Bachelorarbeit

Angewandte Informatik

Authentifizierung in einer Gastronomie-App mit wechselnden Geräten und einmaliger Kopplung der Kasse

Name:	Simon Galle
Matrikelnummer:	26201
Erstbetreuer:	Prof. Dr. Sven Karol
Zweitbetreuer:	Steven Schwarznau
Unternehmen:	ALVARA Digital Solutions
Abgabetermin:	07.05.2025

Hochschule Merseburg
Prof. Dr. Sven Karol

Aufgabenstellung für die Bachelorarbeit

von

Simon Galle

Thema: Authentifizierung in einer Gastronomie-App mit wechselnden Geräten und einmaliger Kopplung der Kasse

Erstbetreuer: Prof. Dr. Sven Karol

Zweitbetreuer: Steven Schwarznau

Aufgabenstellung

Die Gastronomie-App der ALVARA GmbH ermöglicht es Bedienkräften, Bestellungen über mobile Geräte zu verwalten und mit der Kasse zu synchronisieren. Obwohl die grundlegende Funktionalität der App bereits implementiert ist, fehlt derzeit noch ein ausgereifter Authentifizierungsmechanismus. Im Rahmen dieser Bachelorarbeit die Konzeption und Implementierung eines Authentifizierungsmechanismus für die Gastronomie-App umgesetzt werden. Dieser soll den spezifischen Anforderungen eines dynamischen Gerätewechsels und der einmaligen Kopplung mit einer bestehenden Kasse gerecht werden. Nach der initialen Kopplung mit der Kasse soll die Authentifizierung der Bedienkräfte erfolgen. Dieser Prozess gewährleistet, dass jede Anmeldung eines Operators mit der fest verbundenen Kasse synchronisiert wird, um Konsistenz und Sicherheit in den Transaktionsabläufen zu gewährleisten.

Ein Schwerpunkt der Arbeit liegt auf dem Entwurf und der Implementierung der Schnittstelle zwischen der Synchronisationssoftware der Kasse und dem Backend der App. Die Herausforderung besteht darin, die Kommunikation zwischen den Komponenten sicher und effizient zu gestalten, sodass der Authentifizierungsprozess nahtlos und zuverlässig in den operativen Ablauf eingebunden wird. Zusätzlich sollen verschiedene Authentifizierungsmethoden hinsichtlich ihrer Leistungsfähigkeit und technischen Umsetzbarkeit analysiert und bewertet werden, um eine sichere und schnelle Anmeldung der Bedienkräfte zu gewährleisten. Unter anderem sollen die Authentifizierung per Fingerabdruck und NFC-Chip untersucht werden.

Schwerpunkte

1. Analyse und Diskussion verschiedener Authentifizierungsverfahren
2. Auswahl von Authentifizierungsverfahren für die Umsetzung in der Gastronomie-App
3. Entwurf der Authentifizierungsschnittelle und einer Architektur zur Kopplung mit dem Kassensystem
4. Prototypische Umsetzung in der App
5. Test und Bewertung der entwickelten Lösung

Merseburg, den 18.12.2024

Prof. Dr. Henning Bostelmann
Vorsitzender des Prüfungsausschusses

Prof. Dr. Sven Karol
Themenstellender Hochschullehrer

Abstract

In der vorliegenden Bachelorarbeit wird ein Authentifizierungsmechanismus für eine Gastronomie-App konzipiert und prototypisch umgesetzt. Die App ermöglicht es Servicekräften, Bestellungen über mobile Endgeräte aufzunehmen und mit der Kassensoftware zu synchronisieren. Eine zentrale Herausforderung besteht dabei in der Absicherung der Anmeldung bei wechselnden Geräten sowie der initialen Kopplung der App mit einer bestimmten Kasse. Zur Lösung dieser Anforderungen wurde zunächst eine Analyse gängiger Authentifizierungsverfahren durchgeführt. Auf Basis der Ergebnisse fiel die Entscheidung zugunsten eines sessionbasierten Verfahrens, das auf die spezifischen Gegebenheiten - begrenzte Geräteanzahl, lokales Netzwerk und klar definierte Nutzungsszenarien - zugeschnitten ist. Ein weiteres Kernelement der Arbeit ist die einmalige Kopplung der App mit der Kasse über einen QR-Code, der Informationen wie IP-Adresse des Kassensystems und einen temporären Lizenz-Token enthält. Der entwickelte Prototyp umfasst die Umsetzung des Loginprozesses in der App, die Verwaltung der Sitzungen auf Serverseite sowie die Prüfung der Sitzungsgültigkeit bei jeder Anfrage. Die Lösung wurde unter praktischen Gesichtspunkten integriert, validiert und ermöglicht eine sichere, effiziente und benutzerfreundliche Authentifizierung im täglichen Einsatz der App.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingsverzeichnis

Abkürzungsverzeichnis

API	**A**pplication **P**rogramming **I**nterface
App	**App**lication
DB	**D**ata**b**ase
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol
HTTPS	**H**yper**t**ext **T**ransfer **P**rotocol **S**ecure
ID	**Id**entity
IP	**I**nternet **P**rotocol
JSON	**J**ava**S**cript **O**bject **N**otation
JWT	**J**SON **W**eb **T**oken
MFA	**M**ulti-**F**actor **A**uthentication
NFC	**N**ear **F**ield **C**ommunication
QR-Code	**Q**uick **R**esponse **Code**
REST	**Re**presentational **S**tate **T**ransfer
UI	**U**ser **I**nterface
URL	**U**niform **R**esource **L**ocator
UUID	**U**niversally **U**nique **Id**entifier
WLAN	**W**ireless **L**ocal **A**rea **N**etwork
XSS	**Cross**-**S**ite **S**cripting

1 Einleitung

Die zunehmende Digitalisierung in der Gastronomie hat in den letzten Jahren zu einem Wandel der Arbeitsprozesse geführt. Mobile Endgeräte ersetzen klassische Bestellblöcke und ermöglichen eine direkte, digitale Übertragung von Bestellungen an das Kassensystem. Diese Entwicklung verspricht eine höhere Effizienz, geringere Fehlerquoten und eine optimierte Ablauforganisation. Vor allem in schnelllebigen Arbeitsumfeld wie der Gastronomie ist es entscheidend, dass Bedienkräfte sich schnell, sicher und zuverlässig an den verwendeten Geräten anmelden können – insbesondere, wenn diese Geräte regelmäßig zwischen Mitarbeitern wechseln oder gemeinsam genutzt werden.

In diesem Kontext gewinnt die Absicherung des Zugriffs auf mobile Applikationen zunehmend an Bedeutung. Der Schutz sensibler Transaktionsdaten, die Einhaltung von Rollen- und Berechtigungsmodellen sowie die Zuordnung von Bestellungen zu klar identifizierbaren Bedienern sind dabei zentrale Anforderungen. Authentifizierungsverfahren stellen somit eine unverzichtbare Komponente moderner Gastronomie-Software dar.

Auch aus persönlichem Interesse an sicherheitsrelevanten Softwarekomponenten sowie aus der praktischen Erfahrung mit App-Entwicklung entstand das Ziel, im Rahmen dieser Bachelorarbeit einen vollständigen Authentifizierungsmechanismus für einen bestehenden Gastronomie-App-Prototypen zu entwerfen und umzusetzen. Das spezifische Thema dieser Arbeit lautet: „Authentifizierung in einer Gastronomie-App mit wechselnden Geräten und einmaliger Kopplung der Kasse“.

Im Zentrum der Arbeit steht die technische und konzeptionelle Umsetzung eines Authentifizierungssystems, das den Herausforderungen eines dynamischen Gerätewechsels sowie der initialen Kopplung der App mit einer bestimmten Kasse gerecht wird. Die Forschungsfrage lautet daher: *Wie kann ein sicherer und praxistauglicher Authentifizierungsmechanismus für eine Gastronomie-App gestaltet werden, der sowohl eine initiale Kopplung mit der Kasse als auch wechselnde Endgeräte unterstützt?*

Der aktuelle Forschungsstand zu Authentifizierungsverfahren unterscheidet im wesentlichen zwischen zustandbehafteten (z.B sessionbasierten) und zustandslosen (z.B. tokenbasierten) Ansätzen [1]. Relevante theoretische Grundlagen wurden unter anderem von Fielding [2] et al. im Kontext von REST-Architekturen, sowie in zahlreichen sicherheitstechnischen Studien zu JSON Web Tokens und deren Sitzungsspeicherung [3]

untersucht. Ein besonderer Fokus liegt auf Aspekten wie Session Management, rollenbasierter Zugriffskontrolle und der Absicherung gegen typische Angriffsvektoren, um Angriffe wie die Übernahme aktiver Sitzungen (Session Hijacking) oder den Diebstahl von Zugangstoken (Token Theft) zu verhindern.

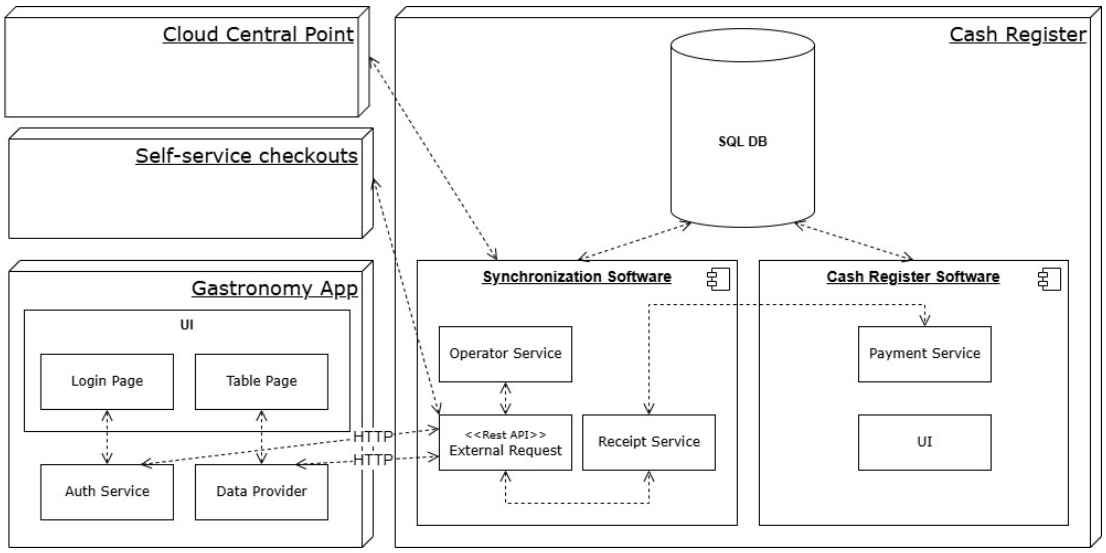
Zur Beantwortung der Forschungsfrage wurde ein methodisches Vorgehen gewählt, das sowohl theoretische Analyse als auch praktische Umsetzung umfasst. Die Arbeit beginnt mit einer Darstellung der vorhandenen App-Architektur und einer Untersuchung klassischer Authentifizierungsmechanismen. Anschließend erfolgt eine Gegenüberstellung relevanter Verfahren und die Auswahl eines geeigneten Konzepts. Darauf aufbauend wird die Authentifizierungsarchitektur inklusive Kopplungsmechanismus konzipiert, implementiert und getestet. Abschließend erfolgt eine Bewertung der Lösung hinsichtlich Funktionalität Sicherheit und Benutzerfreundlichkeit.

2 Die Gastronomie-App

Im Vorbereitung auf diese Arbeit wurde in den letzten sechs Monaten ein innovativer Prototyp der Gastronomie-App entwickelt. Dieser soll als zentrales Element für die Verwaltung von Bestellungen in Verbindung mit einem Kassensystem dienen.
Der Prototyp wurde mithilfe des plattformübergreifenden Open-Source-Frameworks Flutter entwickelt, das von Google bereitgestellt wird [4]. Flutter ermöglicht es, mit einer einzigen Codebasis Applikationen für Android und iOS zu erstellen, wobei die Entwicklung in der Programmiersprache Dart erfolgt. Die Entscheidung für Flutter basiert auf der hohen Entwicklungsproduktivität, der flexiblen Benutzeroberflächengestaltung und der nahezu nativen Performance, die durch die eigene Rendering-Engine erreicht wird [4].

