Launcher

Table of Contents

About arc42

arc42, the template for documentation of software and system architecture.

Template Version 8.2 EN. (based upon AsciiDoc version), January 2023

1. Einführung und Ziele

In diesem Kapitel werden die grundlegenden Ziele des Luddite Launcher Projekts sowie die wichtigsten Anforderungen und Stakeholder beschrieben.

1.1. Aufgabenstellung

Das Luddite Launcher Projekt ist Teil eines Projektes von mir mit dem Ziel ein Phone zu erstellen, welches den Social Media Zugang einschränkt. Die Anwendung dient als primäre Benutzeroberfläche (Launcher) für ein modifiziertes Android-Betriebssystem, das ohne herkömmlichen Webbrowser ausgeliefert wird.

Der Luddite Launcher ermöglicht Benutzern den Zugang zu ausgewählten Web-Anwendungen durch ein Capacitor Browser Plugin, wodurch nur bestimmte, zuvor definierte Websites geöffnet werden können. Dies verhindert unkontrolliertes Surfen und die damit verbundenen Ablenkungen.

Zentrale Funktionen des Launchers sind:

- Anzeige und Suche von verfügbaren Web-Apps
- Nutzerauthentifizierung (Login)
- Öffnen von Web-Apps in einer kontrollierten Browser-Umgebung
- Möglichkeit, neue Web-App-Vorschläge einzureichen
- Integration mit nativen Apps wie WhatsApp

1.2. Qualitätsziele

Ausgehend von den grundlegenden Anforderungen an das System wurden folgende Qualitätsziele definiert:

Priorität	Qualitätsziel	Motivation
1	Aktualität der App-Liste	Die Liste der verfügbaren Web-Apps muss bei jedem Öffnen der App vom Backend geladen werden, um stets aktuell zu bleiben
2	Benutzerfreundliche Suche	Benutzer müssen schnell und einfach nach verfügbaren Web-Apps suchen und diese finden können
3	Kontrollierte Web-Nutzung	Das Öffnen von Web-Apps muss in einer kontrollierten Umgebung erfolgen, bei der Benutzer die URL nicht ändern können
4	Erweiterbarkeit durch Feedback	Benutzer sollen Vorschläge für neue Web-Apps einbringen können, um die Nützlichkeit des Systems kontinuierlich zu verbessern

Priorität	Qualitätsziel	Motivation
5	Zugangskontrolle	Nur angemeldete Benutzer sollen die App nutzen können, um personalisierte Einstellungen zu ermöglichen
6	Integration nativer Anwendungen	Native Apps wie WhatsApp sollen nahtlos integriert werden, um ein vollständiges Nutzungserlebnis zu bieten
7	Zeitmanagement	Für Social Media-Seiten soll eine Zeitbeschränkung möglich sein, um die Bildschirmzeit zu reduzieren und digitale Abhängigkeiten zu bekämpfen

1.3. Stakeholder

Rolle	Erwartungen
Endbenutzer	Möchten ein Smartphone, das ihnen hilft, digitale Abhängigkeiten zu reduzieren, ohne auf wichtige Funktionen verzichten zu müssen
Entwickler	Benötigen eine gut strukturierte, erweiterbare Architektur, die einfach zu warten und zu aktualisieren ist
Dozent	Erwarten eine klar dokumentierte Lösung, die architektonische Entscheidungen und deren Begründungen nachvollziehbar darstellt

2. Randbedingungen

Die Entwicklung des Luddite Launchers unterliegt verschiedenen technischen und organisatorischen Randbedingungen, die im Folgenden zusammengefasst werden.

2.1. Technische Randbedingungen

Randbedingung	Erläuterung
LineageOS als Basis	Die Anwendung muss auf einem modifizierten LineageOS (für Google Pixel 6) lauffähig sein, bei dem der Standard-Browser entfernt wurde
Angular & Capacitor	Die Frontend-Entwicklung erfolgt mit Angular und Capacitor für die Erstellung der hybriden mobilen App
Capacitor InAppBrowser Plugin	Das Capacitor InAppBrowser Plugin wird für die kontrollierte Darstellung von Web-Inhalten verwendet
Node.js Backend	Das Backend wird mit Node.js implementiert
MongoDB	MongoDB wird als Datenbank für die Speicherung von Benutzerdaten, App-Listen und Wunschlisten verwendet

2.2. Organisatorische Randbedingungen

Randbedingung	Erläuterung
WebLab	Das Projekt wird im Rahmen eines Schulprojekts durchgeführt, mit entsprechenden Dokumentationsanforderungen
Hosting	Die Backend-Container und MongoDB werden auf einem Infomaniak Server in einer Openstack Public Cloud Infrastruktur gehostet

2.3. Konventionen

Konvention	Erläuterung
Versionskontrolle	Das Projekt wird mit Git verwaltet, mit GitHub Actions für CI/CD
Dokumentation	Die Architektur wird mit arc42 dokumentiert und via CI automatisiert deployed.
Codestruktur	Angular-Komponenten werden nach Funktionalität organisiert

3. Kontext und Abgrenzung

3.1. Fachlicher Kontext

Der Luddite Launcher fungiert als zentrale Schnittstelle zwischen dem Benutzer und den verschiedenen Anwendungsmöglichkeiten. Im Gegensatz zu herkömmlichen Android-Launchern kontrolliert er den Zugang zu Webinhalten und bietet nur Zugriff auf ausgewählte Websites und native Anwendungen.

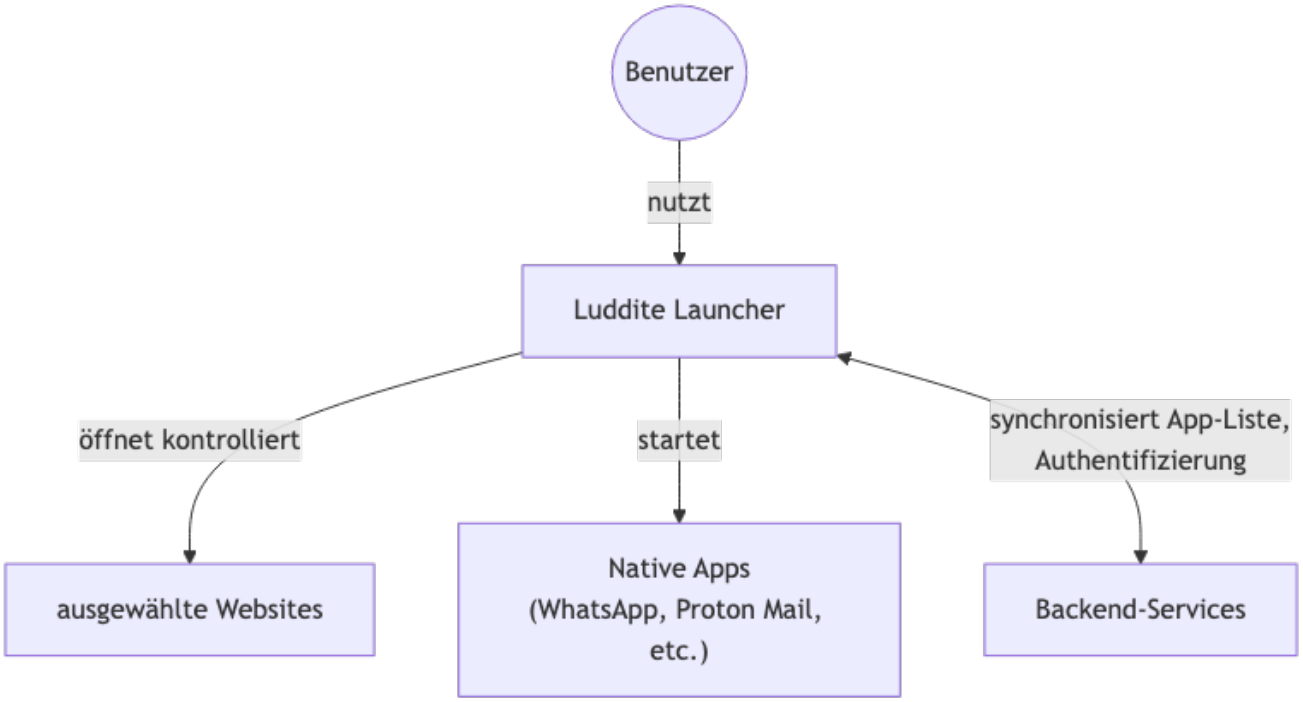


Figure 1. Fachlicher Kontext des Luddite Launchers

Die fachlichen Schnittstellen umfassen:1

Schnittstelle	**Beschreibung**	**Daten**
Web-App-Zugriff	Kontrolliertes Öffnen einer kuratieren Website	URL der Website
Native-App-Zugriff	Starten einer installierten nativen Anwendung	Intent zum App-Start
App-Liste-Synchronisation	Aktualisieren der verfügbaren Apps vom Backend	Liste der Web-Apps und nativen Apps
Authentifizierung	Benutzeranmeldung und -verwaltung	Benutzername, Passwort, Token
Wunschliste	Einreichung von Vorschlägen für neue Web-Apps	Name, URL, Kommentar

3.2. Technischer Kontext

Der Luddite Launcher ist in eine technische Umgebung eingebettet, die aus mehreren Komponenten besteht.

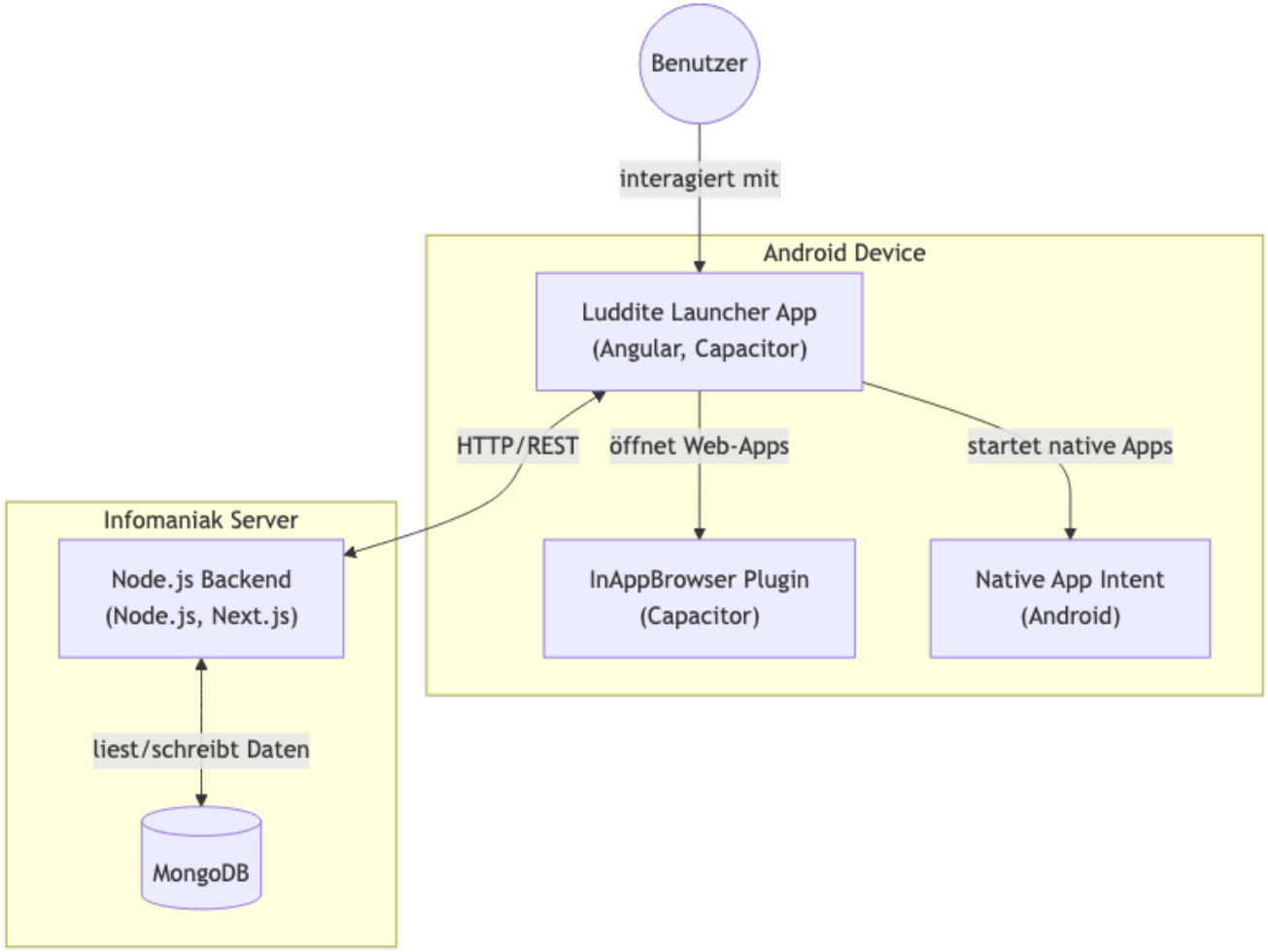


Figure 2. Technischer Kontext des Luddite Launchers

Die technischen Schnittstellen umfassen:

