungsprozess nach Auswahl der gewünschten Sicherheitsstufe («Individualisierbarkeit»), im Hintergrund abläuft, damit der/die Benutzer_in sich auf andere Aufgaben konzentrieren kann («Benutzerfreundlichkeit»). Hinsichtlich Verschlüsselung setzt PSC auf bewährte und bekannte Algorithmen («Transparenz»), wobei die Programmstruktur mit zukünftigen Algorithmen ergänzbar ist («Langlebigkeit»). Die erwähnten Nutzen sind in Abbildung 2 grafisch dargestellt.

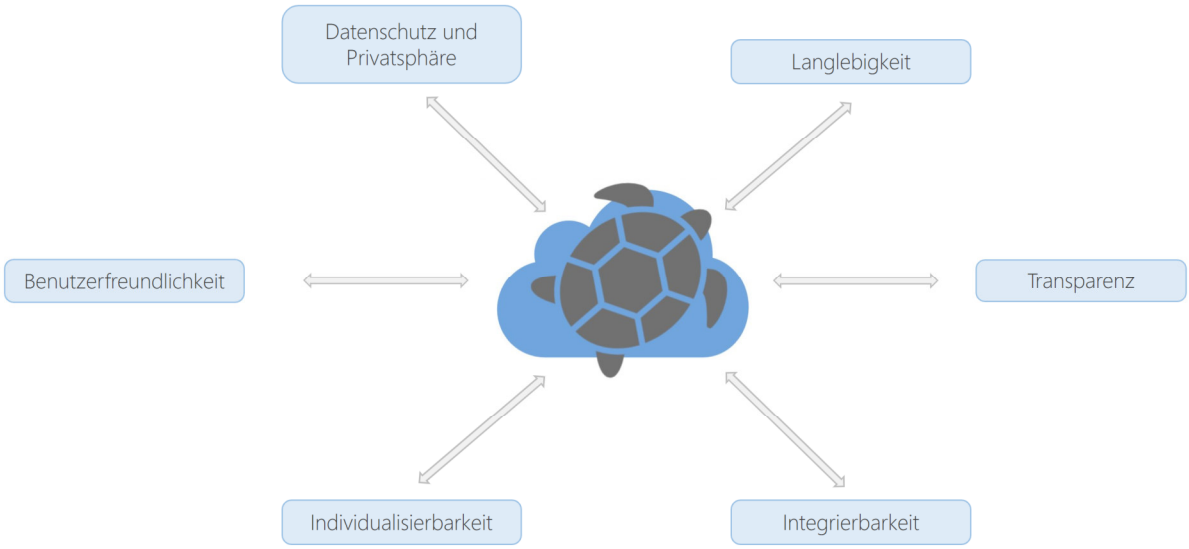


Abbildung 2 Übersicht der Nutzen

1.3 Alleinstellungsmerkmale

Es gibt aktuell diverse Konkurrenzanbieter, welche ebenfalls Lösungen für die Ver- und Entschlüsselung von Dateien anbieten [4]. Die Spannweite reicht von kostenpflichten Tools bis hin zu kostenlosen und insbesondere *open source* Tools. Die Alleinstellungsmerkmale von PSC sind:

- Minimalistische Benutzeroberfläche für eine intuitive Benutzung der Applikation.
- Transparente und verständliche Hilfestellungen innerhalb der Applikation.
- *open source*, d.h. der Quellcode von PSC wird öffentlich einsehbar sein.
- Auswahl der gewünschten Sicherheitsstufe resp. des anwendbaren Verschlüsselungsalgorithmus.
- Option zur Verschlüsselung von lokal gespeicherten Dateien.
- Fokus auf bekannte Verschlüsselungsalgorithmen.
- Neue Verschlüsselungsalgorithmen können zusätzlich implementiert werden.

1.4 Kontextszenario

Die PSC Anwendung richtet sich an alle Nutzer_innen, welche sensible Daten nicht ohne zusätzlichen Schutz in einem Cloud-Speicher sichern möchten. PSC hat eine eigene Benutzeroberfläche, soll aber auch ohne Benutzeroberfläche, im betriebssystemeigenen Dateiverwaltungsprogramm, verwendbar sein. Die Verwendung von PSC soll im nachfolgenden Kontextszenario nähergebracht werden.

Die Personalberaterin Ursula Günther (55 Jahre) möchte zuhause die Bundesordner mit all ihren Steuer- und Bankdokumenten digitalisieren, um für ihre neuen Zimmerpflanzen Platz zu schaffen. Schnell merkt sie, dass mit dem Einscannen der Dokumente, die Arbeit noch nicht getan ist. Wenn ihr Computer kaputt gehen würde, wären alle Dokumente verloren. Nach Recherchen im Internet stösst sie auf die 3-2-1 Regel[1] für Backups und möchte diese nun umsetzen. Doch Ursula will ihre vertraulichen Dokumente nicht ohne weiteres auf den ausländischen Servern ihres Cloud-Speicheranbieters speichern. Hier kommt PSC ins Spiel. PSC kennt Ursula Günther schon, denn sie verwendet sie auch im Büro, um vertrauliche Dokumente Mitarbeitenden im Homeoffice zur Verfügung zu stellen.

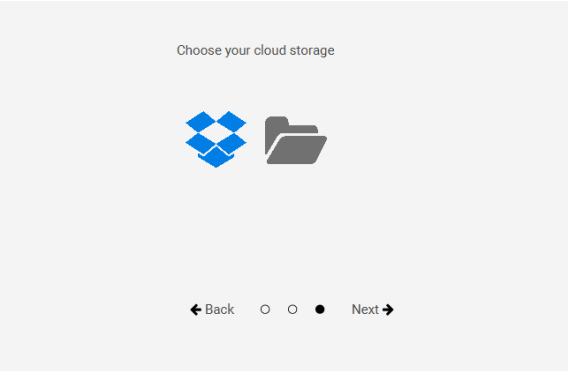


Abbildung 3 Auswahl Cloudanbieter

Ursula Günther wird vom Installationsassistenten durch die Installation von PSC geführt. Als Erstes muss Ursula Günther ihren Account registrieren. Hierzu gibt sie einen Benutzernamen, eine E-Mailadresse sowie ein Passwort ein. Im zweiten Schritt (siehe Abbildung 3) wählt sie den gewünschten Cloud-Speicherdienst aus. Im dritten und letzten Schritt kann sie die gewünschte Verschlüsselungsart auswählen. Es stehen verschiedene Verschlüsselungsalgorithmen zur Auswahl. So kann Ursula auswählen, ob sie den Fokus auf eine schnelle Ver- und Entschlüsselung oder auf erhöhte Sicherheit legen möchte. Da Ursula Günther ihre Dokumente nicht täglich entschlüsseln und öffnen muss, wählt sie die höchste Sicherheitsstufe. Im Anschluss erscheint im Fenster ihre Ordnerstruktur, welche derjenigen in ihrem Online. Die Steuer und Bankdokumente kann sie nun per Drag-and-drop in den gewünschten Ordner legen. Danach werden die Dokumente lokal auf ihrem Rechner verschlüsselt und in die Cloud hochgeladen. Wie auf den Abbildung 8 dargestellt, kann sie den Status des Verschlüsselns und Hochladen mithilfe der Fortschrittleiste im Interface der PSC nachverfolgen.

[1] Die 3-2-1 Regel sieht folgende Strategie vor: Drei Kopien aller kritischen Daten auf zwei unterschiedliche Medien/Datenträger, wobei eine Kopie der Daten an einem geografisch getrennten Standort gelagert werden soll [5].

Möchte Ursula ihre verschlüsselten Dokumente ansehen, kann sie die PSC öffnen, das gewünschte Dokument markieren und auf den «Herunterladen» Knopf drücken (siehe Abbildung 4). Das verschlüsselte Dokument wird nun heruntergeladen, entschlüsselt und in einem vordefinierten Verzeichnis abgespeichert.

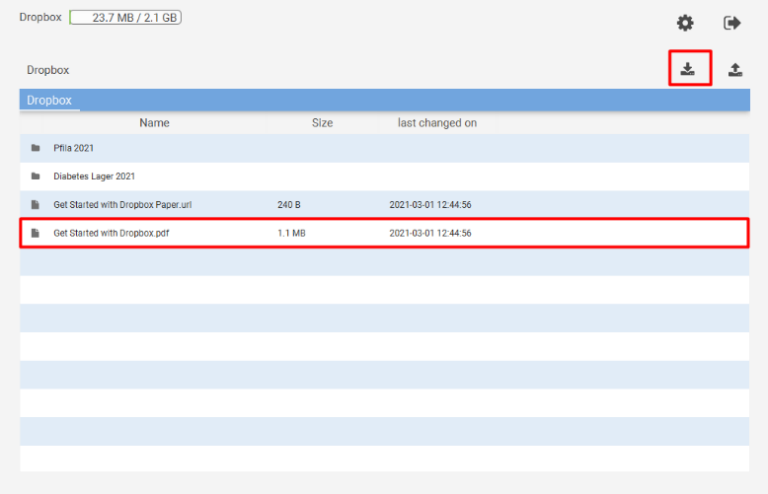


Abbildung 4 Auswahl des Dokumentes und Herunterladen Button (rot markiert)

2 Analyse

In diesem Kapitel finden sich Details zu allen identifizierten Anforderungsspezifikationen von PSC. Die Auflistung bezieht sich auf den Entwicklungszeitraum der Betaversion.

2.1 Use-Case

Im folgenden Abschnitt werden die identifizierten Use-Cases «Verschlüsseln und Upload» sowie «Download und Entschlüsseln» ausführlich *(d.h. fully dressed)* erläutert, sowie die dazugehörigen UML-Diagramme präsentiert. Die beiden Use-Cases werden neben dem gemeinsamen Use-Case-Diagramm jeweils mittels eines System-Sequenzdiagramms, eines UI-Sketches sowie eines Bildes der Benutzeroberfläche gemäss Betaversion ergänzt (der ursprüngliche UI-Sketch soll als Referenzpunkt dienen). Der Use-Case «Registrierung eines neuen Accounts» wird als informeller Use-Case erläutert *(d.h. casual)*.

Die Interaktion zwischen Benutzer_in und PSC wird im nachfolgenden Use-Case-Diagramm aufgezeigt (Abbildung 5). Daraus wird ersichtlich, dass die Hauptanwendungsfälle für die Benutzung «Verschlüsselung und Upload» und «Download und Entschlüsselung» sind. Weiter ist erkennbar, dass der/die Benutzer_in das System initialisieren und gewisse Einstellung vornehmen muss («Set up»).

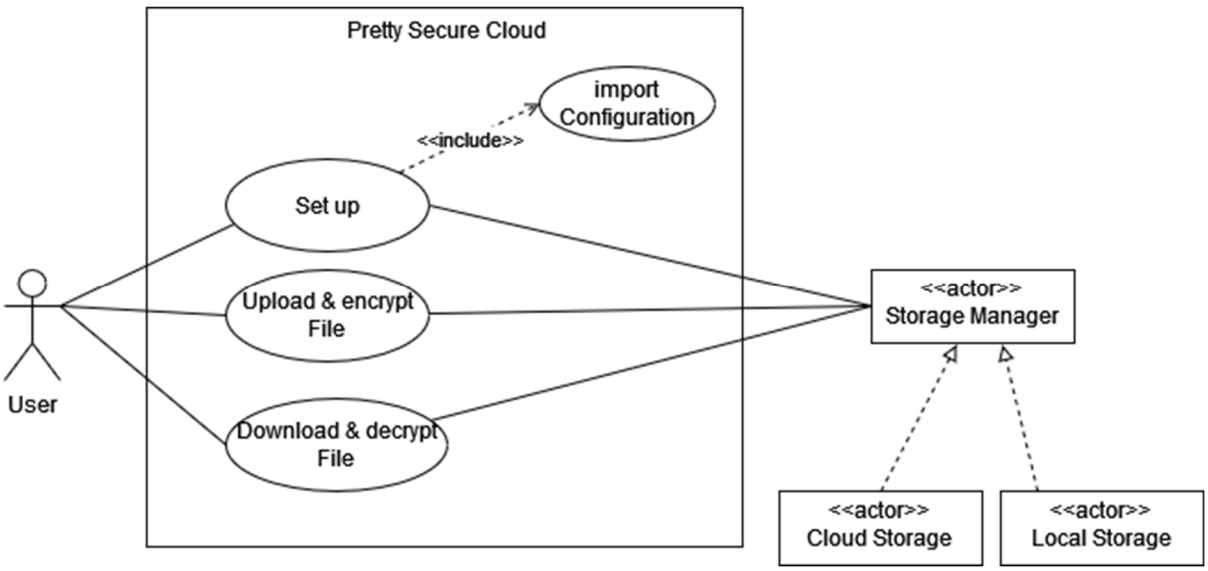


Abbildung 5 Use-Case-Diagramm

2.1.1 Verschlüsselung und Upload

2.1.1.1 Beschreibung

Im Use-Case «Verschlüsselung und Upload» geht es um den Prozess, welcher durchlaufen werden muss, um ausgewählte Dateien mittels der PSC verschlüsseln und auf einen ausgewählten Cloud-Dienst hochzuladen.

Tabelle 1 Use-Case 1 - Verschlüsseln und Upload

Primärakteur_in	Cloudbenutzer_in
Stakeholder / Interessen	Cloudbenutzer_in will: • Daten verschlüsseln • Dateien schnell und problemlos auf Cloud laden
Vorbedingungen	• Benutzer_in ist identifiziert und authentifiziert vom System • Gültige Logindaten bei Cloud-Dienst • Funktionierende Internetverbindung
Nachbedingungen	• Die Datei wurde verschlüsselt in einen Cloud-Dienst hochgeladen • Benutzer_in ist bewusst, dass Dateien nur mit dem entsprechenden Schlüssel wieder entschlüsselt werden können
Hauptablauf	• Benutzer_in wählt Dateien zum Verschlüsseln und Hochladen aus [Alt1: Dateien mit selben Namen befinden sich in dem Cloud-Verzeichnis] • Das System erhält die nötigen Angaben, um die Dateien zu verschlüsseln und auf die Cloud zu laden. • System prüft Cloud-Anbindung [Alt2: Zugriff auf Cloud nicht möglich] • System benachrichtigt Benutzer_in über den Status des Uploads • System bietet Möglichkeit weitere Dateien hochzuladen [Alt 3: Benutzer_in möchte weitere Dateien hochladen] oder Use Case zu verlassen [Use Case beendet]
Alternative Abläufe	Alt1: • System informiert Benutzer_in, dass Dateien mit selben Namen sich in Cloud-Verzeichnis befinden • System bietet Möglichkeit, Dateien, auf der Cloud zu überschreiben, eindeutige Name zu vergeben oder die Aktion abzubrechen. [Use Case beendet] • Hauptablauf Punkt 2 Alt 2: Benutzer_in wird informiert über fehlerhafte Cloud-Anbindung [Use Case beendet] Alt 3: Hauptablauf, Punkt 1
Spezielle Anforderungen	• Upload Möglichkeit via Cloud API • Sichere Aufbewahrungsmethode für Schlüssel
Häufigkeit des Auftretens	Abhängig von Intensität der Cloud-Nutzung

2.1.1.2 UI-Sketch

Im nachfolgenden UI-Sketch wird der behandelte Use-Case anhand von beispielhaften Skizzen des möglichen Benutzeroberfläche visualisiert (Abbildung 6). Der/die Benutzer_in gelangt zum Hauptfenster (links), welches eine Übersicht über die hochzuladenden Dateien inkl. Statusanzeige gibt. Dort ist ersichtlich, dass die Selektion der hochzuladenden Dateien via Drag-and-Drop vorgenommen werden kann.

Abbildung 6 UI-Sketch Verschlüsselung und Upload

In den folgenden Grafiken ist der aktuelle Stand der graphischen Benutzeroberfläche ersichtlich. Im Unterschied zum UI-Sketch, wurde in der Betaversion das gesamte Fenster als Bereich für das Hochladen von Dateien via Drag-and-Drop implementiert (Abbildung 7).

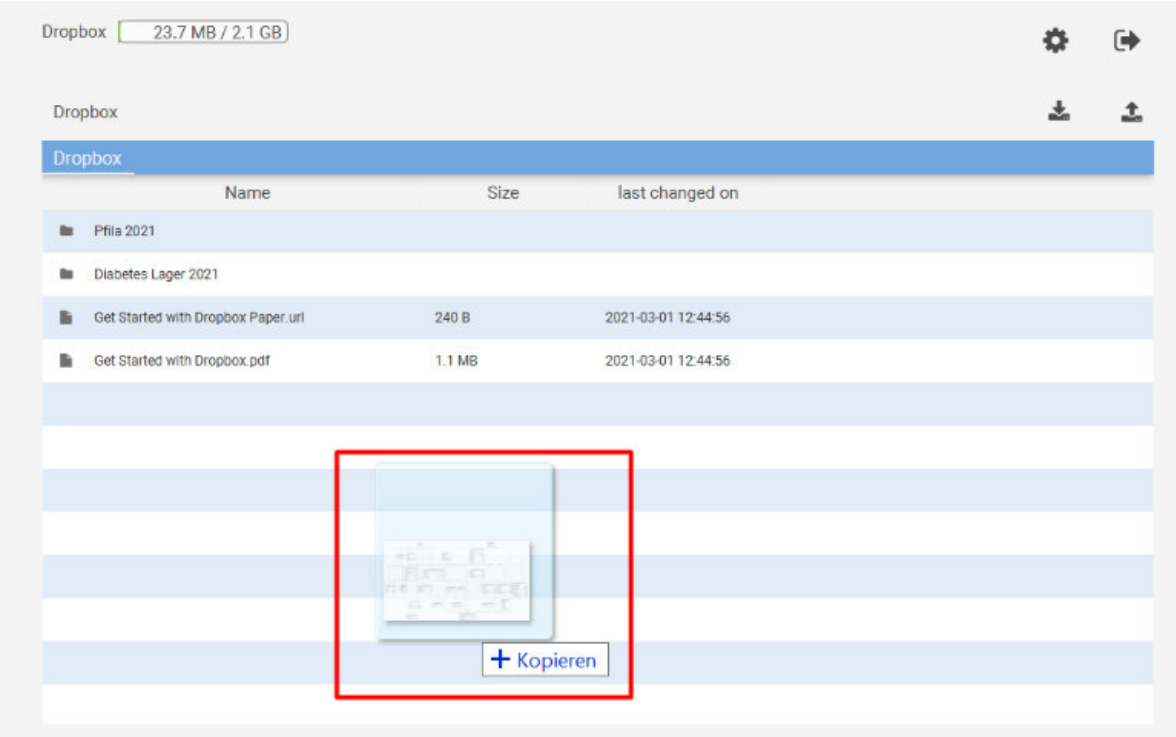


Abbildung 7 Platzieren einer Datei mit Drag and Drop

Ebenfalls wurde die Statusanzeige durch eine Benachrichtigung in Textform ersetzt (Abbildung 8). In zukünftigen Versionen wäre denkbar, dass ein Verlauf der bisherigen Statusänderungen aufgerufen werden kann.

