Hochschule Rhein-Waal
Fakultät Kommunikation und Umwelt
Prof Dr.-Ing. Ido Iurgel
Prof Dr. Kai Essig

Flutter als Entwicklungsplattform für eingebettete Linux Systeme: eine Fallstudie

Bachelorarbeit
im Studiengang
Medieninformatik
Zur Erlangung des akademischen Grades

Bachelor of Science

vorgelegt von
Fabian Baldeau

Matrikelnummer:
27516

Abgabedatum:
xx. xxxx 2024

Zusammenfassung

Die Integration von Flutter's plattformübergreifenden UI-Fähigkeiten mit der Anpassungsfähigkeit von Embedded Linux verspricht die Rationalisierung der Entwicklung von Benutzeroberflächen (UIs) für eingebettete Systeme. Diese Arbeit untersucht die Eignung von Flutter in diesem Kontext durch eine Fallstudie: die Entwicklung eines Flutter App Prototyps auf einem Raspberry Pi mit einer angepassten eingebetteten Linux Distribution. Leistungsmesswerte, Benutzerfreundlichkeit und der Entwicklungsprozess werden bewertet. Die Ergebnisse zeigen, dass Flutter, wenn es angepasst wird, erfolgreich zur Erstellung von attraktiven und benutzerfreundlichen eingebetteten Schnittstellen mit wenigen Kompromissen bei der Leistung genutzt werden kann.

Die Arbeit beschäftigt sich im Wesentlichen mit der Entwicklung einer prototypischen Anwendung mit Flutter für ein eingebettetes Linux System. Dabei wurde ein Konzept ausgearbeitet werden, dass zeigt wie Flutter in eingebetteten Linux Systemen genutzt werden kann und es wurde anhand der prototypischen Anwendung untersucht, wie gut Flutter für diesen Verwendungszweck geeignet ist. Die Ergebnisse der Untersuchung zeigen ein großes Potential für die Nutzung von Flutter in eingebetteten Linux Systemen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Auflistungsverzeichnis

1 Einleitung

1.1 Motivation

Die Entwicklung moderner und ansprechender Benutzeroberflächen (GUIs) für eingebettete Linux-Systeme ist eine Herausforderung. Herkömmliche GUI-Toolkits stoßen häufig an ihre Grenzen, wenn es um die Erfüllung der Anforderungen an Leistung, plattformübergreifende Kompatibilität und Design geht. Flutter (Google Inc., o. D. a), ein quelloffenes Benutzerschnittstellen-Entwicklungs-Kit von Google, stellt eine vielversprechende Alternative dar. Es zeichnet sich durch native Kompilierung für hohe Performance und geringen Ressourcenverbrauch, eine flexible Architektur, die an verschiedene Plattformen angepasst werden kann, und seinen Fokus auf moderne und intuitive Benutzeroberflächen aus.

1.2 Relevanz

In dieser Arbeit wird der Einsatz von Flutter für die Entwicklung von GUIs für eingebettete Linux-Systeme untersucht. Die Arbeit leistet einen wichtigen Beitrag zu diesem Forschungsbereich, indem sie:

- die Leistungsmerkmale von Flutter auf eingebetteter Hardware detailliert untersucht und wertvolle Hinweise für die Auswahl des richtigen Toolkits liefert.
- den Entwicklungsprozess detailliert beschreibt und Herausforderungen sowie bewährte Praktiken bei der Verwendung von Flutter in einem eingebetteten Linux-Kontext aufzeigt.
- eine reale eingebettete GUI mit Flutter implementiert und dabei die Fähigkeiten und potenziellen Anwendungsfälle des Toolkits hervorhebt.

Die Ergebnisse dieser Arbeit sind relevant für Entwickler im Bereich eingebetteter Systeme, die moderne, performante und ansprechende GUIs für Linux-basierte Geräte entwickeln möchten.

1.3 Zielsetzung

Ziel der Arbeit ist es, die Potenziale und Herausforderungen von Flutter im Bereich der GUI-Entwicklung für eingebettete Linux-Systeme zu beleuchten und Entwicklern wertvolle Hinweise für die Auswahl des richtigen Toolkits zu liefern.

1.4 Problemstellung

Beantwortet werden soll die Frage, inwieweit sich Flutter für die Entwicklung von GUIs für eingebettete Linux-Systeme eignet. Dazu wird in dieser Arbeit geklärt: Wie einfach und effizient ist die Entwicklung und Wartung von GUIs mit Flutter? Welche Auswirkungen hat Flutter auf die Performance und den Ressourcenverbrauch im eingebetteten Kontext? Wie hoch ist der Lernaufwand für Entwickler? Welche Community und Support-Ressourcen stehen für Flutter-Entwickler zur Verfügung? Um diese Fragen zu beantworten wird die folgende Methodik angewendet.

1.5 Methodik

Um die Forschungsfrage zu beantworten, wird eine Literaturrecherche, eine experimentelle Untersuchung und eine Auswertung der Untersuchung durchgeführt. In der Literaturrecherche werden aktuelle wissenschaftliche Quellen zu GUI-Entwicklung für eingebettete Systeme und Flutter ausgewertet. Die experimentelle Untersuchung umfasst die Implementierung einer realen eingebetteten GUI mit Flutter und die Bewertung der Performance, des Ressourcenverbrauchs und der Benutzerfreundlichkeit der GUI. Die Ergebnisse der Literaturrecherche und der experimentellen Untersuchung werden in einem Bewertungsschema zusammengefasst und abschließend bewertet. Basierend auf den Ergebnissen wird ein Leitfaden für die Entwicklung von GUIs für eingebettete Linux-Systeme mit Flutter erstellt.

1.6 Struktur der Arbeit

Zu ergänzen wenn Arbeit fertig ist…

1.7 Problematik

Zu ergänzen falls nötig…

2 Grundlagen

2.1 Eingebettete Systeme

Traditionelle Ansichten über Computer beschreiben sie oft als eigenständige Geräte wie Desktops oder Laptops. Im Gegensatz dazu skizzierte Mark Weiser in seinem einflussreichen Paper “The Computer of the 21st Century” 1999 eine Vision des “Ubiquitous Computing”. (Weiser, 1999) Diese Vision beschreibt eine Welt, in der nahtlos integrierte Geräte in unserem Leben “verschwinden”. Eingebettete Systeme verkörpern heute diese Philosophie. Sie sind keine Allzweckcomputer, sondern spezialisierte Computer, die in Geräten und Systemen eingebettet sind. Sie sind in der Regel auf eine spezielle Aufgabe oder Funktion zugeschnitten und sind oft in Geräten des täglichen Lebens zu finden.
Beispiele für eingebettete Systeme finden sich in allen Lebensbereichen:

- **Automobilindustrie**: Infotainment System
- **Industrie**: Automatisierungssysteme, Steuerung von Maschinen
- **Medizintechnik**: Medizinische Geräte, Diagnosegeräte
- **Haushaltsgeräte**: Waschmaschinen, Kühlschränke
- **Consumer Electronics**: Smartphones, Smartwatches

Eingebettete Systeme sind oft auf Echtzeitfähigkeit, Energieeffizienz und Zuverlässigkeit ausgelegt. Sie sind in der Regel nicht so leistungsfähig wie Desktop- oder Server-Computer, aber sie sind in der Lage, spezialisierte Aufgaben zu erfüllen.
Auch Experten fällt eine klare Definition von eingebetteten Systemen in der heutigen Zeit nicht leicht, wie auch Jack Ganssle, ein Experte für eingebettete Systeme mit über 30 Jahre Berufserfahrung, in einem Blog Beitrag schreibt. (Ganssle, 2008) So konnte man laut Ganssle früher Systeme, die mit einem Mikrocontroller oder Mikroprozessor ausgestattet sind und in einem Gerät eingebettet sind, klar als eingebettete Systeme bezeichnen. Doch heutzutage werden oft auch Systeme, die mit einem vollwertigen Betriebssystem ausgestattet sind, als eingebettete Systeme bezeichnet. Dazu gehört das Linux Betriebssystem, welches in vielen eingebetteten Systemen zum Einsatz kommt und auch Windows ist laut Ganssle in eingebetteten Systemen anzutreffen. Als weiteres Beispiel dazu nennt er das Handy, welches als es nur zum telefonieren genutzt wurde klar als eingebettetes System bezeichnet werden konnte, doch heutzutage ist es ein vollwertiger Computer, der auch als solcher für eine Vielzahl von anderen Funktionen genutzt werden kann.

In weiterer Literatur zu eingebetteten Systemen finden sich die folgenden Definitionen:

> "An embedded system is a microprocessor based system that is built to control a function or a range of functions." (Heath, 2002)

> "Embedded systems are information processing systems embedded into enclosing products." (Marwedel, 2021)

Auch aus diesen Definitionen geht hervor, dass die Definition von eingebetteten Systemen nicht eindeutig ist. Da es sowohl eine Funktion oder eine Reihe von Funktionen steuern kann, als auch speziell ein System ist, das in ein Produkt eingebettet ist.
Ein eingebettetes System ist also meistens ein Computersystem, das spezialisiert ist auf eine bestimmte Funktion oder Aufgabe. Mittlerweile heißt das jedoch auch, dass es sich oft um Systeme handelt, welche eigentlich vollwertige Computer sind und daher viele Funktionen ausführen könnten, doch der Hersteller oder Entwickler das System auf gezielte Funktionen beschränkt. So würde ein Bankautomat mit eingebetteten Linux nur auf die speziellen Aufgaben beschränkt werden, welche ein Bankautomat zu erfüllen hat, obwohl er eigentlich ein vollwertiger Computer sein könnte und viele weitere Funktionen ausführen könnte. Besonders bei eingebetteten Systemen mit Touchscreens könnte eine schier unendliche Anzahl von Funktionen auf dem Display angezeigt und angesteuert werden. Daher kommt die Definition eines eingebetteten Systems auch auf die Definition des Systems vom Hersteller oder Entwickler an.