Schnittstelle	**Technologie**	**Beschreibung**
Frontend-Backend-Kommunikation	HTTP/REST-API	Der Launcher kommuniziert mit dem Backend über REST-Endpunkte
Web-App-Anzeige	Capacitor InAppBrowser Plugin	Ermöglicht das kontrollierte Öffnen von Websites
Native-App-Start	Android Intent API	Startet native Anwendungen über Android-Intents
Backend-Datenbank	MongoDB Driver	Verbindung zwischen Node.js Backend und MongoDB

4. Lösungsstrategie

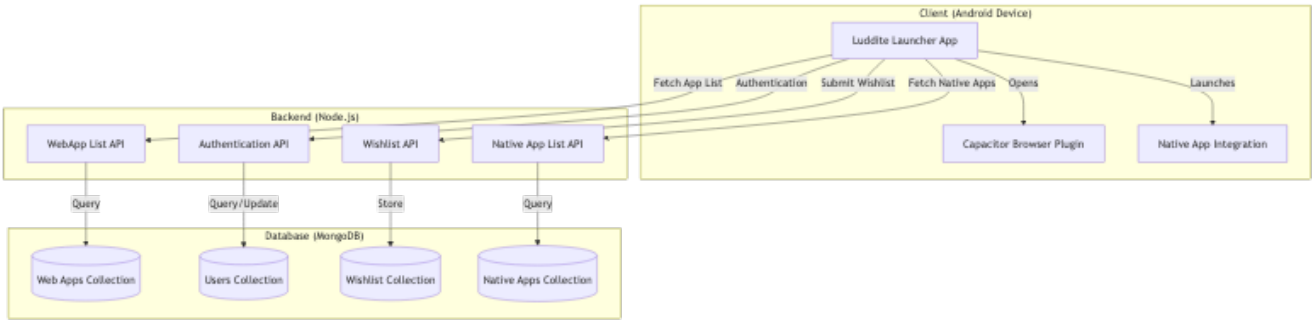
Dieses Kapitel bietet einen kurzen Überblick über die grundlegenden Entscheidungen und Lösungsansätze, die für die Architektur des Luddite Launchers getroffen wurden.

4.1. Technologieentscheidungen

Technologie	Entscheidung	Begründung
Frontend Framework	Angular	Angular bietet ein robustes Framework für die Entwicklung komplexer Single-Page-Anwendungen mit TypeScript. Es unterstütz durch die vorgegebene Struktur Einsteiger beim Aufbau der Applikation. Ausserdem wurde Capacitor ursprünglich für Angular Applikationen entwickelt und funktioniert reibungslos.
Hybride App-Entwicklung	Capacitor	Capacitor ermöglicht die Entwicklung einer Web-App mit nativen Funktionalitäten. Insbesondere das Capacitor Browser Plugin ist entscheidend für das Kernkonzept des Launchers, da es kontrollierte Webansichten ohne Zugriff auf URL-Änderungen ermöglicht.
Backend	Node.js	Node.js bietet eine leichtgewichtige, skalierbare Plattform für die Implementierung des API-Servers. Die Event-driven, nicht-blockierende Architektur eignet sich gut für eine Anwendung mit mehreren gleichzeitigen Benutzern.
Datenbank	MongoDB	Als NoSQL-Datenbank bietet MongoDB Flexibilität bei der Datenspeicherung und eignet sich gut für die Speicherung von App-Definitionen, Benutzereinstellungen und Wunschlisten-Einträgen. Die dokumentenorientierte Struktur passt gut zum JSON-basierten Datenaustausch der App.
Hosting	Openstack Public Cloud	Die Infrastruktur wird auf einem existierenden Infomaniak Server in der Openstack Public Cloud gehostet. Das vereinfacht den Zugriff auf das Backend vom nativen Gerät aus.
Containerisierung	Docker	Docker wird für die Containerisierung des Backends verwendet, was die Konsistenz zwischen Entwicklungs-, Test- und Produktionsumgebungen gewährleistet.

4.2. Architekturansatz

Der Architekturansatz für den Luddite Launcher folgt einem Client-Server-Modell mit einer klaren Trennung zwischen Frontend, Backend und Datenbank:



4.3. Qualitätsziele und ihre Umsetzung

Qualitätsziel	Lösungsansatz	Implementierungsprinzipien
Aktualität der App-Liste	Pull-basierte Synchronisierung	Die App synchronisiert die Liste der verfügbaren Apps bei jedem Start und bietet manuelles Aktualisieren im Einstellungsbereich. Lokale Kopien werden für Offline-Zugriff gespeichert.
Benutzerfreundliche Suche	Suchfunktion	Implementierung einer dynamischen Suchfunktion, die Ergebnisse in Echtzeit filtert, während der Benutzer tippt.
Kontrollierte Web-Nutzung	Capacitor Browser Plugin	Verwendung des Capacitor InAppBrowser Plugins mit eingeschränkten Berechtigungen, die Benutzer daran hindern, URLs zu ändern oder neue Tabs zu öffnen.
Erweiterbarkeit durch Feedback	Wunschlisten-System	Implementierung eines Formulars, mit dem Benutzer neue Web-Apps vorschlagen können. Diese werden in einer separaten MongoDB-Collection gespeichert und können später vom Administrator überprüft werden.
Zugangskontrolle	JWT-basierte Authentifizierung	Verwendung von JSON Web Tokens für die Authentifizierung.
Integration nativer Anwendungen	Capacitor App Launcher	Verwendung des Capacitor App Launcher Plugins, um native Apps zu starten.
Zeitmanagement	Event-basiertes Tracking	Konnte noch nicht umgesetzt werden.

4.4. Organisatorische Entscheidungen

- **Entwicklungsansatz**: Agile Entwicklung
- **Versionsverwaltung**: Git
- **CI/CD**: GitHub Actions für automatisierte Builds und Deployments
- **Dokumentation**: arc42 für die Architekturdokumentation mit CI Pipeline

4.5. Technische Schulden und Risiken

Für die aktuelle Schulprojektphase wurden bewusst folgende Einschränkungen akzeptiert:

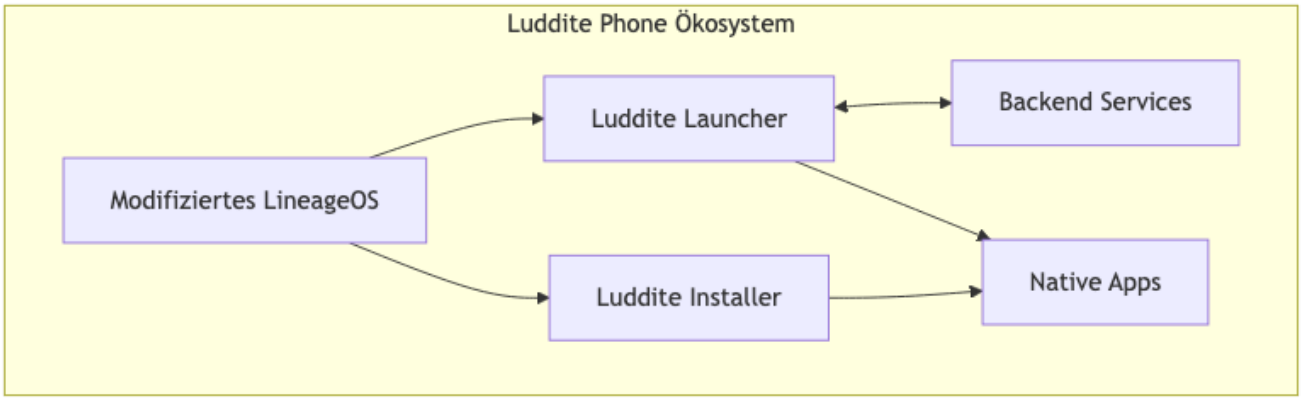
- Die Implementierung der Zeitbegrenzungsfunktion wurde aufgrund von Zeitbeschränkungen zurückgestellt
- Die Benutzeroberfläche konzentriert sich zunächst auf Funktionalität; erweiterte Anpassungsoptionen sind für spätere Versionen vorgesehen
- Die Passwörter werden im Klartext auf der MongoDB gespeichert. Aus Zeitgründen konnte das Authentifizierungsverfahren nur vereinfacht umgesetzt werden.
- Die Kommunikation mit dem Server läuft via http, nicht https. Zurzeit wird der Server via IP Adresse und nicht einer Domain angesprochen was das installieren von Zertifikaten erschwert.

5. Bausteinsicht

1. 5.1 Whitebox Gesamtsystem

Der Luddite Launcher ist als spezialisierte Benutzeroberfläche (Launcher) für ein modifiziertes Android-System konzipiert, das darauf abzielt, digitale Ablenkungen zu reduzieren. Er arbeitet zusammen mit anderen Komponenten im "Luddite Phone"-Ökosystem, um ein fokussiertes digitales Erlebnis ohne uneingeschränkten Web-Zugang zu bieten.

1.1. Übersichtsdiagramm



1.2. Motivation

Der Luddite Launcher dient als primäre Benutzeroberfläche für die ablenkungsfreie Smartphone-Umgebung. Durch die Kontrolle darüber, welche Anwendungen und Web-Ressourcen zugänglich sind, hilft er Benutzern, ihren digitalen Konsum zu verwalten und die Abhängigkeit von sozialen Medien zu reduzieren.

1.3. Enthaltene Bausteine

Baustein	Beschreibung
Luddite Launcher (Angular + Capacitor)	Die Hauptanwendung, die die Benutzeroberfläche für den Zugriff auf zugelassene Web-Anwendungen und native Apps bereitstellt
Backend-Dienste (Node.js)	Stellt Authentifizierung, App-Listen bereit und speichert Benutzereinstellungen und App-Vorschläge
Native Apps-Integration	Integrationsschicht für den Zugriff auf installierte Android-Anwendungen

Baustein	Beschreibung
Web-Apps-Verwaltung	Steuert den Zugriff auf vorab genehmigte Web-Anwendungen durch eine eingeschränkte Browser-Umgebung

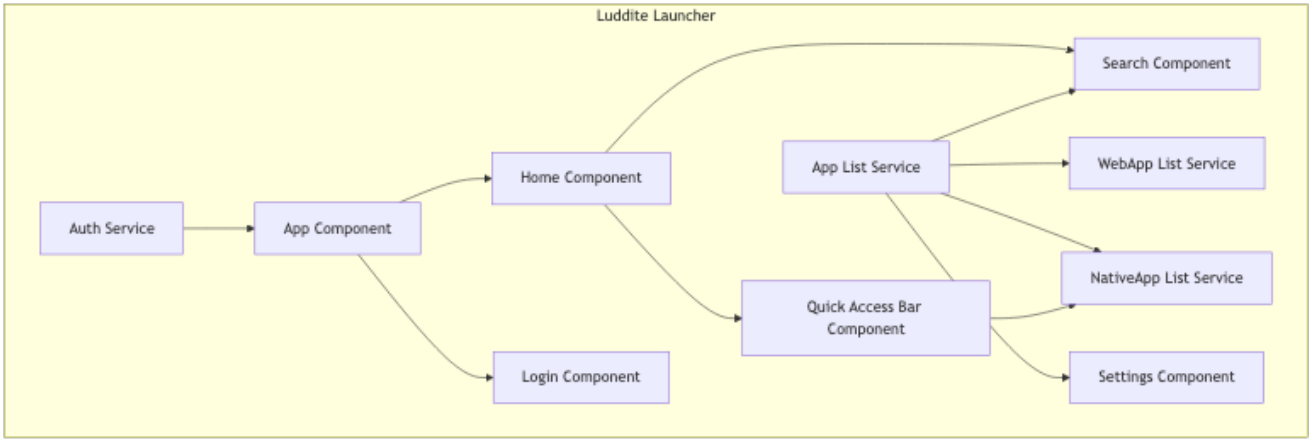
1.4. Wichtige Schnittstellen

Schnittstelle	Beschreibung
Backend-API	REST-API für Authentifizierung, Abrufen von App-Listen und Einreichen von App-Vorschlägen
Capacitor Browser Plugin	Ermöglicht das Öffnen von kontrollierten WebViews für zugelassene Websites
App Launcher API	Ermöglicht den Start von nativen Apps über das Android-System
Preferences Storage	Speichert Benutzereinstellungen und Zustände lokal auf dem Gerät

2. 5.2 Ebene 2: Whitebox Luddite Launcher

Der Luddite Launcher ist der zentrale Baustein des Systems und besteht aus mehreren Angular-Komponenten und Diensten, die zusammenarbeiten, um die Hauptfunktionalitäten bereitzustellen.

