hochschule mannheim

Fakultät Informationstechnik
Studiengang Technische Informatik

Bericht zum praktischen Studiensemester

Vorgelegt von Marius Cerwenetz

Studiengang Technische Informatik

Firma Mint Medical GmbH

Name	Marius Cerwenetz
Matrikelnummer	1824006
Studiengang	Technische Informatik
Fachsemester	6
Praktikumszeitraum	23.03.2021 bis 17.09.2021
Präsenztage	111
Firma	Mint Medical GmbH
Standort	Heidelberg
Abteilung	Entwicklung
Betreuer	Johannes Kast

Datum, Johannes Kast

Firmenstempel

Selbstständigkeitserklärung

Ich versichere, dass ich diesen Bericht zum praktischen Studiensemester selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Die Stellen, an denen Inhalte aus den Quellen verwendet wurden, sind als solche eindeutig gekennzeichnet. Die Arbeit hat in gleicher oder ähnlicher Form bei keinem anderen Prüfungsverfahren vorgelegen.

Datum, Ort	Marius Cerwenetz

Sperrvermerk

Der vorliegende Bericht enthält interne und teilweise vertrauliche Daten der Firma Mint Medical GmbH. Der Bericht darf daher zu keinen anderen als Prüfungszwecken verwendet werden. Insbesondere ist die Vervielfältigung und Veröffentlichung von Berichtinhalten oder Teilen davon nur mit Zustimmung des Unternehmens erlaubt.

Zusammenfassung

Mint Lesion ist eine Software die in der Radiologie für sturkturierte Beefundungen eingesetzt wird. Sie wird vom Unternehmen Mint Medical entwickelt und vertrieben. Um die Anwendung ortsunabhängig und dynamisch benutzen zu können wurde in der Vergangenheit eine Webanwendung, Mint Web, entwickelt, die das Benutzen essentieller Funktionen von Mint Lesion unterstützt. Die Softwarelösung Mint Analytics erstellt aus Daten von Mint Lesion Statistiken und Übersichten der aufgenommen Cases und Studien.

In Zusammenarbeit mit den Entwickler:innen wurde Mint Analytics in Mint Web integriert. Relevante Anforderung waren dort der automatisierte, sicher implementierte Anmeldevorgang. Ein Authentifizierungs- und Konfigurationskonzept der Vereinigung der beiden Lösungen wurde erarbeitet und umgesetzt. Zur Verwendung kamen dabei unterschiedliche Konfigurationen von JSON Webtokens. Die Authentifizierung konnte an bereits vorhandene Schnittstellen von Mint Analytics angebunden werden. Des weiteren erfolgte Auslagerung Rechenintensiver Aufgaben sowie Konfigurationsschritte an Mint Lesion.

Die Lösung unterstützte die gesetzten Anforderungen. Nutzer, die bereits in Mint Analytics angemeldet waren konnten direkt auf Mint Analytics zugreifen. Durch eine Anpassung der Konfiguration von Mint Analytics konnten bisherige Sicherheitsstandards eingehalten werden.

Inhaltsverzeichnis

Kapitel 1

Einführung

Der Umfang der Aufgaben die Radiologinnen oder Radiologen in ihrem Beruf ausführen müssen hat sich in den letzen 20 Jahren erweitert. Heutzutage müssen Bilddaten befundet, strukturierte Befundungen gestellt und Studien vorangetrieben werden. Das Unternehmen Mint Medical GmbH aus Heidelberg entwickelte für die ersten beiden Tätigkeitsbereiche Lösungen. Mit Mint Lesion ist es beispielsweise möglich aus Computertomographie (CT)-Bildern mit Bildbearbeitungstools konkrete Befundungen zu stellen und zu speichern, sowie diese nach einem auf Regeln basierenden System, vorhandenen Studien zuzuführen. Um diese Funktionalitäten auch Standortunabhängig anbieten zu können wurde Mint Web entwickelt. Dies ist eine Webanwendung, die die Funktionalitäten von Mint Lesion abstrahiert und sie über eine Weboberfläche bereitstellt. Ein weiteres Produkt des Unternehmens, Mint Analytics, bietet eine automatische Datenauswertung der in Mint Lesion anfallenden Daten und stellt diese auf Dashboards dar.

Bei den beiden Produkten Mint Web und Mint Analytics handelt es sich um Webanwendungen, die getrennt von einander existieren. Beide Anwendungen verfügen über eine Nutzerverwaltung, die mit unterschiedlichen Nutzersystemen arbeiten kann. Kunden die beispielsweise in Mint Web arbeiten und auf Mint Analytics zugreifen wollen, müssen sich, obwohl Sie in MintWeb bereits eingeloggt waren, unter Umständen mit einem anderen Nutzeraccount wieder in Mint Analytics einloggen.

Um die Benutzerfreundlichkeit zu verbessern und um ein erneutes Anmelden mit anderen Nutzerdaten zu ersetzen, sollte eine Signle-Sign-On-Lösung (SSO) entwickelt werden. Es soll möglich sein Mint Analytics direkt in Mint Web zu verwenden. Der dafür vorgesehene Mechanismus soll sicher sein, so dass trotz der Anbindung, also Aufweichung, keine Gefahr durch Angreifer:innen entstehen sollte. Des weiteren soll die Verwendung von unterschiedlichen Nutzerrollen mit unterschiedlichen Rechten unterstützt werden. Die Authentifizierung soll ebenfalls verschlüsselt ablaufen, so, dass beide Dienste Login-Daten nicht im Klartext austauschen. Risiken in der IT-Sicherheit spielen bei MintMedical, gerade als Medizinprodukthersteller eine große Rolle.

Kapitel 2

Mint Medical GmbH

Mint Medical aus Heidelberg ist ein medizintechnisches Unternehmen, das in der Softwareentwicklung tätig ist. Seit 2010 werden dort Programme für Radiologinnen und Radiologen entwickelt, die eine computergestützte Befundung nach internationalen medizinischen Standards ermöglicht. Das Unternehmen ging aus einem Projekt innerhalb des deutschen Krebsforschungszentrums (DKFZ) als eigenständiger Betrieb hervor. Die strukturierten Befundungen werden für medizinische Studien verwendet, bei denen Sie durch Auswertung unter anderem für die Pharmazeutische Industrie Erkenntnisse erbringen. Im Jahr 2018 konnte durch eine Studie, die die Softwareprodukte von MintMedical verwendete, ein Break-Through-Medikament zugelassen werden. 2021 wurde das Unternehmen Teil der Brainlab AG aus München, die ebenfalls ein medizintechnisches Unternehmen ist. Brainlab spezialisiert sich auf Hard- und Softwareentwicklung für Systeme die in Operationen und in der Strahlentherapie eingesetzt werden.[12]. 2021 entwickelte Mint Medical zusammen mit anderen Unternehmen und Forschungseinrichtungen wie dem DKFZ, der TU-Darmstadt, Racoon, ein multizentrisches Forschungsnetzwerk für Covid19-Studien. Deutschlandweit zentralisieren dort Radiologinnen und Radiologen aus verschiedenen Unikliniken ihre Befundungen und Studien. [5] Die Zentralisierung soll es vereinfachen Erkenntnisse über Covid19 zu sammeln und potentielle Behandlungen auszumachen. Zusätzlich soll es eine epidemiologische Übersicht über die Covid-Fälle darstellen.

Ein Weiteres Projekt ist die Integration von Smart on FIHR. FIHR ist eine Abkürzung und steht für Fast Healthcare Interopable Resource. SMART steht für Substitutable Medical Applications, reusable technologies. Es handelt sich um einen Datenaustausch-Standard für Daten im Gesundheitswesen, wie zum Beispiel Patient:innendaten. SMART on FIHR bietet eine Art AppStore-Konzept für Elektronische Patientendaten und deren Verarbeitung. Anwendungen können so, unter Voraussetzung des Einverständnisses des/der Patient:in, dediziert auf die minimal notwendigen Daten zugreifen. Aktuell findet der Standard in Deutschland trotz elektronischer Patientenakte noch nicht viel Unterstützung in der Softwareentwicklung. In den USA ist er allerdings weit verbreitet. Eine Anbindung an SMART on FIHR ist somit auch mit einer größeren Kompatibilität anderer Softwarelösungen

verbunden.

MintMedical hat zwei Standorte: Heidelberg-Neuenheim und Hamilton, in New Jersey in den USA. Aktuell arbeiten circa 50 Mitarbeiter:innen in Deutschland und den USA, die unter Leitung des Geschäftsführers Dr. Matthias Baumhauer die Entwicklung vorantreiben. Das Hauptprodukt Mint Lesion ist sowohl in den USA als auch in Deutschland von der FDA und dem TÜV als Medizinprodukt zugelassen.

Kapitel 3

Mint Lesion

Mint Lesion ist eine Software, die zur strukturierten Befundung von von radiologischen Untersuchungen verwendet wird. Die Einsatzorte sind vielfältig. Weltweit arbeiten Artztpraxen, Pharmaunternehmen, Kliniken oder Auftragsforschungsinstitute (Clinical Research Organisations bzw. CROs) damit. Letztere sind private Unternehmen, die klinische Studien überwachen, begleiten, überprüfen und organisieren. [1] Sie sind gerade für die Studienverwaltung eine interessante Interessentengruppe. Mint Lesion bietet die Möglichkeit Befundungen und Läsionen auf Grundlage medizinischer Bildaufnahmen wie CT, Magnetresonanztomographie (MRT) oder Positronen-Emissions-Tomographie (PET) zu erstellen. Durch Bilderkennungsalgorithmen kann die Größe eines Tumors bestimmt werden. Auch 3D-Erfassungen sind möglich. Befunde können bei mehreren Untersuchungen auch aktualisiert werden. Mint Lesion erstellt aus den Befundungen automatisch einen Bericht nach krankheitsspezifischen Bewertungskriterien, wie RECIST oder PI-RADS. Durch diese Kriterien ist ist es außerdem möglich Studien mehrerer Patient:innen mit jeweils mehreren Befundungen zusammenzufassen. Dadurch können Wirksamkeitsstudien mit relativ wenig Aufwand und Klarheit generiert werden. Eine Nutzerin oder ein Nutzer durchläuft dabei sequenziell die Reiter *Verwalten*, *Befunden* und *Berichten*.

Unter *Verwalten* können Patient:innen und Studien erzeugt, betrachtet, verändert oder gelöscht werden. Radiologinnen und Radiologen arbeiten meist in Kliniken in getrennten Bereichen und besitzen für die Ausstellung von Befundungen entsprechende Rechte. Nutzer:innen müssen so unterschiedliche Aufgaben erfüllen, wie zum Beispiel eine Befund freigeben oder bestätigen. Diese Aufgaben sind unter der Liste *Tasks* zusammengefasst. Für die Befundung sind Läsionen an Bilddaten nötig. Läsionen sind Untersuchungen auf zum Beispiel CT-Bilddateien. Diese Bilddateien können von einem externen Speicher in die Software geladen werden. Sie müssen im Digital Imaging and Communications in Medicine (DICOM)-Format vorliegen und können aus dem Picture Archiving and Communication System (PACS) des Krankenhausinformationssystems (KIS) importiert werden.

Unter *Befunden* kann der Nutzer oder die Nutzerin mit Hilfe von Werkzeugen Läsio-

nen erstellen. Diese können planar oder volumetrisch sein. Also 2-Dimensional oder 3-Dimensional. Größe und Form des markierten Bereichs werden mit Hilfe von Bildverarbeitungsalgorithmen automatisch erkannt und umschlossen. Um die Läsion mit vergangenen Läsionen zu vergleichen oder mehrere unterschiedliche Schichten des gleichen Bilds gleichzeitig anzuzeigen können so bis zu vier Bilder parallel betrachtet werden. Nachdem die Läsionen erstellt wurden kann der Nutzer oder die Nutzerin einen nach einem Befundungsstandard erstellten Fragebogen, dem Electronic Case Report Form (eCRF) ausfüllen um einen Befund anzustellen. Der Fall (Case) hat nun nach erster Diagnose den Zustand *In Progress* und muss erst von einem Nutzer oder einer Nutzerin mit erhöhten Rechten auf *Approved* gestellt werden.