2.2 Linux

Linux ist die Bezeichnung für ein freies und quelloffenes Betriebssystem, das auf dem Linux-Kernel und dem GNU-Betriebssystem basiert. Hinter der Bezeichnung GNU steht das GNU-Projekt, das 1983 von Richard Stallman ins Leben gerufen wurde. Das GNU-Projekt hat das Ziel, ein vollständig freies Betriebssystem zu entwickeln, das auf freier Software basiert. Im allgemeinen Sprachgebrauch wird Linux oft als Bezeichnung für das gesamte Betriebssystem genutzt, obwohl es sich eigentlich um das Betriebssystem GNU/Linux handelt. Richard Stallman, der Gründer des GNU-Projekts und der Free Software Foundation, erläutert in einem Blog-Beitrag, dass es wichtig wäre, GNU zu erwähnen, da es viele der Kernkomponenten des Betriebssystems zusammen mit dem Linux-Kernel bildet und dass die Bezeichnung GNU/Linux diesen wichtigen Beitrag

anerkennt. Zudem würde es dabei helfen, die Idee des GNU-Projekts und der Free Software Foundation zu verbreiten. (Richard Stallman, o. D.)

In der Arbeit wird die Bezeichnung “Linux” für das Betriebssystem verwendet, da es sich um die allgemein gebräuchliche Bezeichnung handelt und die Bezeichnung “GNU/Linux” in der Literatur und im allgemeinen Sprachgebrauch nicht so verbreitet ist. Besonders im eingebetteten Bereich wird es allgemein als “Eingebettetes Linux” bezeichnet und nicht als “Eingebettetes GNU/Linux”.

Linux gehört zu den weltweit am weitesten verbreiteten Betriebssystemen und zählt zur Gruppe von “unixoide” oder unix-ähnlichen Betriebssystemen, da es Konzepte des 1969 von Bell Laboratories entwickelten Betriebssystems Unix aufgreift.

Nutzer von Linux verwendet oft eine sogenannte “Distribution” von Linux wie Ubuntu. Eine Linux-Distribution, auch als Distro bezeichnet, ist ein vollständiges Betriebssystem, das auf dem Linux-Kernel aufbaut. Der Kernel ist das Herzstück des Betriebssystems. Zusätzlich zum Kernel gibt es weitere wichtige Bestandteile einer Distro, wie Systembibliotheken, Dienstprogramme und Softwareanwendungen. Die Auswahl dieser wird vom Ersteller der Distro getroffen. Distros gibt es in vielen Varianten, jede mit ihren eigenen Zielen, Zielgruppen und Softwareauswahl. Das bedeutet, dass es für viele verschiedene Anwendungsfälle eine passende Distro gibt. (Dikky Ryan Pratama, o. D.)

2.2.1 Eingebettetes Linux

Unter eingebetteten Linux versteht man die Nutzung des Linux-Betriebssystems im Umfeld eingebetteter Systeme. Der einzige wesentliche Unterschied zu dem normalen Linux-Betriebssystem ist dabei, dass das System auf die speziellen Anforderungen eingebetteter Systeme zugeschnitten ist und vor allem sparsamer mit den Ressourcen umgeht. Es hält jedoch nichts davon ab, jede Art von Linux-Distribution auf einem eingebetteten System zu installieren und als “eingebettetes Linux” zu betreiben. Um jedoch das volle Potenzial eines eingebetteten Systems auszuschöpfen, ist es ratsam, eine speziell für eingebettete Systeme entwickelte Linux-Distribution zu verwenden. Yocto ist ein Projekt mit dem eine solche spezielle Distribution für eingebettete Systeme erstellt werden kann.

2.2.2 Das Yocto Projekt

Yocto ist ein Open-Source-Projekt der Linux Foundation, das es ermöglicht, maßgeschneiderte Linux-Distributionen für eingebettete Systeme zu erstellen. Es bietet hierzu eine

Sammlung von Tools, Vorlagen und Methoden, die vor allem auf den Prinzipien der quellbasierten Linux-Distribution Gentoo basieren. Eine quellbasierte Distribution ist eine Linux-Distribution, die aus den Quellcodes der Software zusammengestellt wird. Das bedeutet, dass die Software auf dem Zielgerät aus den Quellcodes kompiliert wird und nicht aus vorkompilierten Binärdateien installiert wird. Dies ermöglicht eine hohe Flexibilität und Anpassbarkeit der Distribution und Software. Yocto erweitert dieses Prinzip speziell auf eingebettete Systeme. Es ist keine fertige Distribution, sondern ein Framework, mit dem eigene Distributionen speziell für eingebettete Systeme erstellt werden können. (Linux Foundation, 2024)

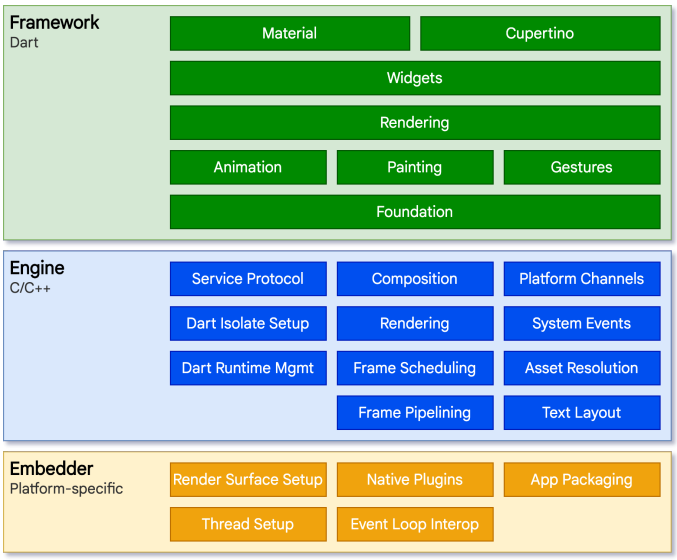
2.3 Flutter

Flutter ist ein von Google entwickeltes quelloffenes Benutzerschnittstellen-Entwicklungs-Kit, welches die effiziente Entwicklung ansprechender Benutzeroberflächen für mobile, Web- und Desktop-Anwendungen ermöglichen soll. Es wurde 2017 von Google veröffentlicht und ist BSD-lizenziert (*The 3-Clause BSD License*, 2011).
Die zunehmende Verbreitung mobiler Geräte und Betriebssysteme stellt die effiziente Anwendungsentwicklung vor Herausforderungen. Plattformübergreifende Frameworks wie Flutter zielen darauf ab, diese Herausforderungen zu mildern, indem sie ein einheitliches Toolset zum Erstellen von Benutzeroberflächen (GUIs) bereitstellen, die nahtlos über verschiedene Plattformen hinweg funktionieren. Dieses Kapitel befasst sich mit der grundlegenden Architektur von Flutter.

2.3.1 Flutters Architektur

Flutter verwendet eine modulare, geschichtete Architektur. Dieses Design besteht aus unabhängigen Bibliotheken, die jeweils von der darunter liegenden Schicht abhängig sind, was lose Kopplung und Flexibilität gewährleistet. Dieser Ansatz fördert die Anpassbarkeit und die Möglichkeit, Framework-Komponenten nach Bedarf zu ersetzen. (*Flutter Architectural Overview*, o. D.)
Die folgende Abbildung 1 zeigt die Architektur von Flutter:

Abbildung 1: Flutter Architektur



Quelle: *Flutter Architectural Overview* (o. D.)

Das Kernelement von Flutter ist die Engine, die in C/C++ geschrieben ist. Sie ist für das Rastern zusammengesetzter Szenen verantwortlich, sobald ein neues Bild gezeichnet werden muss. Sie übernimmt alle Aufgaben zur Darstellung der Benutzeroberfläche.
Auf der Engine liegt das Flutter-Framework, das in der Programmiersprache Dart geschrieben ist. Diese Ebene bietet ein modernes, reaktives Framework, das zum Aufbau der Benutzeroberfläche verwendet wird. In der Entwicklung von Flutter Anwendungen wird meistens nur diese Flutter-Framework Ebene verwendet und die komplette Flutter Anwendung wird meist nur im Framework Teil in Dart geschrieben.
Die niedrigste Ebene ist der Embedder, dessen Aufgabe es ist, die Rendering-Fläche bereitzustellen, die Eingaben zu verarbeiten und Zugriff zu Dart Anwendungs-Snapshots zu ermöglichen. (Garde, 2018) Die Flutter Engine hat hierzu eine C API, welche in einer einzelnen C Header-Datei definiert ist. (The Flutter Authors, o. D.) Diese API Punkte müssen von dem Embedder angesprochen werden.

2.3.2 Deklarative Programmierung in Flutter

Deklarative Programmierung ist ein Programmierparadigma, bei dem der Programmierer den gewünschten Endzustand der Anwendung beschreibt, anstatt die Schritte zu definieren, die zur Erreichung dieses Zustands erforderlich sind. Dieses Paradigma ist in Flutter zentral, da es die Entwicklung von Benutzeroberflächen erleichtert. Die Benutzeroberfläche wird als Baum von Widgets dargestellt, die jeweils den gewünschten Zustand der Benutzer-

oberfläche beschreiben. Wenn sich der Zustand ändert, wird der Baum der Widgets neu gerendert, um den neuen Zustand widerzuspiegeln. (Google Inc., o. D. b)

Auflistung 1: Beispiel für deklarative Programmierung mit Flutter

```
class TextWidget extends StatelessWidget {
  const TextWidget({required this.text, super.key});
  final String text;

  @override
  Widget build(BuildContext context) {
    return Text(text);
  }
}
```

Wie im Code-Beispiel zu erkennen definiert der Entwickler hier ein neues Widget `TextWidget`, welches einen Text als Parameter erhält und diesen als Text-Widget mit dem im Parameter übergebenen Text zurückgibt. Dieses Widget wird dann in der "build" Methode erstellt. An einer gegebenen Stelle im Code kann der Entwickler nun dieses `TextWidget` folgendermaßen verwenden:

Auflistung 2: Nutzung des TextWidgets

```
TextWidget("Hello World");
```

Hier wird der Text "Hello World" als Parameter an das `TextWidget` übergeben und das Widget wird an der beliebigen Stelle in der Anwendung angezeigt wo es so eingesetzt wurde. Wenn nun aber eine dynamische Variable an den Text übergeben werden soll, kann dies einfach durch die folgende Deklaration geschehen:

Auflistung 3: Nutzung des TextWidgets mit dynamischer Variable

```
String text = "Hello World";
TextWidget(text);
```

Falls nun die Variable `text` sich ändert, wird das `TextWidget` automatisch neu gerendert und der neue Text wird angezeigt. Der Entwickler muss sich also nicht darum kümmern, dass das Widget neu gerendert wird, das ist deklarative Programmierung in Flutter. Das Framework übernimmt das für den Entwickler automatisch.