2.1. Whitebox Luddite Launcher



2.1.1. App Component

Verantwortlichkeiten: - Dient als Root-Komponente der Anwendung - Verwaltet den Authentifizierungsstatus - Entscheidet, ob Login-Komponente oder Home-Komponente angezeigt wird

Schnittstellen: - Interagiert mit dem Auth Service - Rendert entweder Login- oder Home-Komponente

2.1.2. Home Component

Verantwortlichkeiten: - Dient als Haupt-Dashboard nach der Anmeldung - Stellt den Hintergrund und das Layout bereit - Integriert Search- und Quick Access Bar-Komponenten

Schnittstellen: - Bindet die Search-Komponente ein - Bindet die Quick Access Bar-Komponente ein

2.1.3. Search Component

Verantwortlichkeiten: - Ermöglicht die Suche nach verfügbaren Web- und nativen Apps - Zeigt Suchergebnisse an - Öffnet ausgewählte Apps über entsprechende Plugins

Schnittstellen: - Nutzt App List Service für Datenzugriff - Verwendet Capacitor InAppBrowser für Web-Apps - Verwendet App Launcher für native Apps

2.1.4. Login Component

Verantwortlichkeiten: - Stellt Benutzeranmeldung und Registrierung bereit - Verwaltet Anmelde- und Registrierungsformulare - Zeigt Authentifizierungsfehler an

Schnittstellen: - Nutzt Auth Service für Anmeldung und Registrierung

2.1.5. Settings Component

Verantwortlichkeiten: - Ermöglicht Bearbeitung von Benutzereinstellungen - Bietet Formular für App-Vorschläge - Erlaubt Daten-Synchronisation mit Backend

Schnittstellen: - Nutzt App List Service für Daten-Synchronisation - Verwendet Backend-API für Vorschlagseinreichungen

2.1.6. Quick Access Bar Component

Verantwortlichkeiten: - Zeigt am häufigsten verwendete Apps an - Ermöglicht schnellen Zugriff ohne Suche - Organisiert Apps nach Priorität

Schnittstellen: - Nutzt NativeApp List Service für App-Daten

2.2. Wichtige Dienste

2.2.1. Auth Service

Verantwortlichkeiten: - Verwaltet Benutzerauthentifizierung - Speichert und aktualisiert Auth-Token - Überprüft Authentifizierungsstatus

Schnittstellen: - Backend-API-Endpunkte für Anmeldung/Registrierung - Capacitor Preferences für Token-Speicherung

2.2.2. App List Service

Verantwortlichkeiten: - Kombiniert Web-Apps und Native-Apps in einer einheitlichen Liste - Stellt Observable-Streams für App-Daten bereit - Synchronisiert App-Daten mit Backend

Schnittstellen: - WebApp List Service und NativeApp List Service für Datenbezug - Backend-API für Datensynchronisation

2.2.3. WebApp List Service und NativeApp List Service

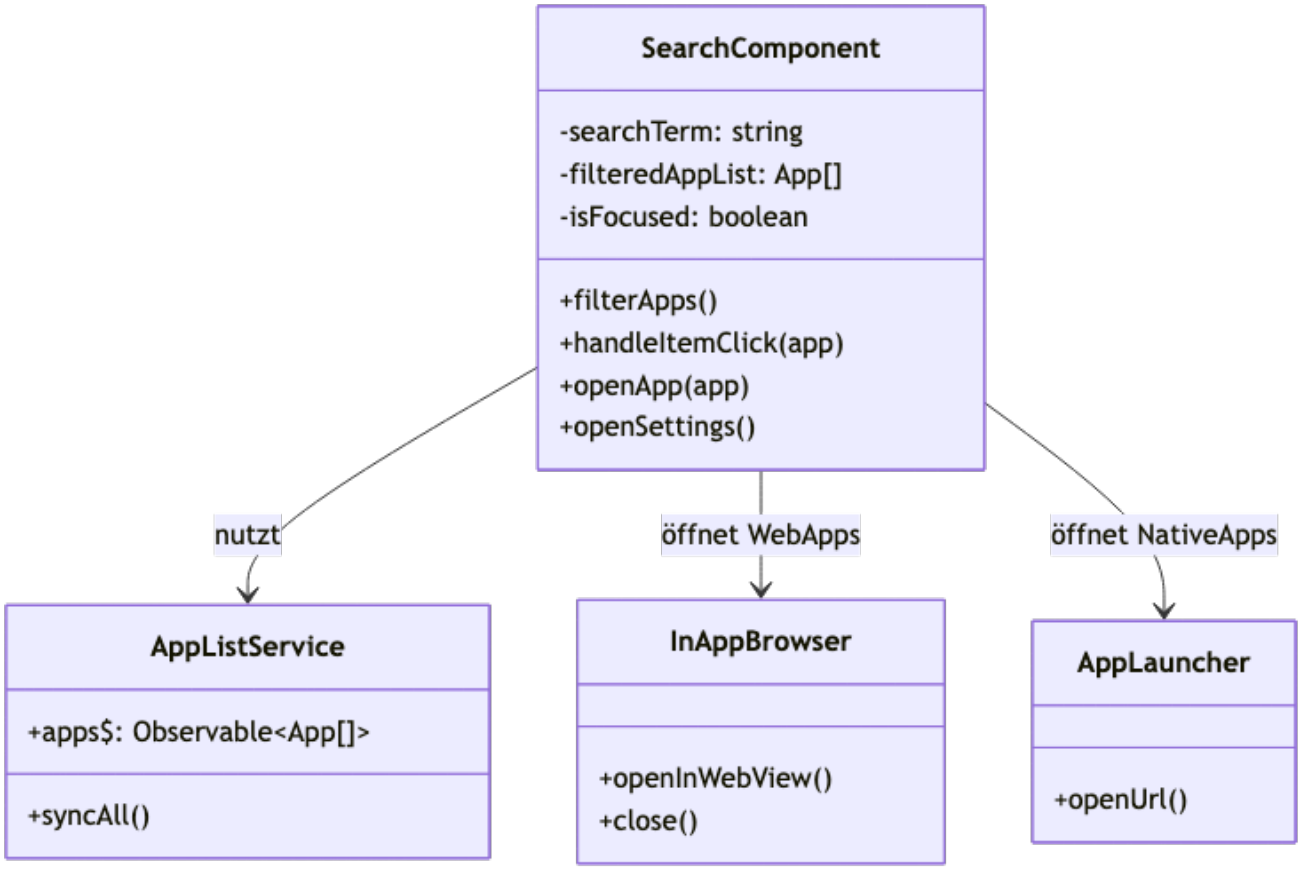
Verantwortlichkeiten: - Verwalten spezifische App-Typen (Web oder Native) - Speichern App-Daten lokal - Synchronisieren mit Backend-Server

Schnittstellen: - Erben von BaseApp Service für gemeinsame Funktionalität - Backend-API für spezifische App-Typen - Capacitor Preferences für lokale Datenspeicherung

3. 5.3 Ebene 3: Zentrale Komponenten im Detail

3.1. Whitebox Search Component

Die Search-Komponente ist eine der wichtigsten Schnittstellen für Benutzer und verwaltet die Suche und den Zugriff auf Apps.

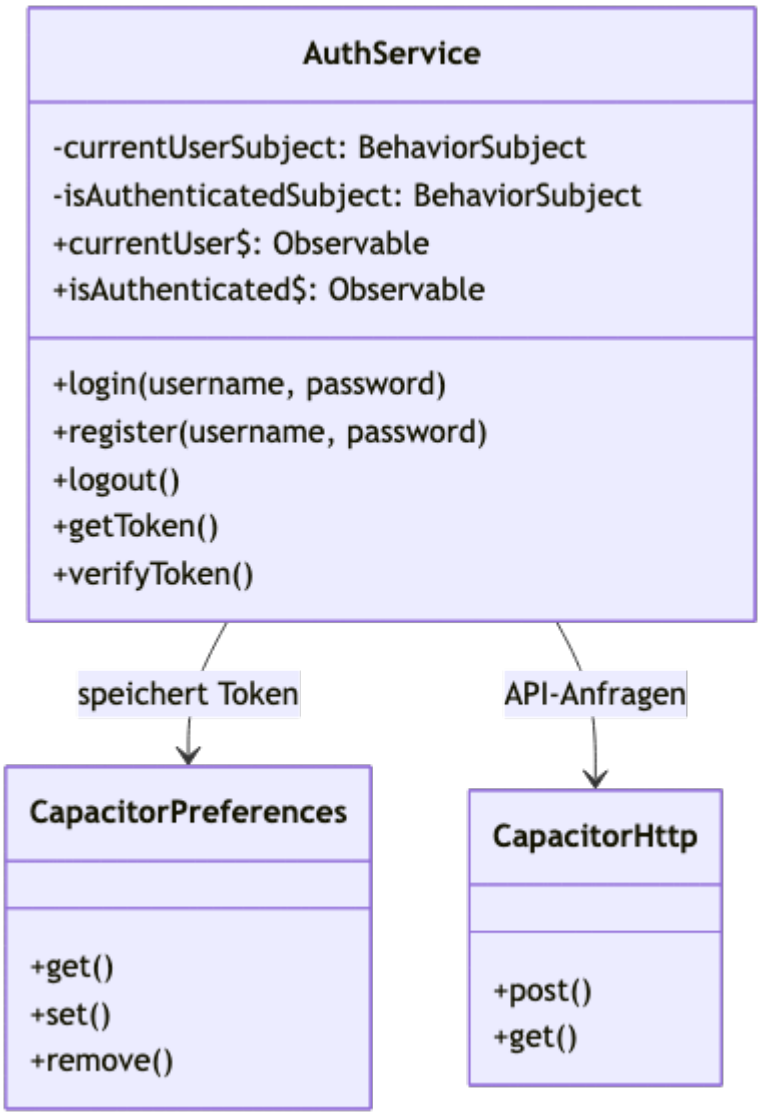


Prozessablauf beim App-Start:

1. Der Benutzer gibt einen Suchbegriff in das Suchfeld ein
2. Die `filterApps()`-Methode wird bei jeder Eingabe aufgerufen
3. Die gefilterten Ergebnisse werden angezeigt
4. Bei Klick auf ein Ergebnis wird `handleItemClick(app)` aufgerufen
5. Je nach App-Typ:
    - Bei Web-Apps: Öffnen durch `InAppBrowser.openInWebView()`
    - Bei nativen Apps: Öffnen durch `AppLauncher.openUrl()`
    - Bei Settings: Navigation zur Settings-Komponente