Die Generierung von Befunden im PDF-Format erfolgt unter dem Reiter *Berichten*. Berichte umfassen sowohl die einzelnen Läsionen und deren Größen und Ort als auch den Verlauf der Läsion über den gesamten Untersuchungszeitraum hinweg. Diese werden visuell von Diagrammen und Graphen untermalt.

Kapitel 4

Single-Sign-On-Lösung

4.1 Einführung

In diesem Kapitel werden die verwendeten Softwarekomponenten vorgestellt. Für die Single-Sign-On-Lösung wurden die beiden Produkte Mint Web und Mint Analytics miteinander verbunden. Diese bauen auf mehreren Frameworks wie gRPC und Elasticsearch auf, die in diesem Kapitel behandelt werden. Die Verknüpfung der beiden Lösungen Mint Web und Mint Analytics geschah unter Verwendung der auf JSON basierenden Access-Token Variante JSON Web Token. Diese Authentifizierungsmethode wird später in einem separaten Kapitel behandelt. Gute Voraussetzungen bot dabei das Security-Plugin von OpenDistro, welche verschiedene Authentifizierungsmechanismen unterstützte.

4.1.1 Mint Analytics

Mint Analytics ist eine Datenvisualisierungssoftware für Mint Lesion. Die Zielgruppe sind Ärzt:innen und medizinisch-technische Mitarbeite:innen, die an Einsatzorten wie Krankenhäusern, Radiologien oder Forschungszentren arbeiten. Durch die Exportfunktion des Lesion Clients können Daten in Logfiles geschrieben und anschließend mit Mint Analytics verarbeitet werden.

Mint Analytics besteht aus drei verschiedenen Komponenten: Logstash, Elasticsearch und Kibana. Sie sind auch als ELK-Stack bekannt und werden von dem Unternehmen Elastic NV entwickelt. Genauer gesagt wird nicht das teilweise proprietäre Produkt Elasticsearch von Elastic eingesetzt, sondern die OpenSource Variante OpenDistro.

Elastic stellte im Januar 2021 nach drei Jahren ihr Lizenzmodell um von einer freien Apache 2.0 Lizenz auf eine ServerSide Public License, wie bei MongoDB und eine selbst entworfene sogenannte Elastic License. [3]. Vorausgegangen war ein Streit zwischen Elastic und Amazon Inc. Amazon stellte den damals noch unter Apache-2.0 Lizenz existierenden Dienst Elasticsearch für ihre Cloudservices bereit. [4] Elastic versuchte das Produkt zu

schützen und stellte Elasticsearch unter ein teilweise proprietäres Lizenzmodell. Amazon spaltete darauf hin das Repository (hard-fork) und entwickelt seitdem unter OpenSource Lizenz an OpenDistro. Dieser Klon soll OpenSource bleiben. Im April 2021 wurde OpenDistro umbenannt in OpenSearch. [2]

OpenDistro, OpenSearch oder Elasticsearch sind Teil des ELK-Stacks. Logstash dient zur Akquirierung von Daten aus Loginformationen. Diese werden mit Hilfe von Elasticsearch gefiltert, sortiert und einkategorisiert um so am Ende von Kibana im Webbrowser präsentiert zu werden. Der Datenexport von Mint Lesion kann zu festgelegten Zeitpunkten oder in Echtzeit erfolgen. Zur Auswertung werden die Daten durch voreingestellte Filter kategorisiert und anschließend in den Dashboards präsentiert. Die Dashboards, also Kibana, sind wahlweise über http oder https auf port 5601 entweder auf localhost oder einer eingestellten anderen Adresse aus erreichbar. Der Elasticsearch-Service ist ebenfalls über http oder https und localhost auf Port 9200 erreichbar. Gestartet werden die Services primär über einen Installer, der mit Hilfe des Windows-Service-Installationstools Non-Sucking Service Manager (NSSM) auf Windowsrechnern die beiden Dienste installiert und startet. Dafür werden jedoch Administratorberechtigungen benötigt, was durch Sicherheitsrichtlinien von Mint Medical nicht auf allen Rechnern möglich ist. Deshalb gibt es die Alternative beide Dienste jeweils aus einer eigenen comannd-line (cmd) heraus mit Hilfe eines batchscripts zu starten. Die Dienste sind an die Lebensdauer der beiden Kommandozeilen gekoppelt und können nicht im Hintergrund ausgeführt werden. Authentifiziert wird an der Kibana-Schnittstelle über das security plugin von OpenDistro. Das Plugin ist Teil von Elasticsearch, nicht von Kibana und ermöglicht vielfältige Authentifizierungsmöglichkeiten. Die Konfiguration erfolgt über unterschiedliche Konfigurationsdateien. Änderungen dieser Dateien haben keinen direkten Einfluss auf das Verhalten von Elasticsearch. Ein Neustart der Dienste bewirkt ebenfalls keine Änderung. Elasticsearch speichert die aktuell gültige Konfiguration intern in einer Datenbank beziehungsweise an einem Index. Änderungen der Konfiguration müssen durch ein batch-script, dem securityadmin-Script, an diesen Index geschrieben werden. Dieses muss nach jeder Änderung der Config-Files ausgeführt werden. Mint Analytics funktioniert als klassische Webanwendung und ist auf Port 5200 erreichbar.

4.1.2 Mint Web

Die Mint-Webapp ist eine Abbildung des Mint Lesion Clients in Form einer webbasierten Browseranwendung. Sie bildet, bis auf die grafische Befundung und Segmentierung, die meisten Funktionen von Mint Lesion ab. Ein Vorteil ist, dass sie nicht auf dem Endgerät installiert sein muss und nur ein Browser vorausgesetzt ist. Als Framework werden ASP.NET und Fluxor/Blazor-Komponenten verwendet. Die Codesprache ist C#. Für die Datenkommunikation mit dem Mint Lesion Client kommt das Protokoll gRPC zum Einsatz.

ASP.NET ist ein Framework von Microsoft mit dem Webanwendungen entwickelt werden

können. Es bietet die Möglichkeit Anwendungslogik auf herkömmliche Weise in C# zu entwickeln, um diese dann an Web-Elemente anzubinden. Dieses Framework wird auch im Blazor Server - Framework verwendet. Blazor bzw. Blazor Server ist ebenfalls ein von Microsoft entwickeltes Framework zur Erstellung von Webanwendungen. Webseiten sind dort aus Komponenten aufgebaut. Komponenten können zum Beispiel ein Formular, ein Dialogfeld oder eine komplette Webseite sein. Für jede Komponente der Webanwendung existiert jeweils eine .blazor und eine .blazor.cs Datei. In der .blazor-Datei wird das Document Object Model der Website beschrieben. Komponenten können geschachtelt platziert und mit entsprechenden Callback-Funktionen verknüpft werden. Beschrieben werden Komponenten in der Razor-Markup Language. In dieser Beschreibungssprache können HTML-Dateien durch kleine Logikfunktionen und Variablen ergänzt werden. Die Komponentenbibliothek die zum Einsatz kommt ist *Blazor Web Components* von SyncFusion. Durch sie können vorgefertigte Seitenelemente wie Buttons oder Textfelder verwendet werden.

Die Anwendungslogik wird allerdings in der .blazor.cs Datei beschrieben. Hier sind Callback-Implementierungen und sonstige Logik vorgesehen. Die Komponenten besitzen einen Lebenszyklus. Ist eine Website eine Komponente und wird diese Website besucht, so erhält das Blazor-Framework einen Aufruf, die Website zu rendern. Bevor das Rendern erfolgt müssen zuererst zwei Methodenaufrufe erfolgen: `OnParametersSet()` und `OnInitializedAsnyc()`. Die erste Methode behandelt Parameter die per GET oder POST-Request an die Webpage geschickt wurden. Sie wird vor dem Rendern aufgerufen, da ihre Ausführung das Aussehen der Website verändern könnte. Danach wird `OnInitializedAsnyc()` aufgerufen. Diese Funktion bietet Entwickler:innen Platz Anwendungslogik im Voraus auszuführen. So können hier Beispielsweise Anfragen an andere Server, Anpassungen von Elementen im DOM oder die Erzeugung neuer Elemente erfolgen. Weitere Anpassungen nach dem Rendern sind auch in den Methoden `OnAfterRender()` oder `StateHasChanged()` möglich. Eine Übersicht dieses Lebenszyklus ist in Abbildung 4.1 zu sehen. Obwohl die Anwendungslogik bei Blazor-Server Serverseitig läuft, sollte bei Webanwendungen generell wenig Rechenleistung in der Anwendung aufgebracht werden. Die MintWebApp ist lediglich als weitere, kleinere Alternative zum MintLesion Client vorgesehen.

Die beiden Programme Mint Web und Mint Lesion laufen getrennt voneinander. Die Webanwendung und der Mint Lesion kommunizieren Daten und Aktionen die auf die gemeinsamen Daten angewendet wurden oder werden sollen per gRPC.

GRPC ist ein im Jahr 2015 von Google entwickeltes und mittlerweile unter OpenSource-Lizenz stehendes Protokoll, das verwendet wird um Funktionen bei verteilten Anwendungen aufzurufen. Dabei werden Daten und Aufrufe als Nachrichten in einer Interface Design Language beschrieben. Sowohl die Nachrichtenformate als auch die Namen der Funktionen werden darin beschrieben. Für den Nachrichtenaustausch stehen vier mögliche Aufrufe zur Auswahl: Unary-Calls bei denen jeder einzelne Aufruf exakt zu einer Antwort führt, Client-seitige Stream-Calls bei denen der Client mit mehreren Antworten in

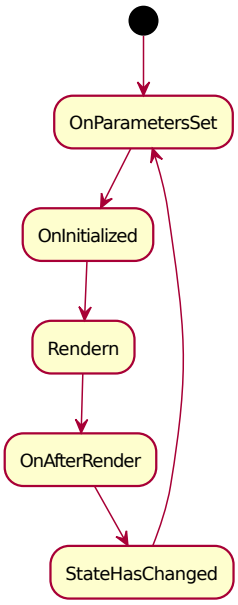

Abbildung 4.1: jwt.io: JWT in dekodierter Form

einem Stream antwortet, Server-seitige Stream-Calls bei deren Aufruf der Server mit mehreren Antworten in einem Stream antwortet und bidirektionale Stream-Calls bei denen Client und Server gleichzeitig auf getrennten Streams senden und empfangen. In diesem Projekt wurden allerdings nur Unary-Calls verwendet. Die Calls können synchron, also blockierend, oder asnychron, also mit paraleller Verarbeitung, aufgerufen werden. Aus der Beschreibungssprache werden mit Hilfe eine Codegenerators Server und Clientcode in der gewünschten Programmiersprache generiert. Auf der Serverseite müssen diese, in der Beschreibungssprache als Service geführten, Funktionsaufrufe an eine interne Funktion angebunden werden, die das entsprechende definierte Nachrichtenformat zurückgeben muss. Der generierte Code dient als vordefinierte Auflistung möglicher API-Aufrufe. Die Aufrufe werden daher auch *Stub* genannt, da sie, wie ein Stümmel, nur dazu dienen Nachrichten entsprechend der vordefinierten Standards auszutauschen. Durch die Möglichkeit Nachrichten mit TLS zu verschlüsseln und die beiden Kommunikationspartner miteinander zu authentifizieren, bietet das auf HTTP/2 basierende Protokoll hohe Sicherheitsstandards. Es bietet eine Datensparsame, schnellere und sicherere Alternative zu herkömmlichen REST Aufrufen.