2.4 Dart

Dart ist die offizielle Programmiersprache für die Entwicklung von Flutter Apps und wurde in dieser Arbeit für die Entwicklung der Demo Anwendung verwendet. Dart ist eine objektorientierte, klassenbasierte Programmiersprache mit Garbage Collection und C-ähnlicher Syntax hat. Die Sprache wurde von Lars Bak und Kasper Lund entworfen und bei Google entwickelt. 2011 wurde sie veröffentlicht (Lars Bak, 2011). Damals wurde Dart speziell für die Entwicklung von Webanwendungen entwickelt. Es sollte besonders dabei helfen komplexe und hochleistungsfähige Webanwendungen zu entwickeln. Google sah es aber explizit nicht als Ersatz für JavaScript, sondern als Ergänzung. Im Buch "What is Dart?" von den Google Entwicklern Kathy Walrath und Seth Ladd betonten diese 2012, dass Google nicht erwarten würde, dass Dart JavaScript ersetze, sondern dass Dart und JavaScript nebeneinander existieren könnten und Dart auch von JavaScript profitieren kann, da Dart Code in JavaScript transpiliert werden kann (Walrath, Kathy, 2012, S. 1–2). Hingegen der Erwartungen von Google wurde Dart jedoch nicht so populär wie erwartet. Google nutzte Dart zwar intern für einige Projekte, doch die Programmiersprache blieb weitesgehend unbekannt und wurde nur von wenigen Entwicklern genutzt. Genaue Gründe sind hierfür nicht bekannt, doch ein Grund könnte sein, dass die JavaScript Community zu diesem Zeitpunkt schon sehr groß war und viele Entwickler nicht bereit waren, auf eine komplett neue Programmiersprache umzusteigen. Zudem wurde 2012 TypeScript von Microsoft veröffentlicht, welches ähnliche Probleme wie Dart löst, jedoch besser in die bestehende JavaScript-Community integriert werden konnte, da es JavaScript als Sprache erweitert und Entwickler so nicht eine komplett neue Sprache lernen mussten.

Erst durch die Einführung von Flutter wurde Dart populärer, da es die offizielle Programmiersprache für die Entwicklung von Flutter Apps ist. Flutter wurde intern bei Google anfangs mit JavaScript entwickelt, doch dies führte zu Problemen, da JavaScript nicht die Performance und Stabilität bieten konnte, die Google für die Entwicklung von mobilen Apps erwartete. Daher wurde JavaScript durch Dart ersetzt und das Flutter und Dart Team haben eng zusammengearbeitet, um Dart so zu verbessern, dass es die Anforderungen von Flutter am besten erfüllt.

Heute ist die neueste Version von Dart die Version 3.3.1. Dart ist eine der wenigen Programmiersprachen, die seit ihrer Veröffentlichung sehr starke Veränderungen durchgemacht hat. So wurde zu der JIT (Just in Time) Kompilierung, die Dart ursprünglich verwendete, eine AOT (Ahead of Time) Kompilierung hinzugefügt, die es ermöglicht, Dart Code

in nativen Code zu kompilieren. Dies war notwendig, um die Performance von Dart zu verbessern und um Dart für die Entwicklung von mobilen Apps zu optimieren, zusätzlich auch weil Apple keine JIT Kompilierung auf iOS Geräten erlaubt. Mit Version 3 wurde eine komplette Sound Null Safety zu Dart hinzugefügt. Sound Null Safety bedeutet, dass der komplette Code und auch Code aus allen Bibliotheken vollständig null-sicher sein muss. Mit Version 3 ist dies auch eine Voraussetzung in Dart, nicht null-sicherer Code kann nicht mehr kompiliert werden (Google Inc., o. D. c).

3 Stand der Technik

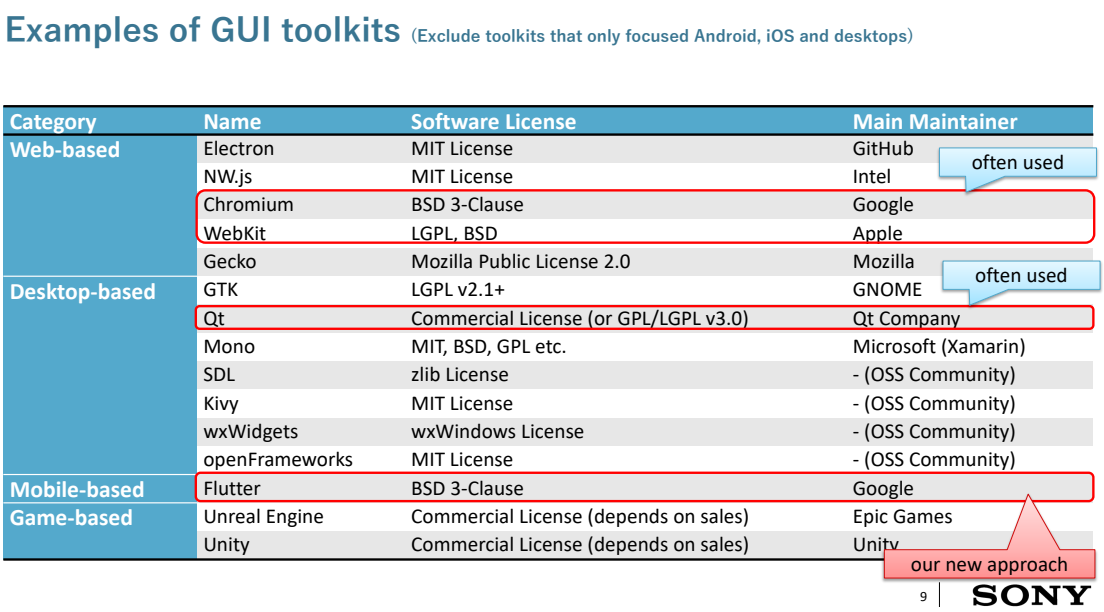
In diesem Kapitel wird der Stand der Technik in Bezug auf die Themenbereiche der Arbeit erläutert. Dabei wird auf die Themenbereiche der Arbeit eingegangen und die aktuelle Situation in der Forschung und Entwicklung dargestellt.

3.1 Toyota

- CES 2024 Automotive Linux mit Flutter (Stephanie Van Ness, 2024)

3.2 Sony

Abbildung 2: Sony



Examples of GUI toolkits (Exclude toolkits that only focused Android, iOS and desktops)

Category	Name	Software License	Main Maintainer
Web-based	Electron	MIT License	GitHub
	NW.js	MIT License	Intel
	Chromium	BSD 3-Clause	Google
	WebKit	LGPL, BSD	Apple
	Gecko	Mozilla Public License 2.0	Mozilla
Desktop-based	GTK	LGPL v2.1+	GNOME
	Qt	Commercial License (or GPL/LGPL v3.0)	Qt Company
	Mono	MIT, BSD, GPL etc.	Microsoft (Xamarin)
	SDL	zlib License	- (OSS Community)
	Kivy	MIT License	- (OSS Community)
	wxWidgets	wxWindows License	- (OSS Community)
	openFrameworks	MIT License	- (OSS Community)
Mobile-based	Flutter	BSD 3-Clause	Google
Game-based	Unreal Engine	Commercial License (depends on sales)	Epic Games
	Unity	Commercial License (depends on sales)	Unity

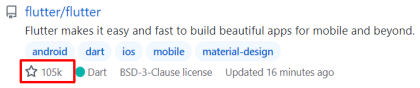
Quelle: Matsubayashi (2020)

3.3 Texas Instruments

Abbildung 3: This is a figure.

Why we choose Flutter?

- You can easily create a modern UI like a mobile app
- Supporting cross-platform (Desktop, Mobile, Web)
- Flutter is popular OSS and there are a lot of information
- Flutter is natively compiled applications (Fast!)



- Flutter provides the custom embedder API-layer for specific platforms
  - https://github.com/flutter/flutter/wiki/Custom-Flutter-Engine-Embedders
- Fewer library dependencies (Flutter Engine)
  - Basically your platform needs only OpenGL/EGL library
- Software license is BSD 3-Clause

21 | SONY

4 Analyse und Design

4.1 Hardware

4.1.1 Hardware für die Demo Anwendung

- Raspberry Pi 4
- Touchscreen 15 Zoll (1920x1080)
- Raspberry Pi 0 2W
- Touchscreen 7 Zoll (1024x600)
- RGB Matrix
- Led

4.1.2 Build Hardware

- AMD 6800U mit 16GB RAM, Linux

4.2 Anforderungen

Abbildung 4: Skizzen der Anwendung lorem ipsum

4.3 Architektur

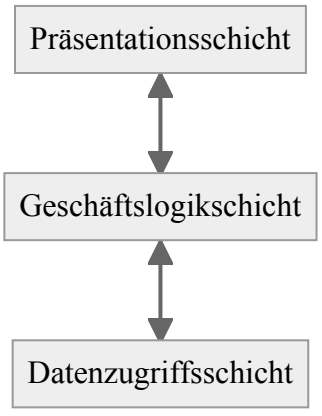
4.3.1 Analyse

4.3.2 Design

Flutter gibt Entwicklern eine große Freiheit bei der Auswahl der Architektur für ihre Anwendungen. Es gibt keine festen Regeln oder Vorschriften zur Ordnerstruktur oder anderen Konventionen bezüglich der Architektur.
Als Architektur für die Anwendung wurde eine drei-schichtige Architektur gewählt. Eine Schichtenarchitektur ist nicht nur Teil der Architektur von Flutter selbst, sondern ist auch eine bewährte Architektur für die Entwicklung von Flutter Anwendungen. Die drei Haupt-Schichten der Architektur der Anwendung sind die Präsentationsschicht, die Geschäftsschicht und die Datenzugriffsschicht. In der Präsentationsschicht werden die Benutzeroberfläche und die Benutzerinteraktionen implementiert. Die Geschäftsschicht ist für die Implementierung der Geschäftslogik verantwortlich. Die Datenzugriffsschicht ist für den Zugriff auf die Datenbank und die Datenverarbeitung verantwortlich. Die drei Schichten
Die drei Schichten sind die **Präsentationsschicht**, die **Geschäftslogikschicht** und die **Datenzugriffsschicht**. Die Präsentationsschicht ist für die Darstellung der Benutzeroberfläche verantwortlich. Die Anwendungsschicht ist für die Geschäftslogik verantwortlich. Die Datenzugriffsschicht ist für den Zugriff auf die Datenbank verantwortlich. Die drei Schichten sind voneinander unabhängig und kommunizieren über definierte Schnittstellen. Die drei Schichten sind in der Abbildung 3.1 dargestellt.