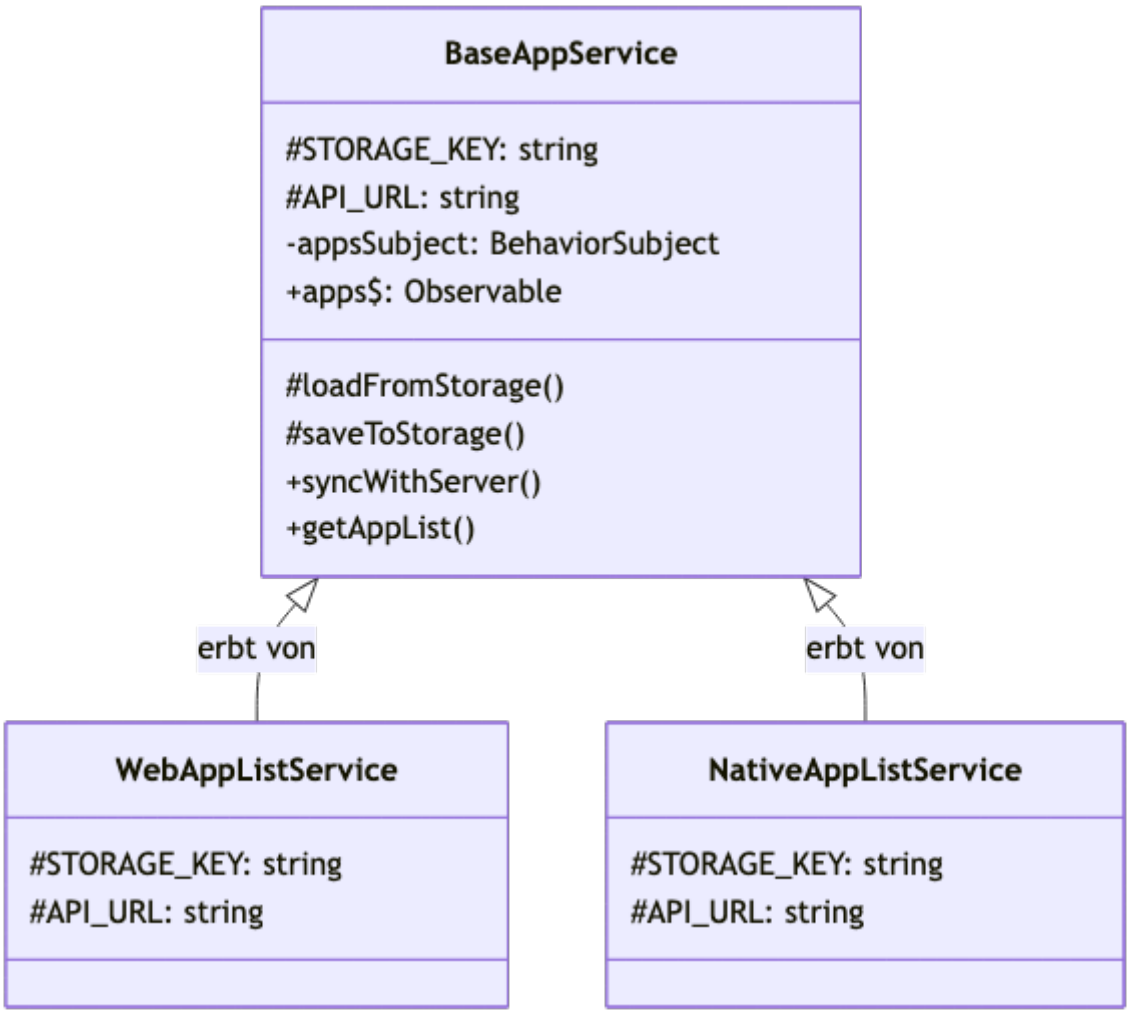
3.2. Whitebox Auth Service

Der Auth Service verwaltet den gesamten Authentifizierungsprozess und ist entscheidend für die Zugriffskontrolle.



3.3. Whitebox WebApp und NativeApp Services

Die App-Services verwalten den Zugriff auf verschiedene App-Typen und teilen gemeinsame Funktionalität über den BaseApp Service.



4. 5.4 Technische Schnittstellen

4.1. Capacitor Plugins als Brücke zum nativen System

Die Anwendung verwendet verschiedene Capacitor-Plugins, um auf native Funktionen zuzugreifen:

Plugin	Zweck
`@capacitor/inappbrowser`	Ermöglicht kontrolliertes Öffnen von Websites in einer WebView
`@capacitor/app-launcher`	Startet native Apps über ihre URI-Schemata
`@capacitor/preferences`	Speichert und verwaltet Benutzereinstellungen und App-Daten
`@capacitor/http`	Führt HTTP-Anfragen an Backend-Dienste durch

4.2. Backend-Schnittstellen

Die Backend-Dienste bieten verschiedene API-Endpunkte:

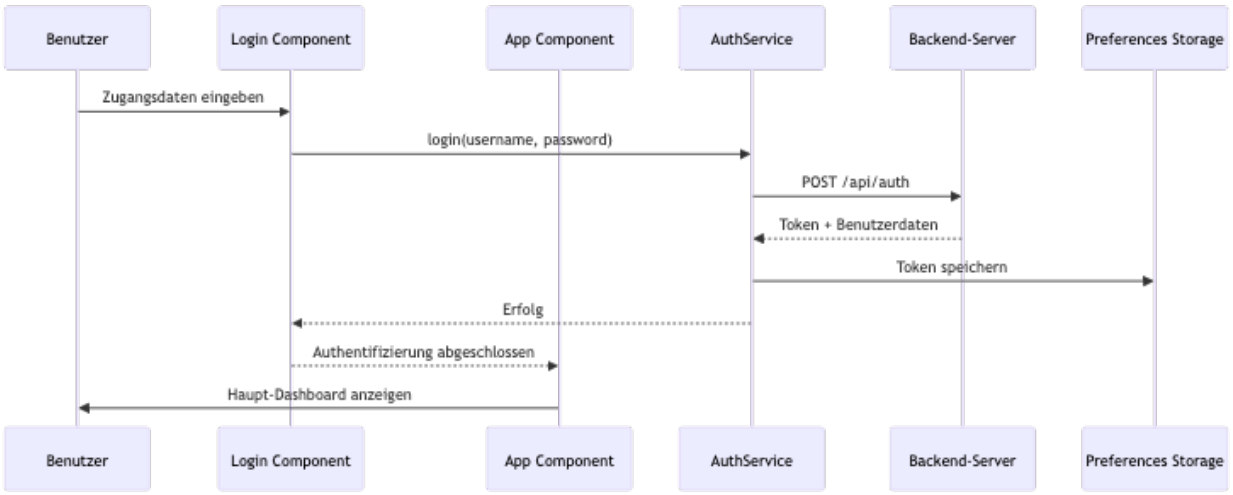
Endpunkt	Beschreibung
`/api/auth`	Authentifizierung (Login) und Token-Verifizierung
`/api/auth/register`	Benutzerregistrierung
`/api/webapps`	Abfrage von verfügbaren Web-Apps
`/api/nativeapps`	Abfrage von Liste mit nativen Apps
`/api/wishlist`	Einreichen von App-Vorschlägen

6. Laufzeitsicht

Die Laufzeitsicht zeigt das Zusammenspiel der Bausteine des Luddite Launchers zur Laufzeit. Die folgenden Szenarien zeigen die wichtigsten Abläufe im System.

1. 6.1 Authentifizierung eines Benutzers

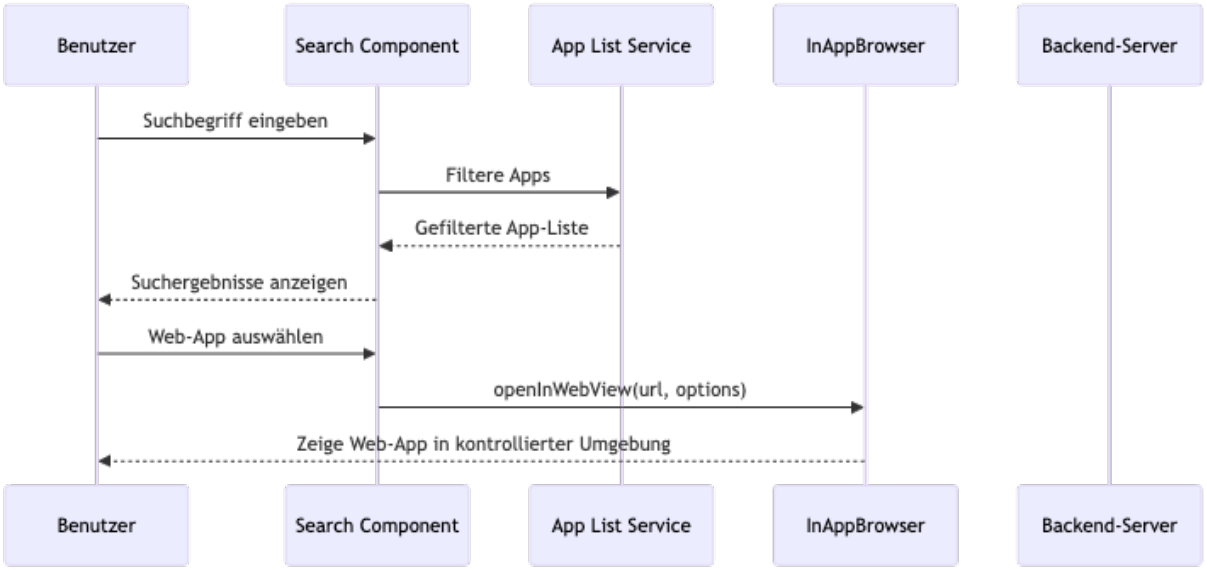
Dieses Szenario zeigt den Anmeldevorgang eines Benutzers und die anschliessende Initialisierung der App.



Das Authentifizierungsszenario ist besonders wichtig, da der Luddite Launcher nur für angemeldete Benutzer zugänglich ist. Nach erfolgreicher Anmeldung wird der Token lokal gespeichert und für alle nachfolgenden API-Anfragen verwendet.

2. 6.2 Suchen und Öffnen einer Web-App

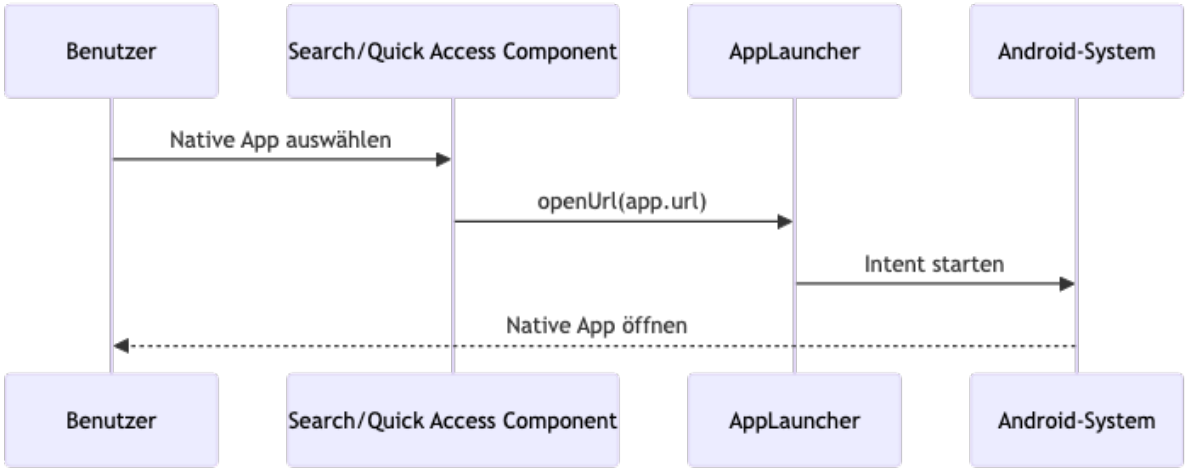
Dieses Szenario beschreibt den Prozess des Suchens und Öffnens einer Web-App, was eine der Kernfunktionen des Launchers darstellt.



Dieses Szenario zeigt, wie der Benutzer über die Suche auf Web-Apps zugreifen kann. Die Search Component filtert die Apps basierend auf dem Suchbegriff und präsentiert die Ergebnisse. Nach Auswahl einer Web-App wird diese in einer kontrollierten Browser-Umgebung geöffnet, wobei der Benutzer die URL nicht ändern kann.

3. 6.3 Starten einer nativen App

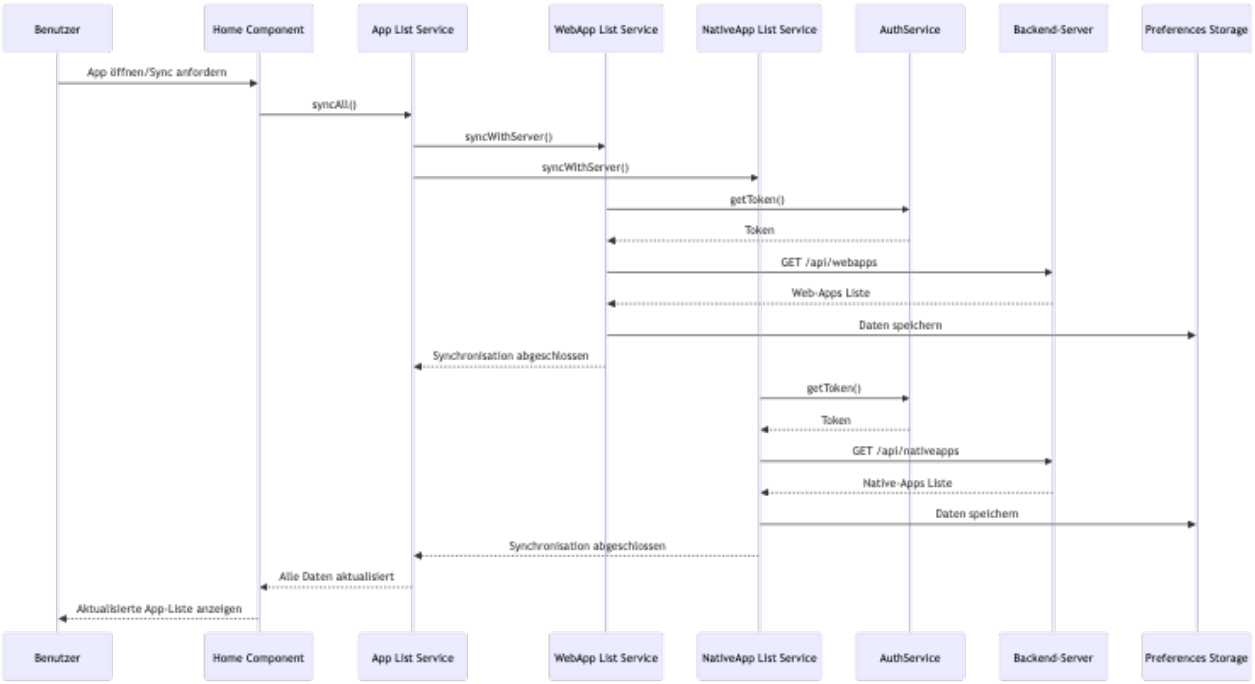
Dieses Szenario zeigt, wie der Benutzer eine native Android-App über den Launcher öffnen kann.