4.2 Authentifizierung

In diesem Kapitel werden die Grundlagen der für die automatische Authentifizierung der SSO Lösung verwendeten JSON Web Tokens behandelt. Außerdem wird der Authentifizierungsablauf der beiden Komponenten Kibana und Elasticsearch erläutert. Dies soll die Grundlage bereiten um bei späteren Erwähnungen Details weglassen zu können.

Für die Authentifizierung von Nutzern oder Clients können neben Benutzername-Passwort-Kombinationen weitere Methoden zum Einsatz kommen um Nutzer:innen beispielsweise beim erneuten Besuchen einer Website nicht neu authentifizieren zu müssen. Eine dieser Möglichkeiten sind JSON Web Tokens (JWT). Sie sind Access-Token die in der Webentwicklung eingesetzt werden. Mit ihnen lassen sich zum Beispiel stateless sessions realisieren. Diese Sitzungen sind nicht serverseitig gespeichert. Sämtliche Sitzungsinformationen sind im JWT gespeichert. Informationen werden in Schlüsselwertpaaren, den sogenannten Claims festgehalten. JWTs bestehen aus einem Header, einer Payload und einer Signatur in welchen jeweils Claims als Felder stehen. Ein Beispiel eines JWTs ist in Listing 4.1 zu finden.

Im Header gibt es zum Beispiel die Claims `typ` und `alg`. `typ` ist immer `jwt` Bei `alg` handelt es sich um den Signaturalgorithmus auf den im Folgenden eingegangen wird.

In der Payload gibt es zum Beispiel die Claims `sub` (subject), `iat` (issued at) und `exp` (expiring at). Das subject, also der Betreff, wird oft verwendet um den Benutzernamen anzugeben. `iat` und `exp` sind Zeitangaben die den Gültigkeitszeitraum des JWT definieren. Beide Felder werden in unixtime angegeben.

Für die Signatur wird durch unterschiedliche Algorithmen aus header und payload eine Hashsumme generiert. Durch sie soll die Datenintegrität sichergestellt werden. Angreifer:innen können den JWT nicht verändern um zum Beispiel Benutzernamen zu ersetzen, ohne gleichzeitig die Signatur auch zu verändern. Beim Erstellen der Signatur ist, je nach Hashalgorithmus, ein Passwort oder ein x.509-Schlüsselpaar erforderlich. Dieses kennen Angreifer:innen nicht und können dadurch für seine/ihre abgewandelten Claims keine Signatur erstellen.

Unterstützte Hashalgorithmustypen des OpenDistro-Securityplugins sind zum Beispiel RS512, HS512, ES512 und PS512. Die Ziffer steht für die Länge des Schlüssels in Bit. Auch kürzere Längen sind möglich. Bei HS512 handelt es sich um einen symmetrischen Schlüssel, also ein 512 Bit langes Passwort. RS512, ES512 und PS512 sind asymmetrische Hashalgorithmen. Sie bilden ihre Signatur auf Grundlage eines Öffentlichen-Privaten-Schlüsselpaares. Die Zahl gibt dort die Länge des private Keys an. Genannte Hashalgorithmen sind jedoch lediglich eine abgekürzte Schreibweise der Algorithmen für das Feld im JWT. Die vollständigen Namen der Algorithmen lauten HMACSHA512 für HS512, RSASHA512 für RS512, ECDSASHA512 für ES512 und RSAPSSHA512 für PS512.

Für die Umsetzung des Projektes wurden lediglich die beiden Signierungsalgorithmen HS512 und RS512 verwendet.

Ein JWT existiert immer in einer Klartextversion und in einer mit dem Kodierungsverfahren base64-url kodierten Version. Während bei base64 drei Bytes zu vier Zeichen zusammengefasst werden, werden bei base64-url noch spezielle Zeichen wie +,/ und = ersetzt, welche in der url-Zeile nicht vorkommen dürfen. Ein Beispiel eines unverschlüsselten JWT ist in Listing 4.1 zu sehen. Die geschweiften Klammern markieren die beiden Bereiche Header und Payload. Darin sind jeweils die durch einen Doppelpunkt getrennten

Claims zu finden.

```
{
  "alg": "HS512",
  "typ": "JWT"
}
{
  "sub": "subject",
  "name": "Username",
  "role": "admin"
}
```

Listing 4.1: jwt klartext

Wie dieser in Listing 4.1 aufgeführte JWT in verschlüsselter Form aussieht ist in Listing 4.2 zu sehen. Header, Payload und Signatur werden durch einen Punkt getrennt.

```
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.
   eyJzdWIiOiJzdWJqZWN0IiwibmFtZSI6IlVzZXJuYW1lIiwicm9sZSI6ImFkbWluIn0
   .-Lmm2YthKZJxeVbcNxgoD5rQQ9HN-0vS6-
   zyI3RMJtL3sGqOUBj9fUodD4TGhA3qLPqnhJhbvLW6wCJdiqaaew
```

Listing 4.2: jwt verschlüsselt

JWTs werden im weiteren Verlauf verwendet um die Authentifizierung zu realisieren und bieten sich für die Lösung der Problemstellung an, die im nächsten Kapitel erläutert wird.

4.3 Motivation

Mint Web und Mint Analytics sind getrennte Anwendungen mit unterschiedlichen Nutzer:innen-Systemen. Nutzernamen und Passwörter können sich unterscheiden. Mint Analytics besaß zum Zeitpunkt der Entwicklung zwar, wie Mint Web, eine inoffizielle Anbindung an LDAP-Systeme, offiziell gab es allerdings nur die beiden Nutzer:innenrollen Administrator und Leser. Die LDAP-Anbindung an Mint Web war hier fortgeschrittener als bei Mint Analytics und in offiziellen Releases bereits implementiert. Wollte ein/e Nutzer:in nun aus der Webanwendung Mint Analytics starten, musste erst in einem separaten Tab oder Fenster die Website von Mint Analytics geöffnet werden, in dem er/sie sich dann zusätzlich anmelden musste. Es war sehr wahrscheinlich dass ein/e Nutzer:in diese Anmeldedaten nicht besäße, da es sich nicht um ihre/seine LDAP-Nutzer:innendaten handelte.

Ziel war es also Mint Analytics in Mint Web einzubinden. Eine Anzeige sollte ohne Login Prozess möglich sein, falls ein/e Nutzer:in sich bereits in Mint Web authentifiziert hatte, was eine engere Kopplung der beiden Produkte darstellte und die Benutzerfreundlichkeit verbesserte. Mint Analytics befand sich zum Zeitpunkt der Entwicklung noch in einem

Pre-Release Zustand. Mit dem Release von Mint Analytics ist geplant dieses Produkt häufiger in andere Lösungen einzubinden. Die Auswertung der anfallenden Daten soll so möglichst simpel einsehbar sein.

4.4 Umsetzung

Während der Entwicklung wurden Anforderungen an die Lösung häufig abgeändert oder erweitert wodurch eine statische, einmalige Kozeptionierung nicht möglich war. Die Umsetzung fand eher nach einem iterativen Ansatz statt, bei dem nach verschiedenen Meilensteinen die Anforderungen neu ausgehandelt wurden. Sie war geprägt von Codeanpassungen und der Verknüpfung erster Implementationsentwürfe mit der bisherigen Infrastruktur. Zusammenfassen lässt sie sich grob in die Abschnitte Integration der beiden Komponenten, automatische Generierung von JWTs und Anpassung der Optionsschnittstelle.

4.4.1 Integration von Kibana in die Webapp

Zuerst wurde die Anbindung beider Komponenten, Mint Analytics bzw. Kibana und Elasticsearch und der Webapp, aneinander realisiert. Eine neue Seite, `Kibana.razor`, wurde in Mint Web eingerichtet. Beim Aufrufen der Seite wurde in der Methode `OnInitializedAsync` ein Iframe von Kibana angezeigt dass das komplette Fenster ausfüllte. Ein Ausschnitt davon ist in Listing 4.3 zu finden. In Zeile 1 wird das Iframe nur geladen, wenn die Member-Variable `m_IframeLoaded` true ist. Der Zweck des Checks ist eine Authentifizierung abzuwarten um erst danach die Seite anzuzeigen. Dies soll verhindern, dass Nutzer:innen sich voreilig einloggen bevor Sie automatisch authentifiziert werden würden.

```
1 @if (m_IframeLoaded)
2 {
3     <iframe src="@IframeUrl" width="100%" height="100%" title="
        kibana"></iframe">
4 }
```

Listing 4.3: Kibana.razor: Iframe

Die beiden Plattformen sind nun rudimentär und statisch miteinander verbunden. Eine automatische Authentifizierung erfolgt noch nicht.

Um eine automatische Authentifizierung durchzuführen mussten in MintAnalytics andere Authentifizierungsschnittstellen verwendet werden. Wie breits in Kapitel 4.1.1 beschrieben, handelt es sich bei MintAnalytics um die Kombination aus Elasticsearch, Logstash und Kibana. Elasticsearch bietet die Möglichkeit über das Securityplugin von OpenDistro alternative Authentifizierungsmechanismen zu verwenden. [8] Die Standardauthentifizierungsmethode ist dabei HTTP Basic. Sie wird für eine Authentifizierung über das Webinterface verwendet. Weitere Möglichkeiten sind zum Beispiel die Nutzung eines Kerberos-Authentifizierungs- und Authorisierungs-Sytems, SAML, LDAP, OpenSSL Zertifikaten, OpenID oder die Authentifizierung über einen vertrauenswürdigen Proxy. Die

Entscheidung welche Authentifizierungsmethode verwendet werden sollte fiel auf die Authentifizierung durch JSON Webtoken, da dort der geringste Implementierungsaufwand erwartet wurde. Der JWT wurde zum Anfang mit Hilfe der Webapplikation JWT.IO erstellt. Die Funktionsweise ist in Abbildung 4.2 zu sehen. Erstellt wird hier ein JWT mit den claims: sub, iat, exp und roles. Er wird mit einem 512 bit langem Code und HS512 signiert. Der JWT läuft am 10.02.2053 um 15:38:27 ab. Die Anwendung generiert daraus

Abbildung 4.2: jwt.io: JWT in dekodierter Form

dann den verschlüsselten beziehungsweise kodierten beziehungsweise signierten JWT, wie er in Abbildung 4.3 zu sehen ist. Die farblichen Teile sind dabei jeweils die durch einen Punkt getrennten Abschnitte Header, Payload und Signatur.

Um die Authentifizierung per JWT auch bei MintAnalytics zuzulassen wurde in der Konfiguration des Opendistro-Securityplugins die JWT Authentifizierungsschnittstelle aktiviert. Auch in der Kibana-Konfiguration mussten Anpassungen durchgeführt werden.

Die Erprobung der Authentifizierung erfolgte zu diesem Zeitpunkt, als das Projekt sich

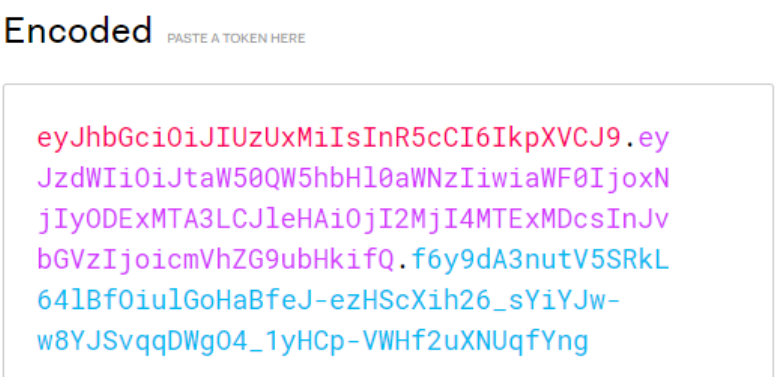

Abbildung 4.3: jwt.io: JWT in signierter Form

noch in der Konzeption befand, mit Hilfe des Programms Postman. Postman ist eine Software des Softwareherstellers Postman Inc., das für das Testen von APIs verwendet werden kann. Damit ist es unter Anderem möglich HTTP Requests zu erstellen um sie dann an einen Server zu senden. Diese Funktion erweist sich als hilfreich um die Authentifizierung bei gleichzeitiger Variierung der Parameter wie Server-Ip-Adresse oder unterschiedlichen Portnummern automatisiert zu testen. Denn auch hier gab es, ähnlich wie bei der Generierung der JWTs oder der Konfiguration von Elasticsearch und Kibana keine oder unzureichende Dokumentation oder Vorlagen seitens der Entwickler von Elastic oder des OpenDistro-Securityplugins. Um die Authentifizierung wirklich zu vollziehen, wird in einem HTTP-GET-Request in das Header Feld `Authorization`, der vorher generierte JWT in seiner verschlüsselten Form eingefügt und abgeschickt. Bei einer fehlerhaften Anfrage wird der Zugriff mit dem Statuscode 403 verwehrt. Durch erproben mit unterschiedlichen Feldern im JWT und variieren der Request-Parameter konnte eine Anmeldung erfolgreich durchgeführt werden. Das Proof of Concept galt als erfüllt.