- three-layer architecture (Hajian, 2024, S. 219)

Abbildung 5: 3-Schichtenarchitektur der Flutter Anwendung



Quelle: eigene Darstellung

4.4 Ordner Konventionen

4.4.1 Layers-First Ordnerstruktur

- schneller und einfacher Start
- einfach zu verstehen
- sehr unübersichtlich, sobald die Anwendung wächst
- zusammengehörende Dateien für ein Feature über das Projekt verteilt

4.4.2 Feature-First Ordnerstruktur

- Dateien, die zu einem Feature gehören, sind an einem Ort zu finden
- Layers in Feature
- Einfache Kommunikation
- Verständnis, was ein Feature ist, muss im Team trainiert werden

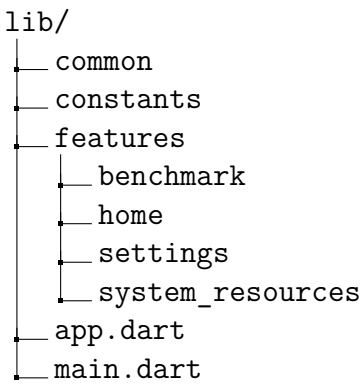


Abbildung 6: Example caption

Quelle: hier Quellangabe

eine Klasse pro File, Ausnahmen eng zusammengehörige Klassen wie zB. Stateful Widgets
state Architektur
android architecture

4.4.3 freezed package

generiert den notwendigen Code einer unveränderlichen Klasse:

- Konstruktor
- toString, equlas, hashCode Methode
- copyWith Methode
- JSON Serialisierung

4.4.4 State Management

Für das State Management wurde Riverpod gewählt. # Implementierung
Im folgenden Kapitel wird die Implementierung des Projektes beschrieben. Dabei wird auf die Entwicklungsumgebung, die Verwendung von Flutter und Dart, die Einrichtung des Raspberry Pi und die Implementierung der Software eingegangen.

4.5 Entwicklungsumgebung

4.5.1 Visual Studio Code

Für die Implementierung der Fallstudie wurde die plattformübergreifende Open-Source-Entwicklungsumgebung Visual Studio Code (VS Code) von Microsoft verwendet. VS Code ist eine sehr beliebte Entwicklungsumgebung für die Entwicklung von Webanwendungen, mobilen Anwendungen und Desktopanwendungen. Die Entwicklungsumgebung bietet eine Vielzahl von Erweiterungen, die die Entwicklung von Anwendungen in verschiedenen Programmiersprachen und Frameworks unterstützen.

4.5.2 Flutter Versionsverwaltung

Das Versionsmanagement von Flutter ist standardmäßig sehr einfach gehalten. Es gibt nur eine globale Installation von Flutter, die manuell auf dem System installiert wird. Diese globale Installation arbeitet mit Git und nutzt verschiedene Branches für die verschiedenen Versionen von Flutter, diese Branches nennt Google Channels. Die Channels sind **stable**, **beta** und **master**. Die **stable** Version ist immer die aktuellste stabile Version von Flutter, diese wird für die Produktion und für die meisten Entwickler empfohlen. Die **beta** Version ist die neueste stabile Version von Flutter, welche neue Features beinhalten kann und nicht für die Produktion empfohlen wird. Diese Version wird später zur nächsten **stable** Version höhergestuft und wird von Google jedoch auch bereits extensiv getestet und sollte sehr stabil sein. Die **master** Version ist die neueste Entwicklungsversion von Flutter, diese Version ist nicht stabil und wird nicht für die Produktion empfohlen. Diese Version beinhaltet alle neuesten Commits welche von Entwicklern in das Flutter Repository gepusht wurden.

Hierbei gibt es jedoch einige Probleme, die durch die globale Installation von Flutter entstehen. Die globale Installation von Flutter ist nicht für die Arbeit mit verschiedenen Projekten und verschiedenen Versionen von Flutter ausgelegt. Es gibt keine Möglichkeit, verschiedene Versionen von Flutter zu installieren und zu verwalten. Außerdem gibt es keine einfache Möglichkeit, eine bestimmte Version von Flutter zu installieren, es wird immer die neueste **stable**, **beta** oder **master** Version installiert. So wurde in der Entwicklung der Fallstudie Flutter Version 3.19.2 verwendet, welche jedoch zu einem späteren Zeitpunkt nicht mehr aktuell ist.
Zur Lösung dieser Probleme gibt es verschiedene Versionsverwaltungs-Tools für Flutter. Die bekanntesten sind FVM (Flutter Version Management) und Puro.
FVM ist ein simples CLI-Tool, welches es ermöglicht, verschiedene Versionen von Flutter zu installieren und zu verwalten. Puro dahingegen ist ein weiteres CLI-Tool, welches neben der Versionsverwaltung von Flutter auch eine Integration mit den bekannten Entwicklungsumgebungen VS Code und IntelliJ bietet.
In der Fallstudie wurde Puro verwendet und das Projekt ist so konfiguriert, dass es die Version 3.19.2 von Flutter verwendet.

4.5.3 Raspberry Pi OS

- Einrichtung als Entwicklungsbetriebssystem mit Flutter Hot-Reload
- für rapide Entwicklung und Testen

4.6 Implementierung der Features

Im Folgenden wird die Implementierung der einzelnen Features des Projektes beschrieben.

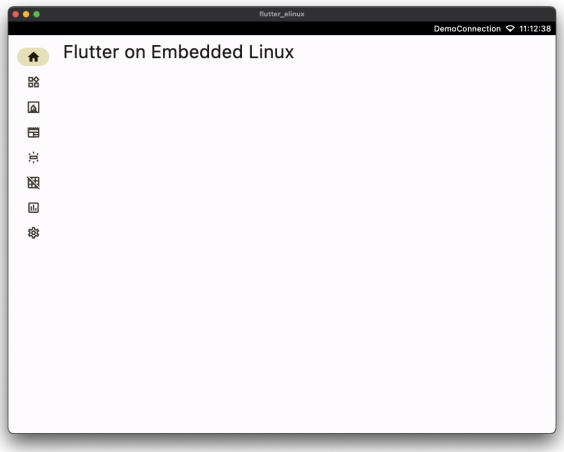
4.6.1 Home Feature

Das Home-Feature ist die Startseite der App. Es besteht aus einem einfachen Screen, der nur den Text “Flutter on Embedded Linux” anzeigt.

Abbildung 7: Ordnerstruktur des Home-Features

```
home/
└── presentation/
    └── home_screen.dart
```

Abbildung 8: Home Screen



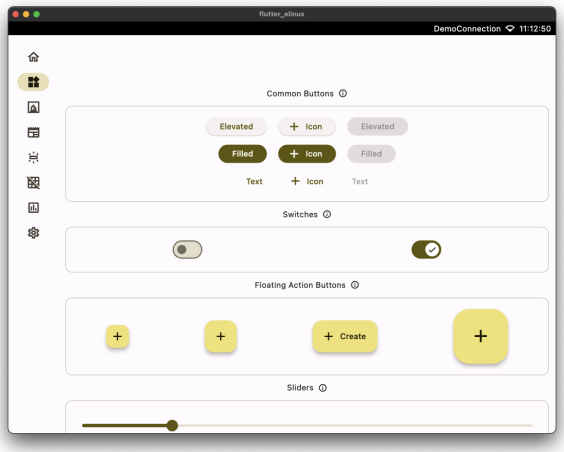
4.6.2 Material Demo Feature

Das Material Demo Feature zeigt eine Auswahl von Material Design Widgets. Es besteht aus einem Screen, der eine Liste von Widgets anzeigt.

Abbildung 9: Ordnerstruktur des Material Demo Features

```
material_demo/
└── presentation/
    ├── material_demo_screen.dart
    └── widgets/
        └── component_decoration.dart
```

Abbildung 10: Material Demo Screen



4.6.3 Benchmark

Das Benchmark Feature zeigt verschiedene Benchmarks, die die Leistungsfähigkeit von Flutter auf Embedded Linux demonstrieren.

Abbildung 11: Example caption

```
benchmark/
├── business/
│   └── flame/
│       ├── rogue_shooter/
│       └── sprite_renderer/
├── data/
│   ├── models/
│   │   ├── benchmark.dart
│   │   └── ember.dart
│   └── benchmarks_list.dart
└── presentation/
    ├── widgets/
    │   └── benchmark_card.dart
    ├── benchmark_screen.dart
    ├── map_benchmark_screen.dart
    ├── rogue_shooter_screen.dart
    ├── sprite_benchmark_screen.dart
    ├── video_benchmark_screen.dart
    └── vsync_benchmark_screen.dart
```