Im Folgenden werden zunächst die grundlegenden Funktionalitäten und die Architektur der App beschrieben, um den Kontext und die Rahmenbedingungen für die Implementierung des Authentifizierungsmechanismus nachvollziehbar zu machen.
Zunächst wird die App-Umgebung betrachtet. Sie setzt sich aus der Gastronomie-App und der Kasse (engl. Cashregister) zusammen, wie in Abbildung 2.1 dargestellt.

Abbildung 2.1: Architekturdiagramm der App: Übersicht der Systemkomponenten und ihrer Interaktionen



Die Kasse gliedert sich in drei Komponenten: die eigentliche Kassensoftware (engl. Cashregister Software), die Synchronisationssoftware sowie die Kassendatenbank. Die Kassensoftware übernimmt dabei die Aufgaben einer herkömmlichen Kasse, beispielsweise das Auswählen von Artikeln oder das Auslösen von Bezahlvorgängen. Zusätzlich ist in der Kassensoftware auch das Frontend integriert, das die Benutzereingabe ermöglicht. Diese Funktion ist jedoch für die Gastronomie-App nicht relevant, da diese die Eingabe an einer stationären Kasse ablösen und so für ein dynamischeres Arbeiten sorgen soll.

Die Synchronisationssoftware ermöglicht über Representational State Transfer (REST) Schnittstellen einen reibungslosen Informationsaustausch zwischen externer Software, der Kasse und der Kassendatenbank. Zu den externen Elementen zählen beispielsweise die Gastronomie-App, der Cloud Central Point, als Monitoring- und Synchronisationsmedium zwischen Kassen und Filialen, sowie die Selbstbedienungskasse (engl. Self-service checkouts). Die Synchronisationssoftware ist eine Anwendung, die ausschließlich im Hintergrund ausgeführt wird, und zu jeder Kasse dazu gehört. REST stellt dabei ein standardisiertes Kommunikationsmodell bereit, bei dem jede Ressource über einen eindeutigen Uniform Resource Locator (URL) adressiert ist und über einheitliche HTTP-Methoden angesprochen wird. Durch seine einfache Struktur und die lose Kopplung eignet sich dieser Ansatz besonders gut für die Anbindung heterogener Systeme in verteilten Architekturen [2].

Folglich kommuniziert die Gastronomie-App ausschließlich mit der Synchronisationssoftware. In Abbildung 2.1 wird die App nur mit den Hauptkomponenten und -seiten dargestellt, die für diese Kommunikation erforderlich sind. Einen detaillierten Einblick in die Funktionen und den Ablauf der App bietet Abbildung 2.2. Dort wird das Ablauf-

Abbildung 2.2: Flussdiagramm des Benutzerablaufs in der Gastronomie-App von der Einrichtung bis zur Zahlungsabwicklung

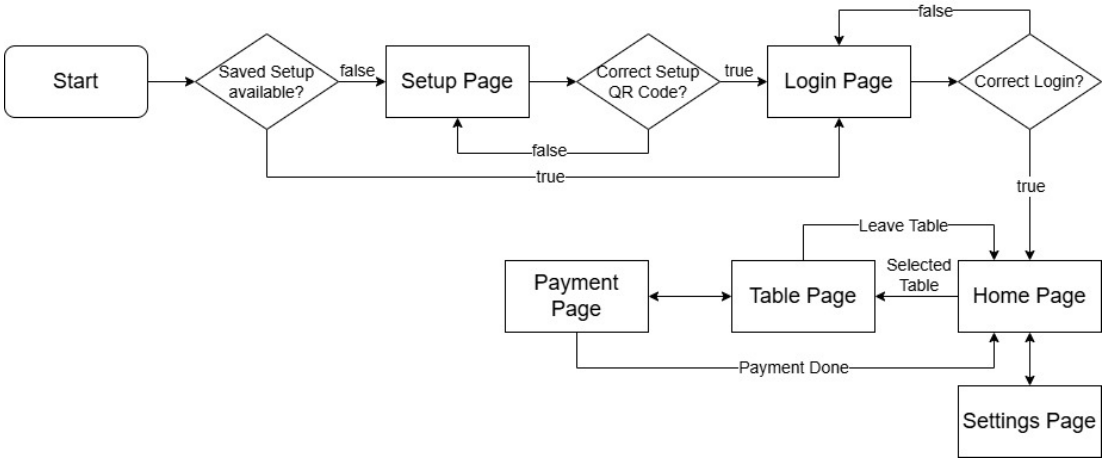


diagramm der Gastronomie-App dargestellt, das veranschaulicht, wie man in der App navigieren kann. Beim erstmaligen Start der App erscheint stets die Einrichtungsseite,

auf der die App mithilfe eines QR-Codes konfiguriert wird. Anschließend überprüft und validiert die Synchronisationssoftware die eingescannten Daten. Bei erfolgreichem Abschluss dieses Vorgangs erfolgt eine Weiterleitung zur Login-Seite, und die Setup-Daten werden für zukünftige App-Starts gespeichert. Danach wird die Anmeldung mittels der Benutzer-Login-Daten durchgeführt, wobei der Benutzer zuvor in der Kasse registriert worden sein muss. Nach erfolgreichem Login gelangt der Nutzer auf die Homepage der App, über die verschiedene Funktionen zugänglich sind. Beispielsweise kann die Einstellungsseite aufgerufen werden, um Anpassungen am Design und an den Systemabfragezeiten vorzunehmen. Alternativ besteht die Möglichkeit, einen Tisch auszuwählen, der anschließend bedient wird. In diesem Fall erfolgt die Weiterleitung auf die eigentliche Bedienungsseite (im Diagramm als „Table Page“ dargestellt), auf der die Bestellungen der Kunden aufgenommen werden können. Anschließend besteht die Möglichkeit, den Tisch zu schließen und auf weitere Bestellungen zu warten oder den Bezahlvorgang zu starten. Hierfür erfolgt eine Weiterleitung auf die Bezahlseite, auf der alle wichtigen Details zur Bestellung ausgefüllt werden können.

Der umfassende Einblick in die Architektur und den Ablauf der Gastronomie-App verdeutlicht, wie die einzelnen Komponenten – von der Kassensoftware über die Kassendatenbank bis hin zur Synchronisationssoftware – ineinandergreifen, um einen reibungslosen Betrieb zu gewährleisten. Diese Darstellung liefert eine Basis, um die Notwendigkeit eines leistungsfähigen Authentifizierungsmechanismus nachvollziehen zu können. Mit diesem Ausblick kann nun im nächsten Abschnitt detailliert auf die eigentliche Authentifizierung eingegangen werden.

3 Klassische Authentifizierungsmechanismen

3.1 Einführung

Um Benutzer nach erfolgreichem Login mit Benutzername und Passwort dauerhaft zu identifizieren und Anfragen sicher zu autorisieren, sind geeignete Authentifizierungsmechanismen erforderlich. In modernen Web- und Mobilanwendungen haben sich dabei vor allem zwei Verfahren etabliert: die sessionbasierte Authentifizierung und die tokenbasierte Authentifizierung [1].

Beide Methoden verfolgen das Ziel, nach dem initialen Login eine kontinuierliche Vertrauensbeziehung zwischen Client und Server aufrechtzuerhalten, unterscheiden sich jedoch grundlegend in ihrer technischen Umsetzung und ihren jeweiligen Vor- und Nachteilen.

Bevor auf die konkreten Verfahren eingegangen wird, ist es sinnvoll, die grundlegenden Begriffe Authentifizierung und Autorisierung voneinander abzugrenzen, da sie oft verwechselt werden, jedoch unterschiedliche Aufgaben im Kontext der Zugriffskontrolle erfüllen. Die Authentifizierung (engl. Authentication) dient dazu, die Identität eines Benutzers zu überprüfen – also sicherzustellen, dass die Person tatsächlich diejenige ist, für die sie sich ausgibt. Typische Beispiele sind die Eingabe eines Passworts, ein Fingerabdruck-Scan oder ein Einmalcode auf dem Smartphone.

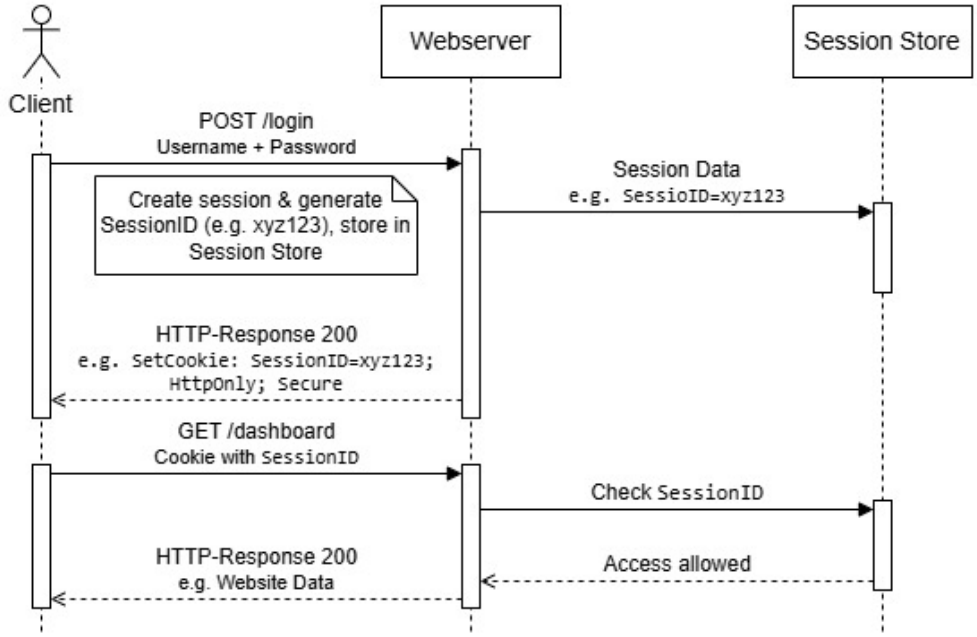
Erst nachdem ein Benutzer erfolgreich authentifiziert wurde, folgt im nächsten Schritt die Autorisierung (engl. Authorization). Dabei wird geprüft welche Rechte dieser Benutzer besitzt – also auf welche Daten oder Funktionen innerhalb des Systems er zugreifen darf. Während Authentifizierung somit die Frage „Wer bist du?“ beantwortet, steht bei der Autorisierung die Frage „Was darfst du tun?“ im Mittelpunkt [5].

Die Unterscheidung ist besonders wichtig, da ein Benutzer zwar erfolgreich authentifiziert sein kann, aber dennoch nicht autorisiert ist, bestimmte Bereiche oder Funktionen zu nutzen. Beide Prozesse bauen aufeinander auf und bilden gemeinsam das Fundament für eine sichere Zugriffskontrolle in modernen IT-Systemen.

3.2 Sessionbasierte Authentifizierung

Die sessionbasierte Authentifizierung zählt zu den klassischen und am weitesten verbreiteten Authentifizierungsmethoden [1]. Sie basiert auf der Idee, nach erfolgreicher Anmeldung eine serverseitig gespeicherte Sitzung zu erzeugen, die dem jeweiligen Nutzer zugeordnet wird. Die Sitzung bleibt über mehrere Anfragen hinweg aktiv und erlaubt eine eindeutige Zuordnung des Nutzers während der gesamten Nutzungsdauer.