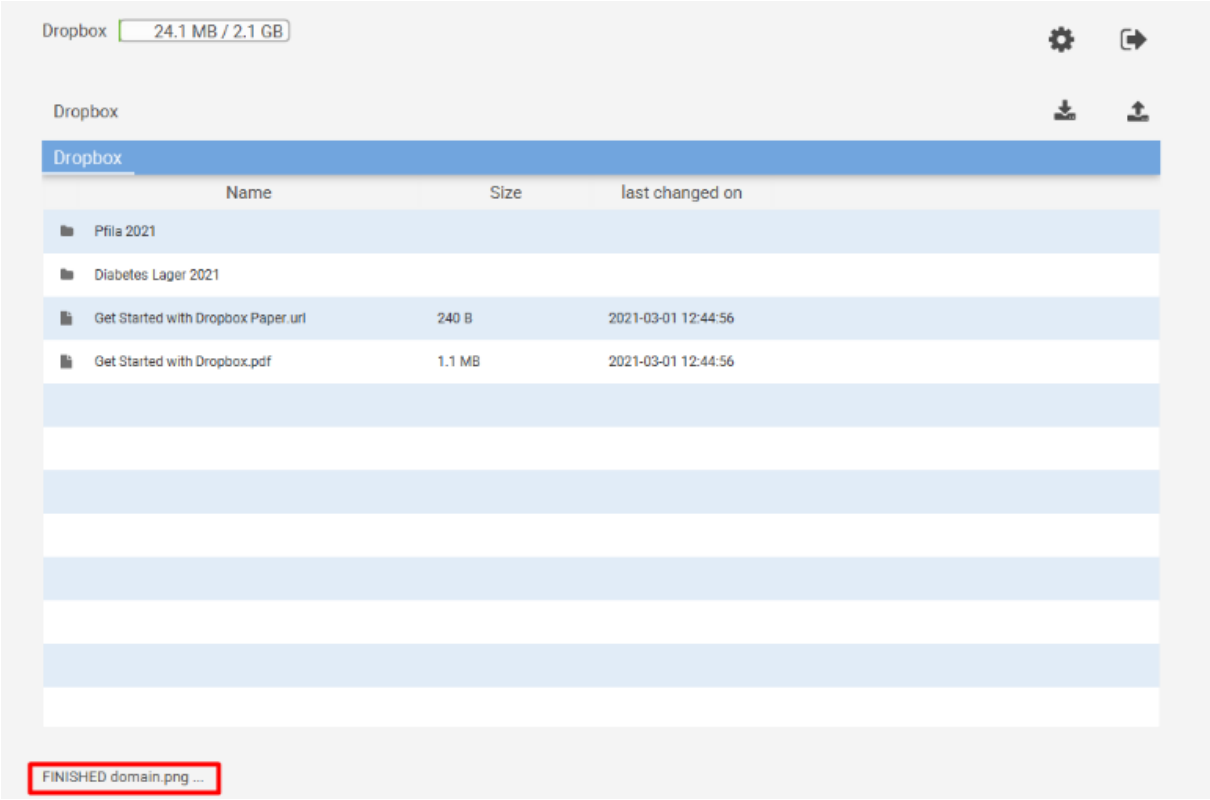


Abbildung 8 Benachrichtigung über erfolgreichen Upload

2.1.1.3 Sequenzdiagramm

Das nachfolgende Sequenzdiagramm veranschaulicht den Use-Case «Verschlüsselung und Upload» mit Blick auf das Verhalten der involvierten Systeme untereinander (Abbildung 9). Im Zentrum stehen insbesondere die sogenannten Lebenslinien eines Systems (vertikal), welche die Aktivität des jeweiligen Systems visualisieren, d.h. wann sie aktiv werden (längliche Rechtecke) respektive inaktiv bleiben, sofern sie nicht interagieren (gestrichelte Linien).

Das Diagramm wird grundsätzlich von oben nach unten gelesen, beginnend beim «User» (oben links), welcher eine oder mehrere Dateien für die Verschlüsselung und den Upload im «Client» selektiert. Der «Client» gibt die erhaltene Datei an den «StorageManager» weiter, welcher sich nun zentral um den Prozess kümmert. Dieser übergibt die Datei zunächst an den «Encrypter», welcher versucht die Datei zu verschlüsseln. Das Resultat wird dann an den «StorageManager» zurückgegeben, welcher die Verschlüsselung dem «Client» bestätigt oder ablehnt. Im letzteren Fall kommt der Vorgang bereits zu einem Ende, da keine verschlüsselte Datei für den Upload zur Verfügung steht. War die Verschlüsselung erfolgreich, so leitet der «StorageManager» die verschlüsselte Datei an den «CloudService» weiter (Upload). Der «StorageManager» wartet dann auf eine Antwort vom «CloudService», ob der Upload erfolgreich war oder nicht und übergibt die erhaltene Information zurück an den «Client», welcher den Status

für die entsprechende Datei aktualisiert. Hervorzuheben ist, dass der/die Benutzer_in nach dem Drag and Drop der Datei keine weiteren Interaktionen mit den involvierten Systemen vornehmen muss. Grundsätzlich kann der/die Benutzer_in aber jederzeit den Status der Verschlüsselung und des Uploads im «Client» nachvollziehen.

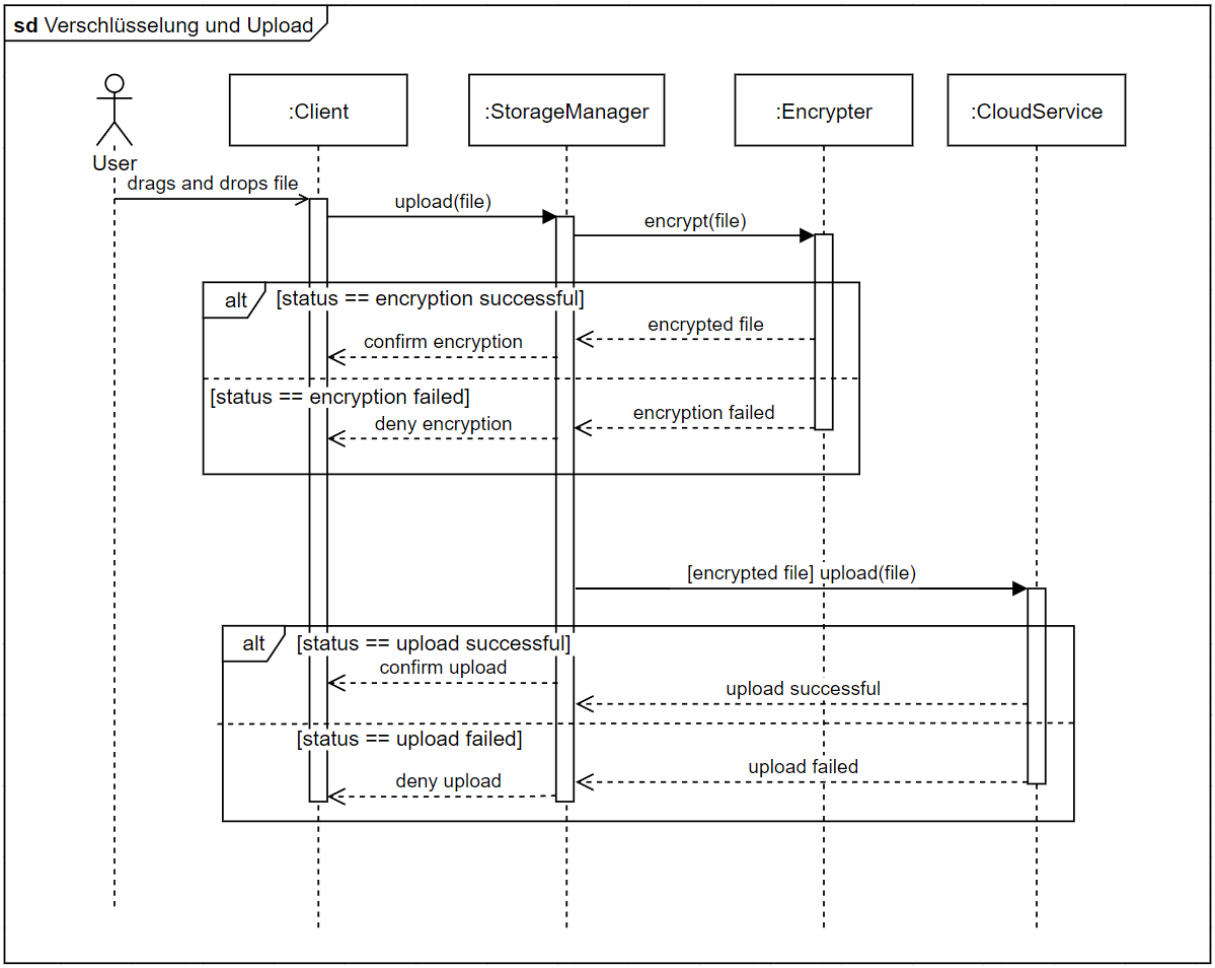


Abbildung 9 Sequenzdiagramm: Verschlüsselung und Upload

2.1.2 Use Case 2: Download und Entschlüsselung

2.1.2.1 Beschreibung

Im Use-Case «Download und Entschlüsselung» geht es um den Prozess, den ein/e Benutzer_in durchlaufen muss, wenn bereits verschlüsselte und hochgeladene Dateien (siehe Abschnitt 2.1.1) mittels PSC vom jeweiligen Speicherort heruntergeladen und entschlüsselt werden sollen.

Tabelle 2 Use-Case 2 - Download und Entschlüsselung

Primärakteur_in	Cloudbenutzer_in
Stakeholder / Interessen	Cloudbenutzer_in will: • Zugriff auf seine/ihre Dateien • Dateien entschlüsseln.
Vorbedingungen	• Benutzer_in ist identifiziert und authentifiziert vom System • Gültige Logindaten bei Cloud-Diensten • Gültiger Schlüssel für Entschlüsselung
Nachbedingungen	Entschlüsselte Dateien sind an gewünschtem Ort gesichert
Hauptablauf	• System prüft Cloudanbindung [Alt1: Zugriff auf Cloud nicht möglich] • System zeigt verschlüsselte Dateien/Ordner, die sich auf der Cloud befinden • Benutzer_in wählt Dateien für den Download und die Entschlüsselung • Das System erhält die nötigen Angaben, um die Dateien zu entschlüsseln und lokal zu speichern. • System benachrichtigt Benutzer_in über erfolgreichen Download und Entschlüsselung • System bietet Möglichkeit weitere Dateien runterzuladen [Alt 3: User möchte weitere Dateien hochladen] oder Use Case zu verlassen [Use Case beendet]
Alternative Abläufe	Alt 1: Benutzer_in wird informiert über fehlerhafte Cloudanbindung [Use Case beendet] Alt 2: • System informiert Benutzer_in, dass Dateien mit selben Namen sich in dem lokalen Ordner befinden • System bietet Möglichkeit, lokale Dateien zu überschreiben, eindeutige Name zu vergeben oder die Aktion abzubrechen. [Use Case beendet] • Hauptablauf Punkt 2 Alt 3: Hauptablauf, Punkt 1
Spezielle Anforderungen	• Downloadmöglichkeit via API der Clouddienste • Sichere Aufbewahrungsmethoden für Schlüssel
Häufigkeit des Auftretens	Abhängig von Intensität der Cloudnutzung

2.1.2.2 UI-Sketch

In dem nachfolgenden UI-Sketch wird der behandelte Use-Case anhand von beispielhaften Skizzen des möglichen Benutzerinterfaces visualisiert (Abbildung 10). Beginnend links, wählt der/die Benutzer_in die Dateien aus, welche vom aktuellen Speicherort heruntergeladen und entschlüsselt werden sollen (dargestellt mit einem grünen Rechteck). Im nächsten Fenster (rechts) erhält der/die Benutzer_in eine Übersicht über die herunterzuladenden Dateien inkl. Statusanzeige. Während dem Prozess, wird der Fortschritt für den Download und die Entschlüsselung unten rechts im Fenster mittels Statusanzeigen visualisiert.

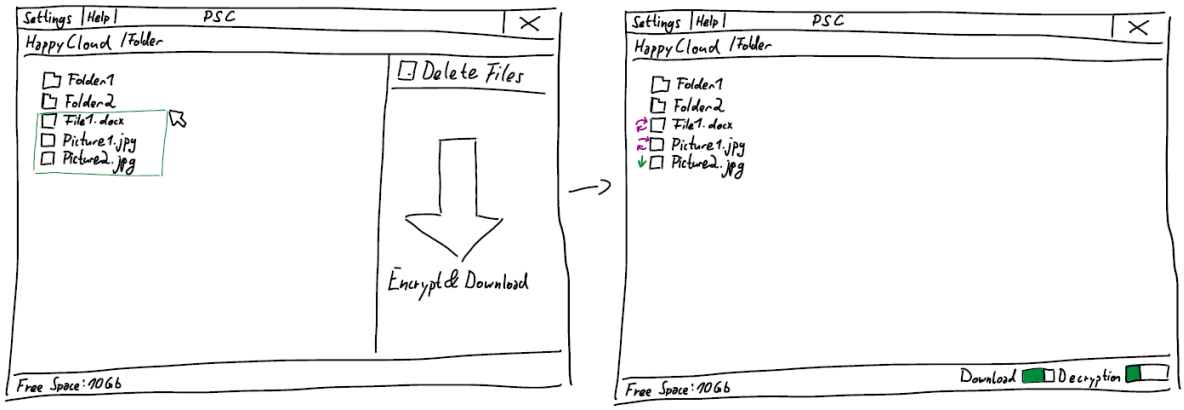


Abbildung 10 UI-Sketch Download und Entschlüsselung

Die nachfolgenden Grafiken widerspiegeln den aktuellen Stand der Benutzeroberflächen von PSC in der Betaversion. Im Unterschied zum UI-Sketch, wurde für die Betaversion die Drag-and-Drop Funktionalität beim Download nicht implementiert.

Abbildung 11 Auswahl der Datei (Dunkelblau hinterlegt)

Die Datei, welche heruntergeladen werden sollen, können allerdings ausgewählt (Abbildung 11) und mit einem anschliessenden Klick auf das Downloadsymbol (siehe rote Markierung in Abbildung 12) heruntergeladen und entschlüsselt werden. Analog zum Use-Case «Verschlüsselung und Upload» wurde die Statusanzeige ebenfalls als Textfeld implementiert (siehe unten rechts in Abbildung 12).