Abbildung 12: Home Screen

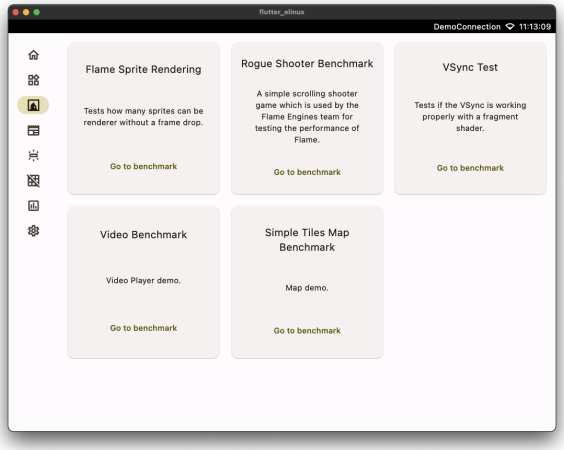


Abbildung 13: Home Screen

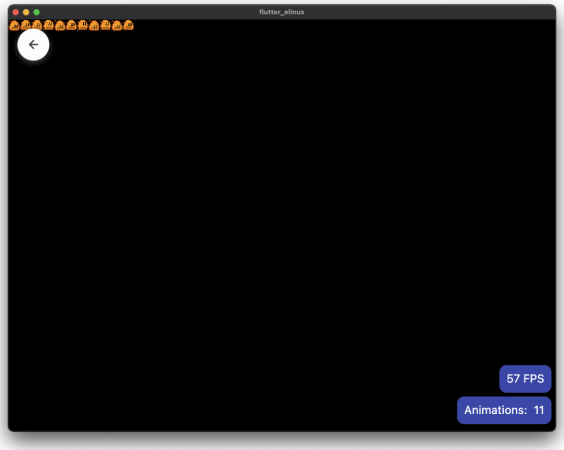


Abbildung 14: Home Screen

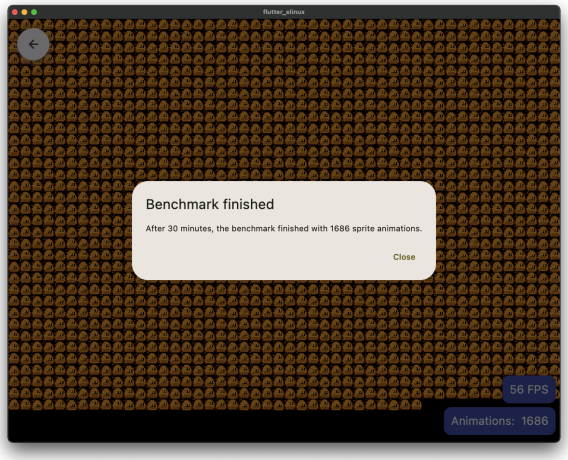


Abbildung 15: Home Screen

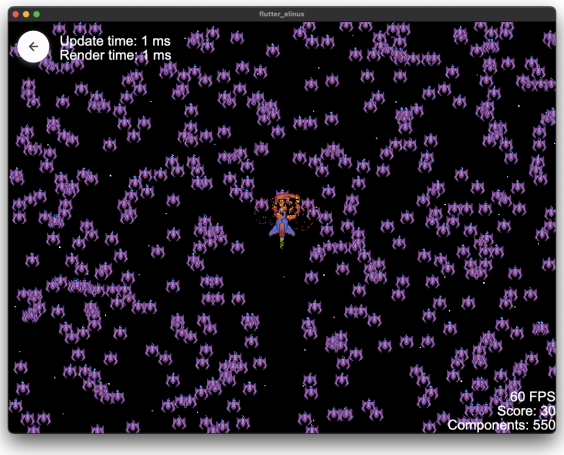


Abbildung 16: Home Screen

Abbildung 17: Home Screen

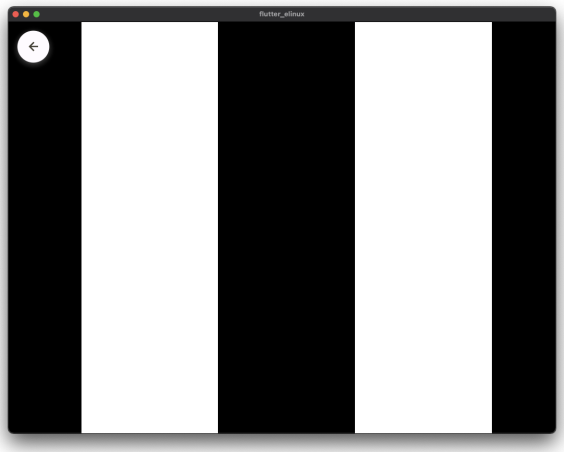


Abbildung 18: Home Screen

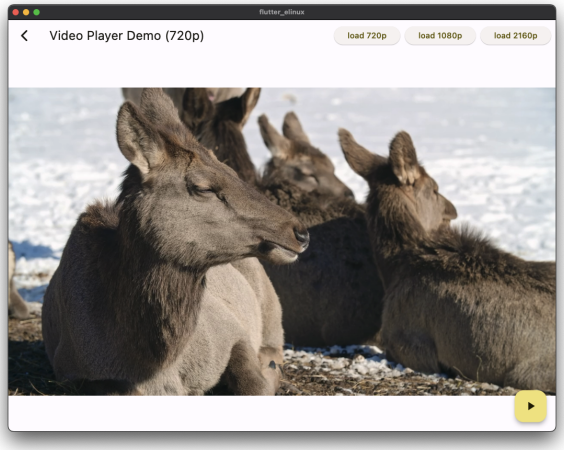
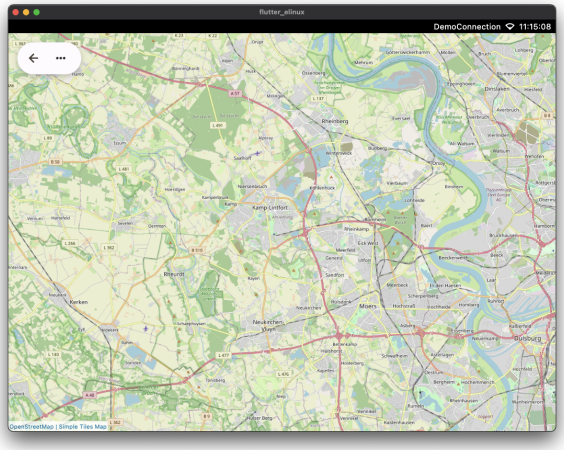


Abbildung 19: Home Screen



4.6.4 News

Das News Feature zeigt eine Liste von Nachrichten der Tagesschau.

Abbildung 20: Example caption

```
news_api/
├── business/
│   ├── news_controller.dart
│   └── scroll_state_controller.dart
├── data/
│   ├── models/
│   │   ├── news.dart
│   │   ├── region_news.dart
│   │   ├── teaser_image_url.dart
│   │   └── teaser_image.dart
│   └── repository/
│       └── news_repository.dart
└── presentation/
    ├── widgets/
    │   ├── news_grid_item.dart
    │   ├── news_grid.dart
    │   ├── news_list_item.dart
    │   └── news_list.dart
    └── news_screen.dart
```

Abbildung 21: Home Screen

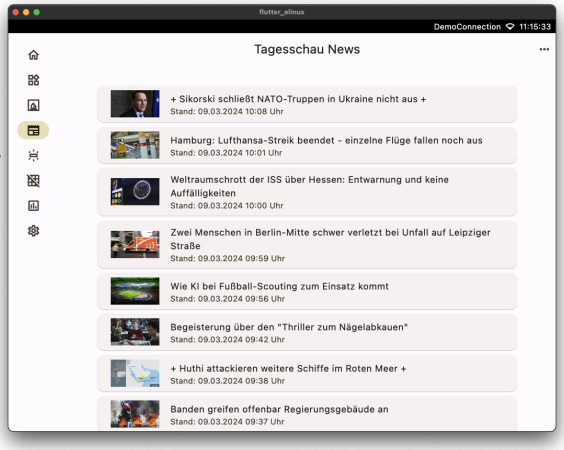
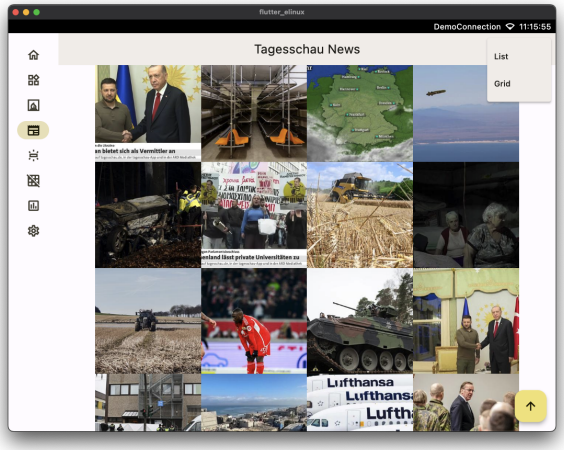


Abbildung 22: Home Screen



4.6.4.1 **Freezed** wurde verwendet, um die Modelle zu generieren.

Auflistung 4: Implementierung der News Klasse

```dart
import 'package:flutter/foundation.dart';
import 'package:freezed_annotation/freezed_annotation.dart';

import 'teaser_image.dart';

part 'news.freezed.dart';
part 'news.g.dart';

@freezed
class News with _$News {
  const factory News({
    String? title,
    TeaserImage? teaserImage,
    TeaserImage? brandingImage,
    String? date,
    String? type,
  }) = _News;

  factory News.fromJson(Map<String, dynamic> json) => _$NewsFromJson(json);
}
```

Auflistung 5: Implementierung des News Controllers

```dart
import 'package:dio/dio.dart';
import 'package:flutter/foundation.dart';

import '../../../../constants/api_urls.dart';

/// The news repository
class NewsRepository {
  /// The Dio instance
  final Dio _dio = Dio(
    BaseOptions(
      baseUrl: TagesschauApiUrls.main,
      responseType: ResponseType.json,
    ),
  );

  /// Fetch the news
```

```
  Future<String> fetchNews() async {
    try {
      final Response<String> response = await _dio.get('/news');
      if (response.statusCode == 200) {
        if (response.data != null) {
          return response.data!;
        } else {
          throw Exception('Failed to fetch news: response data is null');
        }
      }
      throw Exception(
          'Failed to fetch news. Status code: ${response.statusCode}.');
    } catch (e) {
      debugPrint("Error fetching news: $e");
      rethrow;
    }
  }

  /// Fetch the next page of news
  fetchNewsPage(String url) async {
    final date = _extractDate(url);
    final Response<String> response =
        await _dio.get('/news', queryParameters: {'date': date});
    return response.data;
  }

  /// private method to extract the date from the url
  String _extractDate(String url) {
    final uri = Uri.parse(url);
    final queryParameters = uri.queryParameters;
    return queryParameters['date']!;
  }
}
```

Auflistung 6: Implementierung des News Controllers

```
import 'dart:convert';

import 'package:flutter/foundation.dart';
import 'package:riverpod_annotation/riverpod_annotation.dart';

```

```dart
import '../data/repository/news_repository.dart';
import '../data/models/region_news.dart';

part 'news_controller.g.dart';

/// The news controller
@riverpod
class NewsController extends _$NewsController {
  @override
  Future<RegionNews> build() async {
    try {
      final String newsResponse = await NewsRepository().fetchNews();
      final Map<String, dynamic> jsonData = json.decode(newsResponse);
      final RegionNews news = RegionNews.fromJson(jsonData);
      return news;
    } catch (e) {
      rethrow;
    }
  }

  /// Append the next page of news to the current news list
  Future<void> appendPage() async {
    try {
      final RegionNews currentNews =
          state.asData?.value ?? const RegionNews(news: []);
      final newsResponse =
          await NewsRepository().fetchNewsPage(currentNews.nextPage ?? 'null');
      final Map<String, dynamic> jsonData = jsonDecode(newsResponse);
      final RegionNews nextNews = RegionNews.fromJson(jsonData);
      final RegionNews appendNews =
          currentNews.copyWith(nextPage: nextNews.nextPage, news: [
        ...currentNews.news,
        ...nextNews.news,
      ]);
      state = AsyncData(appendNews);
    } catch (e) {
      debugPrint(e.toString());
    }
  }
}
```