Abbildung 3.1: Sequenzdiagramm der sessionbasierten Authentifizierung mit zentralem Session Store



Der Ablauf beginnt typischerweise mit den Absenden der Zugangsdaten (z.B. Benutzername und Passwort) an den Server, in Abbildung 3.1 mit `POST /login`. Nach erfolgreicher Prüfung der Zugangsdaten wird eine Session initialisiert und eine eindeutige Session-ID generiert, die in Abbildung 3.1 an den Session Store übertragen wird. Diese wird dem Client, meist über einen Cookie, übermittelt und bei jeder weiteren Anfrage automatisch mitgeschickt. Der Server nutzt die Session-ID, um die zugehörigen Sitzungsdaten abzurufen und den Benutzer entsprechend zu autorisieren.
In sessionbasierten Authentifizierungssystemen dienen HTTP-Cookies als zentrales Transportmittel zur Übermittlung der Sitzungskennung zwischen Client und Server. Nach erfolgreicher Authentifizierung generiert der Server eine eindeutige Session-ID, die in einem Cookie gespeichert wird. Adam Barth definiert in [6] das Verhalten von HTTP-Cookies standardisiert und beschreibt unter anderem sicherheitsrelevante Attribute wie `Secure` und `HttpOnly`, welche beispielhaft in Abbildung 3.1 enthalten sind. Ersteres stellt sicher, dass Cookies ausschließlich über verschlüsselte HTTPS-Verbindungen übertragen werden, während `HttpOnly` verhindert, dass clientseitige Skripte – wie er

beispielsweise im Falle eines in Abschnitt 3.4 näher erläuterten Cross-Site Scripting (XSS) Angriffs eintreten könnte – Zugriff auf den Cookie-Inhalt erhalten. Diese Schutzmechanismen sind entscheidend für die Integrität und Vertraulichkeit der Benutzersitzung [6].

Ein entscheidendes Merkmal der sessionbasierten Authentifizierung ist ihr zustandsbehaftetes (stateful) Verhalten: Der Server speichert für jeden eingeloggten Benutzer spezifische Daten, was bei wachsender Nutzerzahl zu einer erhöhten Serverlast führen kann. Darüber hinaus kann das Sitzungsmanagement in verteilten oder skalierenden Systemen (z.B. bei mehreren Servern) komplex sein, da die Sitzungsinformationen zentral abgeglichen oder persistiert werden müssen.

Trotz dieser Einschränkungen bleibt die sessionbasierte Authentifizierung besonders für klassische Webanwendungen relevant, in denen Benutzerinteraktionen in einem festen Kontext stattfinden und keine hohen Skalierungsanforderungen bestehen. Insbesondere für Anwendungen, die stark auf serverseitige Verarbeitung setzen, stellt sie nach wie vor eine praktikable Lösung dar [1].

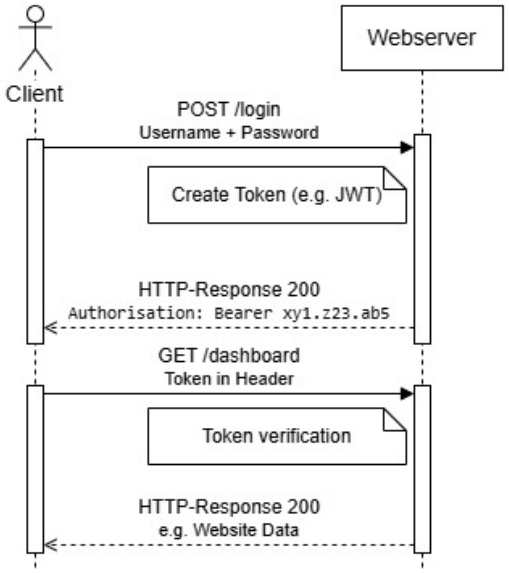
3.3 Tokenbasierte Authentifizierung

Die tokenbasierte Authentifizierung stellt eine moderne Alternative zur klassischen Sitzungsverwaltung dar. Im Gegensatz zum zustandsbehafteten Ansatz speichert der Server keine Benutzersitzung, sondern überträgt dem Client nach erfolgreichem Login ein digitales Token, das bei jeder weiteren Anfrage erneut mitgeschickt wird. Das Verfahren basiert auf dem Prinzip der Zustandslosigkeit (statelessness).

Der typische Ablauf der tokenbasierten Authentifizierung ist in Abbildung 3.2 dargestellt. Nach der Authentifizierung generiert der Server ein Token - häufig in Form eines JSON Web Tokens (JWT), das signiert, aber nicht verschlüsselt ist und Informationen über den Benutzer enthält – beispielsweise eine Benutzer-ID oder Rolleninformationen. Dieses Token wird vom Client bei jeder Anfrage im HTTP-Header mitgeführt und dient als Nachweis für eine gültige Authentifizierung. Der Server validiert das Token bei jedem Zugriff neu, ohne dabei einen Benutzersitzungszustand speichern zu müssen.

Ein Token kann unterschiedliche Formen annehmen, wie z.B. Zugriffstoken, Sitzungstoken oder Refresh-Token, wobei sich insbesondere Zugriffstoken in zustandslosen Architekturen durchgesetzt haben. Die Übertragung erfolgt üblicherweise über das sogenannte Bearer-Schema im Authorization-Header [1], etwa in folgender Form:

Abbildung 3.2: Sequenzdiagramm der tokenbasierten Authentifizierung mit serverseitiger Verifizierung



`Authorization: Bearer <token>` [1]

„Bearer“ heißt übersetzt „Träger“ – also „trägt“ der Client das Token mit sich und nutzt es, um sich gegenüber dem Server zu authentifizieren. Das vorangestellte Wort „Bearer“ signalisiert dem Server, dass es sich hierbei um einen Token handelt.

Ein wesentlicher Vorteil dieses Ansatzes ist die hohe Skalierbarkeit, da keine serverseitigen Sitzungsdaten verwaltet werden müssen. Insbesondere in verteilten oder cloudbasierten Systemen kann so auf die komplexe Synchronisierung von Sitzungen verzichtet werden. Gleichzeitig vereinfacht das Verfahren die domänenübergreifende Authentifizierung (Cross-Origin Requests), da das Token bei jeder Anfrage flexibel mitgesendet werden kann - unabhängig davon, auf welchem Server die Anfrage verarbeitet wird [1].

Ein weiterer Aspekt ist die Verwendung standardisierter Formate wie JWT, bei denen das Token aus drei Teilen besteht: Header, Payload und Signatur. Die Signatur schützt die Daten vor Manipulation, während die übrigen Bestandteile vom Server ausgelesen werden können, ohne dass zusätzliche Datenabfragen notwendig sind. Diese Eigenschaft reduziert den Aufwand bei der Autorisierung einzelner Anfragen erheblich.

Trotz der zahlreichen Vorteile bringt das Verfahren auch Herausforderungen mit sich. Da das Token clientseitig gespeichert und bei jeder Anfrage mitgeschickt wird, müssen Mechanismen zur Absicherung gegen Token-Diebstahl (z.B. durch Cross-Site Scripting (XSS) [3]) etabliert werden. Zudem muss sichergestellt sein, dass abgelaufene oder kompromittierte Tokens nicht mehr verwendet werden können, was in der Praxis durch ein

Ablaufdatum sowie gegebenenfalls durch die Nutzung von Refresh Tokens umgesetzt wird.

Insgesamt bietet die tokenbasierte Authentifizierung eine leistungsfähige, flexible und moderne Lösung für Anwendungen mit hohem Skalierungsbedarf oder verteilter Infrastruktur. Besonders in REST-basierten Architekturen, in denen Zustandslosigkeit gefordert ist, stellt sie eine bevorzugte Methode zur sicheren und performanten Nutzeridentifikation dar [1, 3].

3.4 Sicherheitsaspekte bei Authentifizierungsmechanismen

Unabhängig vom verwendeten Authentifizierungsverfahren ist die Absicherung gegenüber potenziellen Bedrohungen ein zentraler Aspekt moderner Anwendungssicherheit. Authentifizierungsmechanismen sind häufig das primäre Ziel von Angriffen, da sie den Zugang zu schützenswerten Ressourcen kontrollieren. Dabei bestehen je nach Verfahren unterschiedliche Angriffsvektoren und Schutzanforderungen – insbesondere im Bereich mobiler Anwendungen, in denen besondere Herausforderungen hinsichtlich Konnektivität, Gerätesicherheit und Nutzerverhalten bestehen.

Ein wesentliches Risiko bei tokenbasierten Verfahren besteht im möglichen Diebstahl oder Abfangen von Tokens, beispielsweise über unsichere Netzwerke, ungeschützte Speicherorte oder durch sogenannte Replay-Angriffe. Ein Angreifer kann dabei ein gültiges Token erneut verwenden, um sich unbefugt Zugriff zu verschaffen. Dieses Risiko wird insbesondere dann relevant, wenn Tokens im `localStorage` oder `sessionStorage` auf dem Gerät abgelegt werden. In solchen Fällen sind sie anfällig für XSS, wenn die Anwendung nicht ausreichend gegen das Einschleusen von schädlichem JavaScript geschützt ist. Um dem entgegenzuwirken, empfiehlt sich die Nutzung von HttpOnly-Cookies sowie der Einsatz von Content Security Policies (CSP), um die Ausführung nicht vertrauenswürdiger Scripte zu verhindern [3, 7].

Auch bei sessionbasierten Verfahren können sicherheitsrelevante Schwächen auftreten. Besonders hervorzuheben ist das Risiko des Session Hijacking, bei dem eine aktive Sitzung durch einen Angreifer übernommen wird – etwa durch das Abfangen der Session-ID über unsichere Verbindungen oder durch unzureichend abgesicherte Cookies. Schutzmaßnahmen wie die ausschließliche Übertragung über HTTPS, die Verwendung von `Secure-` und `HttpOnly`-Flags sowie eine automatische Session-Invalidierung bei Inaktivität sind hier zentrale Schutzmechanismen.

Mobile Anwendungen bringen zusätzliche sicherheitsrelevante Aspekte mit sich. So erfolgt der Zugriff häufig über öffentliche oder instabile Netzwerke, und die Endgeräte sind nicht selten gemeinsam genutzt oder besonders anfällig für Verlust oder

Diebstahl. In solchen Szenarien bietet der Einsatz von Multi-Faktor-Authentifizierung (MFA) einen erhöhten Schutz [7]. Dabei wird der Zugang nur gewährt, wenn mehrere unabhängige Faktoren – etwa ein Passwort und ein biometrisches Merkmal – erfolgreich nachgewiesen wurden. Selbst bei der Kompromittierung eines Faktors bleibt der Zugriff auf das System so weiterhin geschützt.

Darüber hinaus sollte das Prinzip der Minimalrechtevergabe (Least Privilege) Anwendung finden. Nutzer sollten nur jene Rechte erhalten, die für ihre jeweilige Aufgabe erforderlich sind, um die potenzielle Angriffsfläche zu minimieren. Dies verhindert, dass im Fall eines erfolgreichen Angriffs größere Schäden entstehen.

Insgesamt zeigt sich, dass die Sicherheit eines Authentifizierungsverfahrens nicht allein von der eingesetzten Technologie abhängt, sondern maßgeblich von deren Umsetzung und dem Einsatz ergänzender Schutzmechanismen beeinflusst wird. Gerade in mobilen Anwendungen, bei denen Benutzerfreundlichkeit, Gerätesicherheit und Netzwerkbedingungen stark variieren, müssen Authentifizierungslösungen so gestaltet werden, dass sie ein ausgewogenes Verhältnis zwischen Sicherheit, Effizienz und Bedienbarkeit gewährleisten [7].

4 Konzeption der Lösung

4.1 Einmalige Kopplung der App an die Kasse

Bevor ein Benutzer die Gastronomie-App regulär verwenden kann, muss diese einmalig mit einer konkreten Kasse gekoppelt werden. Dieser Schritt stellt sicher, dass die App eindeutig einer Kasseneinheit zugewiesen wird und nur im Rahmen einer gültigen Lizenz betrieben werden kann. Die Kopplung erfolgt durch das Scannen eines QR-Codes, der voraussichtlich direkt an der Kasse generiert wird.