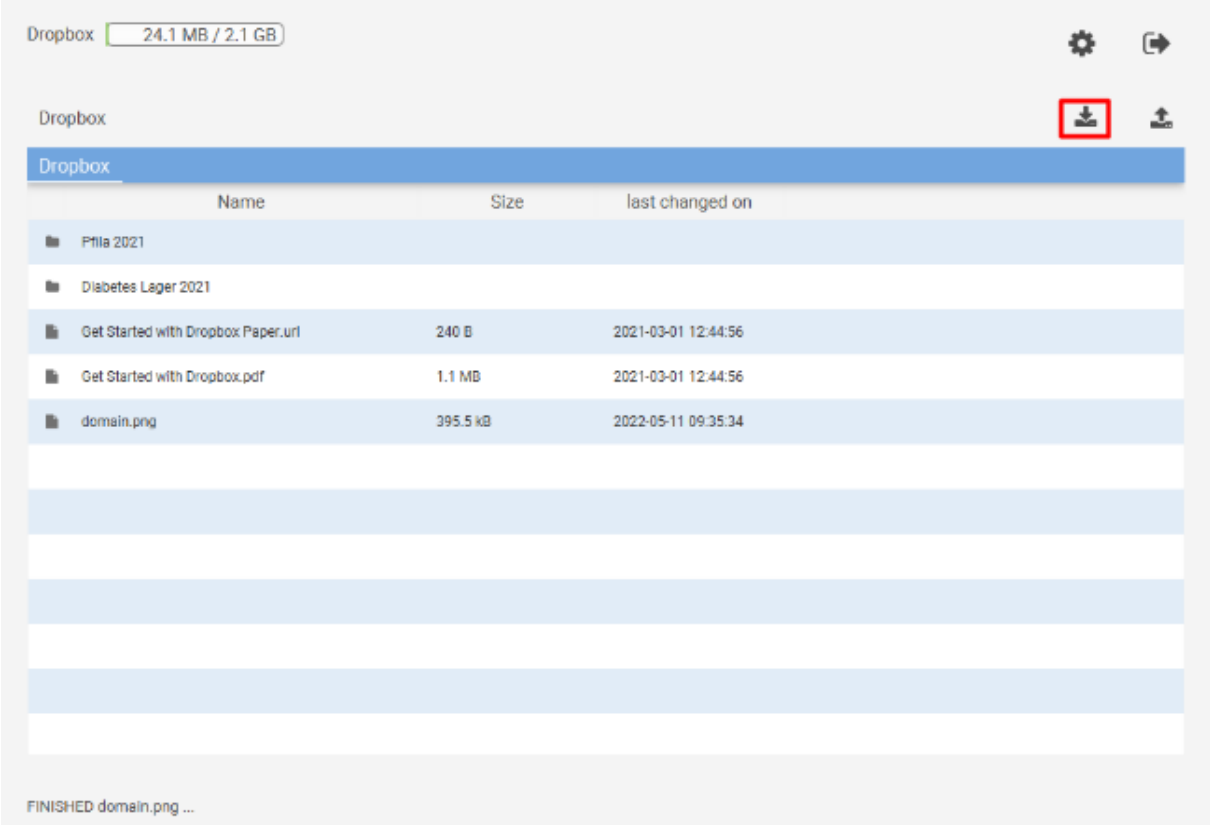


Abbildung 12 Download Button (rot markiert) und Benachrichtigung bei erfolgreichem Download

2.1.2.3 Sequenzdiagramm

Das nachfolgende Sequenzdiagramm veranschaulicht den Use-Case «Download und Entschlüsselung» mit Blick auf das Verhalten der involvierten Systeme untereinander (Abbildung 13). Im Zentrum stehen insbesondere die sogenannten Lebenslinien eines Systems (vertikal), welche die Aktivität des jeweiligen Systems visualisieren, d.h. wann sie aktiv werden (längliche Rechtecke) respektive inaktiv bleiben, sofern sie nicht interagieren (gestrichelte Linien).

Das Diagramm wird grundsätzlich von oben nach unten gelesen, beginnend beim «User» (oben links), welcher eine oder mehrere Dateien für den Download und die Entschlüsselung im «Client» selektiert. Der «Client» gibt die erhaltene Dateiauswahl an den «StorageManager» weiter, welcher sich nun zentral um den Prozess kümmert. Dieser übergibt die Dateiauswahl zunächst an den «CloudService» und beantragt den Download der angeforderten Dateien. Das Resultat des Downloads wird anschliessend an den «StorageManager» zurückgegeben, welcher den Download dem «Client» bestätigt oder ablehnt. Im letzteren Fall kommt der Vorgang bereits zu einem Ende, da keine verschlüsselten Dateien für die Entschlüsselung zur Verfügung stehen. War das Herunterladen erfolgreich, so leitet der «StorageManager» die verschlüsselte Datei an den «Decrypter» weiter. Der «StorageManager» wartet dann auf eine Antwort

vom «Decrypter», ob die Entschlüsselung erfolgreich war oder nicht und übergibt die erhaltene Information zurück an den «Client», welcher den Status für die entsprechende Datei aktualisiert. Hervorzuheben ist, dass der «User» nach dem Selektieren der Dateien keine weiteren Interaktionen mit den involvierten Systemen vornehmen muss. Grundsätzlich kann der «User» aber jederzeit den Status des Downloads und der Entschlüsselung im «Client» nachvollziehen.

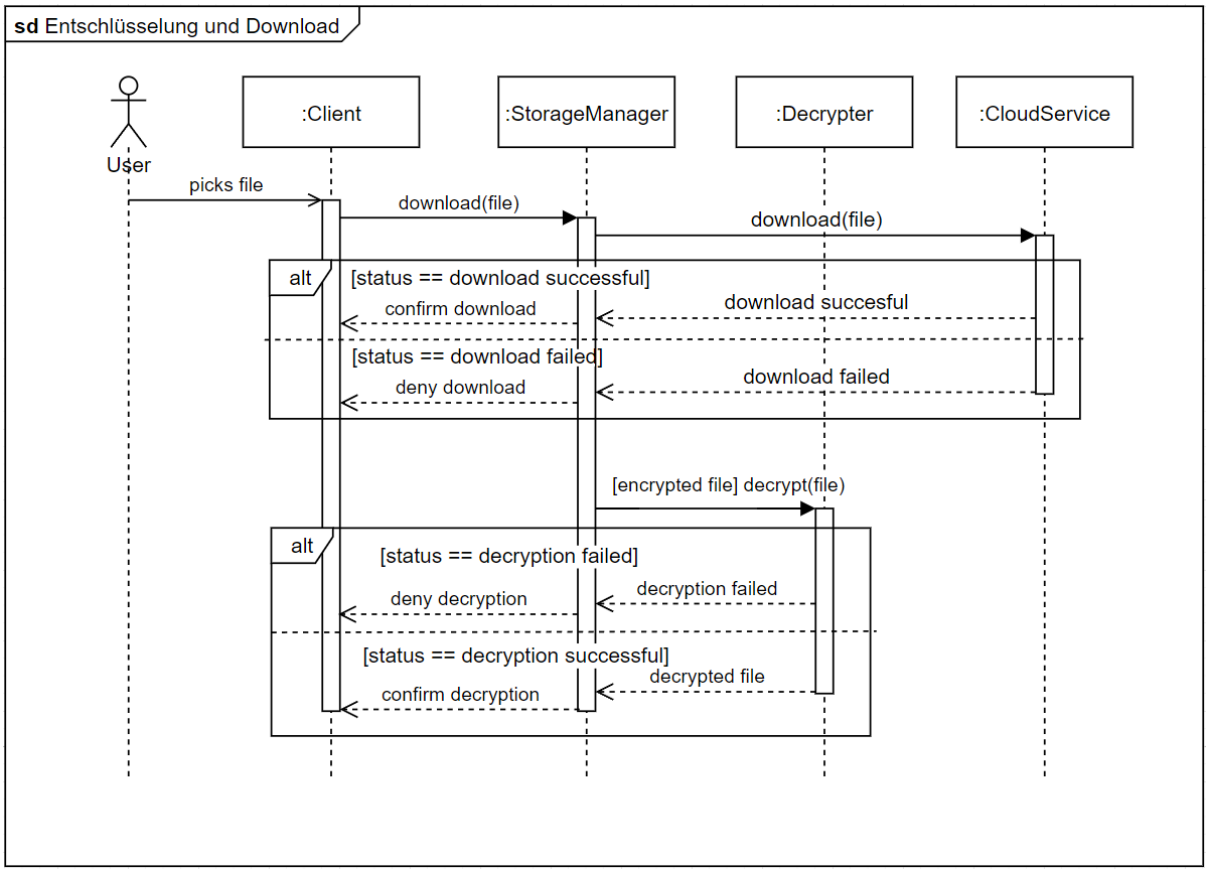


Abbildung 13 Sequenzdiagramm: Download und Entschlüsselung

2.1.3 Use-Case 3: Registrierung eines neuen Accounts

Für die Inbetriebnahme von PSC muss der/die Benutzer_in einen Speicherort definieren respektive seinen/ihren bestehenden Online-Dateispeicherdienst an PSC anbinden. Das Erfolgsszenario für die Registrierung eines neuen Accounts lässt sich wie folgt beschreiben:

- Zur Registrierung bei dem Cloudanbieter werden die Login-Daten angegeben.
- Nach dem Absenden und Überprüfen der Login-Daten wird dem/der Benutzer_in zurückgemeldet, ob die Registrierung erfolgreich war.
- Falls Registrierung nicht erfolgreich war, wird der/die Benutzer_in aufgefordert, Login-Daten nochmals einzugeben.

2.2 Zusätzliche Anforderungen

2.2.1 Funktionalität *(Functionality)*

Die Hauptfunktionalitäten der Anwendung ist das Ver- und Entschlüsseln von Dateien sowie das Hoch- und Herunterladen auf bzw. von verbundenen Speicherdiensten. Als zusätzliche Funktionen ist die Möglichkeit von ***Drag and Drop*** im Zusammenhang mit Dateibewegungen über die Benutzeroberfläche. Ausserdem wird der **Status** der erwähnten Hauptfunktionalität pro Datei in der Benutzeroberfläche visualisiert.

Um die Wiederverwendbarkeit der Dateien, z.B. wenn der spezifische Schlüssel für den verwendeten Verschlüsselungsalgorithmus verloren geht, zu gewährleisten, werden **nur gebräuchliche Verschlüsselungsalgorithmen** eingesetzt. Dadurch kann die Entschlüsselung der Dateien grundsätzlich auch von Drittanwendungen vorgenommen werden. Durch die Verwendung von bekannten und modernen Verschlüsselungsalgorithmen wird eine grösstmögliche **Datensicherheit** angestrebt.

2.2.2 Gebrauchstauglichkeit *(Usability)*

Die Benutzeroberfläche ist nach dem *«Keep it simple and stupid (KISS)»*-Prinzip aufgebaut. Somit wird gewährleistet, dass die Anwender_innen **keine Bedienungsanleitung** oder dergleichen benötigen, um die PSC-Applikation zu bedienen. Als Regel wird festgelegt, dass der/die Benutzer_in innert «zwei Minuten» die Anwendung installieren, einstellen und verwenden kann. Dies wird insbesondere durch einen **schlanken Aufbau der Benutzeroberflächen** erreicht.

2.2.3 Zuverlässigkeit *(Reliability)*

Durch die Verwendung von **ausschliesslich modernen Verschlüsselungsalgorithmen** wird ein zuverlässiger Verschlüsselungs- und Entschlüsselungsprozess für alle gebräuchlichen Dateitypen gewährleistet. Sollte zu einem gegebenen Zeitpunkt keine aktive Internetverbindung vorhanden sein, können die verschlüsselten **Dateien auch lokal gespeichert** werden. Bei Bedarf können diese zu einem späteren Zeitpunkt auf einen angebundenen Speicherdienst hochgeladen werden.

2.2.4 Performanz (*Performance*)

Um unerwünschte Verzögerung oder gar Abstürze während der Benutzung der Anwendung zu vermeiden, wird der Prozess der Ver- und Entschlüsselung sowie des Hoch- und Herunterladens in **einzelne Threads pro Datei** zerlegt. Wenn ein Prozess fehlschlägt, wird dies in der Benutzeroberfläche angezeigt und der/die Benutzer_in kann diesen erneut initiieren.

2.2.5 Unterstützbarkeit *(Supportability)*

Im gesamten Softwareentwicklungsprozess sollten fortlaufend Tests für die Klassen geschrieben werden, so dass bei Änderungen einer Klasse, diese direkt für die geforderte Funktionalität getestet werden kann. Bei jedem Antrag auf Programmänderung werden sämtliche **Tests automatisch von GitHub durchgeführt** und gegebenenfalls abgelehnt, sollte einer der vorhandenen Tests fehlschlagen (siehe Abschnitt 4.1).

2.3 Domänenmodell

Das Domänenmodell (Abbildung 14) repräsentiert die Abhängigkeiten in der Applikation. Eine zentrale Rolle in der Domäne nimmt der/die Benutzer_in («User») und die verknüpften Speicherdienste («StorageManager») ein. Vereinfacht formuliert, steuert der «User» über den «StorageManager» das Hochladen (inkl. Verschlüsselung) sowie das Herunterladen (inkl. Entschlüsselung) der ausgewählten Datei. Der «StorageManager» verwaltet die möglichen Speicherorte («CloudStorage» und «LocalStorage») und koordiniert in dieser Position das Hoch- und Herunterladen auf/von den erwähnten Speicherorten. Für jeden «User» können ein (symmetrische Verschlüsselung) oder zwei (asymmetrische Verschlüsselung) Schlüssel («Key») angelegt werden, welche sodann für alle Verschlüsselungs- und Entschlüsselungsvorgänge verwendet werden («Encryption» und «File»). Aus dem Domänenmodell ist weiter ersichtlich, dass für jede Datei der Verschlüsselungszustand gemerkt wird («EncryptionState»).

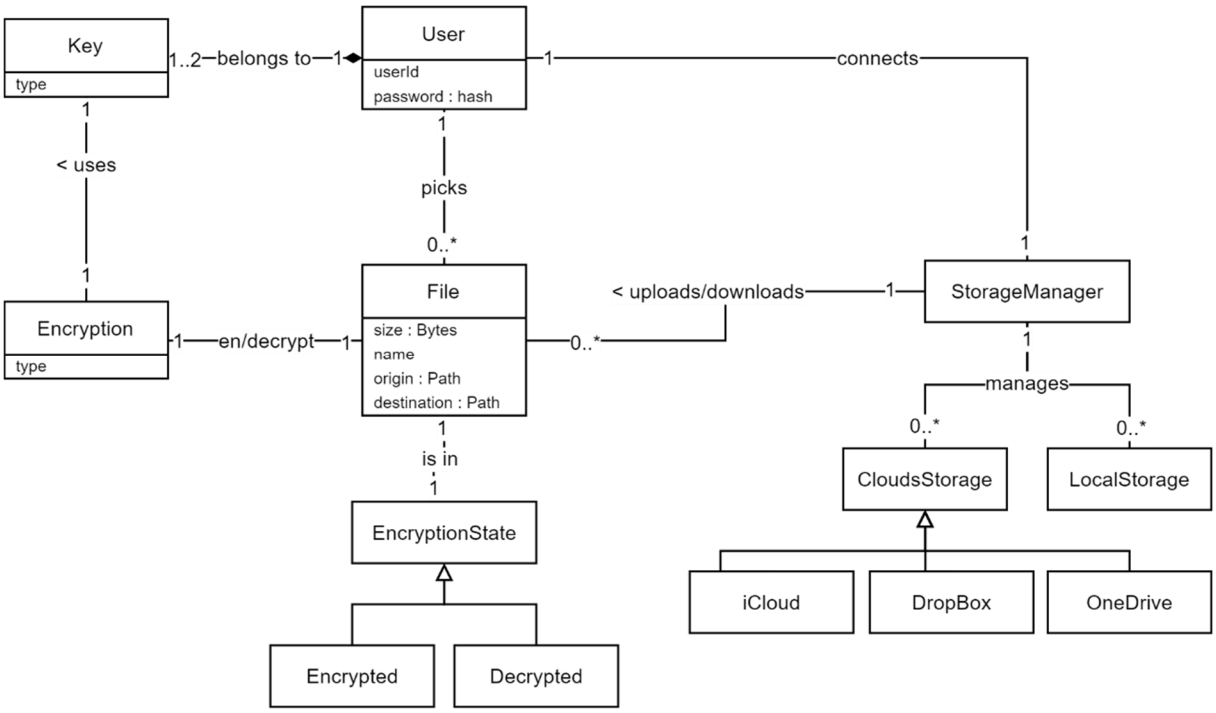


Abbildung 14 Domänenmodell Pretty Secure Cloud

3 Design

3.1 Softwarearchitektur

3.1.1 Package-Diagramm

Für die Applikation wird eine *three-tier-architecture* umgesetzt. Das heisst, die Applikation wird in drei Schichten aufgeteilt: Präsentation («presentation»), Logik («domain») und Datenhaltung («datasources»). Dabei darf jeweils nur die obere Schicht auf die untere zugreifen, wobei das Überspringen einer Schicht verboten ist. Dieses Architekturprinzip zeichnet sich dadurch aus, dass eine beliebige Schicht ausgetauscht werden kann und dies einen geringen oder bestenfalls keinen Einfluss auf die übrigen Schichten hat.

Das nachfolgende Package-Diagramm (Abbildung 15) visualisiert, die im obigen Absatz beschriebene Programmstruktur mit sogenannten *packages*, sowie deren Interaktionen bzw. Abhängigkeiten untereinander. Das Package-Diagramm gibt insbesondere Einsicht in die verschiedenen Schichten der Programmarchitektur und Auskunft über die verwendeten Frameworks. Die Anwendung als Ganzes wird mit dem *package* «ch.psc» dargestellt.

Ausgehend vom *package* «presentation» ist zu erkennen, dass das Java-Framework JavaFX für die Präsentationsschicht verwendet wird. JavaFX eignet sich insbesondere für die Umsetzung des angestrebten *«Model-View-Controller (MVC)»*-Patterns, welches nachfolgend erläutert wird. Der sogenannte *controller* ist für die Kommunikation zwischen den Benutzeroberflächen (*view*) und der Domäne (*model*) zuständig und wird als zentraler Programmteil im *package* «controller» innerhalb des *package* «presentation» abgelegt. Die Benutzeroberflächen (sogenannte *views*) werden als FXML-Dateien umgesetzt. Die Verwendung von FXML-Dateien für die Entwicklung der Benutzeroberflächen erlaubt den Gebrauch des grafischen Tools SceneBuilder von Gluon, welcher wiederum die Entwicklung erheblich unterstützen kann.

