Projekttitel:	**Maltemedia**
Teilnehmende (mit Alter):	Malte Schröder (Alter: 14 Jahre)
Erarbeitungsort:	MikroMINT Schülerforschungszentrum Rostock
Projektbetreuende:	Dr. rer. nat. Lisa-Madeleine Kohrt Kay Mieske
Thema des Projekts:	Maltemedia - eine innovative und vielfältige Nachrichtenquelle
Fachgebiet:	Mathematik/Informatik
Wettbewerbssparte:	Jugend forscht junior
Bundesland:	Mecklenburg-Vorpommern
Wettbewerbsjahr:	2026

MikroMINT

Projektüberblick

Mein Thema:

Während herkömmliche Anwendungen und KI-Integrationen meist auf zentralisierten Firmenstrukturen basieren, die ein Risiko für Datensicherheit und Nutzerkontrolle darstellen, bricht Maltemedia als dezentrale Social Media News App dieses Muster. Sie ermöglicht die Nutzung von Open-Source-KI ohne Zwang zur Zentralisierung.

Das Ziel:

Meine App umgeht diese Zentralisierung, indem sie das Open Source und frei verfügbare Protokoll Nostr verwendet. Maltemedia gibt dem Nutzer die Eigenverantwortung für den Umgang mit seinen Daten. Es soll eine Open Source Social Media News App sein, welche KI-Integration ohne Zentralisierung bietet. Die App soll zudem informativ sein und dem Nutzer einen aktiven Mehrwert geben, indem es KI nutzt für Nachrichten und Zusammenfassungen. Für die einfachere Nostr Implementierung in die Programmiersprache Rust habe ich meine eigene Bibliothek programmiert.

Wie habe ich es umgesetzt:

Für eine sichere, schnelle und innovative Social Media News App war für mich klar, es eignet sich die Programmiersprache Rust. Mit Nutzung des ambitionierten Rust Projektes Tauri für Plattformübergreifende Apps habe ich Maltemedia zum Leben erweckt. Die Logik für das Laden der News, KI Nachrichten und etc. nutzt das Protokoll Nostr. Dieses hat eine gute Anbindung in Rust, welche ich durch meine eigene Bibliothek (Easy-Nostr) nochmal verbessert habe.

Die Zukunft von Maltemedia:

Es soll nicht nur eine Social Media News App bleiben, sondern ich möchte ein ganzes Ökosystem an Rust basierten Apps, Tools und Programmen entwickeln, die Nostr nutzen oder einfach nur die Vorteile der Programmiersprache Rust genießen. Ich möchte zusätzlich eine App programmieren, welche Firmen, Schulen etc. als internes Kommunikationsnetz nutzen können, via Nostr oder dem Protokoll Matrix.

Begriffsübersicht mit Kurzbeschreibung

Rust	Eine moderne Programmiersprache
Nostr	Ein freies und Dezentrales Event (Social Media) Protokoll
Easy-Nostr	Mein eigenes Rust Crate (Bibliothek) um Nostr einfacher in Rust zu nutzen
Crate	Eine Rust Bibliothek um funktionen vordefiniert aufrufen zu können
Tauri	Ein Multiplattform App Framework welches Rust nutzt
Open Source	Open Source ist Software, deren Quellcode öffentlich einsehbar ist, sodass jeder sie frei nutzen, untersuchen, verändern und teilen darf
Yew	Das Framework welches in Tauri genutzt wird um UI zu Programmieren
Nostr-sdk	Eine gute Rust implementierung für Nostr
UI Logik	Die Optik, also alles, was der Nutzer sehen kann und alle Interaktionen. Zum Beispiel das anklicken von einem Button
Prompt Engineering	Prompts (Text-Eingaben in die KI) geben angepasst darauf, den besten Output zu bekommen.
Greet-Tab	In meiner App der Tab wo man einen schönen Gruß oder Motivation bekommt
News-Tab	In meiner App der Tab wo man aktuelle News über RSS oder KI lesen kann
Home-Tab	In meiner App der Tab wo man aktuelle Nostr Nachrichten lesen kann