Der Launcher ermöglicht den direkten Zugriff auf installierte native Apps wie WhatsApp oder Proton Mail, ohne dass der Benutzer auf den Standard-App-Drawer des Android-Systems zugreifen muss. Dies erfolgt über die Capacitor AppLauncher-API, die einen Intent an das Android-System sendet.

4. 6.4 Synchronisation der App-Daten mit dem Backend

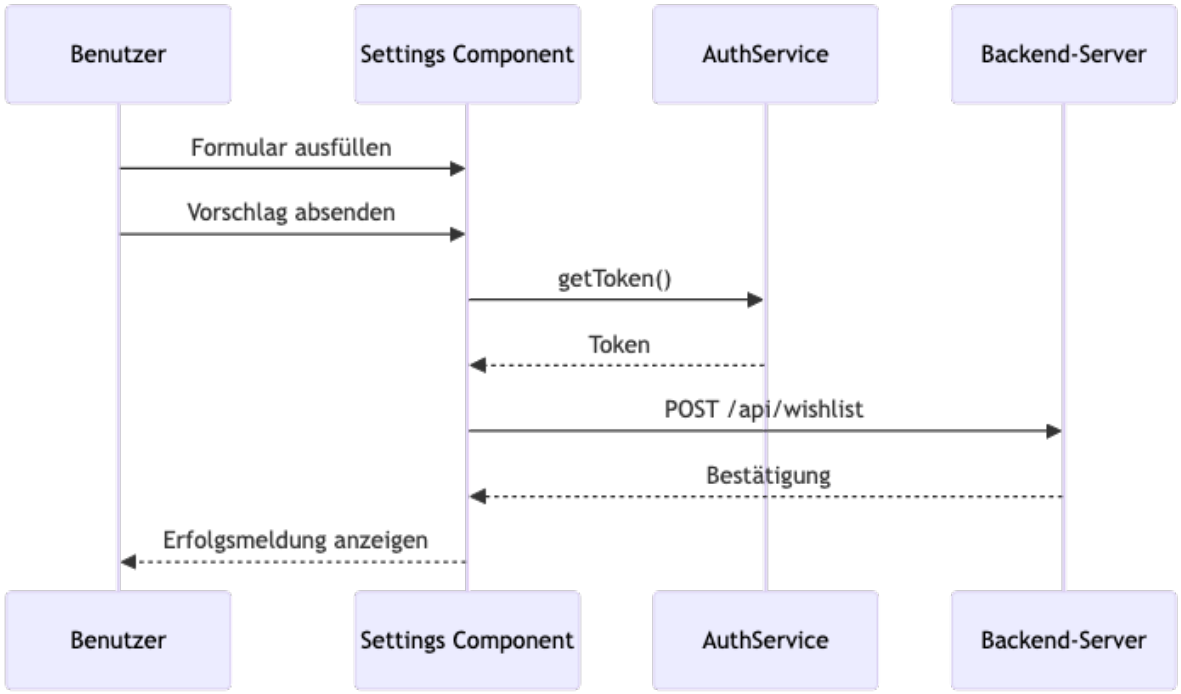
Dieses Szenario beschreibt, wie der Launcher die Liste der verfügbaren Apps beim Start oder auf Benutzeranfrage mit dem Backend synchronisiert.



Die Synchronisation stellt sicher, dass der Benutzer stets auf die aktuelle Liste der verfügbaren Apps zugreifen kann. Dies ist besonders wichtig, da neue Web-Apps vom Administrator im Backend hinzugefügt werden können und dem Benutzer ohne App-Update zur Verfügung stehen sollen.

5. 6.5 Einreichen eines App-Vorschlags

Dieses Szenario zeigt, wie ein Benutzer einen Vorschlag für eine neue Web-App einreichen kann.



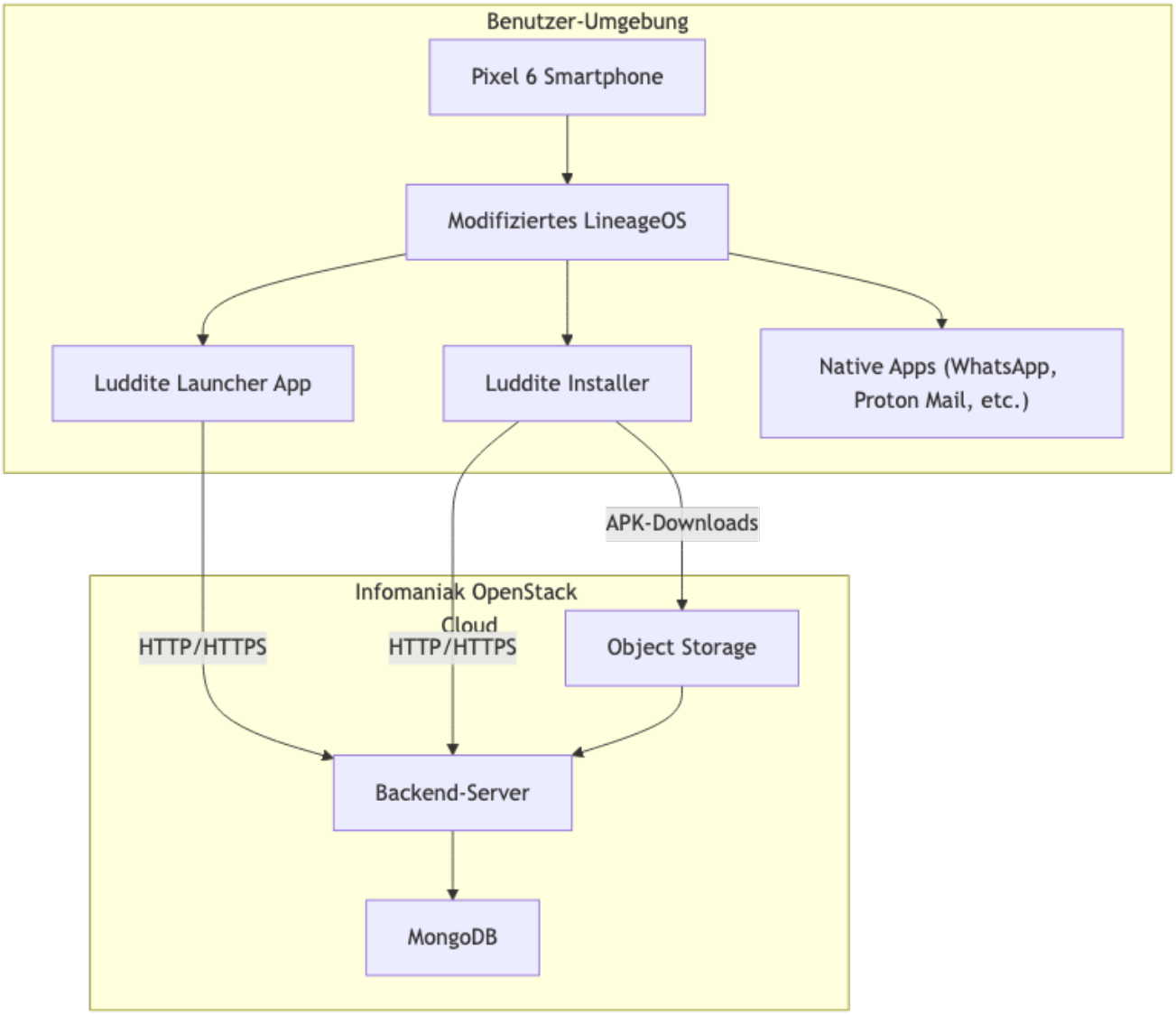
Dieses Feature ermöglicht es Benutzern, aktiv an der Erweiterung des App-Angebots teilzunehmen, indem sie Vorschläge für neue Web-Apps einreichen können. Die Vorschläge werden in einer separaten Collection in der MongoDB gespeichert und können vom Administrator geprüft und gegebenenfalls implementiert werden.

6. 7. Verteilungssicht

6.1. 7.1 Infrastruktur Ebene 1

Die Verteilungssicht zeigt die Infrastruktur, auf der das Luddite Launcher-System betrieben wird. Diese umfasst den Luddite Launcher (Frontend), das Backend und die Datenbank.

6.1.1. Übersichtsdiagramm



6.1.2. Motivation

Die Verteilungsarchitektur wurde so gestaltet, dass sie den Hauptzielen des Projekts entspricht, insbesondere der Schaffung einer kontrollierten Smartphone-Umgebung, die die Nutzung sozialer Medien einschränkt und gleichzeitig wichtige Funktionen beibehält. Die Infrastruktur unterstützt:

1. Ein modifiziertes Android-Betriebssystem ohne nativen Browser
2. Einen benutzerfreundlichen Launcher mit kontrolliertem Web-Zugang
3. Zentralisierte Verwaltung von zulässigen Web-Apps

4. Sichere Verteilung von geprüften nativen Apps
5. Künftige Implementierung von Nutzungszeitbeschränkungen

6.1.3. Qualitäts- und Leistungsmerkmale

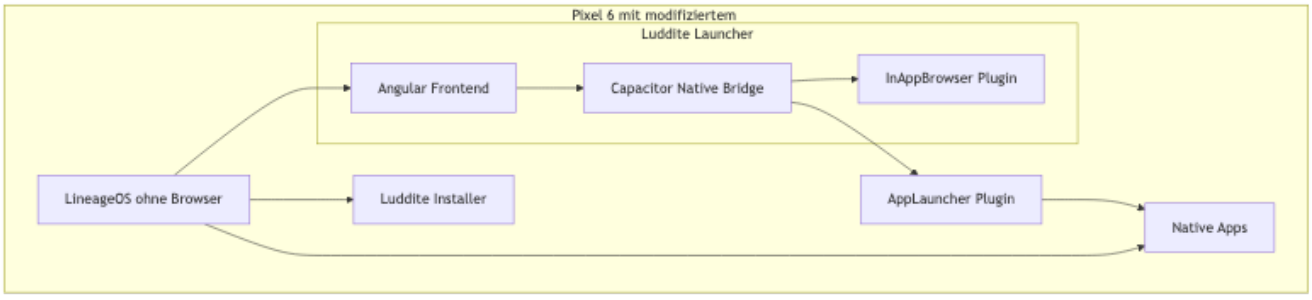
- **Verfügbarkeit**: Die Backend-Infrastruktur in der Infomaniak OpenStack Cloud gewährleistet hohe Verfügbarkeit für die App-Listen und Authentifizierungsdienste.
- **Sicherheit**: Die zentrale Verwaltung erlaubt strikte Kontrolle darüber, welche Apps installiert werden können.
- **Skalierbarkeit**: Die Cloud-Infrastruktur kann mit wachsender Benutzerbasis skaliert werden.
- **Netzwerkbandbreite**: Die Kommunikation zwischen App und Backend beschränkt sich hauptsächlich auf kleine Datenpakete (Authentifizierung, App-Listen), mit Ausnahme der gelegentlichen APK-Downloads.