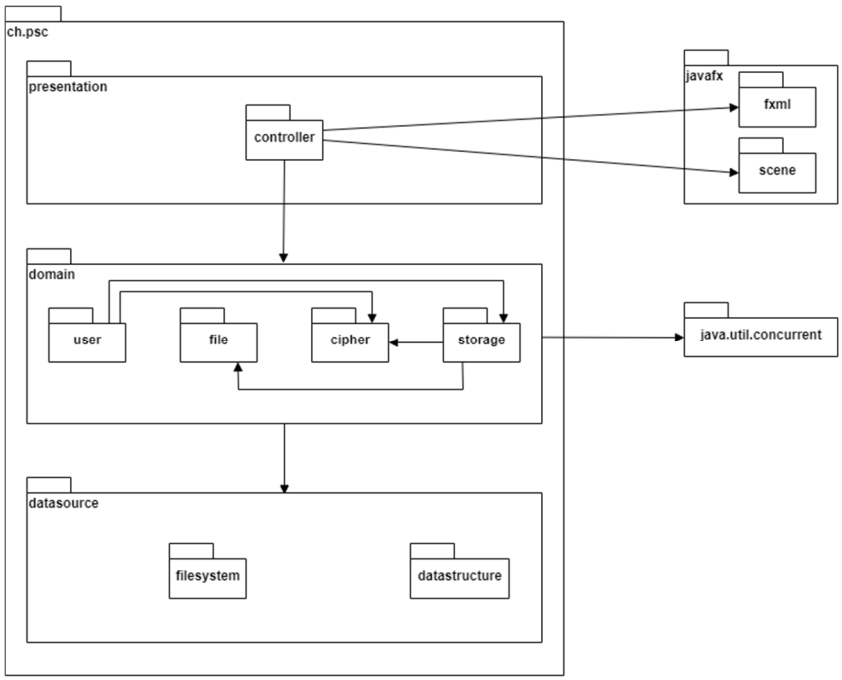


Abbildung 15 Package-Diagramm

3.1.2 Klassendiagramm

Das Klassendiagramm visualisiert die aktuelle Programmarchitektur mit Blick auf die verwendeten Klassen sowie deren Eigenschaften, Methoden und Interaktionen untereinander (Abbildung 16).

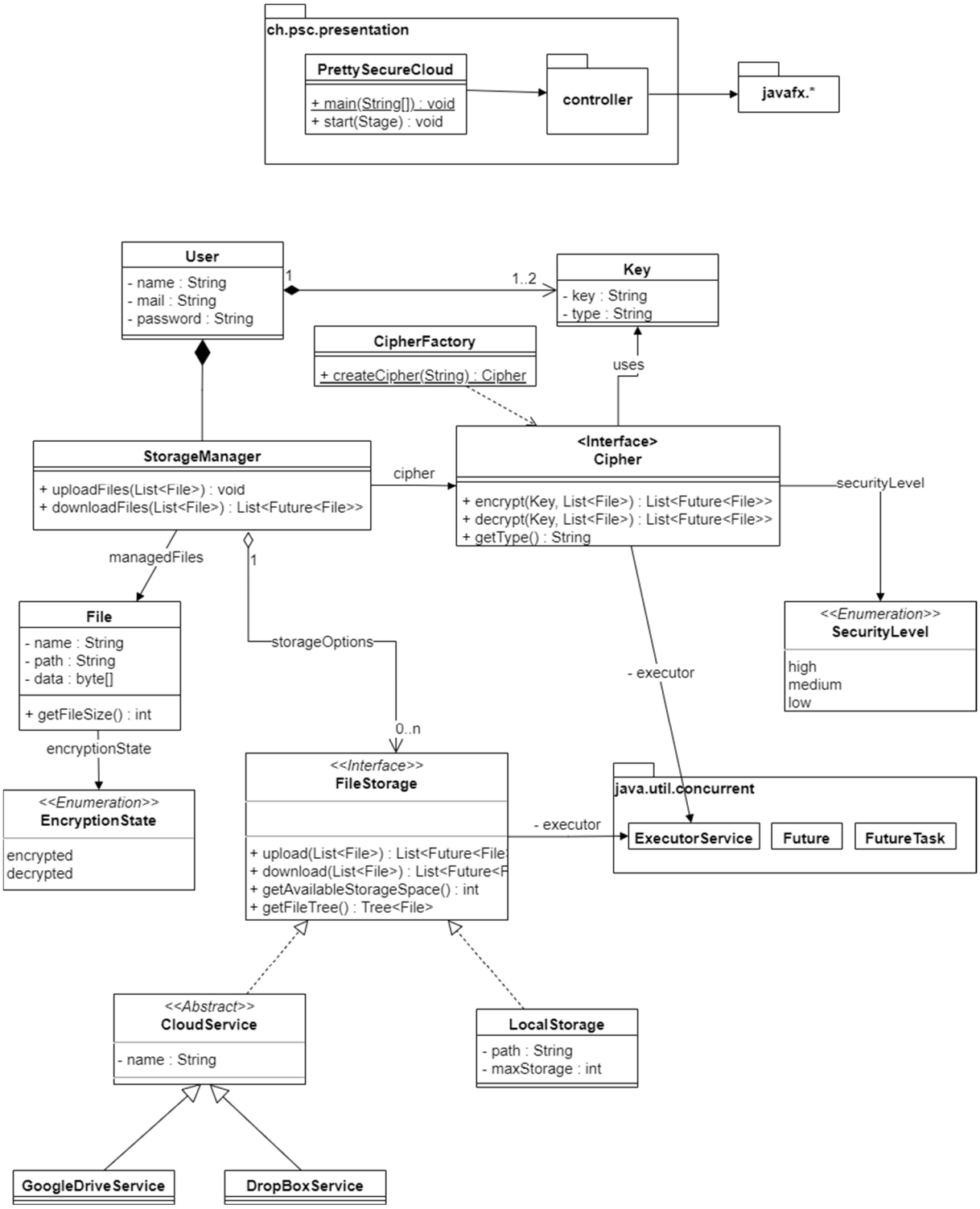


Abbildung 16 Klassendiagramm (Projektbeginn)

In der Logikschicht «domain» spielen sich mehrere Prozesse ab, welche über eine längere Zeit andauern können. Java bietet eine Bibliothek «java.util.concurrent», die geeignete Methoden beinhaltet, um diese Prozesse asynchron auszuführen und zu einem späteren Zeitpunkt wieder zusammenzuführen. Voraussichtlich wird «java.util.concurrent» an mehreren Stellen im Programm verwendet werden, wie bspw. von den Interfaces «Cipher» und «FileStorage».

Pro «User» existiert genau ein «StorageManager», welcher für den «User» die Dateien verwaltet. Somit kann die Dateiverwaltung von der Klasse «User» wegabstrahiert werden, vorausgesetzt eine starke Beziehung, sogenannte Komposition, zwischen den beiden Klassen existiert. Der «StorageManager» weiss, wo sich die unterschiedlichen Dateien «File» befinden, und gibt die Ver- oder Entschlüsselung der Dateien in Auftrag (durch Verwendung von verschiedenen «Cipher» Implementationen).

Um unterschiedliche Verschlüsselungsverfahren anzubieten, wird das Interface «Cipher» definiert. Der Schlüssel «Key» selbst beinhaltet die Verschlüsselungsmethode, sodass die «CipherFactory» mithilfe des Schlüssels automatisch die dazugehörige Implementation des «Ciphers» erzeugen kann [6].

Damit wird gewährleistet, dass ziemlich einfach neue Verschlüsselungsmethoden angeboten werden können, ohne Änderungen am restlichen Code vornehmen zu müssen.

Das effektiv implementierte Klassendiagramm zum Zeitpunkt der Betaversion kann dem Anhang entnommen werden (automatisch generierte Darstellung basierend auf dem aktuellen Code, siehe Anhang 9.1 und 9.2) Eine elektronische Version dieser Darstellung wird ebenfalls dem *repository* beigelegt.

3.2 Designkonzept

Die einheitliche Präsentation sämtlicher Programmelemente, welche für die Benutzer_innen sichtbar sind, werden basierend auf der folgenden Farbpalette entworfen (Tabelle 3). Damit das Programm ein modernes Look-and-Feel hat, wird die moderne Schriftart «Roboto» verwendet.

Tabelle 3 Farbpalette

HEX-Code	Verwendung	HEX-Code	Verwendung
E1ECF7	Akzent III	EE4035	Status: Fehler
AECBEB	Akzent II	F37736	Status: Warnung II
83B0E1	Akzent I	FDF498	Status: Warnung I
71A5DE	Hauptakzent (Hover: Knopf)	7BC043	Status: OK
4A4A4A	Hauptschrift	717171	Sekundärschrift (Vorschau/Icon)

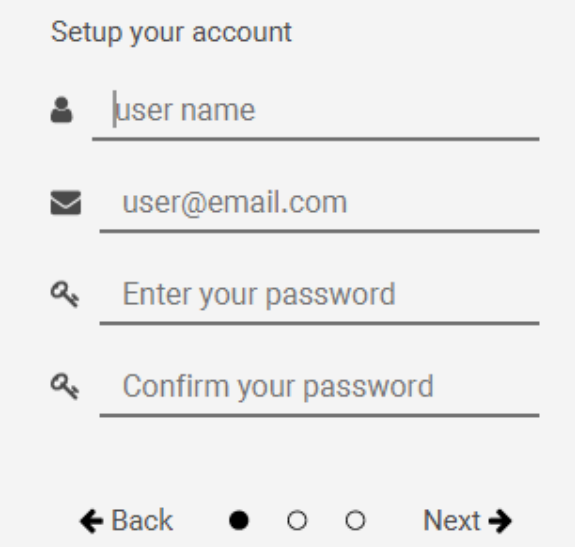


Abbildung 17 Registrierungsformular

Wichtige Knöpfe und Eingabefelder werden mit einem einheitlichen Set von Symbolen unterstützt (Abbildung 17).

4 Implementation

4.1 Testkonzept

Mit dem Ziel, möglichst effizient zu testen, Regressionen (Fehler als Folge von Änderungen von bereits getesteter Software) vorzubeugen und demzufolge eine Steigerung der Softwarequalität zu gewährleisten, wurde das folgende Testkonzept in Abbildung 18 entwickelt.

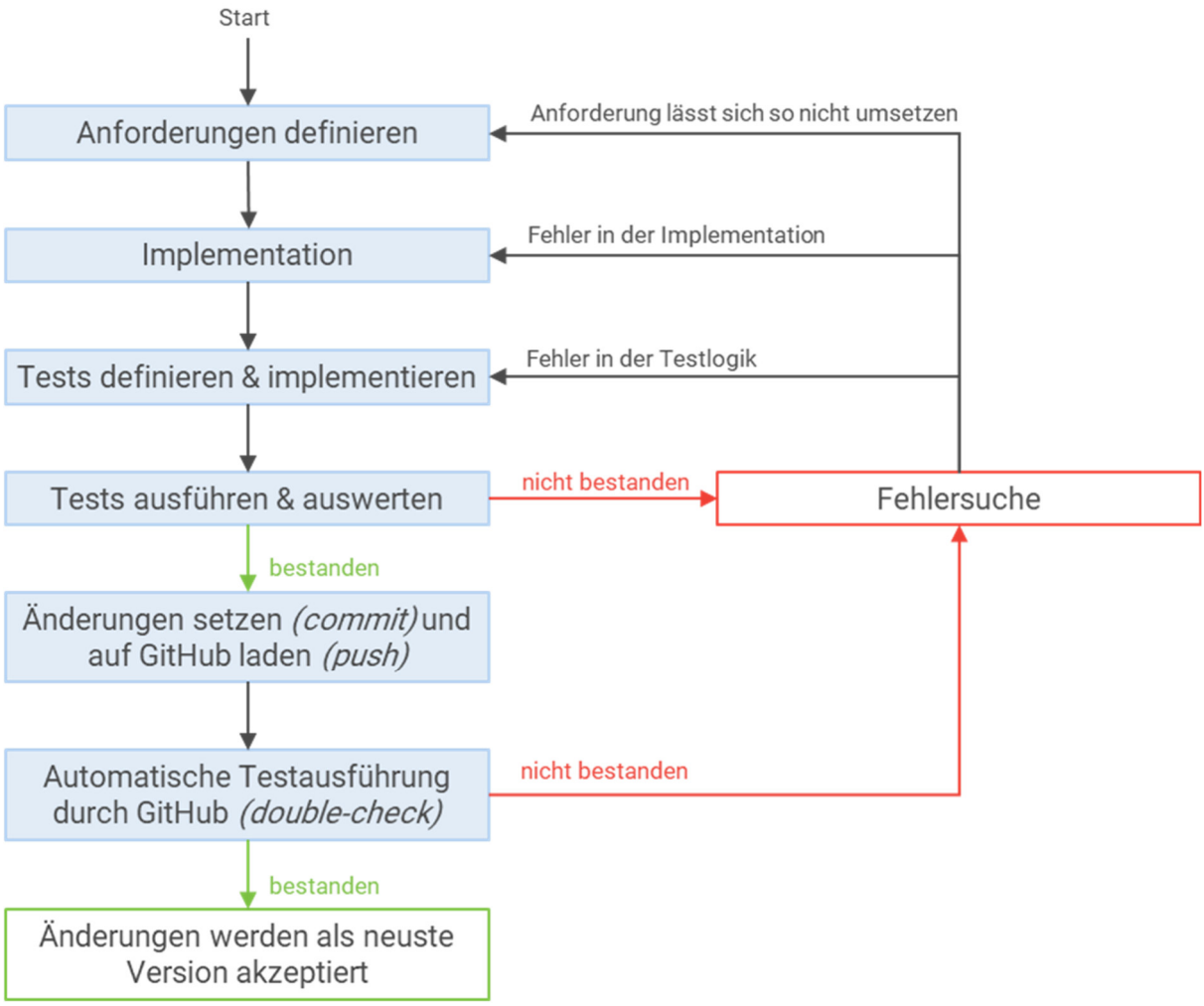


Abbildung 18 Testkonzept

Sobald Programmteile implementiert wurden, werden diese entsprechenden Tests qualitativ getestet. Die Tests werden mit Hilfe des JUnit 5 Frameworks implementiert. Der Umfang dieser Tests orientiert sich an Äquivalenzklassen. Äquivalenzklassen klassifizieren Wertebereiche, d.h. wann die Programmteile als funktionsfähig gelten oder nicht, und ermöglichen somit trotz geringer Anzahl von Tests eine hohe Fehlerentdeckungsrate. Spezieller Fokus bei den Tests wird auf Grenzfälle gelegt (z.B. bei einer geforderten, maximalen Elementanzahl von 10, wird im Test versucht genau 10 respektive 11 Elemente zu verwenden). Ergeben sich Fehler aus den Tests, muss die Ursache ergründet werden. Diese kann bei der gewählten Implementation (z.B. nicht funktionierende Methoden), Testlogik (z.B. falsch definierte Wertbereiche) und/oder eingangs definierter Anforderungen liegen.

Werden die Tests in einem ersten Schritt («Tests ausführen & testen») erfolgreich durchlaufen, so wird anschliessend versucht, die neuen Programmteile auf die Versionskontrolle (hier GitHub) hochzuladen und diese dadurch dem Projektteam zur Verfügung zu stellen («Änderungen setzen *(commit)* und auf GitHub laden *(push)*»). Mit anderen Worten werden kleine, aber funktionierende Programmteile stetig in die Applikation eingepflegt und anschliessend die nächsten Schritte definiert (sogenannte *continuous integration*). Wichtig hierbei ist, dass die gewählte Versionskontrolle GitHub alle bestehenden Tests erneut ausführt und die Änderungen nur dann zulässt, wenn alle Tests erfolgreich durchlaufen wurden («Automatische Testausführung durch GitHub *(double-check)*»). Durch diese doppelte Testausführung ist sichergestellt, dass die neuen Programmteile zu keinen Fehlern bei den bestehenden Programmteilen führen.

4.2 Getestete Programmteile

Sämtliche Komponenten der Domänenlogik (namentlich die Ent- und Verschlüsselungsverfahren, die Authentifizierung des Users oder die Anbindung an Cloud-Dienste) werden mit Modultests *(unit tests)* überprüft. Charakteristisch für Modultests ist, dass sich diese nur mit den betroffenen Programmteilen *(unit)* befassen, d.h. keine anderen Programmteile in den Tests miteinbeziehen.

Die Interaktion zwischen Klassen im selben Packet (namentlich im Packet «Cipher») wurde mit Integrationstests abgedeckt. Dies bedeutet, dass die Tests mehrere Klassen miteinschliessen und diese auf ihre Abhängigkeiten hin prüfen (sogenannte Integrationstests).

Das UI wurde manuell geprüft (siehe Anhang 9.3). Automatisierte Tests für den Bereich UI sind kritisch zu betrachten, da sie sehr aufwändig sind. Ihre Berechtigung erhalten sie, sobald die Software einen gewissen Umfang bzw. Komplexität erreicht. Für die Betaversion wurde auf die erwähnten UI-Tests verzichtet. Für künftige Versionen von PSC sollten diese aber ebenfalls definiert werden.

Abschliessend ist zu erwähnen, dass ebenfalls keine Systemtests für die Betaversion definiert wurden. Bei diesen wird das ganze System bezüglich der zu Beginn festgelegten Anforderungen getestet. Diese sollten wiederum für zukünftige Versionen durchgeführt werden.

Zusammenfassend kann festgehalten werden, dass die Betaversion als funktionsfähig im Sinne der oben implementierten Tests zu qualifizieren ist. Eine detaillierte Übersicht der Testergebnisse befindet sich im Anhang 9.4.

Wie in Abbildung 19 ersichtlich ist, erreichen die definierten Tests eine beinahe 100%-ige Abdeckung (sogenannte code coverage für das *package* «domain».