Der QR-Code enthält ein kompaktes JSON-Objekt mit zwei essenziellen Parametern: der IP-Adresse der zugehörigen Synchronisationssoftware sowie einem temporär gültigen Lizenz-Token.
„JavaScript Object Notation (JSON) ist ein leichtgewichtiges, semi-strukturiertes Datenformat, das auf den Datentyp der Programmiersprache JavaScript basiert“ [8]. Es wird als Schlüssel-Wert-Struktur modelliert, welche beispielhaft in Listing 4.1 dargestellt ist und eignet sich insbesondere zur serialisierten Übertragung komplexer Objekte über Webschnittstellen. Im gezeigten Beispiel fungiert `userName` als Schlüssel, während `Maximusti` den zugehörigen Wert darstellt. Aufgrund seiner Flexibilität und Kompaktheit wird JSON heute weitverbreitet als Standardformat für den Datenaustausch im Webumfeld verwendet und kommt in zahlreichen Datenmanagementtechnologien zum Einsatz [8].

```
1 {
2         "userName": "Maximusti",
3         "hashedPassword": "5994471abb01112afcc18159f6cc74b4f511b998"
4 }
```

Listing 4.1: Beispiel eines gültigen JSON-Objekts

Während die IP-Adresse notwendig ist, um alle weiteren REST-Kommunikationen korrekt adressieren zu können, dient der Lizenz-Token als Schutzmechanismus gegen unautorisierte Nutzung. Nach dem Einlesen des QR-Codes sendet die App die enthaltenen Daten zusammen mit der MAC-Adresse des Geräts an die angegebene IP-Adresse der

Synchronisationssoftware. Die MAC-Adresse ermöglicht eine eindeutige Identifikation des verwendeten Geräts und dient als zusätzliche Maßnahme zur Missbrauchsvermeidung.

Die Lizenzprüfung erfolgt ausschließlich serverseitig innerhalb der Synchronisationssoftware. Diese prüft nicht nur die Gültigkeit des Tokens, sondern überwacht auch, wie viele App-Geräte aktiv mit der Kasse verbunden sind und ob dies der im Lizenzmodell hinterlegten Anzahl entspricht. Eine erfolgreiche Prüfung führt zur dauerhaften Kopplung der App an die Kasse. Dabei werden die IP-Adresse und der Lizenz-Token lokal in den `SharedPreferences` der App gespeichert, sodass bei zukünftigen Starts keine erneute Einrichtung erforderlich ist.

Sollte der Lizenz-Token zu einem späteren Zeitpunkt ungültig oder abgelaufen sein – etwa durch Erreichen eines Quartals- oder Monatsendes gemäß Lizenzmodell – wird beim nächsten App-Start eine erneute Validierungsanfrage an die Synchronisationssoftware gestellt. Verläuft diese erfolgreich, wird die Sitzung reaktiviert. Schlägt die Prüfung fehl, ist ein erneutes Scannen eines gültigen QR-Codes erforderlich, um den Betrieb der App fortzusetzen.

Die einmalige Kopplung der App bildet somit die Grundlage für eine kontrollierte und lizenzkonforme Nutzung der Anwendung und gewährleistet, dass alle weiteren Interaktionen ausschließlich mit einer autorisierten Kasseneinheit erfolgen.

4.2 Vergleich und Bewertung der Authentifizierungsverfahren

Im vorherigen Kapitel wurden zwei gängige Authentifizierungsverfahren – die sessionbasierte und die tokenbasierte Authentifizierung – ausführlich beschrieben. Beide verfolgen das Ziel, den Benutzer nach erfolgreicher Anmeldung eindeutig zu identifizieren und den Zugriff auf geschützte Ressourcen zu kontrollieren. Ihre jeweilige Eignung hängt jedoch stark vom Anwendungskonzept sowie den konkreten funktionalen und nicht-funktionalen Anforderungen ab.

Für die in dieser Arbeit betrachtete Gastronomie-App ergibt sich ein Anwendungskontext, in dem typischerweise nur wenige Geräte pro Kasse im Einsatz sind. Eine hohe Skalierbarkeit oder der Zugriff von vielen externen, wechselnden Clients – wie sie in zustandslosen, tokenbasierten Architekturen üblich ist – ist in diesem Szenario daher nicht erforderlich. Der Fokus liegt vielmehr auf der zuverlässigen Steuerung von Benutzeranmeldungen und insbesondere darauf, das gleichzeitige Zugriffe mehrerer Bediener auf denselben Tisch ausgeschlossen werden können. Um dies zu gewährleisten, ist eine zentral verwaltete Sitzungskontrolle notwendig, mit der sich aktive Sitzungen gezielt verwalten und zugehörige Ressourcen eindeutig zuordnen lassen.

Tabelle 4.1: Vergleich sessionbasierter und tokenbasierter Authentifizierungsverfahren

Kriterium	**Sessionbasiert**	**Tokenbasiert**
Speicherort der Sessiondaten	Serverseitig	Clientseitig
Skalierbarkeit	Begrenzt	Hoch
Komplexität der Implementierung	Einfach	Komplexer
Sicherheit bei lokalem Einsatz	Hoch	Abhängig von Client-Schutz
Geeignet für REST-Architektur	Eingeschränkt	Optimal
Mehrgeräteeinsatz	Ja, aber synchronisiert	Uneingeschränkt
Zentralisierte Kontrolle	Vollständig	Nicht direkt möglich

Eine Gegenüberstellung zentraler Eigenschaften beider Verfahren ist in Tabelle 4.1 dargestellt. Sie verdeutlicht die strukturellen und funktionalen Unterschiede, die bei der Auswahl eines geeigneten Mechanismus berücksichtigt wurden.

Die tokenbasierte Authentifizierung bringt zwar Vorteile hinsichtlich Skalierbarkeit und Flexibilität mit sich, führt aber in diesem Fall zu einem unnötig hohen Implementierungs- und Sicherheitsaufwand, insbesondere bei der Verwaltung und Absicherung von Tokens auf den jeweiligen Geräten. Auch wen die App und das Kassensystem innerhalb eines lokalem, kundeneigenen Netzwerks (WLAN) betrieben werden, ist der Einsatz verschlüsselter Übertragungsverfahren wie HTTPS dringend zu empfehlen. Durch die Nutzung von HTTPS wird sichergestellt, das sensible Daten (insbesondere Zugangsdaten und Session-IDs) nicht im Klartext übertragen werden und somit vor dem Zugriff durch unberechtigte Dritte geschützt sind. Die Absicherung erfolgt idealerweise durch die Konfiguration eines TLS-Zertifikats auf dem Server, wodurch alle REST-Endpunkte der Synchronisationssoftware ausschließlich über HTTPS angesprochen werden [9].
Wie aus Tabelle 4.1 ersichtlich bietet die sessionbasierte Authentifizierung in diesem Szenario entscheidende Vorteile: Durch die serverseitige Verwaltung der Benutzersitzung lässt sich gezielt steuern, welcher Benutzer auf welchen Tisch zugreifen darf, und gleichzeitige Zugriffe können zuverlässig unterbunden werden. Darüber hinaus kann der Session-Token durch zusätzliche serverseitige Informationen ergänzt werden – etwa durch eine eindeutige Tischkennung –, um konsistentes Verhalten in der Anwendung sicherzustellen und Schreibkonflikte in der Datenbank zu vermeiden.

Insgesamt zeigt sich, dass die sessionbasierte Authentifizierung die Anforderungen der Anwendung ideal erfüllt. Sie ermöglicht eine einfache und robuste Verwaltung aktiver Benutzer, reduziert die Komplexität der Implementierung und erlaubt eine gezielte Kontrolle über kritische Vorgänge innerhalb der App – wie den Tischzugriff –, ohne dabei unnötige sicherheitstechnische oder infrastrukturelle Anforderungen aufzubauen.

4.3 Auswahl und Begründung des Authentifizierungsverfahrens

Basierend auf den in Abschnitt 4.2 durchgeführten Vergleichen wurde die Entscheidung getroffen, in der Gastronomie-App eine sessionbasierte Authentifizierung einzusetzen. Ausschlaggebende für diese Wahl war neben den funktionalen Anforderungen insbesondere die Art und Wiese, wie sich Benutzer in der App anmelden, wie Geräte im Filialbetrieb verwendet werden und wie die Kommunikation mit dem Backend abläuft.

Die Anmeldung in der App wird direkt vom Benutzer initiiert und erfolgt durch die Eingabe eines Benutzernamens und Passworts. Die eigentliche Benutzerverwaltung liegt dabei nicht in der App selbst, sondern in der zentralen Synchronisationssoftware, die als Schnittstelle zwischen Kasse, Datenbank und externen Komponenten agiert. Diese Struktur spricht für ein serverseitiges Authentifizierungsmodell, bei dem die Anmeldung nicht dezentral über Token, sonder kontrolliert über Sitzungsdaten erfolgt.

Die gesamte Kommunikation zwischen App, Kasse und Synchronisationssoftware erfolgt über REST-Schnittstellen [2]. Dabei wird beim Login geprüft, ob die Anmeldedaten korrekt sind. Nach erfolgreicher Authentifizierung wird eine Sitzung erstellt und ein Session-Token generiert, der bei allen weiteren Anfragen mitgeführt und serverseitig validiert wird. Der Fokus liegt hier nicht auf Skalierbarkeit, sondern auf klarer Zustandsverwaltung und Zugriffskontrolle über eine begrenzte Anzahl an Geräten pro Filiale.

Ein weiterer Aspekt ist das Nutzungsszenario innerhalb der Filialen. Es ist vorgesehen, dass ein Gerät von mehreren Benutzern nacheinander verwendet werden kann. Dazu muss sich der aktuelle Benutzer entweder manuell abmelden oder wird automatisch abgemeldet, sobald er sich auf einen anderen Gerät mit denselben Zugangsdaten anmeldet. Diese Logik lässt sich in einem zustandsbehafteten System leicht umsetzten, da jede neue Sitzung die vorherige überschreibt. Ein Neustart der Synchronisationssoftware – beispielsweise im Rahmen eines nächtlichen Wartungsfensters – führt zudem automatisch zur Invalidierung aller Sessions und demzufolge zur Abmeldung aller Geräte.

Die Anforderungen an eine langfristige Session-Persistenz sind in diesen Szenario begrenzt. Nach einem Geräte-Neustart ist eine erneute Anmeldung erforderlich, was dem Bedienerwechsel im laufenden Betrieb zusätzliche Sicherheit verleiht. Gleichzeitig wird durch das zentrale Sessionhandling gewährleistet, dass kein Bediener versehentlich mit einer veralteten Sitzung weiterarbeitet.

Auch wenn die App ausschließlich innerhalb eines privaten Netzwerks betrieben wird, sollten grundlegende Sicherheitsmaßnahmen wie eine verschlüsselte Datenübertragung

Tabelle 4.2: Zusammenfassung zentraler Anforderungen an die Authentifizierungslösung

Kategorie	**Anforderung**
Funktional	Nur ein Benutzer pro Gerät gleichzeitig
Sicherheit	Session muss nach Logout ungültig sein
Technisch	Session-ID muss bei jeder Anfrage geprüft werden
Infrastruktur	Keine Token-Signierung nötig (lokales WLAN)

über HTTPS dennoch umgesetzt werden, um die Integrität und Vertraulichkeit sensibler Informationen – insbesondere des Session-Tokens – zu gewährleisten [9]. Dadurch lassen sich potentielle Angriffsvektoren wie Man-in-the-Middle-Angriffe bereits auf Netzwerkebene effektiv unterbinden. Die Notwendigkeit einer clientseitigen Speicherung von langgültigen Tokenstrukturen entfällt dennoch, da das gewählte Modell auf kurzfristige Sitzungen setzt, die serverseitig verwaltet werden. Der gewählte Ansatz erlaubt es zudem weitere Zustände – wie den geöffneten Tisch – direkt an die Sitzung zu binden, wodurch konkurrierende Zugriffe frühzeitig erkannt und verhindert werden können.

Insgesamt ergibt sich damit ein Authentifizierungskonzept, dass auf den konkreten Nutzungskontext zugeschnitten ist: überschaubare Geräteanzahl, serverseitige Benutzerverwaltung, planbarer Zugriffsrhythmus und zentral steuerbare Zustände. Die sessionbasierte Authentifizierung erfüllt diese Anforderungen in technischer wie konzeptioneller Hinsicht optimal und bietet zugleich die nötige Flexibilität für zukünftige Erweiterungen.

4.4 Anforderungen an die Authentifizierung

Die Authentifizierung für die Gastronomie-App muss verschiedene funktionale, technische und sicherheitsrelevante Anforderungen erfüllen, die sich aus dem konkreten Nutzungsszenario innerhalb der Filialen ergeben. Ziel ist es, einen zuverlässigen und kontrollierten Zugriff auf die Funktionen der App zu gewährleisten, gleichzeitig jedoch eine einfache und praktikable Handhabung im Arbeitsalltag zu ermöglichen.