6.1.4. Zuordnung der Bausteine zu Infrastruktur

Baustein	Infrastrukturelement
Luddite Launcher	Pixel 6 mit modifiziertem LineageOS
Luddite Installer	Pixel 6 mit modifiziertem LineageOS
Native Apps	Pixel 6 mit modifiziertem LineageOS
Backend API	Node.js Server in Infomaniak OpenStack Cloud
App-Datenbank	MongoDB in Infomaniak OpenStack Cloud
APK-Speicher	Object Storage in Infomaniak OpenStack Cloud

6.2. 7.2 Infrastruktur Ebene 2

6.2.1. Smartphone-Umgebung



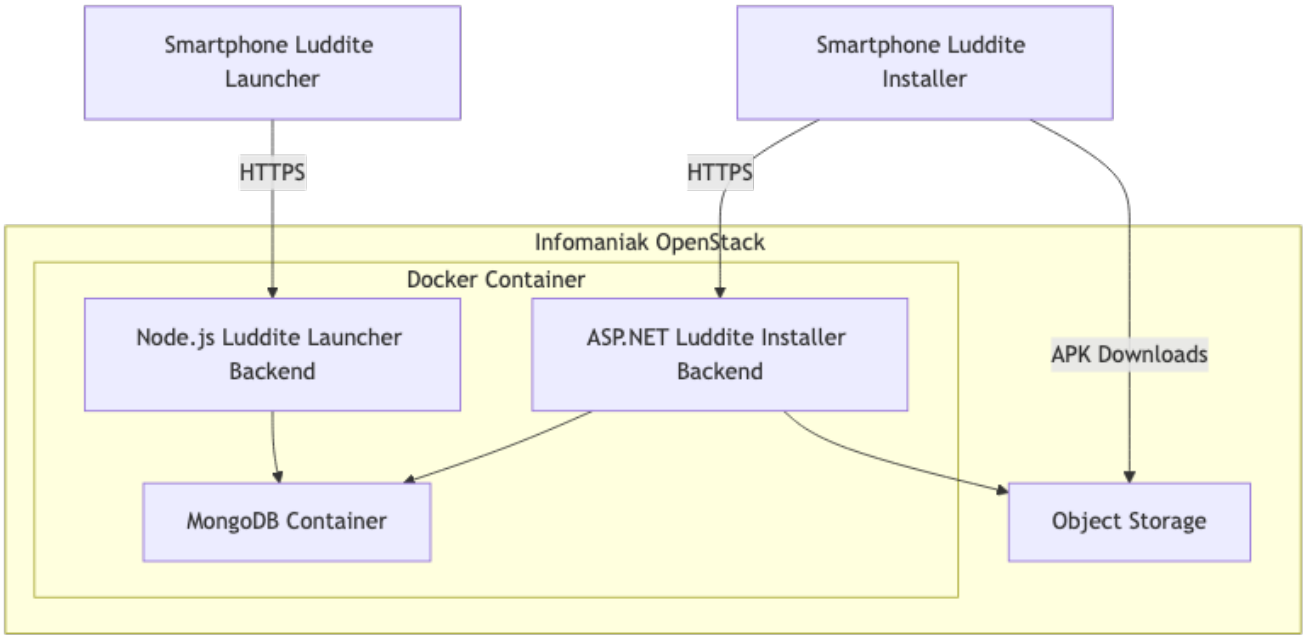
Der Luddite Launcher läuft auf einem Pixel 6 Smartphone mit einer angepassten LineageOS-Version, bei der der standardmässige Webbrowser entfernt wurde. Der Launcher selbst ist eine Angular-Anwendung, die mittels Capacitor auf native Funktionen zugreifen kann.

Besonderheiten:

- Das InAppBrowser-Plugin ermöglicht kontrollierten Zugriff auf ausgewählte Webseiten

- Das AppLauncher-Plugin ermöglicht den Start nativer Apps
- Die modifizierte LineageOS-Version verhindert die Installation nicht-autorisierter Apps (mit Ausnahme über ADB, was in zukünftigen Versionen weiter eingeschränkt werden soll)
- Der Luddite Installer ist die einzige Möglichkeit, autorisierte Apps zu installieren

6.2.2. Backend-Infrastruktur



Das Backend wird in der Infomaniak OpenStack Cloud gehostet und besteht aus Docker-Containern:

6.3. 7.3 Technische Infrastruktur

6.3.1. Hardware-Anforderungen

Smartphone:

- Pixel 6 oder LineageOS 22.1 kompatibles Gerät

Backend:

- 1 vCPU
- 4 GB RAM
- 20 GB SSD Speicher
- Object Storage mit 10 GB (erweiterbar)

6.3.2. Software-Umgebung

Smartphone:

- Modifiziertes LineageOS ohne Browser-Komponenten
- Android API Level 26+

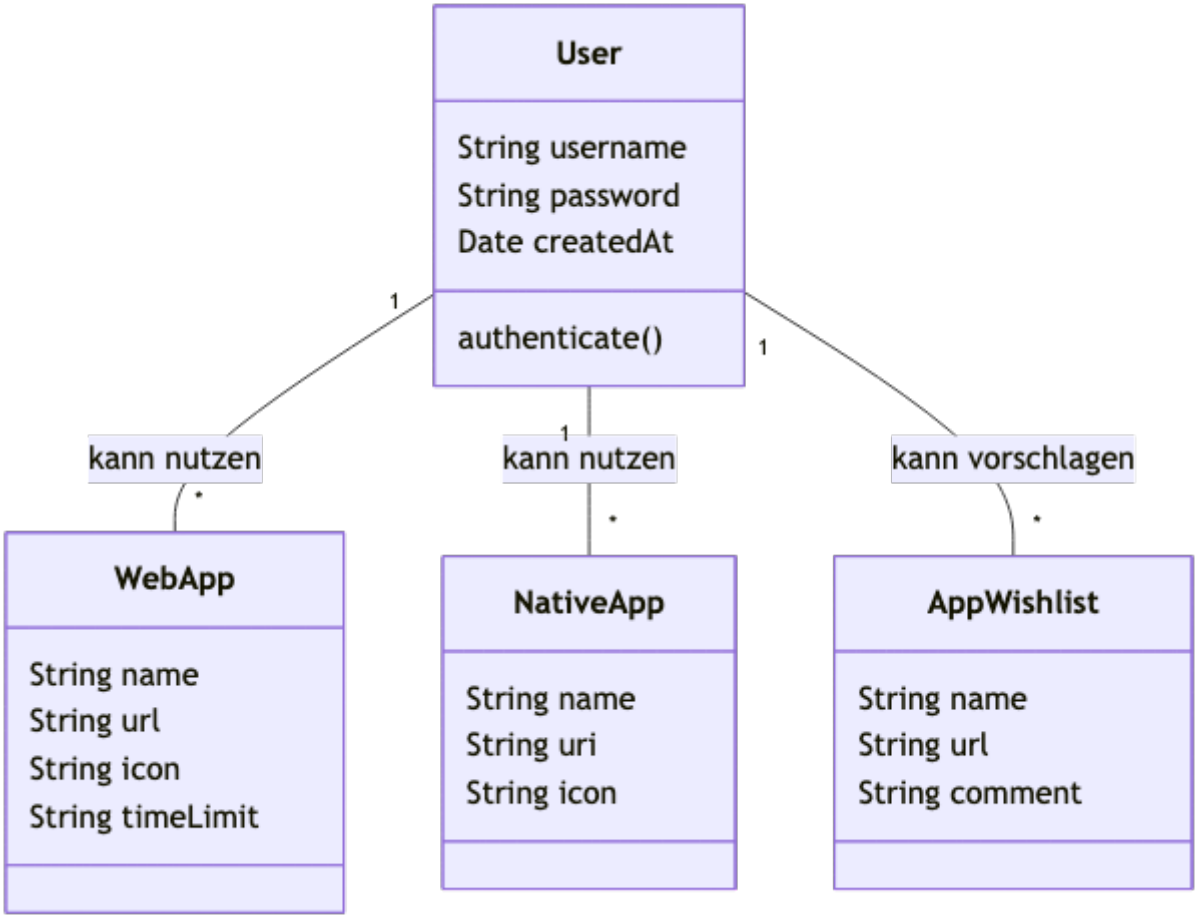
Backend:

- Docker Containers
- Node.js 20.x
- MongoDB 6.x
- OpenStack-basierte Cloud-Infrastruktur

7. 8. Querschnittliche Konzepte

7.1. 8.1 Domänenmodell

Das Domänenmodell des Luddite Launcher Systems umfasst die folgenden Kernelemente:



Diese Entitäten bilden die Grundlage des Domänenmodells und sind in der MongoDB gespeichert.

7.2. 8.2 Nutzererfahrung und UI-Konzept

Der Luddite Launcher setzt auf eine minimalistische, nicht ablenkende Benutzeroberfläche, die sich an folgenden Prinzipien orientiert:

- **Funktionsorientiert statt unterhaltungsorientiert**: Die Oberfläche präsentiert Tools und nützliche Anwendungen, nicht Unterhaltungs- oder Social-Media-Inhalte.
- **Reduzierte visuelle Reize**: Einfache Farbpalette, reduzierte Animationen, keine Benachrichtigungsbadges für Social-Media-Apps.
- **Fokussierte Suche**: Zentrales Suchfeld als Haupteinstiegspunkt zur App-Nutzung.
- **Bewusste Nutzung fördern**: Zukünftig geplante Zeitlimits werden visuell dargestellt, um Nutzer zur bewussten Verwendung anzuregen.

7.3. 8.3 Sicherheitskonzept

Das Sicherheitskonzept des Luddite Launcher Systems umfasst folgende Aspekte:

7.3.1. 8.3.1 Authentifizierung und Autorisierung

- JWT-basierte Authentifizierung für die Kommunikation zwischen App und Backend
- Passwortgeschützte Benutzerkonten zur Personalisierung der Einstellungen
- Sichere Token-Speicherung mit Capacitor Preferences API

7.3.2. 8.3.2 Systemsicherheit

- Eingeschränkter Browser-Zugriff nur auf definierte URLs
- Verhinderung von URL-Änderungen im In-App-Browser
- Installation neuer Apps nur über den kontrollierten Luddite Installer (mit Ausnahme von ADB, was in zukünftigen Versionen weiter eingeschränkt werden soll)

7.4. 8.4 Persistenzkonzept

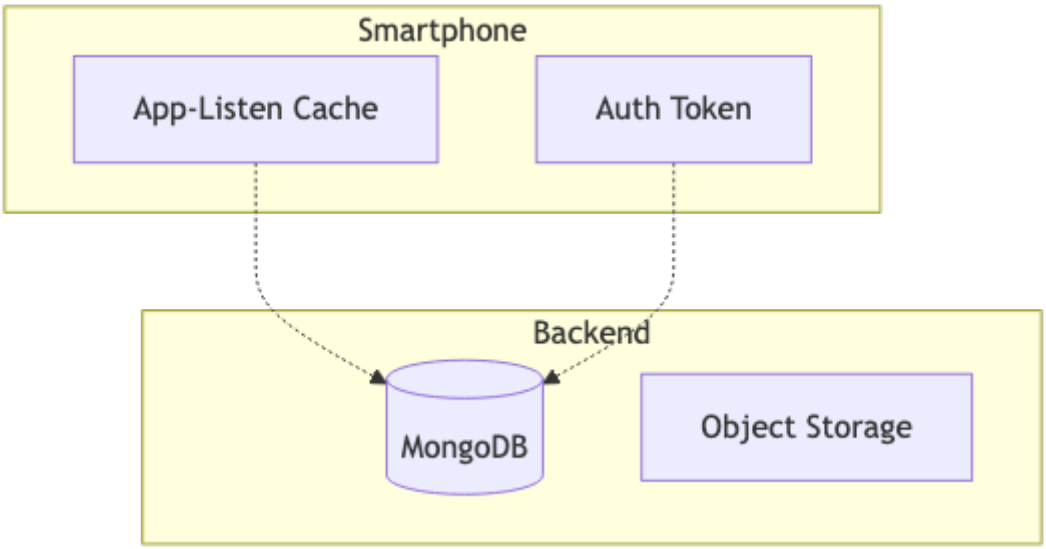
Die Persistenz im Luddite Launcher System ist auf verschiedenen Ebenen implementiert:

7.4.1. 8.4.1 Server-seitige Persistenz

- MongoDB-Datenbank für Benutzerdaten, App-Kataloge und Berechtigungen
- Kollektionen für:
  - Benutzer (users)
  - Web-Apps (webapps)
  - Native Apps (nativeapps)
  - App-Wünsche (app_wishlist)
- Object Storage für die Speicherung von APK-Dateien

7.4.2. 8.4.2 Client-seitige Persistenz

- Lokales Caching der App-Listen mittels Capacitor Preferences API
- Speicherung von Authentifizierungstokens
- Pufferung von App-Daten für Offline-Funktionalität



7.5. 8.5 Fehlerbehandlungs- und Erholungskonzept

Das System implementiert folgende Strategien zur Fehlerbehandlung:

- **Offline-Funktionalität**: Bei Verbindungsabbrüchen kann der Launcher weiterhin auf gecachte App-Listen zugreifen
- **Fehlerprotokollierung**: Kritische Fehler werden im Backend protokolliert

7.6. 8.6 Testkonzept

Für die aktuelle Entwicklungsphase des Luddite Launchers wurde ein pragmatischer Testansatz gewählt:

- **Manuelle Tests**: Aufgrund der überschaubaren Anzahl an Funktionen werden aktuell manuelle Tests durchgeführt. Diese umfassen:
  - Funktionelle Tests der Kernfunktionen (Suche, App-Öffnung, Web-App-Darstellung)
  - UI/UX-Tests für verschiedene Ansichten
  - Integrationstests mit den Backend-Services
- **Geplante Teststrategie**: Für zukünftige Versionen ist der Ausbau der Teststrategie geplant:
  - Unit-Tests für Angular-Komponenten und Services
  - Automatisierte End-to-End-Tests mit Cypress oder ähnlichen Frameworks
  - Continuous Integration Testing mit GitHub Actions

Das aktuelle manuelle Testverfahren ist für den Projektumfang angemessen, wird jedoch mit wachsender Funktionalität und Nutzerbasis entsprechend erweitert werden.