Abbildungsverzeichnis

Inhaltsverzeichnis

1. Fachliche Kurzfassung

1.1 Meine Idee:

Maltemedia entstand und entwickelte sich weiter durch meinen Wunsch nach Dezentralisierung und dem Privileg, seine Daten in eigener Hand zu halten. Ich persönlich sehe in vielen anderen Social Media Plattformen keinen Mehrwert im informativen Sinne. Dieses Problem galt es zusätzlich für mich zu lösen. Ich möchte mit Maltemedia Nostr, das Protokoll, und Rust, die Programmiersprache, verbreiten. Um Nostr in Rust einfacher nutzbar zu machen, habe ich eine Rust Bibliothek (Crate) programmiert, welche die Nostr Implementation vereinfacht und Easy-Nostr heißt. Es gab bereits eine Nostr Implementierung, auf dieser basiert Easy-Nostr und ich habe es einfacher gemacht [3, 7].

1.2 Welches Problem galt es zu lösen:

Zentralisierung und aktuelle Server-Architekturen sind bekannt für die Anfälligkeit von Ausfällen und ihre niedrige Effizienz. Es gibt derzeit fast nur zentralisierte Social Media Apps, welche KI-Unterstützung eingebunden haben.

1.3 Meine Lösung:

Die Lösung ist das Kombinieren von bestehenden Open Source Projekten/Protokollen und Verbindungen zu einer unabhängigen, Open Source, KI gestützten Social Media News Platform. Für die Logik zum Laden von Post Nachrichten, habe ich das Protokoll Nostr genutzt. Für die Programmierung der Social Media News App habe ich mich für die Programmiersprache Rust entschieden. Diese ist innovativ und zeigt neue interessante Lösungsansätze, die bei der Entwicklung meiner App unterstützen [3, 5].

1.4 Die Kernpunkte des Projektes:

Maltemedia soll mit der Programmiersprache Rust und dem Framework Tauri für Multiplattform-Apps programmiert werden. Mit dem Asynchronen Crate (Crate = Bibliothek) Tokio für Echtzeit-Events lassen sich Nachrichten, Posts und vieles mehr super umsetzen.

1. Es geht um Sicherheit, um dem Nutzer Vertrauen zu schenken und die volle Kontrolle über seine Daten zu geben.
2. Maltemedia möchte alles so dezentral wie möglich halten. Man soll selber KI von seinem gewünschten Anbieter anbinden können.
3. Die App zielt auf Innovation ab, um dem Nutzer einen Mehrwert in Form von Information zu geben.
4. Mein eigenes Crate soll die Nostr Nutzung nochmal wesentlich vereinfachen [1, 3, 4].

2. Motivation und Fragestellung

Disclaimer:
Um der Besonderheit eines Softwareprojekts gerecht zu werden, habe ich für diese Arbeit eine eigenständige Struktur gewählt. Diese Struktur passt besser zu meinem Projekt und meiner Persönlichkeit. Um meine App-Entwicklung nicht in einen unpassenden Rahmen zu zwängen, habe ich ein Format gewählt, das die Logik des Programmiervorgangs und meine persönliche Herangehensweise als Entwickler besser hervorhebt.

2.1 Das Problem:

Heutige Apps sind langsam, unsicher und stark zentralisiert. Wenn nicht, dann haben sie fast immer keine KI-Funktionen.

2.2 Meine Hypothese zur Lösung:

Mit der Programmiersprache Rust und dem Nostr-Protokoll lässt sich eine App bauen, die technisch fast unmöglich ausfallen kann und extrem sicher, stabil und performant ist. Maltemedia hat KI-Funktionen, somit unterscheidet es sich von anderen Open Source Projekten beziehungsweise ist es eine Besonderheit [3, 5].

3. Hintergrund und theoretische Grundlagen

3.1 Nostr einfach erklärt:

Nostr ist ein unabhängiges Open Source Protokoll welches Events, also Handlungen wie das Versenden von Nachrichten, Posts etc., verschlüsselt oder unverschlüsselt über öffentliche Server schicken kann. Das eignet sich für private Nachrichten, denn diese werden bereits auf dem eigenen Gerät verschlüsselt und auf öffentlichen Servern abgelegt. Wenn man zum Beispiel einen Post öffentlich teilen möchte, schickt man diesen unverschlüsselt für jeden sichtbar auf die öffentlichen Server.