Nun war eine automatische Generierung der JWTs, sowie die Anbindung der WebApp an die Schnittstelle von Kibana zu implementieren. Hierfür wurde nun beim Aufruf der Seite, wie in Postman, ein GET Request gestartet, in dem im Authorization-Feld des Headers der verschlüsselte JWT stand. Der konkrete Ablauf ist in Listing 4.4 zu finden.

In Zeile 11 wird die Zuweisung des JWT verkürzt dargestellt. Der JWT wird zu diesem Zeitpunkt noch nicht automatisch generiert. Ein Blanko-jwt kommt bisher zum Einsatz. Er wurde ebenfalls mit dem Webtoo jwt.io erstellt. Das Setzen der Authentifizierung erfolgt in Zeile 12. Bei der Instanziierung des `HTTPClientHandlers` in Zeile 1 bis 4 wurde die Zertifikatsvalidierung für HTTPS Verbindungen deaktiviert, da Sie zum Testen auf Systmen mit selbst unterschriebenen Zertifikaten eher hinderlich gewesen wäre. Die Testsysteme bei MintMedical besitzen keine durch eine zentrale CA unterschriebene SSl-Zertifikate. Wird über den `HTTPClientHandler` eine Verbindung über HTTPS aufgebaut, wird diese normalerweise abgebrochen, wenn das TLS-Zertifkat des Website nicht durch eine öffentliche CA unterschrieben ist. Die Funktion lässt sich jedoch deaktivieren. Das Request

wird in Zeile 6 erzeugt und erhält sowohl die Ziel-URL, als auch die HTTP-Methode. In Zeile 17 wird das Request über den `HttpClient` Client abgesendet.

```
1  var httpClientHandler = new HttpClientHandler
2  {
3        ServerCertificateCustomValidationCallback = (message, cert,
             chain, sslPolicyErrors) => true
4  };
5  var client = new HttpClient(httpClientHandler);
6  var request = new HttpRequestMessage
7  {
8        RequestUri = new Uri(m_URL),
9        Method=HttpMethod.Get,
10 };
11 var jwt="JWT";
12 request.Headers.Authorization = new AuthenticationHeaderValue(
       jwt);
13 HttpResponseMessage response=null;
14 var success=false;
15 try
16 {
17       response = await client.SendAsync(request);
18       success = true;
19       }
20 }
```

Listing 4.4: Kibana.razor.cs::GetCookie(): HTTP Request

Im Falle der erfolgreichen Authentifizierung wird in der Response eine Anweisung `Set-Cookie` gesendet. Der Cookie beinhaltet einen security_token, der für Kibana, beziehungsweise Elasticsearch, als Session-Cookie mit impliziter Autorisierung verwendet wird. Käme jemand in den Besitz dieses security-tokens und würde ihn sich in seinem Browser als Cookie speichern, würde Mint Analytics ihn/sie automatisch authentifizieren. Der bei der erfolgreichen Authentifizierung in der Antwort mitgesendete Cookie wird aus der Antwort extrahiert. Die Funktionsweise ist in Listing 4.5 zu finden. Dieses Listing knüpft direkt an Listing 4.4 an. In Zeile 6 wird mit Hilfe eines Linq-Statements der Cookie aus der Antwort extrahiert. Linq ist ein Feature von C#, das ermöglicht mittels Statements Datenfilterungen in iterierbaren Containern zu realisieren [7].

Um diesen Cookie nun in den Browser des/der Nutzer:in zu injizieren wurde ein spezielles Feature von ASP.NET benötigt. Microsoft bietet in seiner Bibliothek JSInterop in .NET die Möglichkeit Javascript-Funktionen im Browser des Clients aufzurufen. So können beispielsweise Elemente im DOM geändert werden um mögliche Elemente zu löschen, zu ändern oder hinzuzufügen. Allerdings können auch Cookies gesetzt werden. Die

Javascript-Funktion `CreateCookie` ermöglicht es in den Browser des Besuchers von Mint Web Cookies zu injizieren. Im Zusammenspiel mit JSInterop und der InvokeVoidAsnyc-Methode konnte der ausgelesene Sicherheitstoken so in den Browser injiziert werden. Dieser Schritt ist in Listing 4.5 in Zeile 19 zu finden.

```
1  finally
2  {
3    if (success)
4    {
5      var responselines = response.Headers.ToString().Split("\n");
6      var str = responselines.Where(str=>str.StartsWith("Set-Cookie
           ")).FirstOrDefault();
7      var cookie="";
8      try
9      {
10       cookie = str.Split(":")[1];
11     }
12     catch (NullReferenceException e)
13     {
14       _logger.LogError("Wrong Credentials for jwt.\n" + e);
15     }
16     var key = cookie.Split("=")[0];
17     var value = cookie.Split("=")[1].Split(";")[0];
18     value += "; secure=true; samesite=None";
19     await _js.InvokeVoidAsync("CreateCookie", key, value);
20     SetIframeLoaded();
21   }
22 }
```

Listing 4.5: Kibana.razor.cs::GetCookie(): Cookie-Extraktion

Der Cookie mit dem Inhalt des Kibana-Securitytokens ist nun im Browserstorage des/der Aufrufer:in der Kibana-Website gespeichert. Beim Laden des Iframes versucht Kibana aus den Cookies des Browsers diesen Sicherheitstoken auszulesen. Kibana erkennt den Cookie und authentifiziert den/die Nutzer:in, da es ihn für den/die vorher authentifizierte:n Nutzer:in hielt, obwohl das Request technisch gesehen nicht vom Browser sondern von Mint Web ausgegangen war.

Genau genommen wurde dadurch eine Browser-Schwachstelle beziehungsweise eine Designentscheidung von Kibana ausgenutzt. Beim Speichern von Cookies ist es bei unverschlüsselten Verbindungen ohne TLS möglich, auf Cookies von anderen Anbietern zuzugreifen. Dies sind die sogenannten Third-Party-Cookies. Sie dienen sonst zur Nachverfolgung von Nutzer:innen im Netz um zum Beispiel zugeschnittene Werbung anzuzeigen. Am 03.09.2019 entschied sich Mozilla third-party-cookies für https in ihrem Browser

Firefox automatisch zu blockieren. [13] Auch Google möchte mit Google Chrome bis Ende 2023 alle third-party-Cookies deaktivieren und Alternativen anbieten, wie beispielsweise FLoC bei der Nutzer:innen nicht mehr einzeln identifizierbar, sondern in einer Kohorte vermerkt werden. [6] Bei beiden Browsern sind allerdings in den Versionen 92.0 (Firefox) und 93.0.4577.82 (Chrome) keine Sicherheitsmaßnahmen vorgesehen um Cookies, die durch den unverschlüsselten Besuch mittels HTTP von Webseiten gesetzt wurden, vor Cross-Site-Tracking zu schützen.

Für dieses Szenario erweist sich der genannte Umstand als unproblematisch, solange Mint Analytics und die Mint Web auf einem gemeinsamen Server mit gleichem Hostname laufen. Websites können nur Cookies der gleichen Domain lesen, verändern oder löschen. Erst bei der Aufteilung der Dienste auf mehrere Server und dem Zugriff des Iframes per verschlüsselter https-Verbindung, wird ein auslesen und injizieren des Cookies unmöglich. Für diesen Fall wurde durch die Änderung der Authentifizierungskonfiguration eine Alternative entwickelt.

Kibana bietet neben der JWT Authentifizierung durch Cookies auch die Möglichkeit den JWT per URL-Option als Parameter zu übergeben. Iframes können diese url mitsamt Optionen aufrufen und die eingebettete Website anzeigen. Dadurch ist es möglich auch bei verschiedenen Servernamen und https die Authentifizierung durchzuführen. Für die Entscheidung welcher der beiden Ansätze, Cookie-Interception oder JWT-Url-Option, beim Aufrufen der Seite gewählt wurde, wurde ein Algorithmus implementiert, der in der Mint Web den aktuellen Hostnamen und die Kibana-Url vergleicht und die Url von Kibana auf das passende Schema, https, überprüft. Sollten beide Voraussetzungen erfüllt sein wird der JWT-Url-Option-Ansatz gewählt, ansonsten der Cookie-Interception-Ansatz.

Die grundlegende Kopplung der beiden Dienste ist somit abgeschlossen.

4.4.2 Konfiguration und Generierung der JWTs

Da der/die Nutzer:in zu diesem Zeitpunkt immer noch mit einem Blanko-JWT authentifiziert wird, der mit dem unsicheren Signaturalgorithmus HMACSHA512 signiert war muss eine automatische Generierung der JWTs unter Berücksichtigung des sichersten Verfahrens implementiert werden.

HMACSHA512 stellt in sofern ein Sicherheitsrisiko dar, als das eine Offenlegung des dafür verwendeten geheimen Schlüssels eine Kompromittierung der Authentifizierung bedeutet hätte. Das hätte auch durch eine Veröffentlichung der Konfigurationsdateien des security-plugins von Opendistro erfolgen können, in denen dieser geheime Schlüssel ebenfalls im Klartext geschrieben steht. Das security-plugin stellte allerdings auch die Verwendung anderer Signaturalgorithmen zur Verfügung. Eine Liste dieser Algorithmen sowie ihre Vor- und Nachteile sind in Kapitel 4.2 zu finden. Die Entscheidung des Alternativalgorithmus fiel auf RS512 bzw. RSASHA512. Bei diesem wird aus einem private und einem public key ein 512 bit langer Hashwert gebildet und als Signatur verwendet. Der Vorteil ist,

dass zur Generierung der Signatur der private und der public Key verwendet werden müssen, zur Validierung lediglich der public-key. Konkret bedeutet das, dass der Host auf dem Kibana läuft, nur den public key des für die Signatur verwendeten Schlüsselpaares benötigt. Dadurch ist sichergestellt, dass der Schlüssel nicht aus Versehen durch die Konfigurationsdatei veröffentlicht wird.