Die Anmeldung erfolgt durch den Benutzer aktiv über die App und wird über Benutzername und Passwort realisiert. Die Verwaltung der Benutzerkonten findet zentral in der Synchronisationssoftware statt. Nach erfolgreicher Anmeldung wird eine serverseitige Sitzung erzeugt, über die der Benutzer eindeutig identifiziert wird. Die App stellt einen Logout-Button zur Verfügung, über den siche Benutzer manuell abmelden können. Zusätzlich findet eine automatische Abmeldung einmal täglich im Rahmen eines

geplanten Neustarts der Synchronisationssoftware statt, wodurch alle aktiven Sitzungen ungültig werden. Eine gleichzeitige Anmeldung auf mehreren Geräten ist nicht zulässig. Meldet sich ein Benutzer auf einem neuen Gerät an, wird die bestehende Sitzung automatisch beendet. Dies stellt sicher, dass ein Benutzer zu jeder Zeit nur auf einem Gerät aktiv ist.

Darüber hinaus ist es erforderlich, dass bestimmte Funktionen – wie beispielsweise das Stornieren von Artikeln – nur durch Benutzer mit entsprechender Berechtigung ausgeführt werden können. Hierfür muss das System in der Lage sein, die hinterlegte Benutzerrolle bei jeder Anfrage zu berücksichtigen. Die Kommunikation zwischen App und Synchronisationssoftware erfolgt vollständig über REST-Schnittstellen. Nach dem Login wird ein Session-Token vergeben, das in einem Cookie gespeichert und bei jeder weiteren Anfrage automatisch mitgeschickt wird. Die Gültigkeit der Sitzung wird bei jeder eingehenden Anfrage überprüft. Aktive Sitzungen werden im flüchtigen Speicher der Synchronisationssoftware gehalten. Ein manuelles Abmelden über die App ist jederzeit möglich und führt zur unmittelbaren Invalidierung der zugehörigen Sitzung auf dem Server. Falls ein Benutzer jedoch vergisst, sich abzumelden, bleibt die Sitzung zunächst bestehen – in diesem Fall sorgt die serverseitige Sessionverwaltung dafür, dass etwa ein geöffneter Tisch nur für fünf Minuten mit der aktiven Sitzung verknüpft bleibt, bevor dieser wieder frei gegeben wird. Zusätzlich wird im Rahmen eines nächtlichen Neustarts der Synchronisationssoftware der gesamte Sitzungsspeicher gelöscht, wodurch alle nicht aktiv beendeten Sessions automatisch ungültig werden. Da die Sitzungen nicht persistent gespeichert werden, ist kein zusätzlicher Inaktivitäts-Timeout notwendig. Bei einem Absturz der App oder einem Neustart des Endgeräts muss sich der Benutzer erneut anmelden. Ein Weiterarbeiten ohne aktive Verbindung ist nicht möglich; nach Wiederherstellung der Verbindung muss die App alle erforderlichen Daten erneut von der Synchronisationssoftware abrufen.

Aus sicherheitstechnischer Sicht ist es notwendig, dass ohne gültige Anmeldung keine Funktion innerhalb der App verfügbar ist. Jeder Benutzer kann grundsätzlich jeden Tisch öffnen, das System verhindert jedoch zuverlässig, dass ein Tisch gleichzeitig von mehreren Benutzern geöffnet wird. Diese Absicherung erfolgt serverseitig, indem Tischbelegungen an aktive Sitzungen gebunden werden. Sobald ein Tisch geöffnet wurde, ist er für andere Benutzer gesperrt. Falls sich ein Benutzer mit einem geöffneten Tisch ausloggt, wird die Tischbelegung automatisch wieder freigegeben. Auch wenn die App ausschließlich innerhalb eines geschlossenen, lokalen Netzwerks betrieben wird, wird für die Übertragung sensibler Daten, wie etwa Anmeldedaten und Session-Token, dennoch eine HTTPS-Verbindung verwendet. Dies stellt sicher, dass Kommunikationsdaten durchgehend verschlüsselt übertragen werden und nicht durch Man-in-the-Middle-Angriffe kompromittiert werden können.

4.5 Konzeption der Sitzungsverwaltung

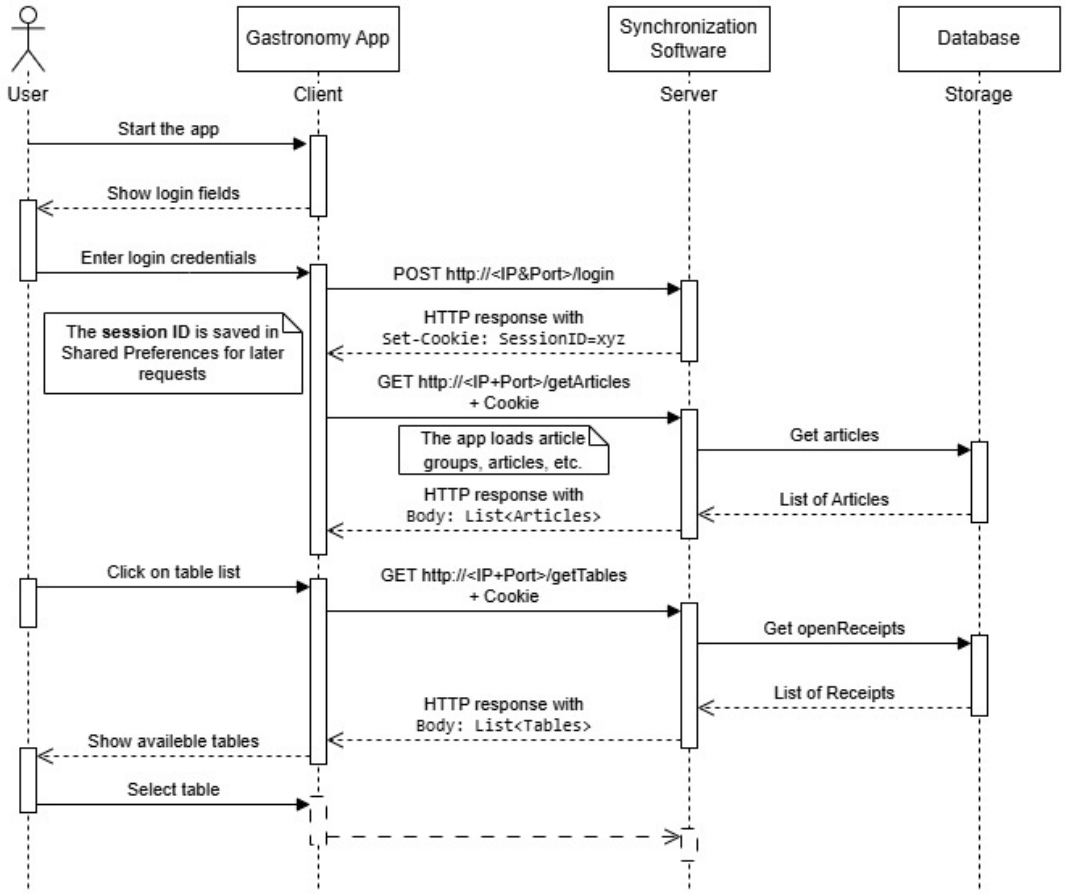
Die Sitzungsverwaltung bildet das technische Rückgrat der Authentifizierungslösung und dient der eindeutigen Identifikation und Steuerung angemeldeter Benutzer während der Nutzung der App. Nach erfolgreicher Anmeldung wird auf Serverseite eine Sitzung initialisiert, die den Benutzer eindeutig identifiziert und über eine Session-ID referenziert wird. Diese Session-ID wird als Cookie an die App übermittelt und dort in den `SharedPreferences` gespeichert, um bei zukünftigen Anfragen automatisch mitgeschickt zu werden.

Die Session bleibt solange gültig, bis sich der Benutzer manuell über den Logout-Button abmeldet oder die Synchronisationssoftware im Rahmen eines täglichen Neustarts automatisch alle aktiven Sitzungen invalidiert. Eine zusätzliche Ablaufzeit oder Inaktivitätsprüfung ist derzeit nicht vorgesehen. Die Sitzungsinformationen werden serverseitig in zwei zugehörigen HashMaps verwaltet, die eine Zuordnung von Bediener zu Token und umgekehrt ermöglichen. Darüber hinaus ist die Sitzung mit dem aktuell geöffneten Tisch verknüpft, sodass konkurrierende Zugriffe auf denselben Tisch zuverlässig erkannt und verhindert werden können.

Verlässt ein Benutzer die App ohne Abmeldung, bleibt die Sitzung zunächst bestehen und der geöffnete Tisch blockiert. Um eine unbeabsichtigte Dauerblockierung zu verhindern, ist vorgesehen, dass Tische nach einem definierten Zeitraum ohne Aktivität wieder automatisch freigegeben werden. Die konkrete Zeitspanne ist zum aktuellen Zeitpunkt noch nicht final festgelegt. Bei einem temporären Verbindungsverlust wird die Sitzung nicht beendet. Sobald die Verbindung wiederhergestellt ist, kann der Benutzer nach kurzer Ladepause nahtlos weiterarbeiten, ohne sich erneut anmelden zu müssen. Die Gültigkeit der Session wird bei jeder Serveranfrage überprüft. Dabei wird die mitgesendete Session-ID ausgewertet und zur Zuordnung des Bedieners sowie zur Validierung des Benutzerkontextes genutzt. Zugriffsrechte für bestimmte Funktionen werden dynamisch über die Benutzer-ID aus der Datenbank abgerufen.

Die Abläufe der Authentifizierung und der weiteren Kommunikation mit der Synchronisationssoftware lassen sich im Sequenzdiagramm in Abbildung 4.1 nachvollziehen. Dort wird der typische Einstieg eines Benutzers in die App sowie der technische Sitzungsaufbau veranschaulicht. Der Prozess beginnt mit dem Start der App, woraufhin dem Benutzer die Login-Oberfläche angezeigt wird. Nach der Eingabe von Benutzername und Passwort sendet die App die Anmeldedaten per `POST`-Anfrage an die Synchronisationssoftware. Im Erfolgsfall wird eine neue Sitzung erzeugt und die zugehörige Session-ID im Set-Cookie-Header der HTTP-Antwort an den Client übermittelt. Diese Session-ID wird in der App gespeichert – in diesem Fall in den `SharedPreferences` – und bei allen folgenden Anfragen automatisch im Cookie mitgesendet.

Abbildung 4.1: Sequenzdiagramm zur Authentifizierung und Sitzungsverwaltung



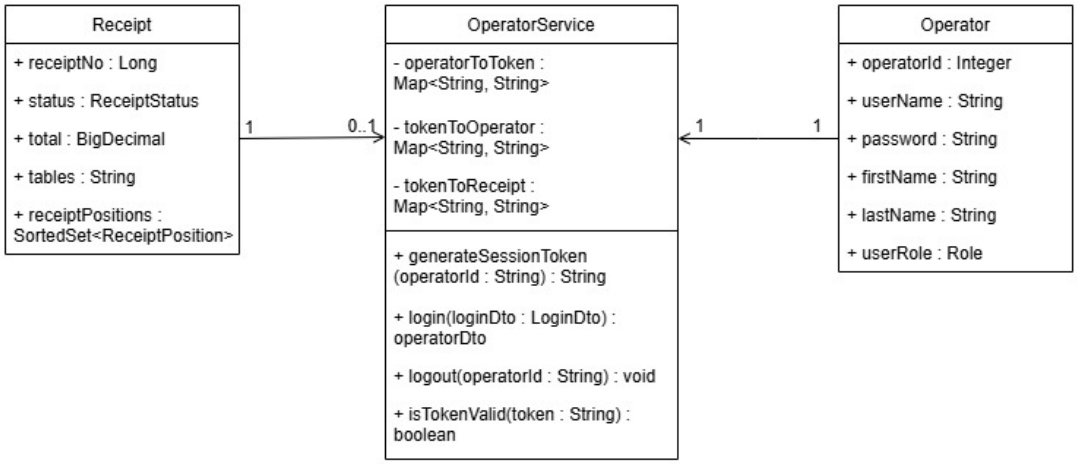
Direkt im Anschluss fordert die App per `GET`-Anfrage die Artikeldaten vom Server an. Diese Anfrage enthält bereits den Cookie mit der Session-ID, worüber die Synchronisationssoftware den Benutzer korrekt identifizieren und autorisieren kann. Nach erfolgreicher Validierung wird die entsprechende Liste an Artikeln zurückgegeben, die in der App anschließend verarbeitet und angezeigt wird.