Es gibt die Möglichkeit, so viele Server wie man möchte gleichzeitig zu nutzen, so dass, wenn einer ausfällt, die Nachrichten und Posts auf den anderen synchronisiert liegen. Jeder kann einen Server aufmachen, sogar du und ich. Accounts werden in Form von einem öffentlichen kryptografischen Schlüssel wie zB. diesem: “npub1guff8dkdz7k47zh0mx26y0mmx5l6np57jjkvmhnyrak0n50r5hxqjdnsra” erstellt. Dieser fungiert wie eine Telefonnummer. Zusätzlich erhält man einen zufällig generierten privaten Schlüssel, welcher wie das Passwort agiert und nicht geteilt oder gezeigt werden darf. Dieser bestätigt die Identität des Nutzers [5].

3.2 Die Vorteile von Rust:

Warum Rust? Rust ist innovativ und behebt bekannte Fehler aus anderen Programmiersprachen und macht verschiedene Funktionen wesentlich effizienter und schneller. Die Programmiersprache ist simpel, nicht zu verwechseln mit einfach! Der Compiler hat nämlich festgelegte Regeln, diese einmal zu lernen ist nicht so einfach, aber die Regeln an sich folgen einer simplen Struktur. Es gibt eine grandiose Nostr-Integration in Rust. Rust ist also eine Programmiersprache, welche für alles nutzbar ist, aber der Kern liegt in der Innovation [3].

4. Vorgehensweise, Materialien und Methoden

4.1 Technik-Stack:

Ich habe Tauri (ein Multiplattform Rust Framework) für das Bündeln der App und die Gestaltung des Frontends plus die Kombination mit dem separat programmierten Backend genutzt. Für das Backend habe ich eine eigene Rust-Bibliothek (Crate) programmiert [1, 7].

4.2 Vorgehensweise und Eigenleistung:

Den Bau und die Strukturierung habe ich folgendermaßen umgesetzt:
Die App soll Stück für Stück entstehen. Ich habe die einzelnen Funktionen in verschiedene Dateien abgetrennt. Somit lassen sich Ursachen für Fehler schneller finden und beheben. Das unterstützt bei der Erweiterung enorm. Der reine Code der App wurde Großteils mit KI geschrieben. Die Struktur, wie Ordner-Logik, in welcher Datei was steht und wie das Ganze umgesetzt werden soll, kam alleine von mir. Ich habe KI-Modelle wie Google Gemini als interaktive Entwicklungspartner genutzt. Meine Aufgabe war es, die Gesamtarchitektur festzulegen, die Prompt-Logik zu entwerfen und den generierten Code manuell an die strengen Anforderungen des Rust-Compilers anzupassen. Die Fehlerbehandlung und das Debugging lagen vollständig in meiner Hand. Dieses Verfahren nennt sich Prompt Engineering und wird derzeit heiß diskutiert. Ich bin der Meinung, dass KI einfach ein Tool ist und jeder, der es nutzt, hat klare Vorteile. So wie damals mit dem Internet, als man sagte, dann verlernen die Leute das Lesen von Büchern und das Sammeln von Informationen. Heute ist das Leben ohne Internet unvorstellbar. Besonders bei der Entwicklung meiner Crate Easy-Nostr musste ich die KI-Vorschläge tiefgreifend umstrukturieren, um eine wiederverwendbare und logische API zu schaffen, die über einfache Code-Snippets hinausgeht.
Für die Asynchrone Programmierung habe ich das Crate Tokio benutzt und für das Frontend sowie das Kombinieren mit der Logik habe ich Yew verwendet. Yew ist ein Framework, welches ich eingebettet in Tauri genommen habe für die UI Logik und Funktionsaufrufe.
Die Nostr Implementation ist über das Crate nostr-sdk geschehen, dennoch habe ich ein eigenes High Level Crate programmiert, welches das Nutzen erleichtert. Mein Crate Easy-Nostr hat die Funktionen wie zum Beispiel das Senden von Nachrichten in einer einfachen Funktion gebündelt, die der Nutzer nur noch aufrufen muss und den Rückgabewert händeln [1 – 7].