Coverage Breakdown

Package	Class, %	Method, %	Line, %
ch.psc.domain.user	100% (3/3)	100% (23/23)	89.5% (111/124)
ch.psc.domain.cipher	100% (11/11)	76.2% (48/63)	79.9% (127/159)
ch.psc.domain.storage	100% (2/2)	64.7% (11/17)	52.9% (45/85)
ch.psc.domain.file	100% (2/2)	81.2% (13/16)	88.5% (23/26)
ch.psc.datasource	100% (1/1)	100% (4/4)	81.8% (9/11)
ch.psc.domain.storage.service	50% (3/6)	28.6% (14/49)	34% (48/141)

Abbildung 19 code coverage

4.3 Codedokumentation

4.3.1 Graphical User Interface (GUI) und Anbindung an Cloud-Speicherdienste

Es ist vorgesehen, dass sämtliche Komponenten der grafischen Benutzeroberfläche (GUI) dynamisch auf Aktionen (z.B. Klicken mit der Maus oder Tastatureingabe des Users) reagieren. Führt der/die Benutzer_in eine Aktion auf einem GUI-Element aus, so wird das dafür zuständige Objekt in der unteren Schicht (d.h. Logikschicht) benachrichtigt. Ein Beispiel hierfür ist, wenn der User eine Texteingabe im E-Mail-Textfeld vornimmt, und zugleich im Hintergrund die Eingabe des Users formal überprüft wird. Dank dem verwendeten Framework «JavaFX» wird das Observer-Pattern mittels sogenannten ***listeners*** zur Verfügung gestellt. Sobald eine Aktion ausgeführt wird, werden alle registrierten *listeners* informiert und reagieren entsprechend (bspw. mit einer Fehlermeldung oder der Aktualisierung von dargestellten Informationen im GUI).

Zudem soll das GUI während eines Uploads oder Downloads von Dateien regelmässig über den Zustand des Prozesses informiert werden. Mithilfe des State-Patterns kann der **Zustand «ProcessState»** (Abbildung 20) über eine definierte Callback-Funktion übermittelt werden (siehe Abbildung 21 für die Verwendung von Callback-Funktionen). Dem Benutzer wird bspw. angezeigt, dass der Upload bzw. Download im Gange ist.

```java
/**
 * States of a download or upload process.
 * Used to notify gui about the current state of process.
 *
 * @author SandroGuerotto
 */
public enum ProcessState {
  DOWNLOADING, DOWNLOADED, ENCRYPTING, ENCRYPTED, FINISHED,
  UPLOADING, UPLOADED, DECRYPTING, DECRYPTED
}
```

Abbildung 20 Enumklasse ProcessState

```java
/**
 * Encrypts the file and uploads it to the selected storage service. After each encrypting process
 * {@link ProcessState} the callback function is called with the appropriate event.
 *
 * @param storage selected storage service
 * @param file file to upload
 * @param callback callback function after each process event
 */
public void uploadFiles(FileStorage storage, File file, Consumer<ProcessState> callback) {
  executorService.submit(() -> {
    try {
      callback.accept(ProcessState.ENCRYPTING);
      PscFile encrypted = encrypt(file);
      InputStream inputStream = createUploadStream(encrypted);
      callback.accept(ProcessState.ENCRYPTED);

      callback.accept(ProcessState.UPLOADING);
      storage.upload(encrypted.getName(), inputStream);
      callback.accept(ProcessState.UPLOADED);

    } catch (IOException | FatalImplementationException | InterruptedException
        | ExecutionException e) {
      e.printStackTrace();
    }
    callback.accept(ProcessState.FINISHED);
  });
}
```

Abbildung 21 Methode uploadFiles der Klasse StorageManager

Da unterschiedliche Speicherdienste angebunden werden können, brauchen die konkreten Implementationen eine gemeinsame Schnittstelle, damit die **Klasse «StorageManager»** einheitlich auf die Funktionalität zugreifen kann. Mit der Umsetzung des Strategy-Patterns konnte dafür ein geeignetes Konzept gefunden werden. Mittels des Interfaces «FileStorage», welches alle Services implementieren, können die Funktionalitäten der einzelnen Services ausgetauscht und/oder erweitert werden.

Die Hauptaufgabe des «StorageManager» ist es den gewünschten Storage und die gewünschten Datenoperationen durchzuführen. In dieser Klasse werden folgende Aufgaben koordiniert:

- Hochladen und Herunterladen von Daten;
- Encrypten und Decrypten von Daten;
- das Laden von Benutzer spezifischen Storages; sowie
- die Benutzerverwaltung sowie Speichern und Löschen von Daten.

Die Applikation PSC steht in der Betaversion nur einem User zu Verfügung. Um nur einen eingeloggten User pro Applikation zu haben, wurde ein **«UserContext»** erstellt. Dadurch können andere Klassen global auf den gleichen User zugreifen, was mit einem Singleton-Pattern vergleichbar ist. In diesem Fall kann die Instanz, allerdings mit einer Setter-Methode gesetzt werden. Damit wird eine bessere Testbarkeit möglich, da der User durch ein Mock-Objekt ersetzt werden kann.

4.3.2 Verschlüsselung und Entschlüsselungslogik

Eine Kernaufgabe des Packets «Cipher» ist das Erstellen von Objekten, welche sich um die Ver- und Entschlüsselung kümmern (Klasse «PscCipher», siehe nächster Abschnitt). Aufgrund der verschiedenen Algorithmen für die Ver- und Entschlüsselung und deren unter Umständen komplexen Erzeugung bot sich im Sinne des Factory-Patterns die Implementation der **zentralen Klasse «CipherFactory»**, welche sich um das Erzeugen der gewünschten *cipher* Objekte kümmert. Wird also ein spezifischer Ver- und Entschlüsselungsalgorithmus benötigt, so muss lediglich der gewünschte Algorithmus als Zeichenkette (z.B. «AES») an die «CipherFactory» übergeben werden und diese gibt das entsprechende Objekt zurück. Ein weiterer Vorteil ist, dass die Klasse «CipherFactory» auch als zentrale Auffangstelle für alle geworfenen *exceptions* fungieren kann und somit das *error handling* einfacher ist (Abbildung 22).

```
public static PscCipher createCipher(CipherAlgorithms type) throws FatalImplementationException {

    try {
        return type.getCipherClass().getConstructor().newInstance();
    } catch (NoSuchMethodException | IllegalArgumentException e) {
        throw new FatalImplementationException("Each class inheriting from PscCipher must implement an empty constructor!", e);
    } catch (InvocationTargetException e) {
        throw new FatalImplementationException("Constructor of " + type.getCipherClass() + " threw an exception:", e);
    } catch (InstantiationException e) {
        throw new FatalImplementationException("Seems like the Enum '" + CipherAlgorithms.class + "is contains abstract classes.
    } catch (SecurityException | IllegalAccessException e) {
        throw new FatalImplementationException("Access to the empty constructor of '" + type.getClass() + "' is denied!", e);
    }

}
```

Abbildung 22 Klasse CipherFactory

Für die Implementation der verschiedenen Verschlüsselungs- und Entschlüsselungsalgorithmen wurde eine **abstrakte Klasse «PscCipher»** implementiert, welche das Bindeglied zwischen den erwähnten Algorithmen und dem restlichen Programm darstellt (siehe Abbildung 23). In dieser abstrakten Klasse wird festgelegt, welche Methoden ein neuer Verschlüsselungs- und Entschlüsselungsalgorithmen *(cipher)* implementieren muss, damit dieser vom Programm verwendet werden kann. Durch diese Abkapselung können zusätzliche *ciphers* einfach implementiert werden. Die Klasse «PscCipher» wurde bewusst nicht als sogenanntes *interface* (Klassen ohne eigene Implementation der definierten Methoden) umgesetzt, da diverse Methoden für alle *ciphers* gleich implementiert werden können.

```java
/** Abstract base class for PSC ciphers.<br /> ...*/
public abstract class PscCipher {

    /** Always provide a public empty constructor */
    public PscCipher() { super(); }

    private ExecutorService executor = Executors.newFixedThreadPool( nThreads: 5);

    /** Which level of security is provided by this Cipher implementation? ...*/
    public abstract SecurityLevel getSecurityLevel();

    /** Returns the amount of bits required for the relevant algorithm. ...*/
    public abstract int getKeyBits();

    /** The name of the cryptographic algorithm (e.g.: AES, RSA). ...*/
    public abstract String getAlgorithm();

    /** Provides the Algorithm, Mode and Padding in the format "Algo/Mode/Padding". ...*/
    public abstract String getTransformation();

    /** Creates the {@link AlgorithmParameterSpec} required for this encryption method. ...*/
    public AlgorithmParameterSpec getAlgorithmSpecification(PscFile file) { return null; }

    /** Encrypts a list of {@link PscFile}s. The encryption is done asynchronously and the meth
    public List<Future<PscFile>> encrypt(Key key, List<PscFile> files) {...}

    /** Decrypts a list of {@link PscFile}s. The decryption is done asynchronously and the meth
    public List<Future<PscFile>> decrypt(Key key, List<PscFile> files) {...}
```

Abbildung 23 Abstrakte Klasse PscCipher

Da die verschiedenen Verschlüsselungs- und Entschlüsselungsalgorithmen verschiedene Arten von Schlüsseln benötigen, wurde die **zentrale Klasse «KeyGenerator»** definiert, welche die für den verwendeten cipher benötigten Schlüssel generiert und zurückgibt (siehe Abbildung 24). Durch diese Abkapselung können wiederum Kodeduplikationen vermieden und zukünftige Sicherheitsaspekte bei der Erstellung von Schlüsseln isoliert implementiert werden.

```java
public Map<String, Key> generateKey(int keyBits, String algorithm) throws FatalImplementationException {
    KeyType type = testKeyType(algorithm);
    Map<String, Key> keyChain = null;

    switch (type) {
        case SYMMETRIC:
            keyChain = generateSymmetricKey(keyBits, algorithm);
            break;
        case ASYMMETRIC:
            keyChain = generateAsymmetricKey(keyBits, algorithm);
            break;
        case NOT_SUPPORTED:
        default:
            throw new FatalImplementationException("Algorithm '" + algorithm + "' is not supported! Please check spelling.");
    }

    return keyChain;
}
```

Abbildung 24 generateKey Methode der Klasse KeyGenerator

4.3.3 Dokumentenverwaltung *(file explorer)*

Die Aufgabe des Pakets «storage» ist die Bereitstellung von verschiedenen Diensten und eines Managers, der diese Dienste in Anspruch nimmt, um das Ziel der Datenverwaltung zu verwirklichen. Wie auch für die Dienste in anderen Paketen, wurde auch im Packet «service» ein **Interface «FileStorage»** implementiert (Abbildung 25). Dieses Interface definiert die nötigen Funktionen, welche für einen Cloud-Speicherdienst definiert werden müssen, damit sie in PSC integriert werden können.

```java
public interface FileStorage {

  /** Uploads a file to the service ...*/
  void upload(String fileName, InputStream inputStream);

  /** Downloads a selected file from the service ...*/
  InputStream download(String path);

  /** Calculates the used memory in bytes ...*/
  BigDecimal getUsedStorageSpace();

  BigDecimal getTotalStorageSpace();

  /** Creates a list with all files/folders in directory, adds filesize, lastModified and isDirectory ...*/
  List<PscFile> getFiles(String path);

  /** Returns the name of the connected storage service. ...*/
  String getName();

  /** Returns used space of the connected storage service. ...*/
  ObjectProperty<BigDecimal> getUsedStorageSpaceProperty();

  /** Return root path. ...*/
  String getRoot();

  /** Return the path separator of the connected storage service. In Dropbox it's "/". ...*/
  String getSeparator();

}
```

Abbildung 25 Interface FileStorage

Wie in Abbildung 25 ersichtlich ist, müssen die verschiedenen Dienste, welche das Interface «FileStorage» umsetzen, folgende Methoden implementieren:

- Hochladen und Herunterladen von Daten;
- gesamter Speicherplatz des Speichermediums in GB zurückgeben;
- den freien Speicherplatz des Speichermediums in GB zurückgeben;
- Daten von einem bestimmten Pfad zurück liefern; sowie
- den Namen des Speichermediums und den Wurzel Pfad sowie den verwendeten Separator zurückgeben.

Das beschriebene Interface-Design bietet sich insbesondere deshalb an, weil die verschiedenen Cloud-Speicherdienste unter Umständen sehr verschiedenen Schnittstellen zur Verfügung stellen. Wenn diese in die definierten Interface-Methoden übersetzt werden müssen, kann sichergestellt werden, dass jeder

zusätzliche Cloud-Speicherdienst die gleichen Methoden innerhalb von PSC zur Verfügung stellt und somit die Funktionsfähigkeit der Applikation nicht tangiert. Dies erleichterten die Wartbarkeit und die Qualität des Quellcodes.

Aufgrund der Vielzahl von möglichen Cloud-Speicherdiensten und deren unter Umständen komplexen Erzeugung, bot sich im Sinne des Factory-Patterns die Implementation der **Klasse «StorageServiceFactory»** als Erzeugungsdienst an. In dieser Rolle ist die erwähnte Klasse für das Erstellen der gewünschten *storage* Objekte verantwortlich.

```
/**
 * Creates a new Services with the configuration of the logged-in user. Example: creating a
 * dropbox service with configuration of logged-in user.
 *
 * <pre>
 * {@code
 *   FileStorage dropbox = StorageServiceFactory.createService(StorageService.DROPBOX,
 *       user.getStorageServiceConfig().get(StorageService.DROPBOX));
 * }
 * </pre>
 *
 * @param service desired service. {@link StorageService}
 * @param accountData configuration and access data
 * @return created and configured service
 * @throws IllegalArgumentException service unknown
 */
public static FileStorage createService(StorageService service, Map<String, String> accountData)
    throws IllegalArgumentException {

  return switch (service) {
    case LOCAL -> createLocalStorageService(accountData);
    case DROPBOX -> createDropBoxService(accountData);
    case GOOGLE_DRIVE -> createGoogleDriveService(accountData);
    default -> throw new IllegalArgumentException("Service not supported");
  };

}
```

Abbildung 26 Klasse StorageServiceFactory

Wie in Abbildung 26 ersichtlich, kann das benötigte *storage* Objekt mit dem relevanten Aufzählungstyp (z.B. «LOCAL») von der «StorageServiceFactory» angefragt werden und diese gibt das entsprechende *storage* Objekt zurück. Im Falle einer Erweiterung um einen zusätzlichen Speicherdienst, müsste lediglich ein weiterer Erstellungsservice in dieser Klasse implementiert werden. Die Dienste des «StorageServiceFactory» wird vom «StorageManager» in Anspruch genommen (siehe Abschnitt 4.3.1).

Für die Erstellung der einzelnen «StorageServices» wird das Factory-Pattern eingesetzt. Die **Klasse «StorageServiceFactory»** mit dem ausgewählten Cloud-Speicherdienst und den User-Daten, den entsprechenden Service. Dank der Factory ist die Klasse «StorageManager» von dem Wissen über bzw. der Erzeugung der Services befreit. Folglich wird dadurch eine losere Kopplung gefördert.

5 Resultate

Die Idee für eine zentrale und vor allem automatische Verschlüsselung für Cloud-Speicherdienste wurde vor rund zwölf Wochen in einer Projektskizze festgehalten und präsentiert. In dieser Zeit nahm das Projekt Woche für Woche Gestalt an. Eine Vielzahl von Teammeetings, Implementationsiterationen und Meilensteinpräsentation später, freut es uns, eine lauffähige Beta-Version von PSC präsentieren zu dürfen. Erfreulich ist insbesondere, dass sämtliche Grundfunktionalitäten gemäss ursprünglichen Use-Cases in der Betaversion realisiert wurden. Namentlich:

- Schlanke und simple Benutzeroberfläche
- Verständlicher Registrierungsprozess inkl. Hilfestellungen
- Anbindung an den Cloud-Dienst «Dropbox»
- Anbindung für lokalen Speicherort
- Wahl zwischen verschiedenen Verschlüsselungs- und Entschlüsselungsalgorithmen (RSA und AES)
- *Drag and Drop* Funktionalität für das Verschlüsseln und Hochladen von Dateien
- Herunterladen und Entschlüsseln von Dateien via Auswahl
- Loginfunktion für mehrere Benutzer_innen

Natürlich ist PSC noch weit entfernt von einer kundenfertigen Version. Allerdings zeigten die vergangenen Wochen, dass die Idee durchaus umsetzbar ist und PSC einen echten Mehrwert liefern kann. Vor diesem Hintergrund sollte nun die gewonnene Geschwindigkeit bei der Entwicklung von PSC aufrechterhalten und die nächsten Funktionen in Angriff genommen werden. Dabei sind insbesondere die folgenden Punkte erwähnenswert:

- Implementation eines hybriden Verschlüsselungssystem (z.B. RSA/AES)
- Herunterladen und Entschlüsseln von Dateien via Drag and Drop
- Visuelle Hilfestellungen für den Status einer Datei (verschlüsselt, unverschlüsselt, wird verschlüsselt, etc.)
- Zusätzliche Einstellungsmöglichkeiten nach erfolgter Registrierung (z.B. nachträgliche Änderung des ausgewählten Verschlüsselungs- und Entschlüsselungsalgorithmus)
- Für Unternehmenskunden notwendige Schnittstellen berücksichtigen
- Barrierefreiheit von PSC für Personen mit Sehschwäche sicherstellen, d.h. prüfen, ob entsprechende Hilfsanwendungen den Inhalt von PSC korrekt wiedergeben können
- Anbindungsmöglichkeit an eine Datenbank über einen Webservice

Viel wichtiger als der bisher erkannte Entwicklungsbedarf sind die Rückmeldungen der/die Benutzer_innen von PSC. Deshalb ist es uns ein grosses Anliegen, PSC einem grösseren Interessentenkreis zur Verfügung zu stellen, um aus den Rückmeldungen lernen und die Entwicklung in die richtige Richtung lenken zu können. Weiter wollen wir auch mögliche Pilotprojekte mit Unternehmen planen, sodass wir etwaige Schnittstellen frühzeitig planen und implementieren können, um eine gute Integrationsfähigkeit von PSC mit der IT-Struktur von Unternehmen zu gewährleisten.