Für die Implementierung sind somit zwei Dinge nötig: Die Konfigurationsmöglichkeit um als Anwender den präferierten Signaturalgorithmus verwenden zu können und public und private key als Optionen verwenden zu können. Mint Lesion verfügt über eine Konfigurationsschnittstelle, die Mint-Options, die über das Administrationstool gesetzt werden können. So wurden verschiedene Optionen eingeführt. Diese wurden in der Klasse `OptionDefinitionsElasticSearchGroup` hinzugefügt. Sie sind in Listing 4.6 zu finden. Jede Option hat einen eindeutigen Identifier, der mit Punkten getrennt ist um mehrere Kategorien zu realisieren. Der Identifier des pubkeys ist in Zeile 2 des Listings zu finden. In Zeile 3 ist der Typ der gespeicherten Option beschrieben. Jede Option benötigt außerdem eine eindeutige Beschreibung welche für den Pubkey in Zeile 4 zu finden ist. Zum Schluss wird ein weiterer Identifier gesetzt um die Option in die richtige Kategorie einordnen zu können. Dies geschieht in Zeile 5.

```
1  options->IntroduceEncryptedStringOption(
2    "es.webapp.jwtSecret",
3    QStringLiteral( "Shared secret for single sign on in the webapp
       . Has to be encoded in base64." ),
4    OptionDefinitionsElasticSearchGroupIdentifier );
5
6  options->IntroduceStringOption(
7    "es.webapp.sso_pubkey_path",
8    QString(),
9    QStringLiteral( "filepath of the public key for the jwt
       generation of kibana sso" ),
10   OptionDefinitionsElasticSearchGroupIdentifier );
11
12
13 options->IntroduceStringOption(
14   "es.webapp.sso_privkey_path",
15   QString(),
16   QStringLiteral( "filepath of the private key for the jwt
       generation of kibana sso" ),
17   OptionDefinitionsElasticSearchGroupIdentifier );
```

Listing 4.6: OptionDefinitionsElasticSearchGroup : RS512-Optionen

Die beiden Optionen dienen der Pfadangabe der Schlüssel. Für die korrekte Auswahl des Signaturalgorithmus wurde ein Signaturbestimmungsalgorithmus implementiert. Dieser

ist in Listing 4.7 zu finden. Sind sowohl die Optionen `es.webapp.sso_pubkey_path` und `es.webapp.sso_privkey_path` gesetzt, wird automatisch RS512 als Signaturalgoritmus angenommen. Dies geschieht in Zeile 10. Sind nicht beide Optionen gesetzt, aber die Option `es.webapp.jwtSecret`, wird HS512 als Signaturalgorithmus angenommen. Dies geschieht in Zeile 14. Sind alle drei Optionen gesetzt wird durch die Reihenfolge der If-Statements automatisch RS512 als Signaturalgorithmus verwendet. Sind keine der drei Optionen gesetzt wird ein Fehler ausgegeben, dass eine Signatur des JWTs nicht möglich ist. Dies geschieht ab Zeile 18.

```
1  const auto options            = Options::GetInstance();
2  const auto hs512OptionKey     = "es.webapp.jwtSecret";
3  const auto& hs512OptionValue  = options->GetString(
       hs512OptionKey ).toStdString();
4  const auto rs512PublicKeyOptionKey = "es.webapp.sso_pubkey_path";

5  const auto& rs512PublicKeyOptionValue = options->GetString(
       rs512PublicKeyOptionKey ).toStdString();
6  const auto rs512PrivateKeyOptionKey = "es.webapp.
       sso_privkey_path";
7  const auto& rs512PrivateKeyOptionValue = options->GetString(
       rs512PrivateKeyOptionKey ).toStdString();
8
9  std::string algo;
10 if ( rs512PublicKeyOptionValue != "" &&
       rs512PrivateKeyOptionValue != "" )
11 {
12   algo = JWTWrapper::rs512Algo;
13 }
14 else if ( hs512OptionValue != "" )
15 {
16   algo = JWTWrapper::hs512Algo;
17 }
18 else
19 {
20   algo  = "";
21   *status = grpc::Status( grpc::StatusCode::INTERNAL, "could not
         autoselect jwt signature algorithm" );
22   return;
23 }
```

Listing 4.7: MSessionMainLoopHandler : Signaturalgorithmus

Zu diesem Zeitpunkt geschieht die Generierung und Signatur der JWTs noch auf Seite

der Webapp. Der implementierte Algorithmus ist für Mint Web zu umfangreich und soll ausgelagert werden. Die Anforderung, belastende Prozesse möglichst nicht auf einem eher langsamen Frontend, sondern auf einem performanten Backend auszuführen, ist eine allgemeine Entscheidung der Entwickler:innen. Darum war im Zuge eines vorangehenden Projekts eines anderen Mitarbeiters bereits die Lösung implementiert worden, einen MintSessionService für die Koppelung von Mint Web mit dem Mint Lesion Client zu verwenden. In Mint Web werden so für große Anfragen Nachrichten generiert und durch das Frameworks gRPC an den MintSessionService gesendet. Dieser in C++ geschriebene Service funktioniert wesentlich effizienter und kann so auch komplexe Anfragen zügig beantworten. Die Datenübertragung an diesen Service erfolgt über gRPC, das Protokoll, das bereits in Kapitel 4.1.2 vorgestellt wurde.

Für die Verwendung von gRPC müssen in Proto-Dateien Services definiert werden. Ein Service kann mehrere RPC-Aufrufe anbieten, die in der gleichen Proto-Datei definiert sind. Die Funktionsimplementierung der RPC-Aufrufe erfolgt allerdings in Handlerklassen. Dort werden die Funktionen nicht nur implementiert, sondern auch an den RPC-Aufruf gebunden.

Die Nachrichtendefinition erfolgt in der Datei `MintModels.proto`. Ein Auszug ist Listing 4.8 zu entnehmen. Als Nachrichtenformate wurden `MClaim`, `MJWTRequest` und `MJWT` eingeführt. MClaim besteht aus einem Schlüsselwertpaar aus strings und MJWTRequest besteht aus wiederholten MClaims, also wiederholten Schlüsselwertpaaren. gRPC-Nachrichten können aus anderen, geschachtelten gRPC-Nachrichtenformaten bestehen. Durch die Kennzeichnung `repeated` kann angegeben werden, dass Felder in der Nachricht wiederholt vorkommen. Die Nachricht MJWTRequest wird auf Seite von Mint Web für die Generierung einer Anfrage einer Signatur eines JWTs verwendet. Der Service erstellt sie wenn möglich und liefert sie in einer Antwort zurück. Für diese Antwort wird das Nachrichtenformat MJWT benötigt.

```
message MJWT
{
  string jwt=1;
}

message MJWTRequest
{
  repeated MClaim claims=1;
}

message MClaim
{
  string key=1;
  string value=2;
}
```

Listing 4.8: MintModels.proto

Die Deklarierung der RPC-Aufrufe erfolgt in der Datei `MSession.proto`. Ein Auszug ist Listing 4.9 zu finden. Neben anderen Methodensignaturen wird hier, in Zeile 10, auch der Aufruf `CreateJWT` beschrieben.

```
1  import "google/protobuf/empty.proto";
2  import "MintModels.proto";
3  service MSession
4  {
5    ...
6    rpc GetNonConformityList( MNonConformityRequest ) returns (
       MNonConformityResponse );
7
8    rpc Check(HealthCheckRequest) returns (HealthCheckResponse);
9
10   rpc CreateJWT(MJWTRequest) returns (MJWT);
11 }
```

Listing 4.9: MSession.proto

Die Definition der Nachrichtenformate ist abgeschlossen. Nun erfolgt die Implementierung und Bindung der gRPC-Methoden. Dafür sind drei Klassen essenziell: `MSessionServer`, `MSessionHandler` und `MSessionMainLoopHandler`. Die Klasse MSessionServer stellt die QT-Applikation dar die gestartet wird verwendet um den gesamten Service zu starten. Die Klasse hält außerdem eine Referenz auf den WorkerThread, der die EventLoop abarbeitet, die von gRPC Aufrufen mit Tasks befüllt werden. Des weiteren

bindet der die Klasse MSessionServer die Handler-Funktionen an die GRPC-Stubs. Wird ein gRPC-Aufruf ausgelöst, wird er zuerst vom MSessionHandler entgegengenommen. In dieser Klasse wird der Name der gewünschten Funktion allerdings nur in die EventLoop eingetragen. Im MSessionMainLoopHandler ist die konkrete Funktion implementiert.

Die RPC-Aufrufe des gRPC-Service MSession werden in der Klasse MSessionHandler entgegengenommen. Der Quellcode dafür ist in Listing 4.10 zu finden. Zu sehen ist in Zeile 8, dass der Aufruf der eigentlichen Methode weitergeleitet wird an die Methode InvokeMethod in der MSessionHandler-Klasse.

```
1  #include "MSessionHandler.h"
2  #include "Options.h"
3  ...
4  grpc::Status
5    MSessionHandler::CreateJWT( ::grpc::ServerContext* context,
         const MJWTRequest* request, MJWT* response )
6  {
7    grpc::Status status;
8    if ( this->InvokeMethod(
9          "CreateJWT", context, &status, Q_ARG( const MJWTRequest*,
               request ), Q_ARG( MJWT*, response ) )
10       == false )
11   {
12     MINT_ERROR( "Not Licensed" );
13     status = ::grpc::Status( grpc::StatusCode::PERMISSION_DENIED,
           "" );
14   }
15   return status;
16 }
```

Listing 4.10: MSessionHandler : CreateJWT

Der Quelltext der InvokeMethod-Methode ist in Listing 4.11 zu finden. Diese Methode überprüft, ob gewisse gRPC-Funktionsaufrufe mit der jeweils gekauften Lizenz von MintLesion überhaupt unterstützt werden. Dies geschieht in Zeile 7. Die Funktion `CheckForLicense` ist für den weiteren Ablauf uninteressant und eine weitere Erklärung dieser wird deshalb übersprungen. In Zeile 10 beziehungsweise 11 ist zu sehen, dass der eigentliche Methodenaufruf auf dem MSessionMainLoopHandler aufgerufen wird. Diese Klasse behandelt die asynchron Aufgerufenen Methoden sequenziell. Die Framework-Methode `QMetaObject::invokeMethod` kann Memberfunktionen auf anderen QObjects ausführen und gibt true zurück wenn der Aufruf erfolgreich war. [10]. In Zeile 9 ist außerdem zu erkennen, dass die Invokation in die Eventloop eingetragen wird.

```
bool MSessionHandler::InvokeMethod(
      const QString& methodName,
      ::grpc::ServerContext* context,
      ::grpc::Status* status,
      Args&&... arguments )
{
  if ( true || CheckForLicense( methodName ) )
  {
    const auto connectionType = m_DirectConnection ? Qt::
       DirectConnection : Qt::BlockingQueuedConnection;
    const auto invokeOk = QMetaObject::invokeMethod(
             MSessionMainLoopHandler::GetInstance(),
             methodName, connectionType,
             Q_ARG( ::grpc::ServerContext*, context ),
             std::forward<Args>( arguments )...,
             Q_ARG( ::grpc::Status*, status ) );

    MINT_ASSERT( invokeOk );

    if ( invokeOk == false )
      MINT_ERROR( "Invoke of MSessionMainLoopHanlder not
         successful for method: ", methodName );

    return invokeOk;
  }
  else
  {
    return false;
  }
}
```

Listing 4.11: MSessionHandler : CreateJWT

Der gRPC-Aufruf CreateJWT wird in der Protokolldefinitionsdatei hinzugefügt, in der Klasse MSessionMainloopHandler implementiert, in der Klasse MSessionServer als UnaryCall gebunden und in der Klasse MSessionHandler auf dem Hauptthread aufgerufen. Ein vollständiger Ablauf der gRPC-Aufrufsbehandlung ist in Abbildung 4.4 zu sehen.