Im weiteren Verlauf des Sequenzdiagramms in Abbildung 4.1 klickt der Benutzer auf die Tischliste. Dies löst eine erneute `GET`-Anfrage aus, die wiederum die Session-ID enthält. Die Synchronisationssoftware prüft daraufhin die offenen Tische, berücksichtigt dabei die aktuelle Sessionzuordnung und stellt eine Liste verfügbarer Tische bereit. Schließlich kann der Benutzer einen Tisch auswählen, der durch die Synchronisationssoftware anschließend mit der jeweiligen Session verknüpft wird.

Das Diagramm verdeutlicht dabei nicht nur die zeitliche Abfolge der Interaktionen, sondern auch die zentrale Rolle der Session-ID als durchgängiges Authentifizierungsmerkmal. Jede Anfrage – unabhängig von Inhalt oder Zeitpunkt – wird über die Session eindeutig einem Benutzer zugeordnet. Damit stellt das Sitzungsmanagement ein konsistentes und sicheres Kontrollinstrument über alle kritischen Interaktionen hinweg dar.

Die strukturelle Umsetzung der Sitzungsverwaltung in der Synchronisationssoftware ist in Abbildung 4.2 dargestellt. Das Klassendiagramm zeigt die zentralen Entitäten und ihre Beziehungen im Kontext der Authentifizierung. Die Klasse `OperatorService`

Abbildung 4.2: Klassendiagramm zur strukturellen Umsetzung der Sitzungsverwaltung



übernimmt dabei die Rolle des steuernden Dienstes für die Benutzeranmeldung und die Verwaltung der Sitzungen. Sie verwaltet die Zuordnungen zwischen Bedienern, Session-Tokens und zugehörigen Bestellvorgängen über interne Datenstrukturen in Form von `Map`-Objekten. Methoden wie `login()`, `logout()` oder `isTokenValid()` ermöglichen die Authentifizierung, Validierung und Beendigung von Sitzungen. Darüber hinaus zeigt das Diagramm die Kopplung zwischen `Operator`, der Session im `OperatorService` und `Receipt`, wobei ein Benutzer nur eine aktive Sitzung haben, die wiederum mit einem Receipt verknüpft sein kann. Durch diese klar definierte Objektstruktur lassen sich Zustände wie „eingeloggter Benutzer“ oder „Tisch geöffnet“ zuverlässig und konsistent abbilden.

Im aktuellen Planungsstand wird die Kommunikation zwischen App und Server über eine HTTPS-Verbindung abgesichert, wodurch Session-Tokens während der Übertragung effektiv vor dem Abfangen durch Dritte geschützt sind. Die Speicherung der Session-ID auf dem Gerät erfolgt dabei in den `EncryptedSharedPerferences` [10] um die clientseitige manipulation zu unterbinden.

5 Prototypische Umsetzung

Auf Basis der in Kapitel 4 beschriebenen Konzeption wurde im Folgenden ein Prototyp entwickelt, der die wesentlichen Funktionen der Authentifizierung innerhalb der Gastronomie-App realisiert. Ziel der prototypischen Umsetzung ist es, eine sichere und benutzerfreundliche Anmeldung, eine zuverlässige Verwaltung aktiver Sitzungen sowie eine kontrollierte Tischbindung zu gewährleisten.

5.1 Initiale Kopplung der Kasse

Bevor ein Benutzer sich in der App anmelden kann, muss eine einmalige Kopplung mit der zugehörigen Kasse erfolgen. Ziel dieses Prozesses ist es, der App die IP-Adresse der Synchronisationssoftware mitzuteilen, damit alle zukünftigen Anfragen korrekt adressiert werden können. Die initiale Konfiguration erfolgt über das Scannen eines QR-Codes, der die erforderliche Verbindungsinformation in einem JSON-Objekt codiert bereitstellt.

Der QR-Code wird direkt an der Kasse generiert und enthält im aktuellen Entwicklungsstand ausschließlich die IP-Adresse der Synchronisationssoftware. Ein serverseitiger Lizenzabgleich findet in dieser Phase noch nicht statt, da die App bislang nicht Bestandteil eines produktiven Lizenzprogramms ist. In künftigen Ausbaustufen ist jedoch vorgesehen, die Kopplung zusätzlich durch einen Lizenz-Token abzusichern.

Die technische Umsetzung des Scanvorgangs ist in Listing 5.1 dargestellt. Nach erfolgreichem Scanvorgang wird in Zeile 6 das aus dem QR-Code gelesene Ergebnis in die Variable `barcode` geschrieben. Wird kein gültiger QR-Code erkannt, gibt die Funktion in Zeile 9 `false` zurück und beendet den Vorgang. Andernfalls wird dem Benutzer ab Zeile 11 ein Ladebildschirm angezeigt, um den fortlaufenden Prozess visuell zu unterstreichen.

In Zeile 20 wird der Scanner deaktiviert und anschließend in Zeile 21 die gelesene IP-Adresse dauerhaft in den `SharedPreferences` unter dem Schlüssel `CRData` (Abkürzung für CashRegisterData) gespeichert. Diese Adresse dient fortan als Kommunikationsziel für sämtliche REST-Anfragen der App. Der Vorgang wird mit `true` abgeschlossen (Zeile 23) und die initiale Kopplung gilt damit als erfolgreich.

```dart
Future<bool> evaluateBarcode(BarcodeCapture barcodeCapture) async {
  final List<Barcode> barcodes = barcodeCapture.barcodes;
  String? barcode;

  for (final code in barcodes) {
    barcode = code.rawValue;
  }
  if (barcode == null) {
    return false;
  }
  showDialog(
    barrierDismissible: false,
    context: context,
    builder: (context) {
      return const AlertDialog(
        backgroundColor: Colors.transparent,
        content: Center(child: CircularProgressIndicator()),
      );
    });
  await scannerController.stop();
  await prefs.setString('CRData', barcode);

  return true;
}
```

Listing 5.1: Verarbeitung des initial gescannten QR-Codes zur Konfiguration der IP-Adresse auf der `CashRegisterLinkingPage`

Bei jedem folgenden Start der App wird überprüft, ob bereits eine Kopplung mit der Kasse erfolgt ist. Dazu wird der Schlüssel `CRData` aus den `SharedPreferences` ausgelesen. Die entsprechende Logik ist in der Klasse `AutGate` implementiert und im Listing 5.2 dargestellt. In Zeile 1 wird der gespeicherte Wert geladen. Falls dieser `null` ist (Zeile 3), bedeutet das, dass noch keine Kopplung durchgeführt wurde – in diesem Fall wird der Benutzer auf die Seite `/cashRegisterLinkingPage` weitergeleitet (Zeile 5). Ist hingegen ein gültiger Wert vorhanden, wird automatisch die Login-Seite geladen (Zeile 8), um den Anmeldeprozess zu starten.

Diese Prüfung stellt sicher, dass die App nur bei erfolgreicher Kopplung mit der entsprechenden Kasse verwendet werden kann. Erst nach dieser initialen Einrichtung wird der Benutzer zur Login-Seite weitergeleitet, über die der eigentliche Authentifizierungsprozess beginnt.

```
_prefs.getString('CRData').then(
  (value) {
    if (value == null) {
      Navigator.pushNamedAndRemoveUntil(
        context, '/cashRegisterLinkingPage', (route) => false);
    } else {
      Navigator.pushNamedAndRemoveUntil(
        context, '/loginPage', (route) => false);
    }
  },
);
```

Listing 5.2: Codeausschnitt zur Prüfung der initialen Kopplung bei App-Start und Weiterleitung zur passenden Seite

5.2 Implementierung Login und Session-Handling in der App

Die Benutzeranmeldung in der Gastronomie-App erfolgt über eine dedizierte Login-Seite, auf der der Benutzer seine Zugangsdaten bestehend aus Benutzername und Passwort eingibt. Nach der Eingabe der Anmeldedaten wird die Funktion `_confirmLogin()` aufgerufen, deren Ablauf in Listing 5.3 dargestellt ist.

Die Funktion übernimmt die Verschlüsselung des eingegebenen Passworts und erstellt anschließend ein neues `LiveApiRequest`-Objekt in Zeile 5, welches in Zeile 9 eine `POST`-Anfrage an die Synchronisationssoftware ausführt. Dabei wird ein `GastroAppLoginDto` mit dem Benutzernamen und dem verschlüsselten Passwort als Nutzlast übermittelt. Als Endpunkt der Anfrage wird `getOperator` verwendet, der in Zeile 6 als `path` definiert wird. Durch die direkte Verknüpfung der Login-Funktion mit dem `getOperator`-Endpunkt entfällt die Notwendigkeit zusätzlicher API-Aufrufe zur Benutzeridentifikation, wodurch die Anzahl der Kommunikationsvorgänge reduziert und die Effizienz der Anmeldung erhöht wird.

Nach erfolgreicher Ausführung der Anfrage wird die Antwort geprüft. Enthält die Antwort einen gültigen Benutzerdatensatz (`OperatorDto`), werden wichtige Informationen wie der Benutzername und die Operatornummer in den lokalen `SharedPreferences` gespeichert. Das vom Server im HTTP-Header zurückgelieferte Session-Token wird hingegen aus Sicherheitsgründen in den `EncryptedSharedPreferences` gespeichert, wie in Listing 5.3 ab Zeile 21 dargestellt. Dieses Token dient der eindeutigen Identifikation des Benutzers bei allen nachfolgenden Anfragen.
Schlägt die Anmeldung fehl oder tritt ein Fehler auf, gibt die Funktion einen entsprechenden Fehlerstatus zurück und die Anmeldung wird abgebrochen.

Durch dieses Verfahren wird sichergestellt, dass die App nach erfolgreicher Authentifizierung alle erforderlichen Benutzerinformationen und das Session-Token lokal verwal-

tet, um bei nachfolgenden Anfragen den Benutzer eindeutig und sicher zu identifizieren.

Um bei weiteren Serveranfragen den Benutzer eindeutig zu authentifizieren, wird das im Rahmen des Login-Prozesses erhaltene Session-Token automatisch in alle zukünftigen HTTP-Anfragen eingebunden. Die Klasse `LiveApiRequest`, die bereits beim Login zum Einsatz kommt, übernimmt dabei auch den Aufbau und das Abschicken der Requests an die Synchronisationssoftware. In der Methode `executePost()` (siehe Listing 5.4, ab Zeile 1) wird das übergebene Objekt serialisiert und gemeinsam mit dem Session-Token im Header als `Session-Token` an den Server übermittelt. Die Funktion `getDefaultHeaders()` stellt sicher, dass das Token aus den `EncryptedSharedPreferences` geladen und korrekt eingebunden wird. Listing 5.4 zeigt den Aufbau dieser beiden zentralen Funktionen.

Durch diese zentrale Einbindung des Session-Tokens in die Header-Konfiguration wird sichergestellt, dass der Benutzer bei jedem Api-Aufruf automatisch authentifiziert ist, ohne dass zusätzliche Authentifizierungsdaten übergeben werden müssen. Gleichzeitig dient das Token als Basis für die serverseitige Autorisierungsprüfung.

5.3 Implementierung Sessionmanagement in der Synchronisationssoftware

Die Synchronisationssoftware übernimmt die zentrale Verwaltung der Benutzer-Sessions, welche über den Endpunkt `getOperator` im `ExternalRequest`-Controller initialisiert werden (Listing 5.8, Zeile 1). Beim Aufruf dieses Endpunkts übermittelt die Gastronomie-App die Benutzerdaten (Benutzername und gehashtes Passwort) in einem `GastroAppLoginDto` an den Server. Die Verifizierung der Anmeldedaten erfolgt anschließend im `OperatorService`, wo geprüft wird, ob ein entsprechender Benutzer existiert und das Passwort übereinstimmt. Listing 5.5 zeigt die zentrale Methode `appLogin()`, die für die Prüfung und das Erzeugen des Benutzerobjekts verantwortlich ist.