7.7. 8.7 Build- und Deployment-Konzept

Das Build- und Deployment-System des Luddite Launcher nutzt folgende Ansätze:

7.7.1. 8.7.1 Frontend-Build

- Angular-basierter Build-Prozess
- Capacitor für die native App-Konvertierung
- GitHub Actions für CI/CD

7.7.2. 8.7.2 Backend-Deployment

- Docker-Container für die verschiedenen Backend-Komponenten
- Docker-Compose für den Server

8. 9. Architekturentscheidungen

In diesem Kapitel werden wichtige Architekturentscheidungen dokumentiert, die für das Luddite Launcher-Projekt getroffen wurden. Diese Entscheidungen haben signifikante Auswirkungen auf die Gesamtarchitektur und den Erfolg des Projekts.

8.1. 9.1 Verwendung von Angular und Capacitor

8.1.1. Kontext

Für die Entwicklung des Luddite Launcher benötigten wir ein Framework, das sowohl eine moderne Web-Entwicklung als auch eine nahtlose Integration mit nativen Mobilgerätefunktionen ermöglicht.

8.1.2. Entscheidung

Wir haben uns für Angular als Frontend-Framework in Kombination mit Capacitor als native Bridge entschieden.

8.1.3. Begründung

- Angular bietet ein robustes, komponenten-basiertes Framework mit starkem Fokus auf Typsicherheit durch TypeScript
- Capacitor ermöglicht die Wiederverwendung von Web-Entwicklungskenntnissen bei gleichzeitiger Nutzung nativer Funktionen
- Die Kombination ermöglicht eine schnelle Entwicklung bei gleichzeitiger nativer Performanz auf dem Gerät
- Angular bietet eine gute Unterstützung für State Management und Services, was für die Verwaltung der App-Listen und Benutzereinstellungen wichtig ist
- Frühere Erfahrungen mit Vue.js haben gezeigt, dass ein stärker strukturiertes Framework für diesen Anwendungsfall vorteilhafter ist

8.1.4. Alternativen

- React Native: Hätte ähnliche Vorteile geboten, aber in der Blockwoche wurde bereits Angular behandelt
- Kotlin/Java (native Android-Entwicklung): Hätte potenziell bessere Performanz geboten, aber hätte deutlich mehr Entwicklungsaufwand bedeutet & wäre auch nicht Teil der Blockwoche gewesen.
- Flutter: War vielversprechend, aber die Integrationsmöglichkeiten mit bestehendem Web-Code waren limitierter

8.2. 9.2 Modifizierte LineageOS-Version ohne Browser

8.2.1. Kontext

Um das Ziel zu erreichen, Social-Media-Ablenkungen zu reduzieren, musste der uneingeschränkte Internetzugang verhindert werden.

8.2.2. Entscheidung

Entwicklung einer modifizierten LineageOS-Version für das Pixel 6, bei der der Standard-Browser vollständig entfernt wurde.

8.2.3. Begründung

- LineageOS bietet als Open-Source-Android-Distribution die Möglichkeit, tiefgreifende Modifikationen vorzunehmen
- Die vollständige Entfernung des Browsers verhindert Umgehungslösungen, die bei einfachen App-Blockern möglich wären
- Dies ermöglicht eine kontrollierte Umgebung, in der nur ausgewählte Web-Apps über den Luddite Launcher zugänglich sind
- LineageOS bietet gute Unterstützung für Pixel-Geräte und regelmässige Sicherheitsupdates

8.2.4. Alternativen

- Nutzung der Android Enterprise-Lösung: Hätte ähnliche Einschränkungen ermöglicht, wäre jedoch mit höheren Lizenzkosten verbunden gewesen
- Entwicklung eines Custom ROMs von Grund auf: Zu hoher Entwicklungsaufwand
- App-basierte Beschränkungen: Wären leichter zu umgehen gewesen

8.3. 9.3 Capacitor InAppBrowser für kontrollierte Web-Apps

8.3.1. Kontext

Es wird eine Lösung benötigt, um ausgewählte Webseiten zugänglich zu machen, ohne einen vollwertigen Browser anzubieten.

8.3.2. Entscheidung

Verwendung des Capacitor InAppBrowser-Plugins für den kontrollierten Zugriff auf Web-Apps.

8.3.3. Begründung

- Das InAppBrowser-Plugin bietet die Möglichkeit, Webseiten in einer kontrollierten Umgebung zu öffnen

- Die Option `openInWebView` ermöglicht es, URLs zu fixieren und Navigationsmöglichkeiten einzuschränken
- Das Plugin ist gut dokumentiert und wird aktiv gewartet
- Events wie `browserClosed` und `browserPageLoaded` ermöglichen die Tracking-Funktionalität für zukünftige Zeitbeschränkungen
- Nahtlose Integration mit unserem Angular-Frontend über die Capacitor-Bridge

8.3.4. Alternativen

- Eigene WebView-Implementierung: Höherer Entwicklungsaufwand ohne signifikante Vorteile
- Progressive Web Apps (PWAs): Hätten weniger Kontrolle über das Nutzerverhalten geboten
- Native App-Wrapper für jede Website: Zu hoher Wartungsaufwand für jede einzelne Web-App

8.4. 9.4 Microservice-Architektur für Backend-Services

8.4.1. Kontext

Das Backend muss verschiedene Funktionen bereitstellen, darunter Authentifizierung, App-Listenmanagement und (zukünftig) Zeitbeschränkungen.

8.4.2. Entscheidung

Implementierung einer Microservice-Architektur mit Docker-Containern für die verschiedenen Backend-Komponenten.

8.4.3. Begründung

- Separate Container für Luddite Launcher Backend, Luddite Installer Backend und MongoDB
- Bessere Skalierbarkeit der einzelnen Komponenten
- Unabhängige Entwicklung und Deployment der verschiedenen Dienste
- Höhere Ausfallsicherheit, da Fehler in einem Service die anderen nicht beeinträchtigen
- Leichtere Wartbarkeit durch klare Trennung der Verantwortlichkeiten

8.4.4. Alternativen

- Monolithische Architektur: Hätte die Entwicklung anfangs vereinfacht, aber langfristig Skalierungsprobleme verursacht
- Serverless-Architektur: Wäre für einige Funktionen gut geeignet, aber schwieriger für die persistente Datenspeicherung

8.5. 9.5 MongoDB als Datenbank

8.5.1. Kontext

Für die Speicherung von Benutzerinformationen, App-Listen und Einstellungen wird eine flexible, skalierbare Datenbanklösung benötigt.

8.5.2. Entscheidung

Verwendung von MongoDB als dokumentenorientierte NoSQL-Datenbank.

8.5.3. Begründung

- Schemafreies Design ermöglicht flexible Anpassungen an den Datenstrukturen während der Entwicklung
- Gute Skalierbarkeit für zukünftiges Wachstum
- JSON-ähnliches Dokumentenformat passt gut zum JavaScript/TypeScript-Stack
- Einfache Integration mit Node.js-Backend
- Hohe Leseperformance für die häufigen Anfragen nach App-Listen

8.5.4. Alternativen

- PostgreSQL/MySQL: Hätten stärkere Konsistenzgarantien geboten, aber weniger Flexibilität
- Firebase Firestore: Hätte serverlose Operationen ermöglicht, aber mit höheren langfristigen Kosten
- Redis: Gut für Caching und temporäre Daten, aber weniger geeignet für die Hauptdatenspeicherung

8.6. 9.6 REST API statt GraphQL

8.6.1. Kontext

Es wird eine API-Strategie für die Kommunikation zwischen Frontend und Backend benötigt.

8.6.2. Entscheidung

Implementierung einer RESTful API für die Backend-Kommunikation.

8.6.3. Begründung

- REST bietet eine vertraute, gut dokumentierte Schnittstelle
- Die Datenaustauschangforderungen sind relativ einfach und gut auf REST-Endpunkte abbildbar
- Geringere Komplexität im Vergleich zu GraphQL, was die Entwicklungszeit reduziert
- Gute Caching-Möglichkeiten, was für mobile Apps mit potenziell instabiler Verbindung wichtig ist
- Einfache Integration mit Angular-Services und Capacitor HTTP-Plugin

8.6.4. Alternativen

- GraphQL: Hätte mehr Flexibilität bei komplexen Abfragen geboten, aber mit höherer Implementierungskomplexität
- gRPC: Hätte bessere Performanz bieten können, ist aber komplexer in der Implementierung und hat weniger Unterstützung für Web-Clients

8.7. 9.7 JWT für Authentifizierung

8.7.1. Kontext

Eine sichere, zustandslose Authentifizierungsmethode ist erforderlich, um Benutzer zu identifizieren und zu autorisieren.

8.7.2. Entscheidung

Verwendung von JSON Web Tokens (JWT) für die Benutzerauthentifizierung.

8.7.3. Begründung

- Zustandslose Authentifizierung passt gut zur Microservice-Architektur
- Tokens können sicher auf dem Client gespeichert werden (Capacitor Preferences API)
- Reduzierter Server-Overhead, da keine Sitzungsverwaltung erforderlich ist
- Einfache Integration mit Node.js-Backend durch verfügbare Libraries
- Ermöglicht eine klare Trennung zwischen Authentifizierung und Autorisierung

8.7.4. Alternativen

- Session-basierte Authentifizierung: Hätte zusätzliche Server-Infrastruktur für die Sitzungsverwaltung erfordert
- OAuth 2.0: Zu komplex für die aktuellen Anforderungen, könnte aber für zukünftige Social-Login-Funktionen relevant werden

8.8. 9.8 GitHub Actions für CI/CD

8.8.1. Kontext

Ein automatisiertes Build- und Deployment-System wird benötigt, um konsistente Releases zu gewährleisten.

8.8.2. Entscheidung

Verwendung von GitHub Actions für CI/CD-Pipelines.

8.8.3. Begründung

- Enge Integration mit dem GitHub-Repository
- Automatisierte Build-Prozesse für Frontend- und Backend-Komponenten
- Automatisierte Tests vor dem Deployment
- Automatisierte APK-Signierung für Android-Releases
- Kosteneffizienz, da innerhalb der kostenlosen GitHub-Kontingente nutzbar

8.8.4. Alternativen

- Jenkins: Mehr Flexibilität, aber höherer Wartungsaufwand
- GitLab CI: Hätte einen Wechsel der Repository-Plattform erfordert
- CircleCI/Travis CI: Ähnliche Funktionalität, aber weniger nahtlose Integration mit GitHub

8.9. 9.9 Object Storage für APK-Dateien

8.9.1. Kontext

Eine skalierbare Lösung für die Speicherung und Verteilung von APK-Dateien wird benötigt.

8.9.2. Entscheidung

Verwendung von Infomaniak Object Storage für die APK-Speicherung.

8.9.3. Begründung

- Kosteneffiziente Speicherung grosser Binärdateien
- Hohe Verfügbarkeit und Zuverlässigkeit
- Einfache Integration mit dem Backend durch REST API
- Skalierbarkeit für wachsende Anzahl von App-Versionen
- Kompatibel mit der bereits genutzten Infomaniak-Infrastruktur

8.9.4. Alternativen

- Datenbankbasierte Speicherung: Ineffizient für grosse Binärdateien
- Filesystem-basierte Speicherung: Schwieriger zu skalieren und zu sichern
- AWS S3 oder Google Cloud Storage: Ähnliche Funktionalität, aber höhere Kosten bei der aktuellen Infrastruktur

9. 10. Qualitätsanforderungen

9.1. 10.1 Qualitätsbaum

Der Qualitätsbaum zeigt die wichtigsten Qualitätsanforderungen basierend auf den ursprünglichen User Stories.