6 Projektmanagement

6.1 Teammitglieder

Guerotto Sandro, Kindle Tristan, Sevimli Ridvan, Sieber Lorenz, Waldburger Safiyya und Walser Christoph Marjan Markus.

6.2 Aufgabenverteilung

Die Aufgabenverteilung für das Projekt wurde nicht statisch festgesetzt, d.h. die Aufgaben können bei Bedarf umverteilt werden. Grundsätzlich werden die Teammitglieder für alle Bereiche eingesetzt. Dadurch wird eine ausgewogene Verteilung der Arbeitslast realisiert. Die Implementation im Besonderen wird in Zweierteams realisiert. Die verschiedenen Aufgabenbereiche wurden prinzipiell wie folgt aufgeschlüsselt.

Projektleitung: Kindle Tristan und Ridvan Sevimli (Stellvertretung).

Dokumentation: Grundsätzlich alle, wobei Kindle Tristan, Waldburger Safiyya und Walser Christoph Marjan Markus für die Aufbereitung und Finalisierung zuständig sind.

Überwachung GitHub Funktionalitäten: Guerotto Sandro und Ridvan Sevimli.

Aktualisierung GitHub *issues*: Alle für die ihnen zugewiesenen *issues*.

Implementation: Grundsätzlich alle, wobei folgende Zweierteams definiert wurden.

- **Anbindung an Cloud-Dienste (Login und Sign Up)**: Guerotto Sandro und Waldburger Safiyya.
- **Dokumentenverwaltung (File-Browser)**: Sevimli Ridvan und Walser Christoph Marjan Markus.
- **Backend (inkl. Verschlüsselung)**: Tristan Kindle und Sieber Lorenz.

Alle Mitglieder haben bis zum Stand der Abgabe gleichermassen zum Projekt beigetragen (die obige Aufzählung soll lediglich die jeweiligen Schwerpunkte hervorheben).

6.3 Vorgehen und Projektplan

Für die Projektleitung ist Tristan Kindle und Ridvan Sevimli (Stellvertretung) zuständig. Zum Aufgabenbereich der Projektleitung gehört die Dokumentation der Projektplanung sowie die zeitliche und fachliche Überwachung des Projektfortschritts. Weiter nimmt die Projektleitung sämtliche Aufgaben im Zusammenhang mit organisatorischem Charakter wahr (z.B. Ansetzen von Besprechungen).

Iterationsreviews: Der unten dargestellte Gesamtprojektplan (inkl. Meilensteinen, siehe Tabelle 4) soll in zweiwöchentlichen Iterationen überprüft und umgesetzt werden. Die Protokolle der Iterationsreviews

werden nach jedem Zyklus mit der Grobplanung abgeglichen und allfällige Abweichungen notiert und mit entsprechenden Massnahmen angegangen.

Zeitliches Budget: Für die Entwicklung einer ersten Version wird ein Aufwand von rund 500 Personenstunden erwartet. Die gemachten Angaben zum zeitlichen Budget werden regelmässig und mündlich innerhalb des Teams besprochen, um unvorhergesehene Aufwände zu erkennen.

Aufgabenverwaltung: Die detaillierten Aufgaben betreffend die Implementation werden via GitHub koordiniert und verwaltet: https://github.com/SandroGuerotto/PrettySecureCloud/projects/1

Diese werden wöchentlich, d.h. nicht nur in jedem Iterationsreview, im Team besprochen und vorangetrieben. Dadurch sollen Probleme frühzeitig erkannt und behoben werden.

Tabelle 4 Projektplan

SW	Meilenstein	Iteration	Aufwand in Stunden		Aufträge	Bemerkung nach Iterationsreview
			Soll	Ist		
1		-	40	40	Projektidee finden, Architektur skizzieren, Projektskizze ausarbeiten, Mockup, Präsentation vorbereiten	Alle Aufträge gemäss Grobplanung und im zeitlichen Budget erfüllt.
3	Projektskizze & Präsentation	1	20	20	Anforderungen, Ziele und Vision formulieren und präsentieren	Alle Aufträge gemäss Grobplanung und im zeitlichen Budget erfüllt.
5		2	60	60	UC detailliert ausformuliert, zusätzliche Anforderungen definiert, UI-Prototyp, Entwurf Domänenmodell	Organisatorische Umlage der Aufträge, d.h. Abweichung vom Grobplanung nicht aber vom zeitlichen Budget. Implementation für UI-Prototyp auf SW 7 verschoben. Dafür erster Entwurf für 'Technischer Bericht I' erstellt.
7		3	80	80	Domänenmodell fertig, Architektur stabil und als PoC verifiziert, Verfassen technischer Bericht	Alle Aufträge gemäss Grobplanung und im zeitlichen Budget erfüllt. UI-Prototyp (aus vorheriger Iteration) erstellt. Architektur wurde verifiziert, keine *show-stopper* erkannt. Technischer Bericht I bereit für finale Kontrolle.
5	Lösungsarchitektur, Technischer Bericht 1	4	20	20	Architektur verifiziert, dokumentiert und präsentiert.	
6		5	100	100	UI Prototyp implementiert, UC realisiert und getestet, Mögliche Showstopper erkennen, Aktuelle *issues* gemäss GitHub lösen	Alle Aufträge gemäss Grobplanung und im zeitlichen Budget erfüllt.
9		6	80	80	Technischer Bericht II erstellen, Aktuelle *issues* gemäss GitHub lösen	Alle Aufträge gemäss Grobplanung und im zeitlichen Budget erfüllt.
11		7	80	80	Tests formuliert und erfüllt, Aktuelle *issues* gemäss GitHub lösen Technischer Bericht II fertigstellen,	Alle Aufträge gemäss Grobplanung und im zeitlichen Budget erfüllt.
13	Beta Release & Technischer Bericht 2	8	20	20	Betaversion fertig implementiert, inkl. Systemtests und Dokumentation	Alle Aufträge gemäss Grobplanung und im zeitlichen Budget erfüllt.

6.4 Risiken

Bei jeder Applikation besteht das generelle Risiko, dass die **Grundfunktionalität** nicht erfüllt ist. Im Fall von PSC wäre das ein Versagen der Hauptfunktionen Hochladen und Verschlüsseln bzw. Herunterladen und Entschlüsseln. Mithilfe einer strukturierte Grobplanung, sorgfältigen Arbeitsweise und regelmässigen Tests kann diesem Risiko substantial entgegengetreten werden.

Weiter ergibt sich ein Risiko mit Blick auf die bestehenden **Konkurrenzprodukte**. Die eingangs erwähnten Alleinstellungsmerkmale der von PSC reduzieren dieses Risiko.

Eine wesentliche Voraussetzung für die Funktionalität der von PSC, ist die Anknüpfung an die **Schnittstelle** des jeweiligen Cloud-Speicher-Dienstes. Ist die Schnittstelle nicht erreichbar oder keine Anbindung (mehr) möglich, müsste ein eigener Speicher-Dienst angeboten werden. Um dieses Risiko bestmöglich zu minimieren, wird den Prinzipien der losen Koppelung sowie Skalierbarkeit bei den betroffenen Programmteilen einen hohen Stellenwert eingeräumt.

Im Verlaufe der Entwicklung der von PSC könnten bisher **unberücksichtigte Features** als notwendig erachtet werden. Dies würde den Umfang des Projekts unerwartet aufblähen, was den Abschluss des Projektes im gesetzten zeitlichen Rahmen gefährden. Eine klare Definition des Projekts sowie regelmässige Testläufe minimiert dieses Risiko.

Überdies gelten **Verschlüsselungsverfahren** mit der Zeit als veraltet und können geknackt werden. Es bedarf einer fortlaufenden Revision, ob die verwendeten Verschlüsselungsverfahren den aktuellen Standards entsprechen.

Fallen **Projektmitarbeitende** aus gesundheitlichen oder anderweitigen Gründen aus, ist mit Verzögerungen oder einer Performanceeinbusse bei der Entwicklung der Applikation zu rechnen. Mit genügend eingeplanter Zeit für die Entwicklung, sollte diese Risiko neutralisiert werden können.

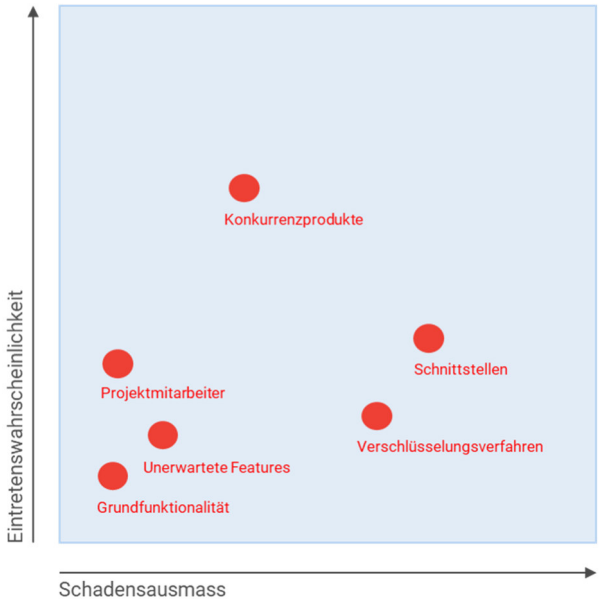


Abbildung 27 Risikomatrix

In Abbildung 27 wurden die oben beschriebenen Risiken in einer Risikomatrix angeordnet. Die Y-Achse gibt Auskunft über die Eintretenswahrscheinlichkeit und die X-Achse über das Schadensausmass. Gemäss Einschätzung des Projektteams, befinden sich die meisten Risiken im unteren, linken Bereich der Matrix, welcher als vertretbarer Risikobereich gewertet werden kann. Mögliche Konkurrenzprodukte sind unvermeidbar und müssen deshalb etwas wahrscheinlicher und schädlicher für PSC eingestuft werden.

6.5 Wirtschaftlichkeit

Das Projektteam besteht aktuell aus sechs Personen. Das Kernteam weist spezialisiertes Fachwissen in Java-Programmierung, Kryptologie, Marketing und Projektmanagement auf, was für die Umsetzung des dargestellten Projekts unabdinglich ist. Entsprechend ist davon auszugehen, dass kaum externes Fachwissen eingekauft werden muss.

Um die Wirtschaftlichkeit des Projekts besser zu visualisieren, wurde eine Analyse modelliert, welche die ersten fünf Jahre nach Start des Projekts umfasst. Zusätzlich wurde dieselbe Analyselogik für ein *best case* sowie *worst case* angewendet. Weiter wird davon ausgegangen, dass der Prototyp für den Beta-Release nach 12 Wochen und das Endprodukt nach weiteren 40 Wochen (also insgesamt ein Jahr) fertiggestellt werden kann.

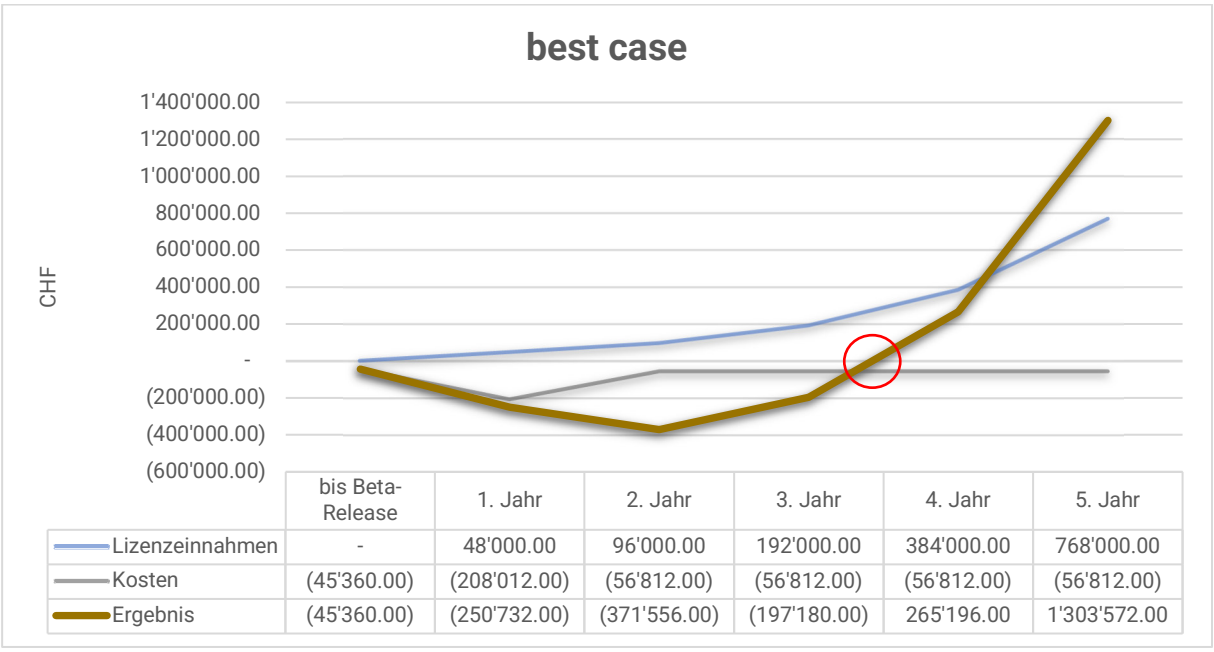
Wie oben bereits erwähnt, gibt es diverse kostenpflichtige sowie kostenfreie Konkurrenzprodukte. Die Preismodelle der kostenpflichtigen Alternativen zu PSC sind in der Regel lizenzbasiert. Dies bedeutet, dass für die Benutzung regelmässige (monatlich/jährlich) Beiträge bezahlt werden müssen, welche schon bei CHF 5.- (Einzellizenz) resp. 20.- (Unternehmenslizenz) pro Benutzerin oder Benutzer pro Monat beginnen. Für die hier dargestellten Berechnungen wird für die Lizenzeinnahmen ein vorsichtig geschätzter Durchschnitt von CHF 6.- *(worst case)* und CHF 10.- *(best case)* pro Benutzerin oder Benutzer pro Monat angenommen (keine Unterscheidung zwischen Einzellizenz und Unternehmenslizenz). Die verkauften Lizenzen werden jährlich um den Faktor zwei erhöht, beginnend mit 400 Lizenzen *(best case)* resp. 200 Lizenzen *(worst case)*.

Für Softwareentwicklungsprojekte üblich, ergeben sich in der Anfangszeit zum Grossteil Entwicklungskosten, welche sich hauptsächlich aus Personalkosten zusammensetzen. Nachdem die Entwicklung des Produktes abgeschlossen ist (nach dem 1. Jahr), stellen sich wiederkehrende Kosten wie zum Beispiel Personal-, Dritt-, Werbe-, Betriebs- sowie Hardware und Cloudkosten ein. Folgende Annahmen wurden für die vermeintlichen Kostenpositionen getroffen:

- Die **Entwicklungskosten** setzten sich vollumfänglich aus Personalkosten zusammen. Gerechnet wurde mit sechs Angestellten, sechs Stunden pro Woche und mit einem Stundensatz von CHF 60.- *(worst case)* resp. 90.- *(best case)*. Damit ergeben sich 504 Arbeitsstunden bis zum Prototyp und 1’680 Stunden bis zum Endprodukt.
- Die jährlichen **Betriebskosten (inkl. Personalkosten)** entsprechen 20% der gesamten Entwicklungskosten, d.h. CHF 26’208.
- Die **Hardware/Cloudkosten** wurde mit einer jährlichen Pauschale von CHF 2’500.- berücksichtigt [7].
- Allfällige **Drittkosten** (spezielles Fachwissen, etc.) wurde mit einer Pauschale von CHF 5’000 berücksichtigt.
- Die jährlichen **Werbekosten** sollen sich auf webbasierte Werbekanäle beschränken, welche maximal CHF 4.- pro Tag in Rechnung stellen, d.h. rund CHF 1’460 pro Jahr [8].

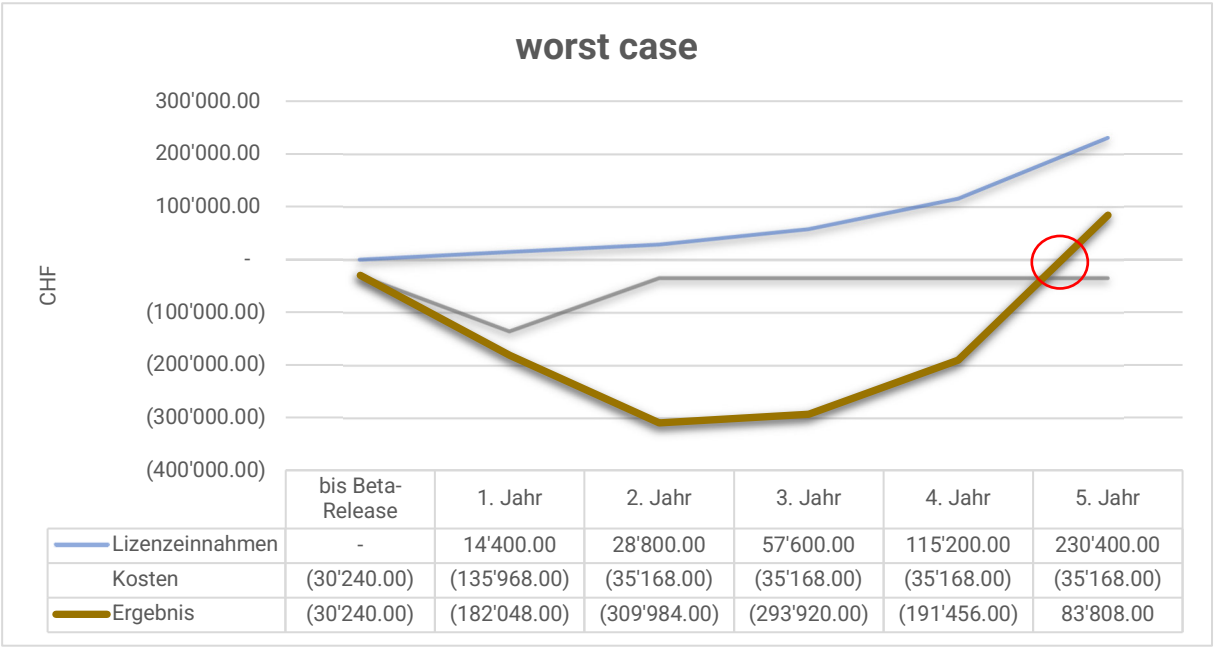
- Eine mögliche **Gewinnverwendung** wird nicht berücksichtigt, d.h. das Jahresergebnis wird vollumfänglich vorgetragen.