Im Erfolgsfall wird aus den Benutzerdaten ein `OperatorDto` erzeugt (siehe Listing 5.5, Zeile 11 ff.), das die wesentlichen Informationen des Benutzers – darunter insbesondere die Benutzer-ID und die zugeordnete Rolle – enthält. Anschließend wird die Methode `generateSessionToken()` im `ExternalRequest`-Controller aufgerufen, wie in Listing 5.8, ab Zeile 7 dargestellt, und ein eindeutiges Session-Token generiert. Das Token basiert auf einem Universally Unique Identifier (UUID), der – wie in Zeile 2 von Listing 5.7 ersichtlich – erzeugt und anschließend serverseitig in zwei separaten `ConcurrentHashMap`-Strukturen gespeichert wird (vgl. Listing 5.6). Die erste Map, `operatorToToken`, ordnet die jeweilige Bedienernummer einem erzeugten Session-Token zu. Die zweite Map, `tokenToOperator`, bildet die Zuordnung in umgekehrter

Richtung ab und ermöglicht die Identifizierung eines Bedieners anhand eines übermittelten Session-Tokens.

Bei jeder nachfolgenden Anfrage an die Synchronisationssoftware ist das Session-Token durch den Client mitzuführen. Wie in Zeile 6 von Listing 5.9 dargestellt, wird die Gültigkeit des Tokens serverseitig überprüft – etwa beim Abruf offener Tische.

Durch diese Validierungslogik wird sichergestellt, dass nur Benutzer mit gültigen und aktiven Session-Token Zugriff auf geschützte Ressourcen erhalten. Sollte das Token ungültig sein, antwortet der Server mit den HTTP-Statuscode 401 (Unauthorized) (vgl. Zeile 9 in Listing 5.9), wodurch die App den Benutzer automatisch auf den Login-Bildschirm weiterleitet.

```dart
Future<bool> _confirmLogin(String username, String password) async {

  // Password hashing (implementation hidden for security reasons)

  LiveApiRequest liveApiRequest = LiveApiRequest(
    path: 'getOperator',
    ipAndPort: await prefs.getString('CRData')
  );
  ApiResponse response = await liveApiRequest.executePost(
    GastroAppLoginDto(
      userName: username,
      hashedPassword: hashedPassword
    )
  );
  if (response.status == Status.SUCCESS) {
    if (response.body != null && response.body!.isNotEmpty) {
      OperatorDto operator = OperatorDto.fromJson(
        jsonDecode(response.body!));
      await prefs.setString('username', username);
      await prefs.setString('operatorNumber', operator.number);
      await encryptedPrefs.setString(
        'sessionToken',
        response.headers?['session-token'] ?? 'empty');
      return true;
    } else {
      return false;
    }
  } else {
    log('Exception: ${response.exception.toString()},
      Body: ${response.body}');
    return false;
  }
}
```

Listing 5.3: Implementierung der Funktion `_confirmLogin()` zum Aufbau der Benutzersitzung in der App

```
Future<ApiResponse> executePost(Object? postObject) async {
  try {
    String serialized = "";
    if (postObject != null) {
      serialized = jsonEncode(postObject);
    }

    http.Response response = await http.post(
      Uri.parse(url),
      headers: getDefaultHeaders(
        await encryptedPrefs.getString('sessionToken') ?? ''),
      body: serialized,
      encoding: Encoding.getByName("utf-8"))
    .timeout(const Duration(seconds: 15));

    return await _handleResult(response);
  } catch (e) {
    log(e.toString());

    return ApiResponse(status: Status.EXCEPTION, exception: e);
  }
}

static Map<String, String> getDefaultHeaders(String sessionToken) {
  Map<String, String> headers = <String, String>{
    "Session-Token": sessionToken,
    "Content-Type": "application/json; charset=UTF-8",
    "Accept-Language": "de-DE",
    "Authorization": "..." // hidden for security reasons
  };

  return headers;
}
```

Listing 5.4: POST-Anfrage mit Session-Token-Header über `LiveApiRequest`

```java
public OperatorDto appLogin(GastroAppLoginDto loginDto)
  throws Exception {
  try {
    logger.info("Start Gastro-App Login Process");
    Operator op = operatorRepository.findByUserName(
      loginDto.getUserName());

    if (op != null && op.getPassword().equals(
      loginDto.getHashedPassword())) {
      logger.info("Gastro-App Login sucess");
      OperatorDto opDto = new OperatorDto();
      opDto.setUserName(op.getUserName());
      opDto.setDisplayName(op.getDisplayName());
      opDto.setNumber(op.getOperatorId().toString());
      opDto.setRole(
        operatorSignInOutService.operatorRoleType(op.getRole()));
      return opDto;
    } else {
      logger.info("Gastro-App Login failed");
      return null;
    }
  } catch (Exception e) {
    throw new InvalidOperatorException("Error during Login of "
      + loginDto.getUserName() + ", Error: " + e);
  }
}
```

Listing 5.5: Prüfung der Anmeldedaten und Rückgabe eines `OperatorDto` bei erfolgreichem Login im `OperatorService`

```java
Map<String, String> operatorToToken = new ConcurrentHashMap<>();
Map<String, String> tokenToOperator = new ConcurrentHashMap<>();
Map<String, String> tokenToReceipt = new ConcurrentHashMap<>();
```

Listing 5.6: Datenstrukturen zur Zuordnung von Bedienern, Session-Tokens und geöffneten Tischen im `OperatorService`

```
public String generateSessionToken(String operatorNumber) {
  String token = UUID.randomUUID().toString();

  if (operatorToToken.containsKey(operatorNumber)) {
    logger.info("Old Session from Operator " + operatorNumber
      + " with token " + operatorToToken.get(operatorNumber)
      + " is replaced by new Session with token " + token);
    tokenToOperator.remove(operatorToToken.get(operatorNumber));
    operatorToToken.remove(operatorNumber);
  } else {
        logger.info("Generated new Session for Operator "
          + operatorNumber + " with token " + token);
  }
  operatorToToken.put(operatorNumber, token);
  tokenToOperator.put(token, operatorNumber);
  return token;
}
```

Listing 5.7: Erzeugung eines neuen Session-Tokens und Aktualisierung der Token-Zuordnungen im `OperatorService`

```
@PostMapping("/app/getOperator")
@ResponseBody
public ResponseEntity<OperatorDto> getOperator(
  @RequestBody GastroAppLoginDto loginDto) throws Exception {
  OperatorDto opDto = operatorService.appLogin(loginDto);
  if (opDto != null) {
    String sessionToken = operatorService.generateSessionToken(
      opDto.getNumber());
    return ResponseEntity.ok().header("Session-Token", sessionToken)
      .body(opDto);
  }
  return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null);
}
```

Listing 5.8: Verarbeitung der Login-Anfrage im `ExternalRequest`-Controller und Rückgabe eines Session-Tokens bei erfolgreicher Authentifizierung

```
@GetMapping("app/getOpenReceipts")
@ResponseBody
public ResponseEntity<List<ReceiptExternalDto>> getOpenAppReceipts(
  @RequestHeader("session-token") String sessionToken)
    throws Exception {
  if (operatorService.isTokenValid(sessionToken)) {
    return ResponseEntity.ok(receiptService.getAllOpenReceipts());
  } else {
    return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null);
  }
}
```

Listing 5.9: Überprüfung der Session-Token-Gültigkeit bei API-Anfragen

6 Tests

6.1 Integrationstest des Login-Prozesses

Zur Überprüfung der korrekten Funktionsweise des Login-Prozesses wurde ein Integrationstest implementiert, der den vollständigen Ablauf von der Benutzerinteraktion bis zur Navigation in die Hauptansicht automatisiert durchläuft. Ziel ist es, die funktionale Korrektheit der Benutzeranmeldung unter realitätsnahen Bedingungen sicherzustellen.

Zunächst wird in Zeile 6 des Listings 6.1 ein Eintrag im `SharedPreferences`-Speicher simuliert, der normalerweise im Zuge der initialen Kopplung über einen QR-Code erzeugt wird. Damit ist sichergestellt, dass die App beim Start nicht zur Kassenkopplungsseite navigiert, sondern direkt zur Login-Seite weiterleitet.

Ab Zeile 14 wird die Hauptanwendung mit den erforderlichen Provider-Komponenten geladen. Nach einer kurzen Initialisierungsphase (Zeile 19), in der alle asynchronen Operationen abgeschlossen werden, beginnt der eigentliche Testablauf: In Zeile 20 wird das Textfeld mit dem Key `'username'` mit einem Testwert befüllt, gefolgt von der Eingabe des Passworts in Zeile 22. Mit dem Tippen auf den Button `'loginButton'` in Zeile 24 wird der Anmeldevorgang ausgelöst.

Der entscheidende Testausdruck befindet sich in Zeile 27: Dort wird überprüft, ob die `HomePage` nach erfolgreicher Authentifizierung im Widget-Tree gefunden werden kann. Ist dies der Fall, gilt der Test als bestanden.

Dieser Test zeigt exemplarisch, wie durch gezielte Simulation von Benutzeraktionen und systematisches Beobachten des App-Zustands eine zuverlässige Absicherung zentraler Funktionalitäten möglich ist.

```
1  void main() async {
2    IntegrationTestWidgetsFlutterBinding.ensureInitialized();
3
4    setUp(
5      () async {
6        SharedPreferences.setMockInitialValues({
7          'CRData': '10.110.1.15:8080'
8        });
9      },
10   );
11   testWidgets(
12     'Login with valid username and password',
13     (widgetTester) async {
14       await widgetTester.pumpWidget(MultiProvider(providers: [
15         ChangeNotifierProvider(create: (context) => ThemeProvider()),
16         ChangeNotifierProvider(create: (context) => DataProvider()),
17       ], child: const MyApp()));
18
19       await widgetTester.pumpAndSettle();
20       await widgetTester.enterText(
21         find.byKey(const Key('username')), 'Tester');
22       await widgetTester.enterText(
23         find.byKey(const Key('password')), '12345');
24       await widgetTester.tap(find.byKey(const Key('loginButton')));
25       await widgetTester.pumpAndSettle();
26
27       expect(find.byType(HomePage), findsOneWidget);
28     },
29   );
30 }
```

Listing 6.1: Integrationstest für den Login mit gültigen Zugangsdaten

6.2 Validierung der Header-Konstruktion mit EncryptedSharedPerferences

Ein weiterer wichtiger Aspekt der getesteten Authentifizierungslogik ist die korrekte Erzeugung der HTTP-Header mit dem gespeicherten Session-Token. In der App wird dieser Token nach dem Login verschlüsselt in den `EncryptedSharedPreferences` abgelegt. Um sicherzustellen, dass der gespeicherte Wert bei nachfolgenden Anfragen korrekt gelesen und in den HTTP-Header integriert wird, wurde ein entsprechender Integrationstest geschrieben.

Der Test initialisiert in Zeile 4 des Listings 6.2 die verschlüsselte SharedPreferences-Instanz mit einem statischen Schlüssel. Anschließend wird in Zeile 8 ein Referenzobjekt auf die Speicherinstanz erzeugt und in Zeile 12 ein fest definierter Session-Token gespeichert. In Zeile 14 wird dieser wieder aus dem Speicher gelesen und übergeben,

um mit der Methode `getDefaultHeaders()` einen vollständigen Header zu erzeugen. Die darauffolgenden Testbedingungen ab Zeile 17 prüfen, ob das Header-Objekt die erwarteten Inhalte aufweist: Der `'Session-Token'` muss korrekt übernommen worden sein, der `'Content-Type'` muss auf `'application/json'` gesetzt sein, die Sprache auf Deutsch und der `'Authorization'`-Header vorhanden sein.

```
1  void main() {
2    IntegrationTestWidgetsFlutterBinding.ensureInitialized();
3
4    testWidgets('getDefaultHeaders with encryptedPrefs',
5      (tester) async {
6      await EncryptedSharedPreferencesAsync.initialize(
7        '1234567890123456');
8      final encryptedPrefs =
9        EncryptedSharedPreferencesAsync.getInstance();
10
11     const testToken = '436913fb-1925-4996-b577-80efcd903dc3';
12     await encryptedPrefs.setString('sessionToken', testToken);
13
14     final headers = LiveApiRequest.getDefaultHeaders(
15       await encryptedPrefs.getString('sessionToken') ?? 'none');
16
17     expect(headers['Session-Token'], equals(testToken));
18     expect(headers['Content-Type'], contains('application/json'));
19     expect(headers['Accept-Language'], equals('de-DE'));
20     expect(headers['Authorization'], isNotNull);
21   });
22 }
```

Listing 6.2: Integrationstest zur Validierung des Session-Headers mit `EncryptedSharedPreferences`

Der erfolgreich durchgeführte Test bestätigt die Integrität der Speicherung und die korrekte Wiederverwendung des Tokens in der Header-Konstruktion – ein zentraler Bestandteil für sichere, zustandsbehaftete Authentifizierungsprozesse.