Für die Implementierung im MSessionMainloopHandler wurde die Klasse JWTWrapper verwendet. Ein Auszug ist am Beispiel der Generierung eines JWTs mit HS512-Signatur in Listing 4.12 zu finden. Zuerst wird ein JWTWrapper-Objekt erstellt. Dieses erhält im Konstruktor den geheimen Schlüssel für die Signatur. Das zweite Argument *mint*

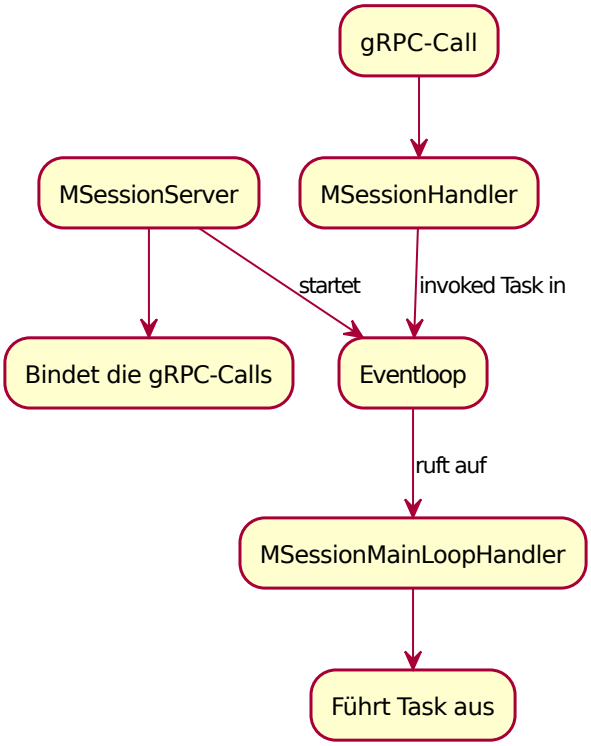

Abbildung 4.4: Ablauf GRPC-Calls

wird in diesem Fall als Aussteller des JWTs eingetragen. Anschließend wird mit der Methode `CreateToken` der eigentliche Token erstellt. Übergeben wird dabei eine Liste aller benötigten Claims `claim_textmap` und der gewünschte Signaturalgoritmus. Die Claims kommen dabei direkt aus der Nachricht MJWTRequest, das in Listing 4.8 gezeigt wurde. Response aus Zeile 3 ist dabei die Antwort im MJWT Format.

```
1 JWTWrapper jwtwrapper( jwtSecret, "mint" );
2 const auto jwtResponse = jwtwrapper.CreateToken( claim_map, algo
      );
3 response->set_jwt( jwtResponse.c_str() );
4 *status = grpc::Status( grpc::StatusCode::OK, ""
```

Listing 4.12: MSessionMainloopHandler::CreateJWT()

Für die weitere Erläuterung ist eine Betrachtung der JWTWrapper-Klasse nötig. Diese Klasse umhüllt die Library jwt-cpp des Autors Dominik Thalhammer welche unter MIT-Lizenz steht. [11] Die Bibliothek jwt-cpp bietet die Möglichkeit jwts mit claims zu erstellen und die Signatur mit verschiedenen Verfahren zu berechnen. Im Wrapper muss dafür vom MSessionMainloopHandler sowohl die Claim-Map, als auch der präferierte Singnaturalgorithmus übergeben werden. Die Implementierung der CreateJWT-Funktion ist in Listing 4.13 zu finden. Verwendet wird hier die Möglichkeit einer Build-Methode in Thalhammers Library. In Zeile 3 bis 6 wird der Header des JWT gesetzt. Alle JWTs

haben eine Gültigkeitsdauer von jeweils einer Stunde. Anschließend werden ab Zeile 8 bis 12 die claims an die payload des jwt angehängt. In Zeile 17 beziehungsweise 21 wird der JWT dann mit dem jeweiligen Signaturalgorithmus signiert. Wichtig ist dabei auch, dass das JWTSecret bei HS512 immer in Base64-Encodierter Form vorgelegen haben muss, da es automatisch dekodiert wird. Das ist ebenfalls in Zeile 17 zu sehen. Kibana benötigt dieses Secret ebenfalls in Base64-Encodierter Form in der Konfigurationsdatei des Securityplugins von Opendistro. Es muss bei der Einrichtung von Kibana nicht zusätzlich dekodiert werden.

```
1  std::string JWTWrapper::CreateToken( const QHash<QString,
       QString>& claims, const std::string& algo )
2  {
3    jwt::builder jwt_build = jwt::create()
4                         .set_type( "JWT" )
5                         .set_issued_at( std::chrono::system_clock::
                             now() )
6                         .set_expires_at( std::chrono::system_clock
                             ::now() + std::chrono::hours{ 1 } );
7    auto i = claims.constBegin();
8    while ( i != claims.constEnd() )
9    {
10     jwt_build.set_payload_claim( i.key().toStdString(), jwt::
         claim( i.value().toStdString() ) );
11     ++i;
12   }
13   std::string jwt_gen;
14   if ( algo == JWTWrapper::hs512Algo )
15   {
16     jwt_gen =
17       jwt_build.sign( jwt::algorithm::hs512{ jwt::base::decode<jwt
           ::alphabet::base64>( m_SessionSecret ) } );
18   }
19   else if ( algo == JWTWrapper::rs512Algo )
20   {
21     jwt_gen = jwt_build.sign( jwt::algorithm::rs512{ m_pubkey,
         m_privkey } );
22   }
23   return jwt_gen;
24 };
```

Listing 4.13: JWTWrapper::CreateToken()

Der signierte JWT liegt nun im base-64-url-Format vor und wird im MSessionMainloopHandler in die Antwort mit dem Nachrichtenformat MJWT geschrieben. Danach wird der gRPC-Call mit einer positiven Statusmeldung beendet. Bricht die Verbindung ab oder tritt ein anderer Fehler, wird auf der Client- und Serverseite jeweils eine Exception geworfen werden.

Die Implementierung auf Seite des Lesion-Clients ist abgeschlossen.

Nun muss noch der gRPC-Aufruf auf Mint Web-Seite realisiert werden. Die Klasse `UserService` ist hierfür die Schnittstelle für gRPC-Methodenaufrufe. Hier wurde die Funktion `CreateJWT` implementiert. Sie ist in Listing 4.14 zu finden. In Zeile 3 bis 7 und 9 bis 13 werden jeweils die beiden Claims *sub* und *roles* erstellt. Diese werden anschließend in Zeile 16 und 17 in ein MJWTRequest eingefügt. Dieses wird in Zeile 21 per gRPC-Aufruf signiert und als MJWT zurückgegeben.

```
public async Task<MJWT> CreateJWT(string role, string username)
{
  var sub = new MClaim
  {
    Key = "sub",
    Value = username
  };

  var roles = new MClaim
  {
    Key = "roles",
    Value=role
  };

  var mJWTRequest = new MJWTRequest();
  mJWTRequest.Claims.Add(roles);
  mJWTRequest.Claims.Add(sub);

  try
  {
    return await (await GetGrpcClient()).CreateJWTAsync(
       mJWTRequest);
  }
  catch (RpcException e)
  {
    m_Logger.LogError(e.ToString());
    throw new RpcException(e.Status);
  }
}
```

Listing 4.14: UserService::CreateJWT()

Somit war die automatische Generierung von JWTs unter Auswahl des Bestmöglichen Signaturalgorithmus unter Angabe von Optionen im Administrationstool implementiert.

4.4.3 Optionen und Crosshosting

Mint Analytics bietet die Möglichkeit die Dashboards in unterschiedlichen Rollen zu betrachten. Nutzer:innen mit der rolle`admin` besitzen mehr Rechte als Nutzer:innen mit der Rolle `readonly`. Sie können neue Dashboards erstellen, bestehende bearbeiten oder löschen oder neue Nutzer:innen hinzufügen. Um die Rolle eine:r Nutzer:in zu konfigurieren ist somit die Einführung einer weiteren MintOption `es.webapp.role` nötig. Die

Rolle kann nahtlos als Claim in das Nachrichtenformat MJWTRequest aufgenommen und validiert werden. Da die Generierung des Requests jedoch auf Mint Web-Seite stattfindet, muss diese ebenfalls an das Optionen-System von Mint Lesion angebunden werden. Die Funktionalität war bereits von einem Mitarbeiter implementiert worden. Sie wurde allerdings etwas rudimentär umgesetzt. Beim initialen Start von Mint Web wird durch die OptionsProvider-Komponente ein gRPC-Aufruf gestartet, der eine Liste der Werte für die Webapp relevanter Optionen zurückliefert. Diese waren jedoch fest im Sourcecode auf Mint Lesion-Seite beschrieben. Um diese Funktionalität etwas dynamischer zu gestalten wurde die Funktion `GetOptions` in der Klasse `MSessionMainLoopHandler` umgestaltet. Es wird nun bei einem Aufruf keine Liste an statisch definierten Optionen mehr zurückgeliefert, sondern nur die Werte, die zuvor in einem `MOptionRequest` angefragt wurden. Diese werden in dem Antwortformat `MOptionReqsponse` zurückgegeben. Die Funktion ist in Listing 4.15 aufgeführt. Mint Options sind Kategorisiert. Sie sind entweder String-Optionen, Boolean-Optionen, Integer-Optionen oder JSON-Optionen. Abfragen nach Werten müssen deshalb mit den jeweiligen Methoden `GetString`, `GetBool`, und so weiter umgesetzt werden. Diese Kategorisierung und Abfrage von jeder in der MOptionRequest genannten Option geschieht ab Zeile 10 bis 30. In Zeile 13 wird der Optionstyp der gewünschten Option abgefragt um den Wert in der korrekten Kategorie abfragen zu können. Hat eine Option keinen Typ kann davon ausgegangen werden, dass sie in keiner Kategorie existiert. In diesem Fall wird sie übersprungen und eine Warnmeldung in das Logfile geschrieben.

```
void MSessionMainLoopHandler::GetOptions(
  ::grpc::ServerContext*,
  MOptionRequest* request,
  MOptionResponse* response,
  ::grpc::Status* status )
{
  const auto options = Options::GetInstance();
  try
  {
    for ( const auto& option : request->options() )
    {
      auto responseOption = response->add_options();
      auto optionType = options->GetOptionType( option.c_str() );
      if ( optionType.isEmpty() )
      {
        MINT_WARN( "Could not find requested Option" );
        continue;
      }
      responseOption->set_type( optionType.toStdString() );
      responseOption->set_key( option );
      if ( optionType == OptionBase::boolType )
      {
        responseOption->set_boolvalue( options->GetBool( option.
            c_str() ) );
      }
      else if ( optionType == OptionBase::integerType )
      {
        responseOption->set_intvalue( options->GetInteger( option.
            c_str() ) );
      }
      ...
    }
    *status = grpc::Status( grpc::StatusCode::OK, "" );
  }
  catch ( const MInterfaceInputException& e )
  {
    *status = grpc::Status( grpc::StatusCode::INTERNAL, "" + e.
        GetStatusCode() + e.GetStatusText() );
  }
  catch ( const std::exception& e )
  {
```

```
39      *status = grpc::Status( grpc::StatusCode::INTERNAL, e.what() )
            ;
40    }
41  };
```

Listing 4.15: MSessionMainLoopHandler::GetOptions()

Die Gegenseite, in der die Abfragen generiert werden, befindet sich in der Webapp in der Klasse `UserService` in der Methode `GetRelevantOptions`. Diese ist in Listing 4.16 aufgeführt. In Zeile 4 ist zu sehen wie nun auf Mint Web-Seite die relevanten Optionen angegeben werden. Die Angabe erfolgt nun in Mint Web immer noch statisch im Quelltext, ist allerdings entkoppelt von der Methode in der Klasse MSessionMainLoopHandler. Die Methode dort kann nun auch für die Abfrage anderer Anwendungen verwendet werden, die möglicherweise andere relevante Optionen haben.

```
1  public async Task<MOptionResponse> GetRelevantOptions()
2  {
3    var optionrequest = new MOptionRequest();
4    optionrequest.Options.AddRange(new[] { "es.webapp.jwtSecret", "
         es.webapp.hostname", ... }) ;
5    try
6    {
7      return await (await GetGrpcClient()).GetOptionsAsync(
           optionrequest);
8    }
9    catch (RpcException e)
10   {
11     m_Logger.LogError(e.ToString());
12     throw new RpcException(e.Status);
13   }
14 }
```

Listing 4.16: UserService::GetRelevantOptions()

Die OptionsResponse aus dem UserService wird anschließend verwendet um in der Komponente `OptionsProvider` die Optionen zu initialisieren. Die Komponente stellt eine Spiegelung der Options-Klasse auf MintLesion-Seite dar. Die darin enthaltenen Optionen können, wie schon auf MintLesion-Seite, einfach abgefragt werden, befinden sich allerdings gepuffert auf Mint Web-Seite. Sollte sich eine Option ändern muss die Website manuell neu geladen werden.