4.6.5 Led Morse

Abbildung 23: Example caption

```
morse_led/
├── business/
│   └── service/
│       └── morse_service.dart
└── presentation/
    └── led_screen.dart
```

Abbildung 24: Home Screen

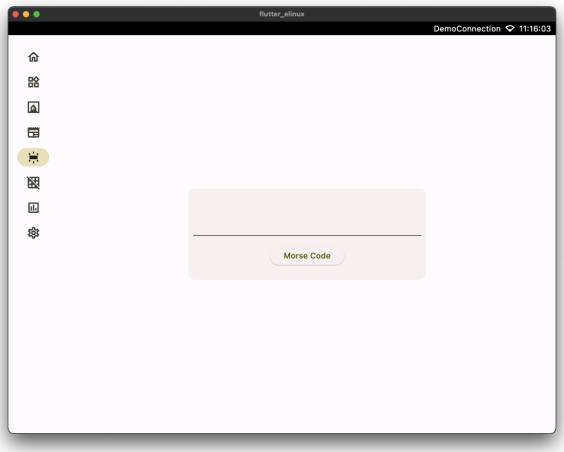
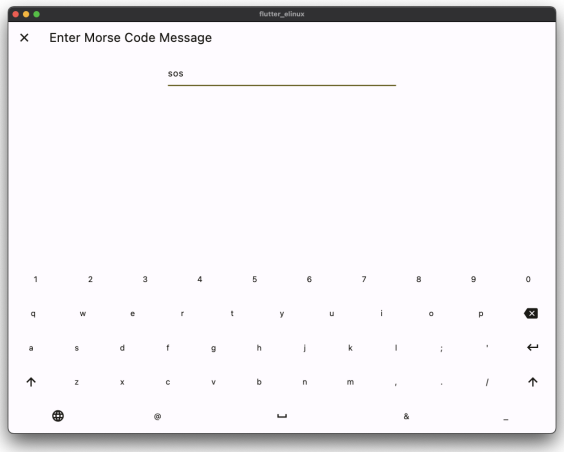


Abbildung 25: Home Screen



Auflistung 7: Implementierung des Morse Service

```dart
import 'package:flutter_gpiod/flutter_gpiod.dart';

class MorseService {
  final _chips = FlutterGpiod.instance.chips;

  printMessage(String message) async {
    final chip = _getRaspberryGPIO();
    final ledPin = chip.lines[18];
    ledPin.requestOutput(consumer: "morse code led", initialValue: false);
    final List<String> chars = message.toUpperCase().split('');
    const int pulseDuration = 100;

    for (var char in chars) {
      final List<String> morseCode = _morseAlphabet[char]?.split('') ?? ['*'];
      for (var code in morseCode) {
```

```
        print('CODE: $code from $morseCode');
        if (code == '*') {
          await Future.delayed(const Duration(milliseconds: pulseDuration * 7));
          break;
        }
        ledPin.setValue(true);
        print('ON');
        if (code == '.') {
          await Future.delayed(const Duration(milliseconds: pulseDuration));
        } else if (code == '-') {
          await Future.delayed(const Duration(milliseconds: pulseDuration * 3));
        }
        print('OFF');
        ledPin.setValue(false);
        // pause between letters
        await Future.delayed(const Duration(milliseconds: pulseDuration));
      }
      print("SPACE BETWEEN LETTERS");
      // space between letters is 3*pulseDuration
      await Future.delayed(const Duration(milliseconds: pulseDuration * 3));
    }

    ledPin.release();
    return true;
  }

  GpioChip _getRaspberryGPIO() {
    /// Retrieve the line with index 23 of the first chip.
    /// This is BCM pin 23 for the Raspberry Pi.
    ///
    /// I recommend finding the chip you want
    /// based on the chip label, as is done here.
    ///
    /// In this example, we search for the main Raspberry Pi GPIO chip and then
    /// retrieve the line with index 23 of it. So [line] is GPIO pin BCM 23.
    ///
    /// The main GPIO chip is called `pinctrl-bcm2711` on Pi 4 and `pinctrl-
    ↪ bcm2835`
    /// on older Raspberry Pis and it was also called that way on Pi 4 with older
    /// kernel versions.
```

```
    final chip = _chips.singleWhere(
      (chip) => chip.label == 'pinctrl-bcm2711',
      orElse: () =>
          _chips.singleWhere((chip) => chip.label == 'pinctrl-bcm2835'),
    );
    return chip;
  }

  // International Morse Code
  final Map<String, String> _morseAlphabet = {
    'A': '.-',
    'B': '-...',
    'C': '-.-.',
    'D': '-..',
    'E': '.',
    'F': '..-.',
    'G': '--.',
    'H': '....',
    'I': '..',
    'J': '.---',
    'K': '-.-',
    'L': '.-..',
    'M': '--',
    'N': '-.',
    'O': '---',
    'P': '.--.',
    'Q': '--.-',
    'R': '.-.',
    'S': '...',
    'T': '-',
    'U': '..-',
    'V': '...-',
    'W': '.--',
    'X': '-..-',
    'Y': '-.--',
    'Z': '--..',
    '0': '-----',
    '1': '.----',
    '2': '..---',
    '3': '...--',
```

```
    '4': '....-',
    '5': '.....',
    '6': '-....',
    '7': '--...',
    '8': '---..',
    '9': '----.',
  };
}
```

4.6.6 Matrix

Abbildung 26: Example caption

```
matrix_rgb/
├── business/
│   ├── service/
│   │   └── matrix_service.dart
│   └── utils/
│       └── find_closest_color_index.dart
├── data/
│   └── models/
│       └── pixel.dart
└── presentation/
    ├── widgets/
    │   └── pixel_painter.dart
    └── matrix_screen.dart
```

Abbildung 27: Home Screen

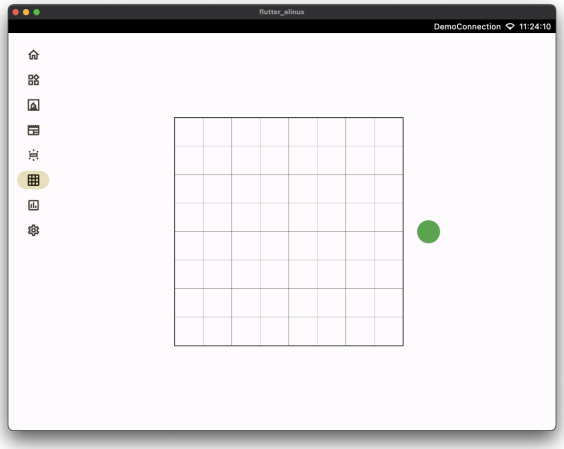


Abbildung 28: Home Screen

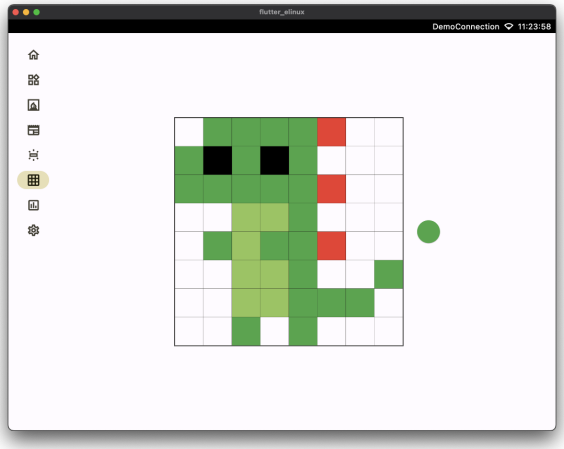


Abbildung 29: Home Screen

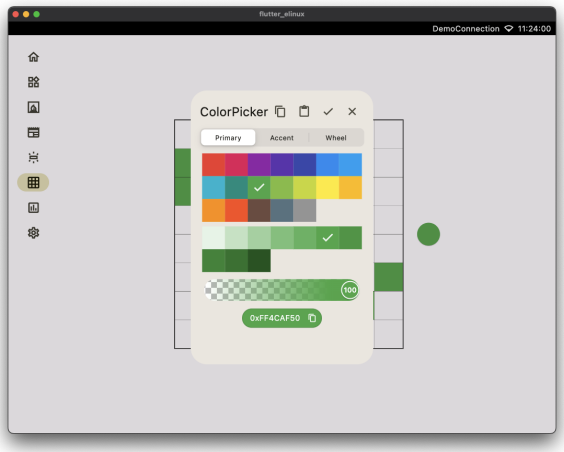
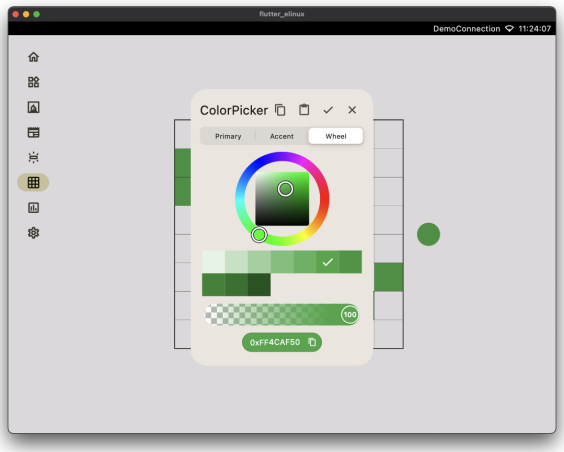


Abbildung 30: Home Screen



Auflistung 8: Implementierung des Matrix Service

```dart
import 'package:dart_periphery/dart_periphery.dart';
import 'package:flutter/foundation.dart';

// linux_spidev was tried out here beforehand,
// but it didn't work and was replaced by dart_periphery
// import 'package:linux_spidev/linux_spidev.dart';

class MatrixService {
  final spi = SPI(0, 0, SPImode.mode0, 35000);
  // final spidev = Spidev.fromBusDevNrs(0, 0);

  void setLedsOn(List<int> pixels) async {
    // final handle = spidev.open();
    try {
      int refresh = 1;
      while (refresh != 0) {
        spi.transfer(pixels, false);
        await Future.delayed(const Duration(milliseconds: 100));
        refresh--;
      }
    } catch (e) {
      debugPrint(e.toString());
    } finally {
      // clean up the resource
      spi.dispose();
    }
```