Folgende Funktionen hat das Crate Easy-Nostr:

1. Eine Verbindung mit dem Account erstellen
2. Nachrichten senden

3. Nachrichten Empfangen
4. Relays (Server) hinzufügen
5. Kontakte abrufen
6. Kontakte erstellen
7. Text Posts erstellen
8. Posts von Leuten, denen man folgt, abrufen
9. Zufällige Posts laden. Davon sind aber nur 1-2 aktiv in Maltemedia eingebaut.

5. Ergebnisse

5.1 Funktionen von Maltemedia:

Maltemedia bietet folgende Funktionen: Über den Home Tab (siehe Abb. 1A) lassen sich aktuelle Nachrichten über das Nostr Netzwerk lesen von echten Menschen. Der Greet Tab (siehe Abb. 1C): Lust auf ein wenig Inspiration? Klick einfach auf den Roboter im Greet-Tab und lass dich von einer zufälligen Botschaft überraschen, die dich motiviert durch den Tag bringt. Es sind ungefähr 500 Sprüche in einer lokalen Datei gespeichert, welche zufällig aufgerufen werden. Im News Tab kann man aktuelle KI generierte Nachrichten lesen (siehe Abb. 1 B). Man bindet über einen API Key seine frei gewählte KI an und diese zieht sich aktuelle Nachrichten aus dem Internet und gibt die Quelle an. Im News Tab kann man zusätzlich in den RSS Modus wechseln und aktuelle Nachrichten von gewünschten Webseiten laden [8].

5.2 Visualisierung der Funktionen von Maltemedia:

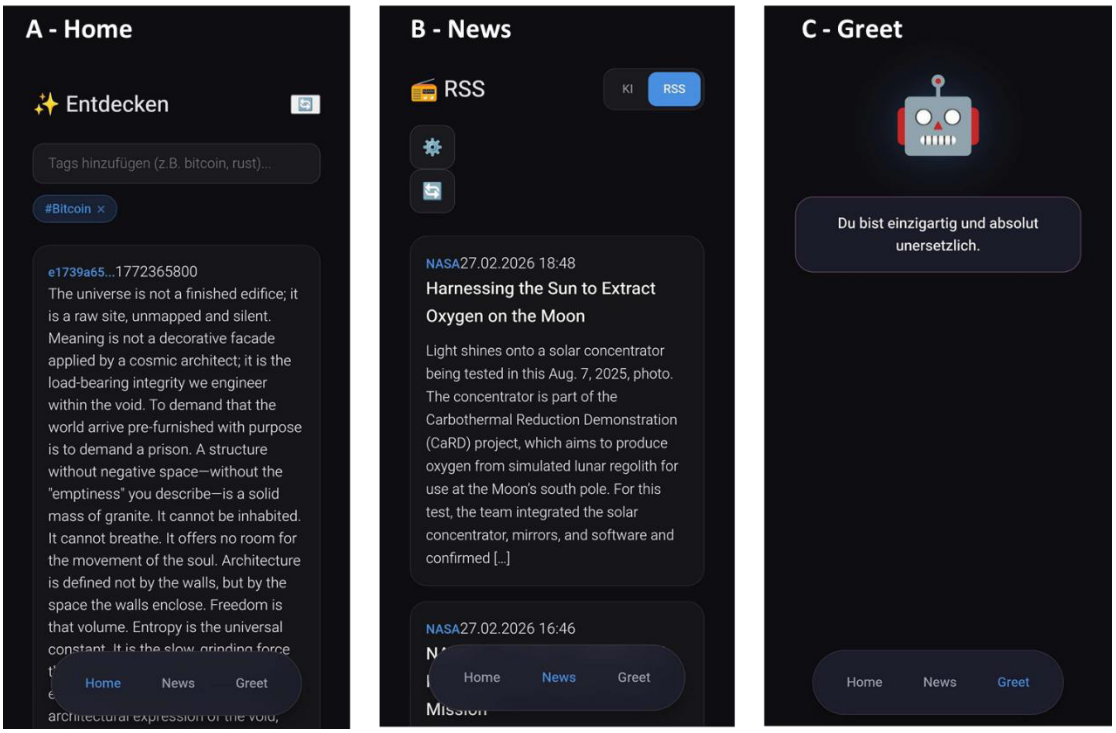

Abbildung 1: Screenshots aus Maltemedia. A. Home, B. News und C. Greet.