9.2. 10.2 Qualitätsszenarien

Die folgenden Qualitätsszenarien basieren direkt auf den User Stories aus dem Anforderungsdokument.

ID	Szenario	User Story	Umsetzungsstatus
QS-1	Bei jedem Öffnen der App wird die Liste der verfügbaren Web-Apps aus dem Backend geladen und aktualisiert. Eine lokale Kopie wird gespeichert, um die App auch bei temporären Verbindungsproblemen nutzen zu können.	User Story 1: Liste mit verfügbaren Apps laden	Vollständig umgesetzt
QS-2	Beim Tippen in das Suchfeld erscheint sofort eine Liste mit den gefilterten Suchresultaten. Diese Liste wird dynamisch angepasst, wenn sich der Suchbegriff ändert. Beim Klicken neben die Suche wird diese geschlossen.	User Story 2: Suchen nach Web Apps	Vollständig umgesetzt
QS-3	Bei Klick auf ein Suchergebnis öffnet sich die Website im InAppBrowser und ist vollständig nutzbar. Der Nutzer kann die URL nicht ändern oder bearbeiten, sodass nur autorisierte Webseiten zugänglich sind.	User Story 3: Öffnen von Web Apps	Vollständig umgesetzt
QS-4	Nutzer können über ein spezielles Formular Vorschläge für neue Web-Apps einreichen. Diese Wünsche werden in einer separaten Collection in der Datenbank gespeichert.	User Story 4: Vorschläge für neue Web Apps einbringen können	Vollständig umgesetzt
QS-5	Nur angemeldete Nutzer können die App nutzen. Ist ein Nutzer nicht angemeldet, erscheint ein Login-Screen. Der Nutzer kann sich jederzeit wieder abmelden.	User Story 5: Authentication	Vollständig umgesetzt
QS-6	Die Suche zeigt auch ausgewählte native Apps wie WhatsApp an. Beim Klick auf eine native App wird diese über einen Intent geöffnet.	User Story 6: Native Apps	Vollständig umgesetzt

ID	Szenario	User Story	Umsetzungsstatus
QS-7	Pro Benutzerkonto können individuelle Maximalzeiten für bestimmte Apps definiert werden. Sobald die Zeit überschritten ist, wird die App in der Suche ausgegraut und kann nicht mehr geöffnet werden.	User Story 7: Zeitbeschränkung	**Nicht implementiert** - vorgesehen für zukünftige Version
QS-8	Pro Benutzerkonto können 2-4 favorisierte Apps festgelegt werden. Diese werden permanent angezeigt und können ohne Suche direkt geöffnet werden.	User Story 8: Beliebte Apps anpinnen	**Teilweise umgesetzt** durch die Schnellzugriffsleiste, aber ohne personalisierte Einstellungsmöglichkeit

9.3. 10.3 Bewertung und Ausblick

Wie aus der Tabelle ersichtlich, erfüllt die aktuelle Implementierung die User Stories 1-6 vollständig:

- Die App-Liste wird aktuell gehalten (QS-1)
- Die Suchfunktion funktioniert wie spezifiziert (QS-2)
- Web-Apps können kontrolliert geöffnet werden (QS-3)
- Vorschläge für neue Apps können eingereicht werden (QS-4)
- Die Authentifizierung ist implementiert (QS-5)
- Native Apps sind in die Suche integriert (QS-6)

Für die Anforderungen QS-7 und QS-8 gilt:

- **Zeitbeschränkung (QS-7)**: Diese Funktion wurde aufgrund des Zeitrahmens für das Schulprojekt zurückgestellt. Die technische Grundlage ist bereits vorbereitet, da das Capacitor Browser Plugin Events für das Öffnen und Schliessen von Web-Apps bietet. Dies ermöglicht die zukünftige Messung der Nutzungszeit. Die Implementierung ist für die nächste Entwicklungsphase eingeplant.
- **Beliebte Apps anpinnen (QS-8)**: Die aktuelle Schnellzugriffsleiste am unteren Bildschirmrand bietet bereits teilweise diese Funktionalität, da häufig genutzte Apps wie Telefon, Messaging und WhatsApp dort angezeigt werden. Allerdings fehlt noch die Möglichkeit für Benutzer, diese Auswahl selbst zu konfigurieren. Die Schnellzugriffsleiste muss um Einstellungsmöglichkeiten erweitert werden, um QS-8 vollständig zu erfüllen.

Die Abhängigkeit zwischen den Qualitätsanforderungen ist wie folgt:

- Die Aktualität der App-Liste (QS-1) ist fundamental für alle anderen Funktionen

- Die Authentifizierung (QS-5) ist Voraussetzung für die personalisierte Nutzung (QS-7, QS-8)
- Die Suchfunktion (QS-2) ist zentral für den Zugriff auf alle Apps (Web-Apps und native Apps)
- Die kontrollierten Web-Apps (QS-3) sind das Kernstück der Digital Wellbeing-Strategie der App

10. 11. Risiken und technische Schulden

In diesem Kapitel werden die aktuellen Risiken und technischen Schulden des Luddite Launcher Projekts dokumentiert. Diese Übersicht dient dazu, potenzielle Problembereiche transparent zu machen und künftige Entwicklungsschwerpunkte zu identifizieren.

10.1. 11.1 Technische Risiken

ID	Risiko	Potenzielle Auswirkung	Priorisierung
R-01	Installation von Apps über ADB weiterhin möglich	Umgehung des kontrollierten App-Ökosystems durch technisch versierte Nutzer	Hoch
R-02	Abhängigkeit von Capacitor-Plugins und deren Aktualisierungen	Mögliche Inkompatibilitäten oder fehlende Funktionalität bei neuen Android-Versionen	Mittel
R-03	Modifiziertes LineageOS ist nicht für alle Smartphones verfügbar	Eingeschränkte Nutzerbasis und potenzielle Kompatibilitätsprobleme	Mittel

10.2. 11.2 Massnahmen zur Risikoreduzierung

Risiko-ID	Massnahmen
R-01	Implementierung einer Root-Erkennung und Sperrung von ADB-Features in modifiziertem LineageOS in einer zukünftigen Version. Alternativ: Entwicklung eines SELinux-Moduls, das nicht-autorisierte App-Installationen blockiert.
R-02	Regelmässiges Testen der App mit neuen Capacitor-Versionen. Erstellung eines umfassenden Test-Katalogs für die Integration mit nativen Funktionen. Bereithalten von Fallback-Mechanismen.
R-03	Auswahl gezielter, weit verbreiteter Smartphone-Modelle für die Unterstützung. Dokumentation des Build-Prozesses für Community-Entwickler, die weitere Geräte unterstützen möchten.

10.3. 11.3 Technische Schulden

ID	Technische Schuld	Auswirkung	Dringlichkeit
S-01	Fehlende automatisierte Tests für die Angular-Komponenten	Erhöhtes Risiko von Regressionsfehlern bei Änderungen und Updates	Hoch
S-02	Redundanzen in der Backend-API mit separaten Endpunkten für Launcher und Installer	Erhöhter Wartungsaufwand und Risiko inkonsistenter Implementierungen	Mittel
S-03	Noch nicht implementierte Zeitbegrenzungsfunktion trotz entsprechender User Story	Nicht erfülltes Kernversprechen der Anwendung zur Reduzierung von Bildschirmzeit	Hoch
S-04	Eingeschränkte Personalisierungsmöglichkeiten der Schnellzugriffsleiste	Weniger benutzerfreundliche Erfahrung und nicht erfüllte User Story	Mittel
S-05	Hardcodierte Konfigurationswerte in einigen Komponenten	Erschwerte Wartbarkeit und Anpassung an unterschiedliche Umgebungen	Niedrig
S-06	MongoDB-Schema ohne ausreichende Validierung	Risiko von Dateninkonsistenzen	Niedrig
S-07	JWT Secret als Konstante im Code statt als Umgebungsvariable	Sicherheitsrisiko bei öffentlichem Code-Zugriff	Hoch
S-08	Passwörter werden im Klartext in der MongoDB gespeichert	Erhebliches Sicherheitsrisiko bei Datenbankzugriff oder -kompromittierung	Sehr hoch

11. 12. Glossar

In diesem Kapitel werden wichtige Begriffe, Abkürzungen und Fachbegriffe definiert, die im Kontext des Luddite Launcher Projekts verwendet werden.

Begriff	Definition
ADB	Android Debug Bridge. Ein Kommandozeilen-Tool, das die Kommunikation mit einem Android-Gerät ermöglicht und zum Installieren von Apps ohne Google Play Store verwendet werden kann.
Angular	Ein TypeScript-basiertes Frontend-Framework für die Entwicklung von Single-Page-Anwendungen, das im Luddite Launcher für die Benutzeroberfläche verwendet wird.
API	Application Programming Interface. Eine Schnittstelle, die es verschiedenen Softwarekomponenten ermöglicht, miteinander zu kommunizieren.
APK	Android Package Kit. Das Dateiformat für die Distribution und Installation von Android-Anwendungen.
Capacitor	Ein Framework, das Web-Anwendungen in native mobile Apps konvertiert und dabei den Zugriff auf native Funktionen ermöglicht.
Digital Wellbeing	Konzept zur Förderung einer gesunden, bewussten Nutzung digitaler Geräte und der Reduzierung von digitalen Ablenkungen.
Docker	Eine Plattform zur Entwicklung, Bereitstellung und Ausführung von Anwendungen in Containern, die in der Backend-Infrastruktur des Luddite Launcher verwendet wird.
InAppBrowser	Ein Capacitor-Plugin, das einen kontrollierten Web-Browser innerhalb der App bereitstellt und für den Zugriff auf ausgewählte Webseiten verwendet wird.
Intent	In Android: ein Mechanismus, der die Kommunikation zwischen verschiedenen Komponenten ermöglicht, z.B. zum Starten einer anderen App.
JWT	JSON Web Token. Ein offener Standard zur sicheren Übertragung von Informationen als JSON-Objekt, der für die Authentifizierung im Luddite Launcher verwendet wird.
LineageOS	Eine freie, quelloffene Custom-ROM-Distribution für Android-Geräte, die als Grundlage für das modifizierte Betriebssystem des Luddite Launcher dient.
Luddite	Historisch: Mitglieder einer Bewegung englischer Textilarbeiter im frühen 19. Jahrhundert, die sich gegen die Industrialisierung wehrten. Im Kontext dieses Projekts: Symbolisch für eine kritische Haltung gegenüber übermässiger Technologienutzung.
Luddite Installer	Eine spezielle App im System, die die Installation von autorisierten nativen Apps ermöglicht.
Luddite Launcher	Die zentrale Anwendung des Projekts, die als Startbildschirm (Launcher) dient und den kontrollierten Zugriff auf Web- und native Apps ermöglicht.

Begriff	Definition
MongoDB	Eine dokumentenorientierte NoSQL-Datenbank, die im Backend des Luddite Launcher zur Speicherung von Benutzerinformationen und App-Listen verwendet wird.
Native App	Eine Anwendung, die speziell für eine bestimmte Plattform (in diesem Fall Android) entwickelt wurde und direkt auf dem Gerät ausgeführt wird.
Node.js	Eine JavaScript-Laufzeitumgebung, die für die Entwicklung des Backend-Servers des Luddite Launcher verwendet wird.
Object Storage	Ein Speichersystem für unstrukturierte Daten, das in der Infomaniak-Cloud zur Speicherung von APK-Dateien verwendet wird.
OpenStack	Eine Open-Source-Cloud-Computing-Plattform, die von Infomaniak für die Bereitstellung der Backend-Infrastruktur verwendet wird.
Schnellzugriffsleiste	Ein UI-Element am unteren Bildschirmrand des Luddite Launcher, das schnellen Zugriff auf häufig genutzte Apps bietet.
Social Media Abhängigkeit	Ein Zustand übermässiger oder zwanghafter Nutzung sozialer Medien, der zu negativen Auswirkungen auf das Wohlbefinden führen kann und dem der Luddite Launcher entgegenwirken soll.
Web-App	Eine Anwendung, die über einen Webbrowser zugänglich ist und im Luddite Launcher über den kontrollierten InAppBrowser aufgerufen wird.
Zeitbegrenzung	Eine geplante Funktion des Luddite Launcher, die die Nutzungsdauer bestimmter Apps, insbesondere sozialer Medien, einschränken soll.