Je nach angewendetem Szenario, stellt sich gemäss Analysemodel ein positives Jahresergebnis nach drei (Abbildung 28) resp. fünf (Abbildung 29) Jahren ein. Die dargestellten Szenarien sind lediglich Prognosen und dienen als Veranschaulichung der zu erwartenden Möglichkeiten. In den nachfolgenden Abbildungen wurden die verschiedenen Positionen für Einnahmen und Kosten stark vereinfach, d.h. zusammengefasst. Der Fokus liegt auf dem Jahr, in welchem das Projekt ein positives Ergebnis erzielt (Break-Even-Punkt) d.h. dort, wo sich der Graph «Ergebnis» und die X-Achse schneiden (siehe roter Kreis). Auf der Y-Achse sind die jeweiligen Beträge in CHF angegeben, in Abhängigkeit des jeweiligen Jahres (X-Achse).



	bis Beta-Release	1. Jahr	2. Jahr	3. Jahr	4. Jahr	5. Jahr
Lizenzeinnahmen	-	48'000.00	96'000.00	192'000.00	384'000.00	768'000.00
Kosten	(45'360.00)	(208'012.00)	(56'812.00)	(56'812.00)	(56'812.00)	(56'812.00)
Ergebnis	(45'360.00)	(250'732.00)	(371'556.00)	(197'180.00)	265'196.00	1'303'572.00

Abbildung 28 best case Szenario



	bis Beta-Release	1. Jahr	2. Jahr	3. Jahr	4. Jahr	5. Jahr
Lizenzeinnahmen	-	14'400.00	28'800.00	57'600.00	115'200.00	230'400.00
Kosten	(30'240.00)	(135'968.00)	(35'168.00)	(35'168.00)	(35'168.00)	(35'168.00)
Ergebnis	(30'240.00)	(182'048.00)	(309'984.00)	(293'920.00)	(191'456.00)	83'808.00

Abbildung 29 worst case Szenario

7 Verzeichnisse

7.1 Literaturverzeichnis

[1] T.Ebert. (01.04.2020). *Filehosting Anbieter für Apple Anwender im Vergleich* [Online]. URL: https://www.apfelpage.de/news/filehosting-anbieter-fuer-apple-anwender-im-vergleich/ [Stand 06.03.2022]

[2] R. Ono. (25.09.2020). *Wieso Telearbeit in der Schweiz so gut funktioniert* [Online]. URL: https://www.swissinfo.ch/ger/wieso-telearbeit-in-der-schweiz-so-gut-funktioniert/46057212 [Stand 10.5.2022]

[3] Bundesamt für Statistik. (o.D.). *Digitale Kriminalität* [Online]. URL: https://www.bfs.admin.ch/bfs/de/home/statistiken/kriminalitaet-strafrecht/polizei/digitale-kriminalitaet.html [Stand 17.05.2022]

[4] T. Joos. (02.01.2019). *Die besten Verschlüsselungs-Tools für Daten in der Cloud* [Online]. URL: https://www.security-insider.de/die-besten-verschluesselungs-tools-fuer-daten-in-der-cloud-a-782048/ [Stand 4.5.2022]

[5] F.Gruber. (01.10.2020). *Die 3-2-1-Backup-Regel* [Online].URL: https://www.backup.ch/die-3-2-1-backup-regel/ [Stand 06.03.2022]

[6] E. Gamma, R. Helm, R.Johnson und J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995 , S. 107 – 116

[7] J. Halstenberg, B. Pfitzinger und T. Jestädt, *DevOps. Ein Überblick*. Wiesbaden: Springer Vieweg 2020, S.18

[8] (o. N.). (07.12.2009). *functionality, usability, reliability, performance, supportability – FURPS* [Online]. URL: https://www.itwissen.info/funcionality-usability-reliability-performance-supportability-FURPS.html [Stand 19.04.2022]

[9] Web-Entwicklung (28.07.2020). *Definition Quellcode* [Online]. URL: https://www.ionos.de/digitalguide/websites/web-entwicklung/quellcode/ [Stand 19.04.2022]

[10] K. Pfeifer (24.04.2019). *Was ist ein Repository?* [Online]. URL: https://www.devguide.at/git/was-ist-ein-repository/ [Stand 19.04.2022]

[11] T. Klein und E. Semenova (21.01.2022). *Tests in der Softwareentwicklung: Ein Klassifizierungsansatz* [Online]. URL: https://www.redbots.de/blog/software-tests-klassifizierung/ [Stand 19.04.2022]

[12] T. Ebert (01.04.2020). *Was ist Versionskontrolle?* [Online]. URL: https://www.atlassian.com/de/git/tutorials/what-is-version-control [Stand 19.04.2022]

7.2 Abbildungsverzeichnis

Tabellenverzeichnis

7.3 Glossar

Cipher:	Methode oder Algorithmus für Ver- und Entschlüsselung
Continuous Integration:	“Continuous Integration” ist eine Methode der agilen Softwareentwicklung. Entwickler fügen neuentwickelter, kleinteiliger und lauffähiger Programmcode regelmäßig in die Anwendung ein, statt sie alle erst zum Abschluss des Projekts zu integrieren [7].
Download:	Herunterladen bzw. Empfangen von Daten von einem bestimmten Medium
FURPS:	Akronym, welches sich auf die englischen Bezeichnungen für ein Qualitätsmodell hinsichtlich Software-Lebenszyklus bezieht [8].
FXML:	XML basierte Markup-Language, spezialisiert, um JavaFX GUIs zu erstellen.
GUI:	Graphical User Interface, Benutzeroberfläche
KISS-Prinzip:	Keep it simple and stupid: Möglichst einfache Lösung für ein Problem
Look-and-Feel:	Bezeichnung für audiovisuelle Design-Aspekte von Software mit grafischer Benutzeroberfläche
Modultests	Tests einzelner Einheiten eines Softwareprogramm
MVC Pattern:	Model View Controller Pattern: Design Pattern für die GUI-Entwicklung
Pull-Anfrage:	Der Begriff «Pull-Anfrage» stammt von git, wobei die <pull Befehl> wird verwendet, um ein anderes Repository mit Ihrem lokalen Repository zusammenzuführen.
QuellCode:	Quellcode versteht man einen für Menschen lesbaren Text, der in einer bestimmten Programmiersprache verfasst ist [9].

Repository:	Als „Software-Repository“ versteht man ein „Projektarchiv“, in dem Quellcode für eine gewünschte Software entwickelt und versioniert wird [10].
Scenebuilder:	Grafisches Tool zur Erstellung einer FXML-Datei
Sequenzdiagramm:	stellt die relevantesten Systemoperationen, die das System für die Anwendungsfälle (Use Cases) zur Verfügung stellen muss, dar
Testen:	Testen beschreibt den Prozess des Testens auf Funktionalität des bereits bestehenden Codes aber im allgemein meist einen Code Abschnitt der neu zu einem Quelltext hinzugefügt werden soll [11].
Three-tier-architecture:	Architekturprinzip in der Softwareentwicklung
UI:	User Interface, Benutzeroberfläche
UI-Sketch:	Benutzeroberfläche als Skizze
UML – Die Unified Modeling Language:	ist eine grafische Modellierungssprache zur Spezifikation, Konstruktion, Dokumentation und Visualisierung von (u.a.) Software.
Upload:	Hochladen bzw. Übertragen von Daten auf ein Medium
Use Case Diagramm:	Diagramm, welches die für die Software relevantesten Anwendungsfälle grafisch darlegt
Use Case:	Anwendungsfall oder Szenario, welches eine Interaktion des Akteurs mit dem System beschreibt
User:	Person, die das Programm verwendet, gleichbedeutend mit der/die Benutzer_in
Versionskontrolle:	Die Versionskontrolle, auch bekannt als Quellcodekontrolle. Versionskontrollsysteme unterstützen die Verwaltung von Änderungen am Quellcode im Zeitverlauf [12].
XML:	Extensible Markup Language: Definiert Regeln, um sowohl Maschinen- und Menschenlesbar zu sein.

8 Selbstständigkeitserklärung

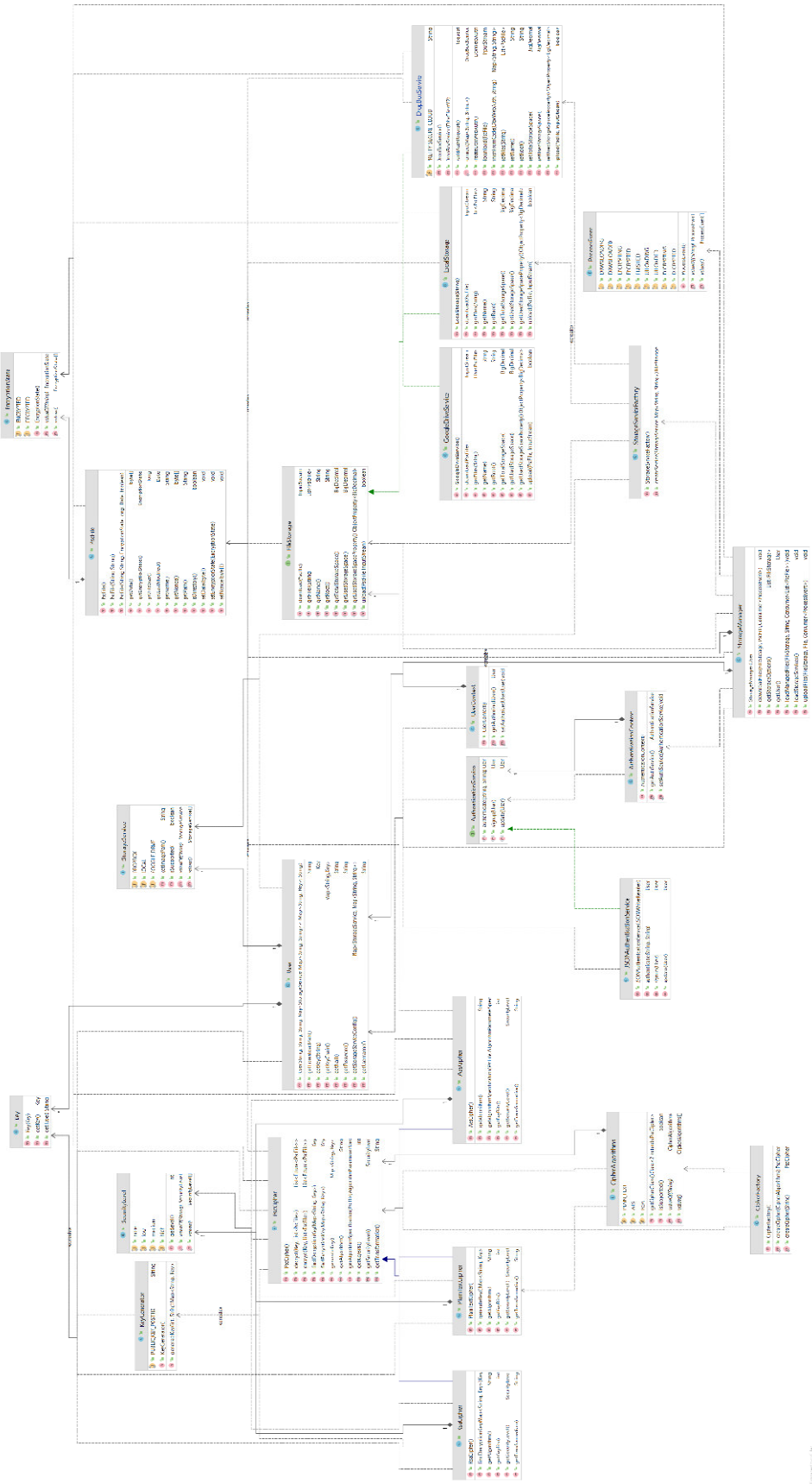
Der vorliegende Bericht wurde von den genannten Personen (siehe Abschnitt 6.1) selbstständig erarbeitet. Es wurden keine anderen als die angegebenen Hilfsmittel und Quellen benutzt. Dies gilt insbesondere für Informationen aus dem Internet. Alle sinngemäss und wörtlich übernommenen Textstellen aus fremden Quellen wurden kenntlich gemacht.