7 Evaluation

Ziel der vorliegenden Arbeit war es, einen sicheren und praktikablen Authentifizierungsmechanismus für eine Gastronomie-App zu konzipieren und prototypisch umzusetzen. Die Evaluation betrachtet, inwieweit die gestellten Anforderungen erfüllt wurden, welche Erkenntnisse aus der Umsetzung hervorgingen und welche Grenzen oder offenen Fragen bestehen bleiben.

Ein zentrales Kriterium für die Bewertung ist die Erfüllung der funktionalen Anforderungen, die sich aus dem praktischen Einsatzkontext ergeben: Die Anmeldung muss zuverlässig erfolgen, auch bei regelmäßig wechselnden Geräten; gleichzeitig darf ein Benutzer stets nur auf einem Gerät aktiv sein. Durch das entworfene sessionbasierte Modell konnte diese Anforderung erfolgreich umgesetzt werden. Die zentrale Sitzungsverwaltung in der Synchronisationssoftware ermöglicht es, bestehende Sitzungen bei erneuter Anmeldung automatisch zu invalidieren und zu ersetzen.

Ein wesentliches Sicherheitsmerkmal, das in der Konzeption (vgl. Kapitel 4.2) vorgesehen ist, wurde im Prototyp bewusst noch nicht umgesetzt: die Übertragung der Kommunikationsdaten über HTTPS. Der Grund hierfür liegt in der technischen Komplexität, da mehrere externe Komponenten – wie beispielsweise die Selbstbedienungskassen (vgl. Kapitel 2) – hätten angepasst werden müssen. Obwohl die aktuelle Implementierung noch auf eine unverschlüsselte Verbindung setzt, wurde in der Konzeption bereits berücksichtigt, dass die Übertragung sensibler Daten über ein verschlüsseltes Protokoll wie HTTPS zwingend erforderlich ist, um langfristig einem professionellen Sicherheitsstandard zu genügen. Der Umstieg auf HTTPS wird daher als eines der vorrangigen Ziele für die nächste Ausbaustufe der definiert.

Auch die einmalige Kopplung der App mit der Kasse konnte erfolgreich umgesetzt werden. Der Scanvorgang sowie die Speicherung der Verbindungsdaten wurden zuverlässig implementiert und stellen sicher, dass nur autorisierte Geräte Zugriff auf die Kasse erhalten. Die Entscheidung, im aktuellen Stadium noch auf eine Lizenzvalidierung zu verzichten, wurde aufgrund des prototypischen Charakters der App bewusst getroffen. Sie erlaubt es, den Mechanismus zunächst technisch zu verankern und bei Bedarf später funktional zu erweitern.

Die durchgeführte Evaluation zeigt, dass die Kombination aus theoretisch fundierter Konzeption und praxisnaher Umsetzung zu einem Authentifizierungssystem geführt

hat, das den Anforderungen der Gastronomie-App gerecht wird. Die klare Trennung zwischen App und Serverlogik sowie die zentralisierte Zustandskontrolle tragen zu einer hohen Robustheit und Wartbarkeit der Lösung bei.

Nichtsdestotrotz bleiben auch gewisse Einschränkungen bestehen. So wurde beispielsweise auf den Einsatz zusätzlicher Sicherheitsverfahren wie Multi-Faktor-Authentifizierung verzichtet. Auch Skalierungsszenarien mit vielen Geräten oder verteilten Infrastrukturen wurden nicht berücksichtigt. Diese Aspekte bieten Raum für zukünftige Weiterentwicklungen, die in Kapitel 8 weiter skizziert werden.

8 Fazit und Ausblick

Im Rahmen dieser Bachelorarbeit wurde ein Authentifizierungskonzept für eine Gastronomie-App entwickelt, umgesetzt und evaluiert. Ziel war es, ein Verfahren zu realisieren, das sowohl den betrieblichen Anforderungen im gastronomischen Umfeld als auch sicherheitstechnische und technische Rahmenbedingungen gerecht wird. Im Fokus standen dabei eine einfache Benutzerführung, eine zuverlässige Sitzungsverwaltung und die Sicherstellung exklusiver Zugriffe auf tischbezogene Bestellungen.

Die Analyse verschiedener Authentifizierungsverfahren zeigte, dass die sessionbasierte Lösung den Anforderungen am besten entspricht. Sie ermöglicht eine serverseitig gesteuerte Benutzerkontrolle und verhindert zuverlässig konkurrierende Zugriffe. Die prototypische Implementierung umfasste sowohl die Authentifizierungslogik in der App als auch das zugehörige Sessionmanagement in der Synchronisationssoftware. Durch gezielte Tests konnte die Funktionsfähigkeit der Lösung überprüft und typische Anwendungsfälle abgedeckt werden. Der Einsatz von `EncryptedSharedPreferences` zur Speicherung sicherheitskritischer Informationen zeigt bereits erste Maßnahmen zur Härtung des Systems. Gleichzeitig wurde deutlich, dass der Umstieg auf eine HTTPS-Verschlüsselung mittelfristig notwendig ist, auch wenn dieser Schritt im Prototyp noch nicht umgesetzt werden konnte.

Die Evaluation der entwickelten Lösung zeigt, dass die Zeile in weiten Teilen erreicht wurden. Die App bietet einen zuverlässigen Login-Mechanismus, verhindert Mehrfachanmeldungen und sichert Zugriffe auf Tische konsistent ab. Herausforderungen bestehen noch bei der automatisierten Aufhebung blockierter Tische und bei der Einführung zusätzlicher Sicherheitsmechanismen - insbesondere im Hinblick auf Token-Gültigkeit und Netzwerkverbindungen außerhalb des geschlossenen WLANs.

Ein besonders praxisnaher Erweiterungsansatz betrifft die Nutzung von Near Field Communication (NFC) Technologie. So wäre es denkbar, dass sich Bediener künftig nicht mehr manuell über Benutzername und Passwort anmelden, sondern stattdessen ein persönlicher NFC-Transponder – etwa integriert in einem Namensschild – zum Login genutzt wird. Zusätzlich können an jedem Tisch stationäre NFC-Chips verbaut werden, über die sich ein Tisch direkt durch kurzes Annähern des Geräts auswählen lässt. Diese Trennung zwischen Benutzeridentifikation und Tischauswahl würde nicht

nur die Benutzerführung deutlich vereinfachen, sondern auch die Geschwindigkeit und Sicherheit im Arbeitsalltag erhöhen.

Darüber hinaus ist geplant, in einer späteren Ausbaustufe eine Benutzerbindung an einen Tisch einzuführen. Sobald ein Tisch von einem Bediener geöffnet wurde, soll dieser exklusiv an dessen Sitzung gekoppelt werden, sodass keine versehentliche Übernahme durch andere Benutzer erfolgen kann.

Auch infrastrukturelle Aspekte sind Teil des Ausblicks. Bei Kunden mit unzureichender WLAN-Abdeckung könnte eine LTE-gestützte Kommunikation in Erwägung gezogen werden. Dies würde eine zuverlässigere Verbindung zwischen App und Kasse ermöglichen, erfordert jedoch zusätzliche Sicherheitsmaßnahmen und gegebenenfalls eine Überarbeitung des bestehenden Netzwerkkonzepts.

Weitere denkbare Funktionserweiterungen umfassen das Starten oder Beenden von Schichten über die App, was bislang nur direkt an der Kasse möglich ist. Auch kundenbezogene Funktionen wie das Einscannen von Bonus- oder Kundenkarten könnte langfristig in das System integriert werden.

Zusammenfassend zeigt sich, dass mit der entwickelten Lösung ein tragfähiges Fundament geschaffen wurde, das sowohl die aktuellen Anforderungen erfüllt als auch zukunftssicher ausgebaut werden kann. Die Arbeit leistet damit einen wertvollen Beitrag zur sicheren und benutzerfreundlichen Digitalisierung von Prozessen im gastronomischen Alltag.

Literaturverzeichnis

[1] Balaj, Yjvesa: *Token-Based vs Session-Based Authentication: A survey.* https://www.researchgate.net/publication/320068250_Token-Based_vs_Session-Based_Authentication_A_survey. Version: 27.09.2017. – [Online; abgerufen am 28. März 2025]

[2] Fielding, Roy T.: *Architectural Styles and the Design of Network-based Software Architectures.* https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. Version: 2000. – [Online; abgerufen am 05. April 2025]

[3] Bulgakova, Oleksandra ; Mashkov, Viktor ; Zosimov, Viacheslav ; Popravkin, Pavlo: *Risk of Information Loss Using JWT Token.* https://ceur-ws.org/Vol-3101/Short21.pdf. Version: 16.09.2021. – [Online; abgerufen am 01. April 2025]

[4] Gupta, Naveen K.: *Impact of Flutter Technology in Software Development.* http://dx.doi.org/10.55248/gengpi.5.0724.1928. Version: Juli 2024

[5] Morowczynski, Mark ; Epping, Michael: *Authentication and Authorization.* http://dx.doi.org/10.55621/idpro.78. Version: 2021

[6] Barth, Adam: *HTTP State Management Mechanism.* https://datatracker.ietf.org/doc/html/rfc6265. Version: April 2011

[7] Hasan, Syed Shabih U. ; Ghani, Anwar ; Daud, Ali ; Akbar, Habib ; Khan, Muhammad F.: *A Review on Secure Authentication Mechanisms for Mobile Security.* http://dx.doi.org/10.3390/s25030700. Version: 01.11.2024

[8] Lv, Teng ; Yan, Ping ; He, Weimin: *Survey on JSON Data Modelling.* http://dx.doi.org/10.1088/1742-6596/1069/1/012101. Version: 2018

[9] Bundesamt für Sicherheit in der Informationstechnik (BSI): *BSI TR-02102-2: Verwendung von Transport Layer Security (TLS).* https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-2.pdf. Version: 09. August 2023

[10] Flutter Community: *encrypt_shared_preferences*. `https://pub.dev/packages/encrypt_shared_preferences`. Version: 2025. – [Online; abgerufen am 15. April 2025]

Erklärung zur Verwendung generativer KI-Systeme und Software

Die Arbeit wurde unter Verwendung folgender generativer KI-Systeme und ähnlich leistungsfähiger Software zur Textproduktion erstellt:

1. Chat-GPT

Ich habe mich über die oben genannten KI-Systeme bzw. Programme und deren Funktionsweise informiert und diese verstanden. Aus den Systemen übernommene Textpassagen und Artefakte habe ich gekennzeichnet und auf ihre Richtigkeit überprüft.

Die oben genannten KI-Systeme bzw. Programme kamen bei den folgenden Arbeitsschritten zum Einsatz:

Arbeitsschritt	**KI-System bzw. Programm**	**Verwendung**
Literaturrecherche	Chat-GPT	Literatursuche im Theorie Teil
Formulierung des Textes der Arbeit	Chat-GPT	Vereinzelte Einleitungs- und Abschlusssätze
Verbesserung bzw. Redigieren des Textes der Arbeit	Chat-GPT	Redigieren vereinzelter Sätze

Ort, Datum

Simon Galle

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als die angegebenen verwendet habe.

Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Ort, Datum

Simon Galle