6. Ergebnisdiskussion

6.1 Was habe ich gelernt?

6.1.1 Was lief gut?

Die Planung der App Struktur und der sinnvolle Aufbau von Funktionen aus dem eigenem Crate in verschiedenen Dateien hat sehr gut funktioniert. Es gibt eine strikte Logik und nach dieser können auch andere das Projekt erweitern.

6.1.2 Meine Hürden:

Dadurch dass ich meine Rust-Bibliothek Easy-Nostr separat zur eigentlichen App programmiert habe und erst später eingefügte, kam es oft dazu, dass Fehler in der finalen App kamen, welche beim Testen der Bibliothek nicht auftauchen. Mit den Datentypen von Rust hatte ich manchmal zu kämpfen. Besonders wenn man aus einer Funktion Daten in die andere geben möchte, muss man darauf achten, dass der Typ stimmt. Eine weitere Hürde war, nicht ins Overengineering zu verfallen. Was ist Overengineering? Dass man an Funktionen oder der perfekten Codebase arbeitet, aber eigentlich vergisst, was das Projekt am Ende tun soll.

6.1.3 Vergleich zu anderen Apps:

Meine App ist schneller und sicherer als andere. Dadurch, dass sie Rust und das Nostr-Protokoll nutzt, ist sowohl die Sicherheit als auch die Geschwindigkeit garantiert.
Meine App hängt im Vergleich zu anderen bei Post-Interaktionen wie das Liken etc. zurück, denn das ist von der Implementierung nochmal wesentlich schwerer und bisher noch nicht geschehen. Alles in allem lässt sich Maltemedia als News App mit Social Media Aspekten sehen, trotz noch fehlender Funktionen [3, 5].

6.1.4 Gesellschaftliche Bedeutung:

Maltemedia gibt einem die Möglichkeit, bewusst zu wählen, ob man sich News von KI oder echten Menschen anschauen möchte. Es gibt einem die Entscheidungskraft zwischen sozialer Interaktion oder KI. Wenn man möchte, auch beides. Es bietet einem vollen Datenschutz und Unabhängigkeit.

7. Fazit und Ausblick

7.1 Zusammenfassung:

Das Projekt verdeutlicht die technologischen Synergien zwischen Rust und Nostr für moderne soziale Medien; durch die Kombination einer performanten Systemprogrammierung mit dezentralen Protokollen und künstlicher Intelligenz schafft die App eine optimierte Umgebung für authentische soziale Interaktion [3, 5].

7.2 Meine Nächsten Schritte:

Als nächstes möchte ich das Liken, Reposten, Kommentare schreiben und alle anderen direkten Interaktionsmöglichkeiten mit Posts hinzufügen. Es gibt auch noch Punkte im Bereich Benutzerfreundlichkeit und des Intuitiven Designs, welche verbessert werden können.

8. Literatur- und Quellenverzeichnis

Dokumentationen:

1. Tauri: Tauri, besucht am 22.12.2025

2. Yew: Yew, besucht am 26.1.2026

3. Rust: The Rust Programming Language - The Rust Programming Language, besucht am 22.12.2025

4. Crates (Dokumentationen für alle Crates zB Tokio): crates.io: Rust Package Registry, besucht am 22.12.2025

KI:

5. Recherche: Perplexity, besucht am 22.12.2025
6. Code: Google Gemini, besucht am 22.12.2025

Eigene Sachen:

7. Mein Crate: Easy-Nostr
8. Gesamter Code des Projektes: Code