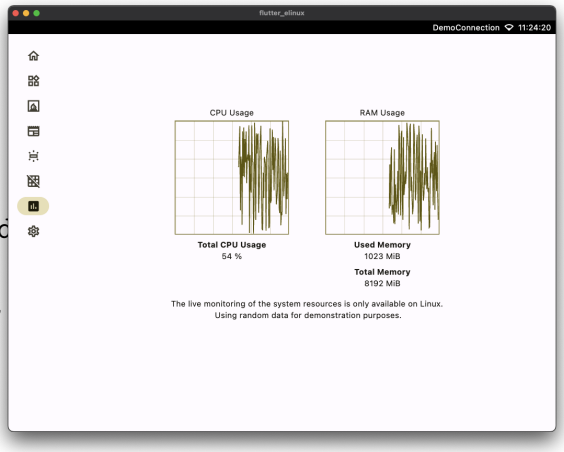
```
27     }
28 }
```

4.6.7 System Informationen

Abbildung 31: Example caption

```
system_resources/
├── business/
│   ├── service/
│   │   └── memory_service.dart
│   └── utils/
│       └── get_cpu_usage_percentage.d
└── presentation/
    └── system_resources_screen.dart
```

Abbildung 32: Home Screen



Auflistung 9: Implementierung des Memory Service

```dart
import 'dart:convert';
import 'dart:io';
import 'dart:math';

import 'package:riverpod_annotation/riverpod_annotation.dart';

part 'memory_service.g.dart';

// riverpod stream builder docs:
// https://riverpod.dev/docs/providers/stream_provider

@riverpod
class MemoryService extends _$MemoryService {
  @override
  Future<double> build() async {
    if (Platform.isLinux) {
      final processMemory = await Process.start(
        'grep',
        [
          'MemTotal',
          '/proc/meminfo',
        ],
      );
```

```
    final awkMemInKB = await Process.start('awk', ['{print \$2}']);
    processMemory.stdout.pipe(awkMemInKB.stdin);

    var result = await awkMemInKB.stdout.transform(utf8.decoder).join();

    return double.parse(result) / 1024;
  } else {
    return 8192;
  }
}

Stream<double> freeMemory() async* {
  while (true) {
    if (Platform.isLinux) {
      await Future.delayed(const Duration(milliseconds: 100));
      final meminfo = await Process.start(
        'cat',
        [
          '/proc/meminfo',
        ],
      );
      String output = await meminfo.stdout.transform(utf8.decoder).join();

      Iterable<String> lines = LineSplitter.split(output);

      int memTotal = _getKeyValue(list: lines, key: 'MemTotal');
      int shMem = _getKeyValue(list: lines, key: 'Shmem');
      int memFree = _getKeyValue(list: lines, key: 'MemFree');
      int buffers = _getKeyValue(list: lines, key: 'Buffers');
      int cached = _getKeyValue(list: lines, key: 'Cached');
      int sReclaimable = _getKeyValue(list: lines, key: 'SReclaimable');

      //  Used memory is calculated using the following "formula":
      // MemUsed = MemTotal + Shmem - MemFree - Buffers - Cached - SReclaimable
      // Source: https://github.com/dylanaraps/pfetch/blob/master/pfetch
      // Source: https://github.com/KittyKatt/screenFetch/issues/386

      final double memoryUsed =
          (memTotal + shMem - memFree - buffers - cached - sReclaimable) /
              1024;
```

```
64
65          yield memoryUsed;
66        } else {
67          await Future.delayed(const Duration(milliseconds: 100));
68          yield Random().nextDouble() * 8192;
69        }
70      }
71    }
72  }
73
74  /// get the key value from a /proc/meminfo line
75  int _getKeyValue({required Iterable<String> list, required String key}) {
76    final String keyLine = list.firstWhere((element) => element.startsWith(key));
77    return int.parse(keyLine.replaceAll(RegExp(r"\s+"), " ").split(' ')[1]);
78  }
```

4.6.8 Einstellungen

Abbildung 33: Example caption

```
settings/
├── business/
│   └── wifi_controller.dart
└── presentation/
    ├── settings_screen/
    │   ├── widgets/
    │   │   └── color_seed_button.dart
    │   └── settings_screen.dart
    └── wifi_settings_screen/
        ├── widgets/
        │   ├── ap_pw_dialog.dart
        │   ├── ap_tile_demo.dart
        │   ├── ap_tile.dart
        │   └── wifi_bar_icon.dart
        ├── accesspoint_screen.dart
        └── wifi_settings_screen.dart
```

Abbildung 34: Home Screen

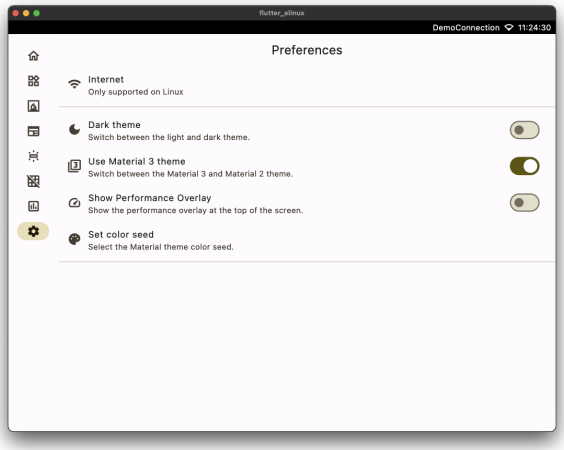


Abbildung 35: Home Screen

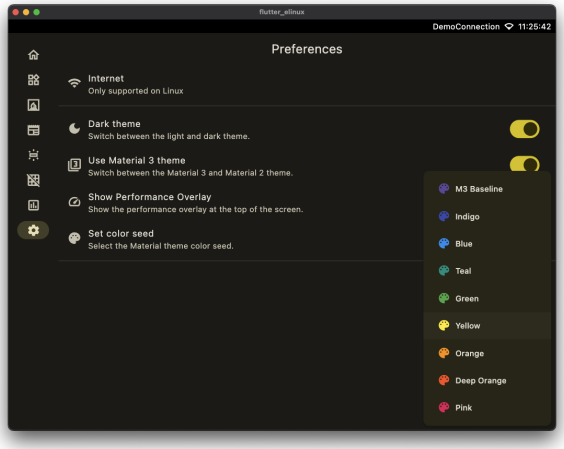


Abbildung 36: Home Screen

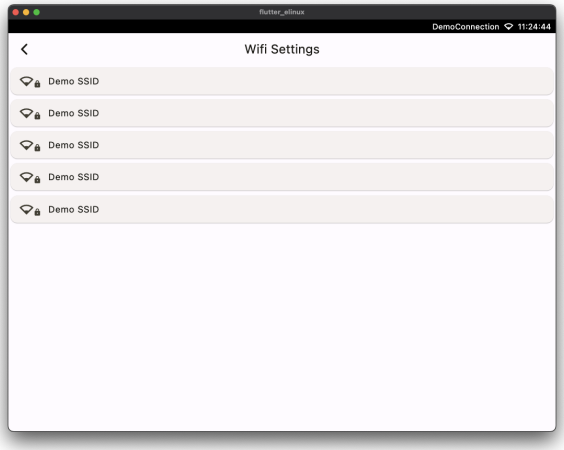


Abbildung 37: Home Screen

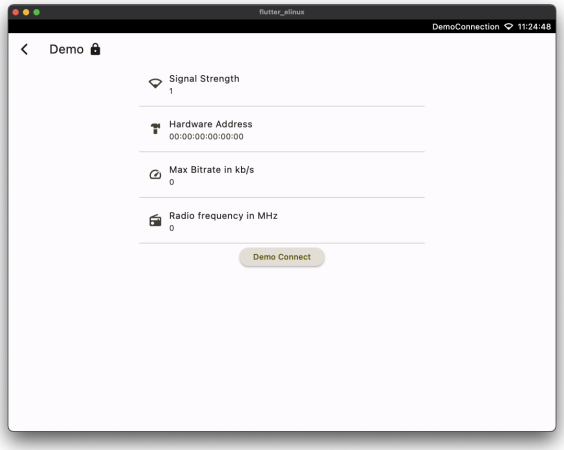


Abbildung 38: Home Screen

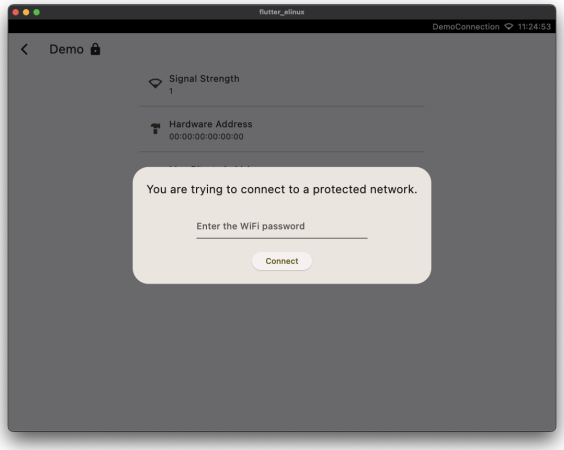
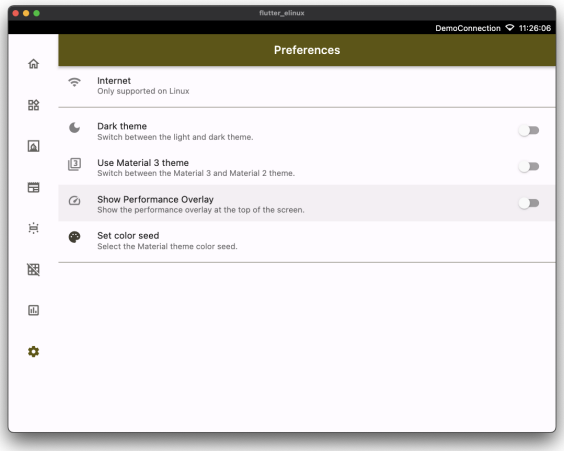