Zürich, 18. Mai 2022

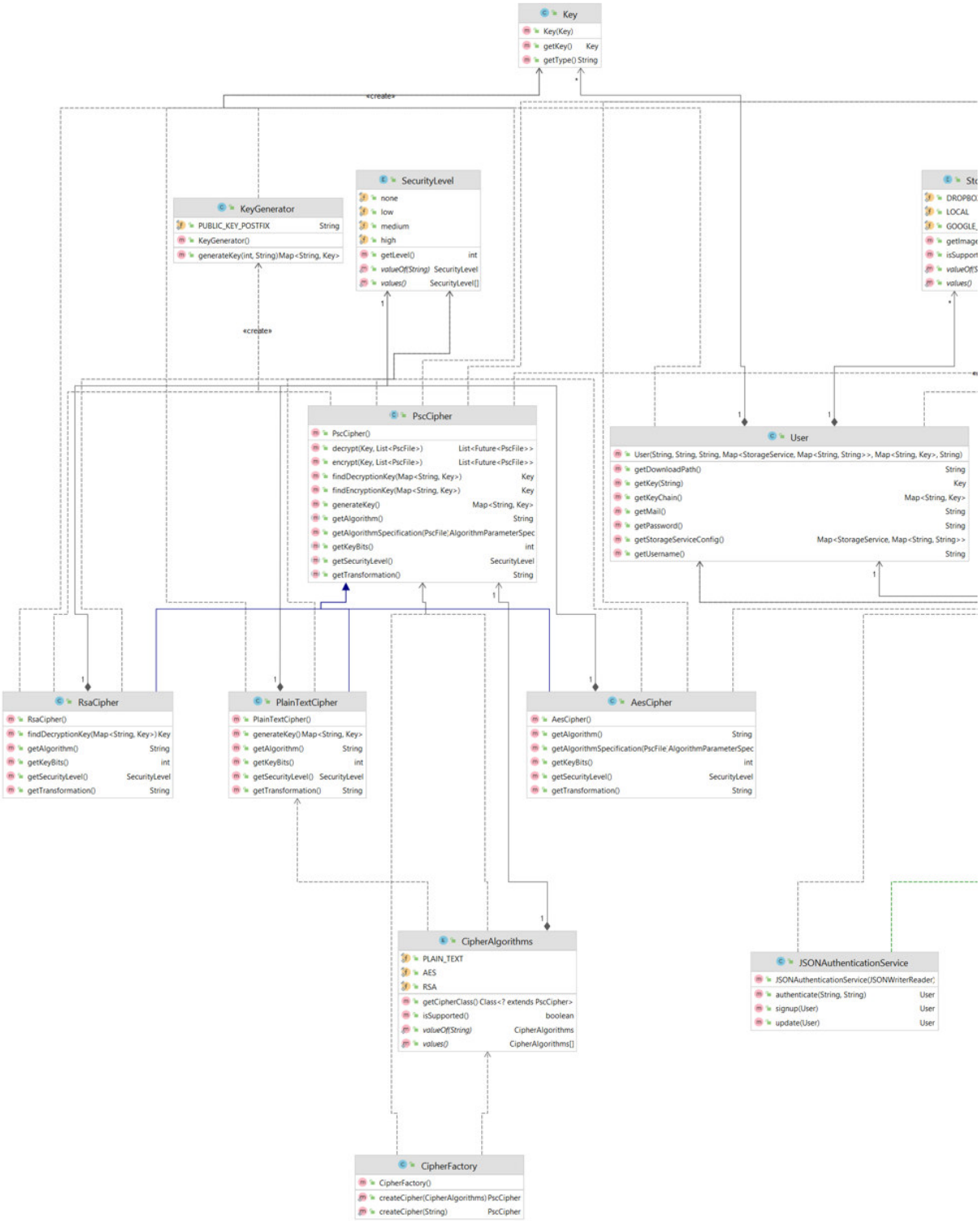
Gruppe 4, Pretty Secure Cloud

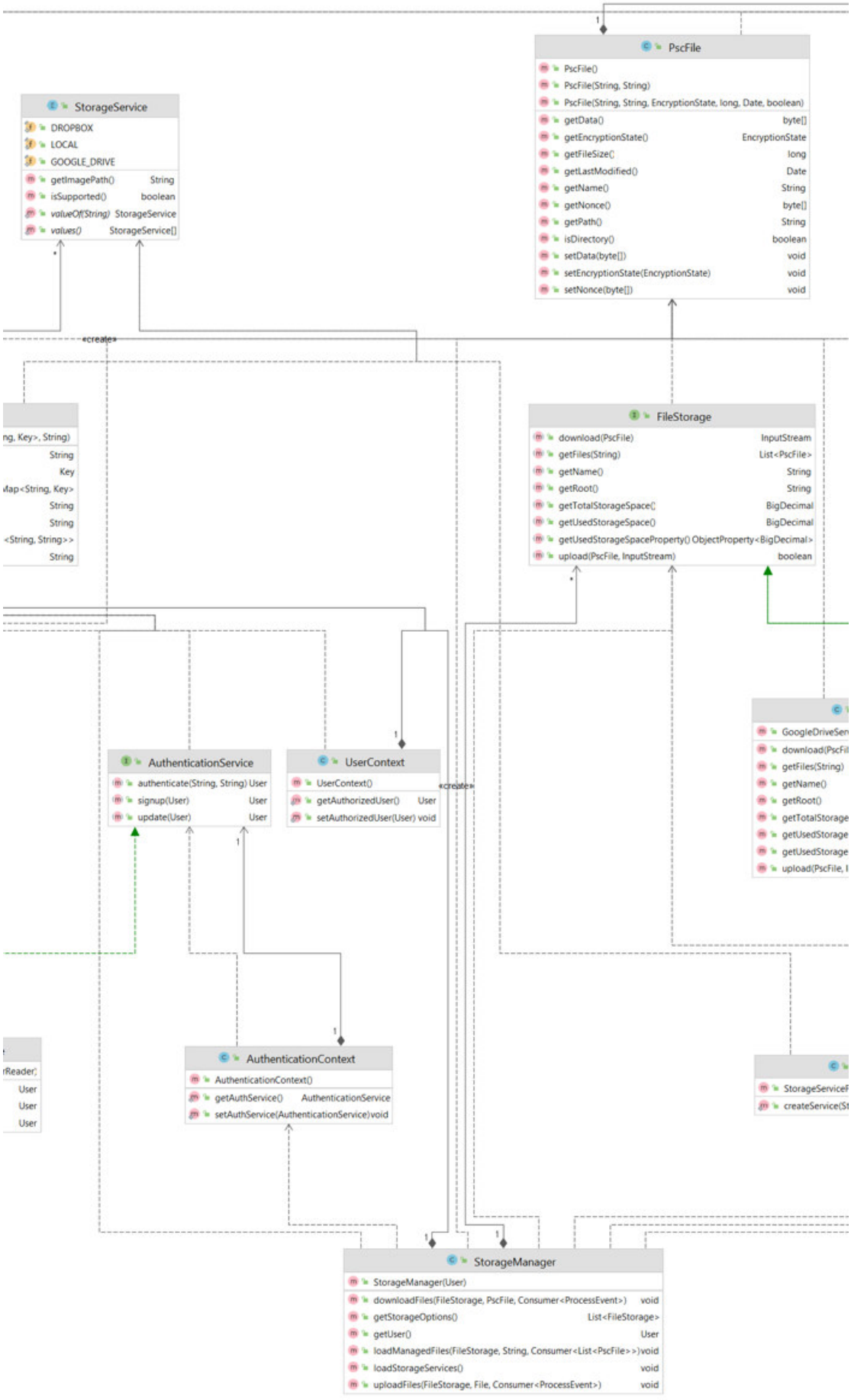
9 Anhang

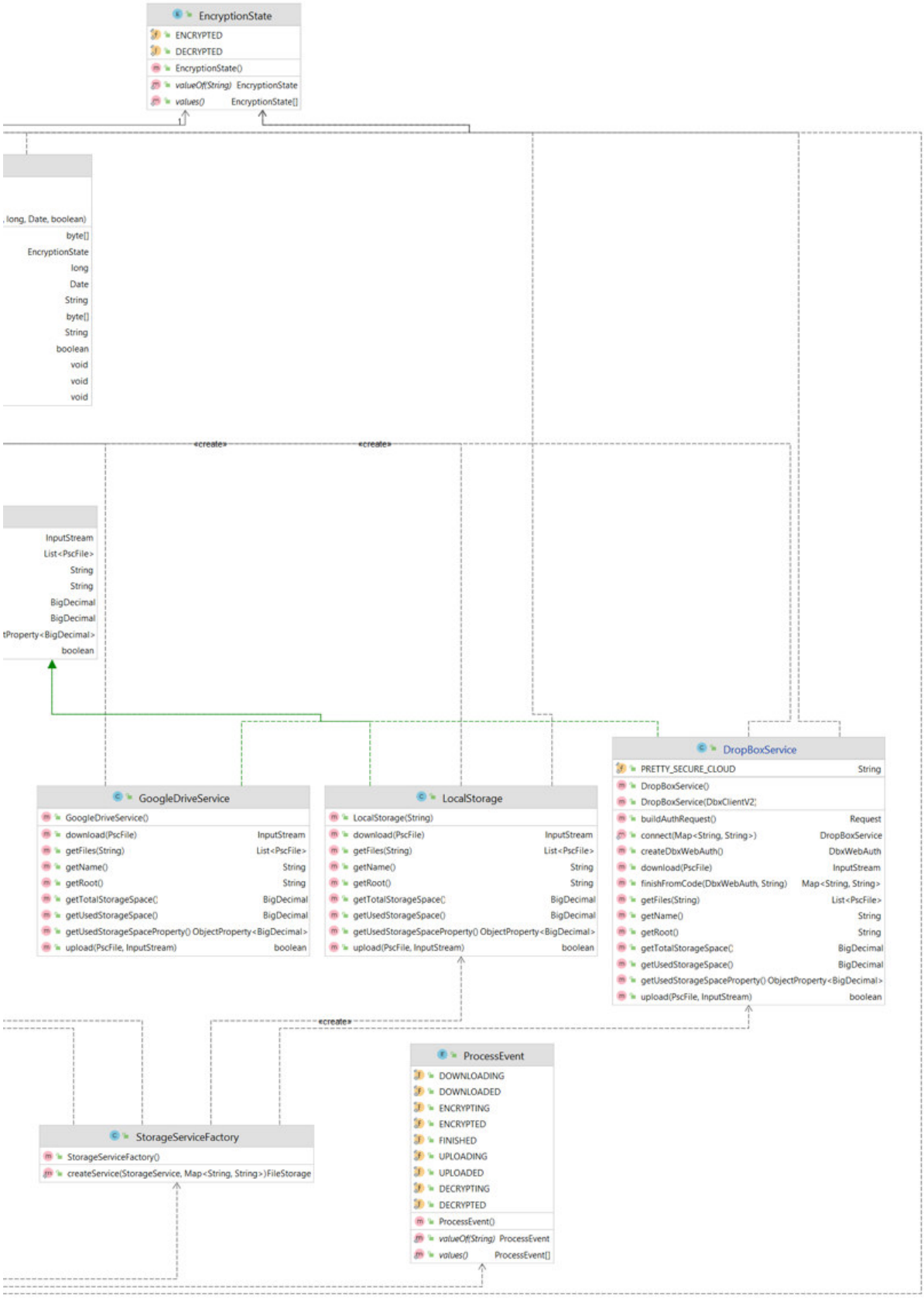
9.1 Automatisch generiertes Klassendiagramm (komplett)



9.2 Automatisch generiertes Klassendiagramm (vergrössert, aufgeteilt)




PscFile
PscFile()
PscFile(String, String)
PscFile(String, String, EncryptionState, long, Date, boolean)
getData() byte[]
getEncryptionState() EncryptionState
getFileSize() long
getLastModified() Date
getName() String
getNonce() byte[]
getPath() String
isDirectory() boolean
setData(byte[]) void
setEncryptionState(EncryptionState) void
setNonce(byte[]) void
StorageService
DROPBOX
LOCAL
GOOGLE_DRIVE
getImagePath() String
isSupported() boolean
valueOf(String) StorageService
values() StorageService[]
«create»
FileStorage
download(PscFile) InputStream
getFiles(String) List<PscFile>
getName() String
getRoot() String
getTotalStorageSpace() BigDecimal
getUsedStorageSpace() BigDecimal
getUsedStorageSpaceProperty() ObjectProperty<BigDecimal>
upload(PscFile, InputStream) boolean
AuthenticationService
authenticate(String, String) User
signup(User) User
update(User) User
UserContext
UserContext()
getAuthorizedUser() User
setAuthorizedUser(User) void
«create»
AuthenticationContext
AuthenticationContext()
getAuthService() AuthenticationService
setAuthService(AuthenticationService) void
StorageManager
StorageManager(User)
downloadFiles(FileStorage, PscFile, Consumer<ProcessEvent>) void
getStorageOptions() List<FileStorage>
getUser() User
loadManagedFiles(FileStorage, String, Consumer<List<PscFile>>) void
loadStorageServices() void
uploadFiles(FileStorage, File, Consumer<ProcessEvent>) void


EncryptionState
ENCRYPTED
DECRYPTED
EncryptionState()
valueOf(String) EncryptionState
values() EncryptionState[]
«create»
«create»
GoogleDriveService
GoogleDriveService()
download(PscFile) InputStream
getFiles(String) List<PscFile>
getName() String
getRoot() String
getTotalStorageSpace() BigDecimal
getUsedStorageSpace() BigDecimal
getUsedStorageSpaceProperty() ObjectProperty<BigDecimal>
upload(PscFile, InputStream) boolean
LocalStorage
LocalStorage(String)
download(PscFile) InputStream
getFiles(String) List<PscFile>
getName() String
getRoot() String
getTotalStorageSpace() BigDecimal
getUsedStorageSpace() BigDecimal
getUsedStorageSpaceProperty() ObjectProperty<BigDecimal>
upload(PscFile, InputStream) boolean
DropBoxService
PRETTY_SECURE_CLOUD String
DropBoxService()
DropBoxService(DbxClientV2)
buildAuthRequest() Request
connect(Map<String, String>) DropBoxService
createDbxWebAuth() DbxWebAuth
download(PscFile) InputStream
finishFromCode(DbxWebAuth, String) Map<String, String>
getFiles(String) List<PscFile>
getName() String
getRoot() String
getTotalStorageSpace() BigDecimal
getUsedStorageSpace() BigDecimal
getUsedStorageSpaceProperty() ObjectProperty<BigDecimal>
upload(PscFile, InputStream) boolean
«create»
ProcessEvent
DOWNLOADING
DOWNLOADED
ENCRYPTING
ENCRYPTED
FINISHED
UPLOADING
UPLOADED
DECRYPTING
DECRYPTED
ProcessEvent()
valueOf(String) ProcessEvent
values() ProcessEvent[]
StorageServiceFactory
StorageServiceFactory()
createService(StorageService, Map<String, String>)FileStorage

9.3 GUI Test Excel

	A	B	C	D	E	F	G
1		1.0.0.0	**Test Status**	**Release**	**Wer testet?**	**Status Funktionell**	**Status Optisch**
2	Getestet = grün						
3	TC0010	Login gültig	OK	1.0	RS	OK	OK
4	TC0011	Login ungültig	OK	1.0	RS	OK	OK
5	TC0012	Registrierung gültig	OK	1.0	RS	OK	OK
6	TC0013	Registrierung ungültig	Retest	1.0	RS	NOT OK	OK
7	TC0014	File upload gültig	OK	1.0	RS	OK	OK
8	TC0015	File upload ungültig	not OK	1.0	RS	NOT OK	NOT OK
9	TC0016	File download gültig	OK	1.0	RS	OK	OK
10	TC0017	File download ungültig	not OK	1.0	RS	NOT OK	NOT OK
11	TC0018	File Browser	OK	1.0	RS	OK	OK
12	TC0019	Log in	OK	1.0	RS	OK	OK
13	TC0020	Setup your account	OK	1.0	RS	OK	OK
14	TC0021	Choose your encryption	OK	1.0	RS	OK	OK
15	TC0022	Choose your cloud storage	OK	1.0	RS	OK	OK
16							

Das entsprechende Dokument befindet sich als elektronische Version im *repository*.

9.4 Gradle Clean Test HTML Reports

Test Summary

39	0	0	4.209s
tests	failures	ignored	duration

100% successful

Packages Classes

Package	Tests	Failures	Ignored	Duration	Success rate
default-package	1	0	0	0s	100%
ch.psc.datasource	2	0	0	0.458s	100%
ch.psc.domain.cipher	26	0	0	1.534s	100%
ch.psc.domain.storage	2	0	0	0.092s	100%
ch.psc.domain.storage.service	4	0	0	2.009s	100%
ch.psc.domain.user	4	0	0	0.116s	100%

Class EncryptionDecryption_AES_ECB_PKCS5Padding

all > default-package > EncryptionDecryption_AES_ECB_PKCS5Padding

1	0	0	0s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
encryptAndDecryptMessageSubsequentlyCompare()	0s	passed

Package ch.psc.datasource

all > ch.psc.datasource

2	0	0	0.458s
tests	failures	ignored	duration

100% successful

Classes

Class	Tests	Failures	Ignored	Duration	Success rate
JSONWriterReaderTest	2	0	0	0.458s	100%

JSONWriterReaderTest

all > ch.psc.datasource > JSONWriterReaderTest

2	0	0	0.458s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
readPlayerData()	0.384s	passed
writePlayerData()	0.074s	passed

Package ch.psc.domain.cipher

all > ch.psc.domain.cipher

26	0	0	1.534s
tests	failures	ignored	duration

100% successful

Classes

Class	Tests	Failures	Ignored	Duration	Success rate
AesCipherTest	5	0	0	0.108s	100%
CipherTest	4	0	0	0.032s	100%
IntegrationTest	2	0	0	0.564s	100%
KeyGeneratorTest	6	0	0	0.120s	100%
PlainTextCipherTest	4	0	0	0.023s	100%
RsaCipherTest	5	0	0	0.687s	100%

AesCipherTest

all > ch.psc.domain.cipher > AesCipherTest

5	0	0	0.108s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
decryptTooShortKey()	0.033s	passed
encryptDecryptTest()	0.029s	passed
encryptTooShortKey()	0.008s	passed
nonceTest()	0.013s	passed
randomResultTest()	0.025s	passed

Package ch.psc.domain.storage

all > ch.psc.domain.storage

2	0	0	0.092s
tests	failures	ignored	duration

100% successful

Classes

Class	Tests	Failures	Ignored	Duration	Success rate
StorageManagerTest	2	0	0	0.092s	100%

CipherTest

all > ch.psc.domain.cipher > CipherTest

4	0	0	0.032s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
generateKeyTestAES128()	0.004s	passed
generateUnimplementedKeyTest()	0.016s	passed
getCipherTestAES()	0.001s	passed
getCipherTestUnimplemented()	0.011s	passed

IntegrationTest

all > ch.psc.domain.cipher > IntegrationTest

2	0	0	0.564s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
aesIntegrationTest()	0.220s	passed
rsaIntegragionTest()	0.344s	passed

KeyGeneratorTest

all > ch.psc.domain.cipher > KeyGeneratorTest

6	0	0	0.120s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
aes128Test()	0.002s	passed
aes192Test()	0.003s	passed
aes256Test()	0.010s	passed
rsa1024Test()	0.032s	passed
rsa2048Test()	0.072s	passed
unsupportedAlgorithmTest()	0.001s	passed

PlainTextCipherTest

all > ch.psc.domain.cipher > PlainTextCipherTest

4	0	0	0.023s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
decryptMultipleTest()	0.006s	passed
decryptionTest()	0.004s	passed
encryptMultipleTest()	0.010s	passed
encryptionTest()	0.003s	passed

RsaCipherTest

all > ch.psc.domain.cipher > RsaCipherTest

5	0	0	0.687s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
decryptTooShortKey()	0.195s	passed
encryptDecryptTest()	0.100s	passed
encryptTooShortKey()	0.089s	passed
nonceTest()	0.192s	passed
randomResultTest()	0.111s	passed

Package ch.psc.domain.storage

all > ch.psc.domain.storage

2	0	0	0.092s
tests	failures	ignored	duration

100% successful

Classes

Class	Tests	Failures	Ignored	Duration	Success rate
StorageManagerTest	2	0	0	0.092s	100%

StorageManagerTest

all > ch.psc.domain.storage > StorageManagerTest

2	0	0	0.092s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
testDownloadFiles()	0.008s	passed
testUploadFiles()	0.084s	passed

Package ch.psc.domain.storage.service

all > ch.psc.domain.storage.service

4	0	0	2.009s
tests	failures	ignored	duration

100% successful

Classes

Class	Tests	Failures	Ignored	Duration	Success rate
DropBoxServiceTest	4	0	0	2.009s	100%

DropBoxServiceTest

all > ch.psc.domain.storage.service > DropBoxServiceTest

4	0	0	2.009s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
buildAuthRequest()	0.005s	passed
finishFromCodeFailed()	0.022s	passed
finishFromCodeSuccessful()	0.085s	passed
getAvailableStorageSpace()	1.897s	passed

Package ch.psc.domain.user

all > ch.psc.domain.user

4	0	0	0.116s
tests	failures	ignored	duration

100% successful

Classes

Class	Tests	Failures	Ignored	Duration	Success rate
JSONAuthenticationServiceTest	4	0	0	0.116s	100%

JSONAuthenticationServiceTest

all > ch.psc.domain.user > JSONAuthenticationServiceTest

4	0	0	0.116s
tests	failures	ignored	duration

100% successful

Tests | Standard error

Test	Duration	Result
authenticateNotSuccessful()	0.005s	passed
authenticateSuccessful()	0.066s	passed
signupSuccessful()	0.032s	passed
updateUser()	0.013s	passed

JSONAuthenticationServiceTest

all > ch.psc.domain.user > JSONAuthenticationServiceTest

4	0	0	0.116s
tests	failures	ignored	duration

100% successful

Tests | Standard error

Test	Duration	Result
authenticateNotSuccessful()	0.005s	passed
authenticateSuccessful()	0.066s	passed
signupSuccessful()	0.032s	passed
updateUser()	0.013s	passed