Durch diese Auslagerung konnten auch weitere Optionen definiert werden. Es zeichnete sich ab, dass mehrere Kunden ein Setup planten, bei dem die beiden Dienste Mint Web und MintAnalytics auf unterschiedlichen Servern laufen würden. Um dieses Crosshosting umzusetzen wurde die Option `es.webapp.hostname` eingeführt. Diese musste die

komplette URL inklusive Schema umfassen. War die Website beispielsweise nur per https erreichbar und nicht auf dem gleichen Server gehosted, war eine Cookie-Injektion wie sie in Kapitel 4.4.1 beschrieben wurde nicht möglich und musste als URL-Option mitgegeben werden. Dies ist im Listing 4.17 dargestellt. In Zeile 1 werden die beiden Bedingungen https und crosshosting geprüft. Ist der Fall erfüllt wird in Zeile 3 über den UserService ein JWT auf herkömmlichem Weg erstellt und in Zeile 4 als URL-Option hinzugefügt. Ansonsten kommt wie vorher bereits die Cookie Interception zum Einsatz deren Funktionsweise Listing 4.4 und Listing 4.5 zu entnehmen ist.

```
1  if (m_URL.StartsWith("https://") && !m_URL.Split("//")[1].
       StartsWith(hostname))
2  {
3    var jwt = (await _userService.CreateJWT(m_Role, m_username)).
       Jwt;
4    m_URL = m_URL + "?token=" + jwt;
5    SetIframeLoaded();
6  }
7  else
8  {
9    await GetCookie();
10 }
```

Listing 4.17: Kibana.razor.cs::OnInitializedAsync()

Die Umsetzung ist damit abgeschlossen. Durch die Rücksichtnahme auf anfallende Eventualitäten funktionierte die Authentifizierung am Ende zuverlässig. Nicht trivial ist dabei der Durchlauf des Authentifizierungsprozesses, der im folgenden Kapitel noch einmal ausführlich behandelt wird.

4.5 Zusammenfassung

In diesem Kapitel wird noch einmal der Grundablauf des Authentifizierungsprozesses erläutert. Der Ablauf ist dabei stark abhängig von der jeweiligen Konfiguration von Mint Analytics. Diese Konfigurationen werden im zweiten Unterkapitel beschrieben.

4.5.1 Funktionsweise

Um den in Kapitel 4.4 beschriebenen Ablauf verständlich darzustellen wird in diesem Kapitel der grobe Ablauf der endgültigen Implementierung erklärt.

Der Algorithmus beginn mit dem Besuch der Website. Loggt der/die Bernutzer:in sich in die WebApp ein und besucht anschließend die Unterseite */kibana*, dann beginnt in der `OnInitializedAsync()` Methode der konkrete Ablauf. Zuerst werden für die

Einsichtrechte, sowie für das Zusammenbauen des Request, die Optionen aus Mint Lesion benötigt. Dazu fragt in Mint Web die Komponente `MOptionsProvider` bei Mint Lesion bzw. dem MintSessionServer per gRPC diese Optionen an. Ein Ablauf ist in Abbildung 4.5 zu sehen.

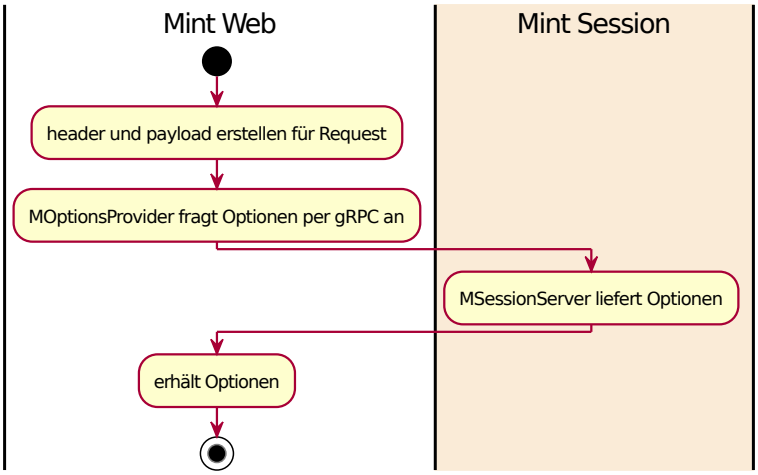

Abbildung 4.5: Optionen holen

Sobald die Optionen geliefert wurden, wird auf Seite von Mint Web eine Entscheidung getroffen welcher konkrete Authentifizierungsmechanismus ausgeführt werden soll. Bei Crosshosting mit https, also wenn die Webanwendung und Mint Analytics nicht unter der gleichen url erreichbar sind und die Seite nur per https erreichbar ist, wird als jwt Authentifizierungsalgorithmus die Methode über die URL-Option verwendet. Dies wird durchgeführt weil in diesem speziellen Fall eine Cookie-Inception nicht möglich ist, da Cookies entweder nur von Websiten gelesen werden können wenn Sie von der gleichen URL sind, oder wenn Sie speziell freigegeben wurden. Ist diese Konfiguration nicht der Fall wird das security_token auf herkömmlichem Wege per Cookie-Interception injiziert. Für beide Wege ist allerdings die Generierung eines JWTs erforderlich. Sie erfolgt auf Seite von Mint Lesion bzw. des Mint SessionServers. Dieser Ablauf wird in Abbildung 4.6 beschrieben. Mint Lesion, beziehungsweise der MintSessionServer, erhält nun per gRPC die Anweisung den JWT zu erstellen. Dieser Ablauf wird in der Abbildung 4.7 beschrieben. Zuerst wird überprüft welche Optionen gesetzt sind. Ist nur die Option `es.webapp.jwtSecret` gesetzt ist davon auszugehen, dass Mint Analytics auf die Annahme von JWTs mit HS512 Signatur eingestellt ist. Dann wird der JWT mit dem HS512-Mechanismus signiert und zurückgesendet.

Sind die beiden Optionen `es.webapp.privkey_path` und `es.webapp.pubkey_path` gesetzt ist davon auszugehen, dass Mint Analytics auf die Annahme von JWTs mit RS Signatur eingestellt ist. Es wird anschließend versucht die beiden Dateien einzulesen und die Schlüssel daraus als Strings zu instantiieren. Funktioniert das nicht, so wird per gRPC eine Exception geworfen, die den Fehler angibt. Funktioniert es, wird der JWT

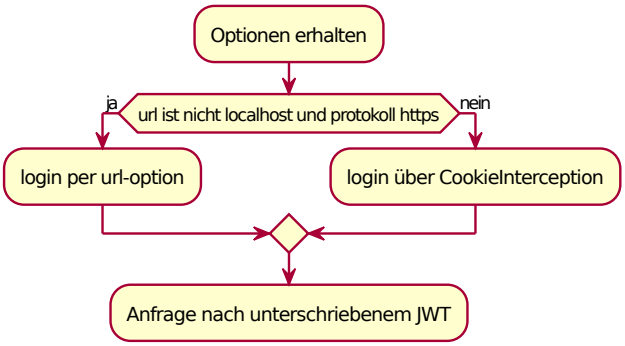

Abbildung 4.6: Ablauf Login Mechanismus

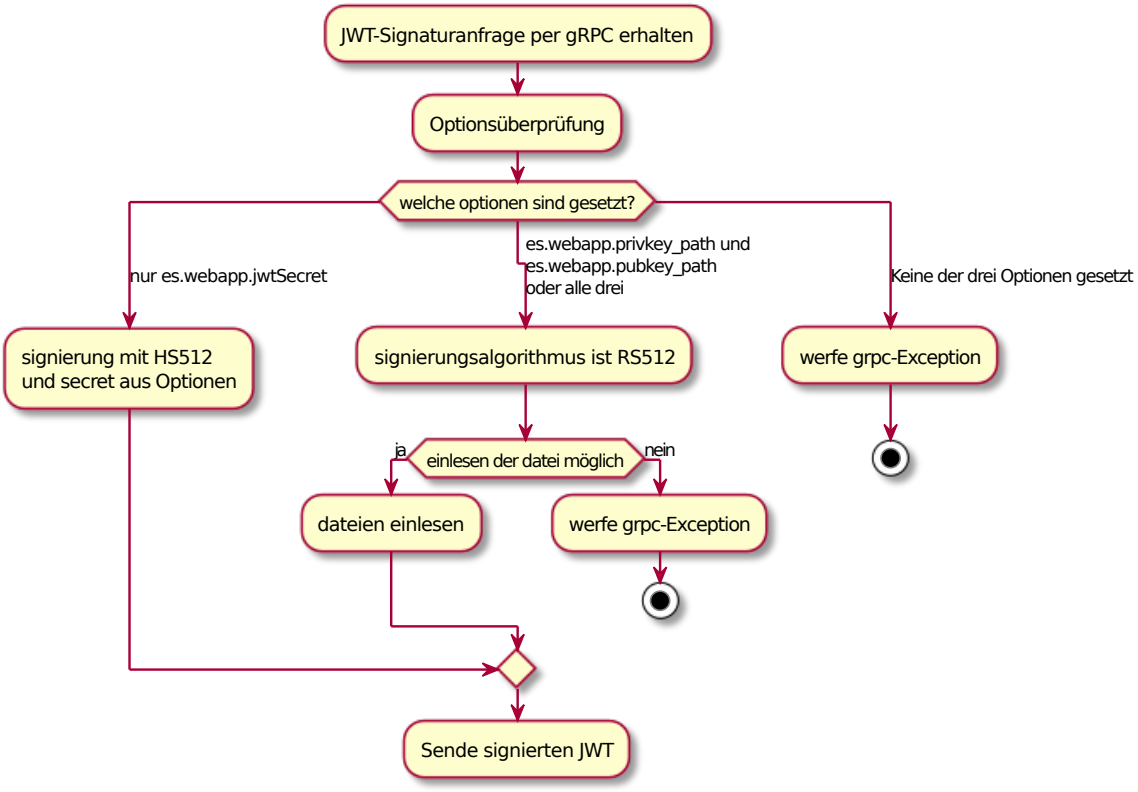

Abbildung 4.7: JWT generierung

mit RS512 signiert und zurückgesendet. Dieser Pfad wird auch eingeschlagen wenn alle drei Optionen gesetzt sind.

Ist keine der Optionen gesetzt wird ebenfalls eine gRPC Exception geworfen.