Abbildung 39: Home Screen

Abbildung 40: Home Screen



Auflistung 10: Implementierung des Wifi Controllers

```dart
import 'dart:async';

import 'package:dbus/dbus.dart';
import 'package:flutter/material.dart';
import 'package:nm/nm.dart';
import 'package:riverpod_annotation/riverpod_annotation.dart';

part 'wifi_controller.g.dart';

@riverpod
class WifiController extends _$WifiController {
  @override
  Future<NetworkManagerClient> build() async {
    try {
      NetworkManagerClient client = NetworkManagerClient();
      await client.connect();

      return client;
    } catch (e) {
      return Future.error(e);
    }
  }

  NetworkManagerDevice get _device => state.value!.devices
      .firstWhere((d) => d.deviceType == NetworkManagerDeviceType.wifi);

  Stream<(NetworkManagerDevice, NetworkManagerClient)> wifiStream() {
    return Stream.periodic(const Duration(milliseconds: 100), (_) {
      return (_device, state.value!);
    });
  }

  void disconnectFromWifiNetwork({
    required NetworkManagerAccessPoint accessPoint,
  }) async {
    try {
      final device = _device;
      var active = await state.value!.activateConnection(device: device);
      await state.value!.deactivateConnection(active);
```

```
    } catch (e) {
      debugPrint(e.toString());
    }
  }

  void connectToWifiNetwork({
    required NetworkManagerAccessPoint accessPoint,
    String? psk,
  }) async {
    try {
      final device = _device;
      // Has password
      if (psk != null) {
        await state.value!.addAndActivateConnection(
          device: device,
          accessPoint: accessPoint,
          connection: {
            '802-11-wireless-security': {
              'key-mgmt': const DBusString('wpa-psk'),
              'psk': DBusString(psk)
            }
          },
        );
      } else {
        await state.value!
            .addAndActivateConnection(device: device, accessPoint: accessPoint);
      }
    } catch (e) {
      debugPrint(e.toString());
    }
  }
}
```

4.7 Yocto Project

Kein source-code, nur meta daten. Die Regeln, wie der Source-Code gebaut wird, werden in Rezepten festgelegt.
docker als build-umgebung. https://hub.docker.com/r/crops/poky
poky = yocto. Yocto Project basierend auf Poky Linux. Ausgesprochen “Pocky” wie das japanische Knabberzeug.
Alle 6 Monate neue Version. Alle 2 Jahre Long-Term-Support Version.
Distro: How I want to put my system together. Machine: the board i want to build for. Image: the selection of packages i want.
build cross-compiler. tools for build time. compile code for target machine.

4.7.1 Layers

Layer contains bunch of recipes.
3 layers: bsp: board support package defines a machine and related board-specific packages contains conf/machine/[MACHINE].conf
distribution: defines a DISTRO such as Poky contains conf/distro/[DISTRO].conf
Software: everything else contains neither conf/machine/[MACHINE].conf nor conf/-distro/[DISTRO].conf libraries, e.g. qt5 languages, e.g. Java tools, e.g. virtualisation or selinux

4.7.2 Recipes

contain instructions on how to fetch, configure, compile and install a software component the body contains BitBake metadata (assignment of variables, mostly); the tasks are written in shell script or Python Recipe files have a suffix .bb may be extended with append recipes with .bbappend suffix
majority of recipes produce packages. /tmp/deploy/rpm/.. often one recipe produces several packages.
IMAGE_INSTALL:append

4.7.3 Images

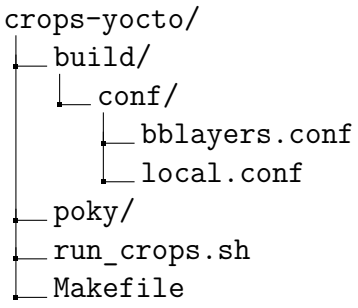
core-image-base: A console-only image that fully supports the target device hardware. core-image-full-cmdline: A console-only image with more full-featured Linux system functionality installed. core-image-minimal: A small image just capable of allowing a

device to boot. core-image-weston: A very basic Wayland image with a terminal. This image provides the Wayland protocol libraries and the reference Weston compositor. For more information, see the “Using Wayland and Weston” section in the Yocto Project Development Tasks Manual.

Tabelle 1: Example caption

Meta-Layer	Branch
https://github.com/kraj/meta-clang	312ff1c39b1bf5d35c0321e873417eb013cea477
https://github.com/meta-flutter/meta-flutter	82e167a7f13161a4ee9c5a426f13df79987d1055
https://github.com/openembedded/meta-openembedded	fda737ec0cc1d2a5217548a560074a8e4d5ec580
https://github.com/agherzan/meta-raspberrypi	9dc6673d41044f1174551120ce63501421dbcd85
https://github.com/jwinarske/meta-vulkan	ceb47bd0ed2a9f657fdae48a901e8a41ba697e74
https://codeberg.org/flk/meta-wayland	cb22939f751c74baa51d9474ba6e8ba647e99756

Abbildung 41: Example caption



5 Zusammenfassung

5.1 Benchmark

Flutter Embedder	Ergebnisse (Animationen nach 30 Minuten)	Durchschnittlich (gerundet)
Flutter-Pi	537, 543, 544	541
Flutter-Auto mit Cage	1025, 1163, 1129	1106

Abbildung 42: Example caption

```
poky/
├── meta/
├── meta-poky/
├── meta-yocto-bsp/
├── meta-selftest/
├── meta-skeleton/
├── mea-clang/
├── meta-flutter/
├── meta-flutter-apps/
├── meta-flutter-elinux/
├── meta-openembedded/
├── meta-raspberrypi/
├── meta-wayland/
└── meta-vulkan/
```

Abbildung 43: Example caption

```
meta-flutter-apps/
└── recipes-graphics/
    └── flutter-apps/
        └── third-party/
            ├── flutter-elinux_git.bb/
            └── flutter-wonders_git.bb/
```

6 Literaturverzeichnis

Dikky Ryan Pratama (o. D.) *What Is Linux Distribution?* Alibaba Cloud Community. Verfügbar unter: https://www.alibabacloud.com/blog/what-is-linux-distribution_599979 (Zugegriffen: 22 Februar 2024).

Flutter Architectural Overview (o. D.). Verfügbar unter: https://docs.flutter.dev/resources/architectural-overview (Zugegriffen: 12 Februar 2024).

Ganssle, J. (2008) *What's Embedded?* Embedded.com. Verfügbar unter: https://www.embedded.com/whats-embedded/ (Zugegriffen: 22 Februar 2024).

Garde, C. (2018) *Flutter on Raspberry Pi (Mostly) from Scratch*. Flutter. Verfügbar unter: https://medium.com/flutter/flutter-on-raspberry-pi-mostly-from-scratch-2824c5e7dcb1 (Zugegriffen: 13 November 2023).

Google Inc. (o. D. a) *Futter - Build for Any Screen*. Verfügbar unter: https://flutter.dev/ (Zugegriffen: 22 Februar 2024).

Google Inc. (o. D. b) *Introduction to Declarative UI*. Verfügbar unter: https://docs.flutter.dev/get-started/flutter-for/declarative (Zugegriffen: 13 Februar 2024).

Abbildung 44: Example caption

```
meta-flutter-elinux/
└── recipes-core/
    ├── custom-scripts/
    │   ├── files/
    │   │   └── flutter-auto-launch/
    │   └── flutter-auto-launch_1.00.bb
    ├── flutter-elinux-rules/
    │   ├── files/
    │   │   └── 90-interfaces.rules
    │   └── flutter-elinux-rules_1.00.bb
    ├── flutter-user/
    │   └── flutter-user_1.00.bb
    ├── systemd/
    │   ├── files/
    │   │   ├── flutter-auto.service
    │   │   └── flutter-pi.service
    │   ├── flutter-auto-service_1.00.bb
    │   ├── flutter-pi-service_1.00.bb
    │   └── systemd_%.bbappend
    └── systemd-conf/
        ├── files/
        │   ├── en.network
        │   ├── eth.network
        │   └── wlan.network
        └── systemd-conf_%.bbappend
```


Google Inc. (o. D. c) *Unsound Null Safety*. Verfügbar unter: https://dart.dev/null-safety/unsound-null-safety/ (Zugegriffen: 9 März 2024).

Hajian, M. (2024) *Flutter Engineering*. Staten House.

Heath, S. (2002) *Embedded Systems Design*. Elsevier.

Lars Bak (2011) *Dart: A Language for Structured Web Programming*. Dart: a language for structured web programming. Verfügbar unter: https://googlecode.blogspot.com/2011/10/dart-language-for-structured-web.html (Zugegriffen: 9 März 2024).

Linux Foundation (2024) *The Yocto Project ® 4.0.16 Documentation*. Verfügbar unter: https://docs.yoctoproject.org/4.0.16/singleindex.html (Zugegriffen: 13 Februar 2024).

Marwedel, P. (2021) *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*. Springer Nature.

Matsubayashi, H. (2020) „Graphical User Interface Using Flutter in Embedded Systems“. Verfügbar unter: https://elinux.org/images/6/61/Oct_27_Graphical_User_Interface_Using_Flutter_in_Embedded_Systems_Hidenori_Matsubayashi.pdf.

Richard Stallman (o. D.) *Why GNU/Linux?* Verfügbar unter: https://www.gnu.org/gnu/why-gnu-linux.en.html (Zugegriffen: 19 Februar 2024).

Stephanie Van Ness (2024) *Racing Toward CES with an Exciting New UI for Automotive Grade Linux IVI Demo*. Verfügbar unter: https://www.ics.com/blog/racing-toward-ces-exciting-new-ui-automotive-grade-linux-ivi-demo (Zugegriffen: 9 März 2024).

The 3-Clause BSD License (2011). Open Source Initiative. Verfügbar unter: https://opensource.org/license/bsd-3-clause/ (Zugegriffen: 18 Februar 2024).

The Flutter Authors (o. D.) *Engine/Shell/Platform/Embedder/Embedder.h at 436f9707b94774d1d049c04b8cda9d81d85aa4a8 · Flutter/Engine*. GitHub. Verfügbar unter: https://github.com/flutter/engine/blob/436f9707b94774d1d049c04b8cda9d81d85aa4a8/shell/platform/embedder/embedder.h (Zugegriffen: 22 Februar 2024).

Walrath, Kathy (2012) *What Is Dart?*

Weiser, M. (1999) „The Computer for the 21st Century“, *ACM SIGMOBILE Mobile Computing and Communications Review*, 3(3), S. 3–11. doi:10.1145/329124.329126.