Der finale Schritt erfolgt dann zwischen Webanwendung und Mint Analytics. Der Ablauf ist in Abbildung 4.8 illustriert. Nachdem Mint Web den signierten JWT erhalten hat, baut es ihn in das Request ein, oder hängt den JWT als Option an die URL des IFrames an. Es kommt dabei darauf an, ob der Fall des Crosshostings mit https, der in Abbildung 4.6 beschrieben wurde, erfüllt ist. Nur wenn er erfüllt ist wird der JWT als URL-Option angehängt. Ist er nicht erfüllt wird das zusammengebaute Request mit dem JWT als Authentifizierungsparameter im Header abgeschickt. Mint Analytics bzw. das security-plugin von OpenDistro überprüft nun in seiner eigenen Konfiguration, ob der jwt mit dem hinterlegten Verfahren und Schlüssel oder Schlüsseln signiert wurde. Ist dies der Fall wird der Nutzer oder die Nutzerin authentifiziert. Danach schickt Mint Analytics eine Response in deren Payload sich der security_token befindet, der als Cookie gesetzt werden muss. Dieser Cookie wird von der WebApp anschließend über JSInterop in den Browser der Nutzerin oder des Nutzers injiziert. Mint Analytics erkennt diesen Cookie aus dem iframe heraus wieder und lässt den Nutzer oder Nutzerin gewähren. Dies ist der

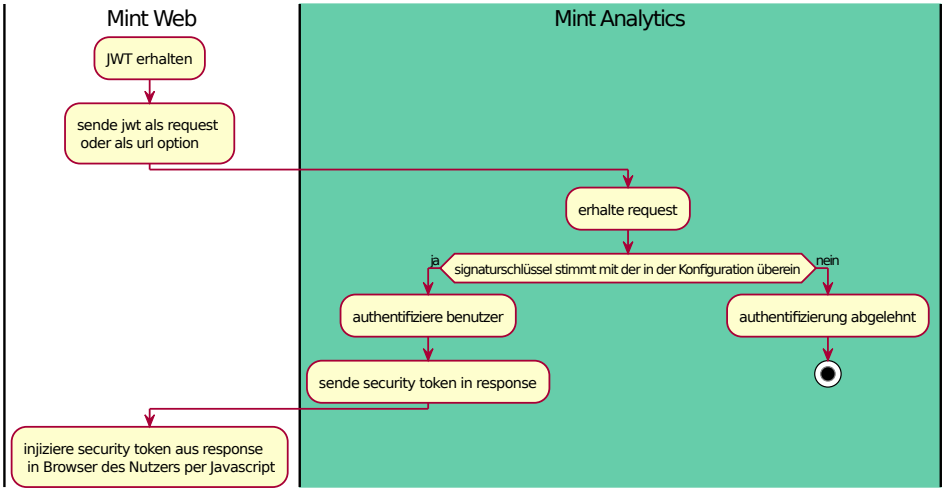

Abbildung 4.8: Finale Authentifizierung

Ablauf der Authentifizierung. Der Vorgang benötigt trotz seines Umfangs und mehrerer HTTP Requests sowie gRPC-Anfragen weniger als 500ms. Diese Zeiten schränken das Nutzererlebnis nicht ein, gerade weil der Prozess für die Benutzung von Mint Analytics einmalig beim ersten Besuch stattfindet.

Als Flaschenhals ist hier das Einlesen der Dateien bei einer RS512-Konfiguration denkbar. Diese kann allerdings bei langsamen Speichermedien auch durch eine HS512-Konfiguration ersetzt werden, in der das Secret sich zur Laufzeit auf dem Stack befindet. Die Vor- und Nachteile der einzelnen Konfigurationen werden im nächsten Kapitel elaboriert.

4.5.2 Konfigurationen

Die verschiedenen Kombinationen aus Signaturalgorithmen, Crosshosting, Url-Options-Authentifizierung und https ließen viele Konfigurationsvarianten zu. Um den Installationsaufwand und die Komplexität zu verringern wurden drei essentielle Konfigurationstypen isoliert und die entsprechenden Konfigurationsfiles für Mint Analytics generiert. In den Konfigurationen müssen so nur noch das Secret beziehungsweise der public-key für die JWT-Validierung eingetragen werden.

Konfiguration A

In dieser Konfiguration ist HTTPS ausgeschaltet, eine URL-Authentifizierung findet nicht statt und der HS512 Algorithmus wird als Signaturalgorithmus für die JWT verwendet. Crosshosting ist ohne HTTPS verfügbar. Diese Varinte ist am Besten geeignet, falls beide Dienste in einem geschützten Infrastrukturbereich und nicht beispielsweise in deiner DMZ verfügbar sind. Durch den HS512 Algorithmus muss das Secret sowohl in der Option `es.webapp.jwtSecret` im Admintool, als auch in der Konfigurationsdatei des securityplugings von elasticsearch angegeben werden. Eine Veröffentlichung dieses Konfigurationsfiles bringt also eine Veröffentlichung des geheimen Schlüssels mit sich. Ein Angreifer könnte somit selbst JWTs generieren und sich im Browser hinzufügen. Außerdem könnte er/sie sich selbst nicht nur als Nutzer:in der Gruppe `readonly`, sondern auch als Nutzer:in der Rolle `admin` einloggen. Im Gegenzug ist die Installation dieser Variante trivial. Es wird lediglich ein 512bit langer Code auf beiden Seiten benötigt.

Konfiguration B

In dieser Konfiguration ist HTTPS ausgeschaltet, eine URL-Authentifizierung findet nicht statt und der RS512 Algorithmus wird als Signaturalgorithmus für die JWT verwendet. In dieser Konfiguration wird das Sicherheitsproblem von Konfiguration A umgangen, indem der in Kapitel 4.2 beschriebene RS512-Algorithmus zum Einsatz kommt. Dieser signiert mit öffentlich-privaten-RSA-Schlüseelpaaren. Die Pfade dieser beiden Schlüsseldateien müssen im Administrationstool in den Optionen `es.webapp.sso_pubkey_path` und `es.webapp.sso_privkey_path` eingegeben werden. In der Konfigurationsdatei des Securityplugins von Elasticsearch muss so lediglich der public-key angegeben werden, da nur der zur Validierung des JWTs benötigt wird. Eine Veröffentlichung der Konfiguration ist somit nicht mehr so verehrend wie in Konfiguration A. Mit dem öffentlichen Signaturschlüssel lassen sich keine JWTs erzeugen. Somit hat ein Angreifer auch nicht die Möglichkeit sich höhere Rechte zu gewähren. Crosshosting ist auch weiterhin möglich. Allerdings, wie in Konfiguration A, ohne HTTPS. Die Installation dieser Variante ist ein weniger komplizierter, da OpenSSL für die Generierung der RSA-Schlüssel installiert sein muss. Diese müssen außerdem händisch per Kommandozeile über das openssl-tool generiert werden.

Konfiguration C

In dieser Konfiguration ist HTTPS eingeschaltet, eine URL-Authentifizierung findet statt

und der RS512 Algorithmus wird als Signaturalgorithmus für die JWT verwendet. Diese Konfiguration ist die Sicherste, da sie sowohl RS512 als Signaturalgorithmus verwendet wird, der wie Konfiguration B eine unbeabsichtigte Veröffentlichung des Secrets unerheblich werden lässt, als auch HTTPS für die Datenübertragung verwendet. Dieser Fall ist speziell für Crosshosting mit HTTPs ausgelegt. Nachteilig ist lediglich, dass die normale Login-Möglichkeit per Browser in Kibana mit Benutzername und Passwort nicht mehr nutzbar ist. Elasticsearch unterstützt zwar eine priorisierte Liste an Login-Möglichkeiten, Kibana allerdings nicht. Schaltet man die Standard-Authentifizierung dort von normaler Authentifizierung um auf JWT-URL-Options-Authentifizierung, wird die normale Authentifizierung deaktiviert.

Kapitel 5

Fazit und Ausblick

Die Anforderungen der sicheren Implementierung des Authentifizierungsmechanismus wurden erfüllt. Eine Laufzeitoptimierung konnte durch eine Auslagerung der arbeitsintensiven Aufgaben, wie der Generierung des JWTs per gRPC, umgesetzt werden. Die SSO ist mit drei unterschiedlichen Konfigurationen betriebsfähig. Alle drei Konfigurationen haben jeweils einen anderen Einsatzbereich. Bei mehreren Angaben wird automatisch die sicherste Konfiguration ausgewählt. Die Konfiguration wurde in bereits bestehende Konfigurationswerkzeuge eingegliedert. Um die Palette der Konfigurationen auch für die Installation der Webanwendung Mint Analytics möglichst unkompliziert zu gestalten wurden mehrere Vorlagen erstellt, die bei Bedarf nur noch geringfügig angepasst werden müssen.

Mögliche Erweiterungen wären zum Beispiel eine engere Integration der TLS-Zertifikate. Aktuell müssen diese im Voraus noch mit dem openssl-commandline-tool händisch erstellt werden. Ein Tool für die automatische Generierung von Schlüsselpaaren und Zertifikaten ist intern bereits in Arbeit. Dies könnte mit wenig Aufwand in das Administrationstool integriert werden um so automatisch korrekte Schlüsselpaare zu erstellen.

Eine weitere Erweiterung besteht in der Aktivitätsüberprüfung der Nutzerin oder des Nutzers bei der Benutzung von Kibana. Die Gültigkeitszeitspanne von einer Stunde für JWTs ist lang. Ein/eine Angreifer:in könnten, sofern Sie den JWT erlangen, Anfragen an das System stellen und Rechte ändern oder Benutzer hinzufügen oder löschen. Die Zeitdauer der JWTs zu limitieren gestaltete sich als schwierig, da Kibana den/die Nutzer:in nach dem Ablaufzeitpunkt des JWTs automatisch ausloggt. Danach würden Anwendender:innen allerdings nicht wieder auf der Startseite von Kibana weitergeleitet werden und wären dadurch wieder eingeloggt, sondern müssten sich händisch mit ihr/ihm wahrscheinlich unbekannten Login-Daten einloggen. Durch eine Aktivitätserkennung könnte bei Inaktivität kurz vor Ablauf der Gültigkeit des JWTs ein automatischer Reload ausgelöst werden um Nutzer:innen automatisch einzuloggen.

Das Security-Plugin von OpenDistro bietet neben JWTs auch noch einige weitere Authenti-

fizierungsmechanismen. So wäre es beispielsweise möglich sich per OpenSSL-Zertifikaten zu Authentifizieren. [9] Intern wird aktuell die Anbindung an SMART on FIHR realisiert. Diese verwendet für die Authentifizierung den Standard OAuth2. Elasticsearch bietet für diese Schnittstelle ebenfalls über OpenID Connect eine Single-Sign-On-Lösung per OAtuh2 an. Die Authentifizierung könnte also auch durch eine Umkonfigurierung von Mint Analytics an die bestehende Infrastruktur angebunden werden. Allerdings wären auch andere Lösungen wie eine Authentifizierung über SAML oder Kerberos möglich.

Literaturverzeichnis

[1] Auftragsforschungsinstitute. https://de.wikipedia.org/wiki/Auftragsforschungsinstitut. [letzter Zugriff: 18. Oktober. 2021].

[2] Amazon. Introducing opensearch. https://aws.amazon.com/blogs/opensource/introducing-opensearch/. [letzter Zugriff: 30. August. 2021].

[3] Elastic. Licensing change. https://www.elastic.co/de/blog/licensing-change. [letzter Zugriff: 29. August. 2021].

[4] Elastic. Why licensing change. https://www.elastic.co/de/blog/why-license-change-AWS. [letzter Zugriff: 30. August. 2021].

[5] Universitätsmedizin Göttingen. Racoon. https://www.umg.eu/forschung/corona-forschung/num/racoon/. [letzter Zugriff: 05. August. 2021].

[6] Niklas Lewanczik. Third party cookies bleiben länger. https://onlinemarketing.de/seo/third-party-cookies-bleiben-laenger-google-verschiebt-deadline. [letzter Zugriff: 23. September. 2021].

[7] Microsoft. linq. https://docs.microsoft.com/en-us/dotnet/csharp/linq/. [letzter Zugriff: 16. Oktober. 2021].

[8] Opendistro. Backend configuration. https://opendistro.github.io/for-elasticsearch-docs/docs/security/configuration/configuration/#json-web-token. [letzter Zugriff: 29. August. 2021].

[9] OpenDistro. Security. https://opendistro.github.io/for-elasticsearch-docs/docs/security/. [letzter Zugriff: 19. Oktober. 2021].

[10] QT. Qmetaobject::invokemethod. https://doc.qt.io/qt-5/qmetaobject.html#invokeMethod. [letzter Zugriff: 16. Oktober. 2021].

[11] Dominik Thalhammer. jwt-cpp. https://github.com/Thalhammer/jwt-cpp. [letzter Zugriff: 13. Oktober. 2021].

[12] Wikipedia. Brainlab - wikipedia. https://de.wikipedia.org/wiki/Brainlab. [letzter Zugriff: 05. August. 2021].

[13] Marissa Wood. Firefox third party. https://blog.mozilla.org/en/products/firefox/todays-firefox-blocks-third-party-tracking-cookies-and-cryptomining-by-default/. [letzter Zugriff: 23. September. 2021].