ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Bachelor-Thesis

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)
an der Hochschule für Technik und Wirtschaft des Saarlandes
im Studiengang Praktische Informatik
der Fakultät für Ingenieurwissenschaften

Eine LaTeX-Vorlage für Abschlussarbeiten im Bereich Informatik/Mechatronik-Sensortechnik (inkl. DFHI) an der htw saar

vorgelegt von
Max Muster

betreut und begutachtet von
Prof. Dr.-Ing. André Miede
Prof. Dr. Thomas Kretschmer

Saarbrücken, Tag. Monat Jahr

Selbständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit (bei einer Gruppenarbeit: den entsprechend gekennzeichneten Anteil der Arbeit) selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich erkläre hiermit weiterhin, dass die vorgelegte Arbeit zuvor weder von mir noch von einer anderen Person an dieser oder einer anderen Hochschule eingereicht wurde.

Darüber hinaus ist mir bekannt, dass die Unrichtigkeit dieser Erklärung eine Benotung der Arbeit mit der Note „nicht ausreichend“ zur Folge hat und einen Ausschluss von der Erbringung weiterer Prüfungsleistungen zur Folge haben kann.

Saarbrücken, Tag. Monat Jahr

Max Muster

Zusammenfassung

Kurze Zusammenfassung des Inhaltes in deutscher Sprache, der Umfang beträgt zwischen einer halben und einer ganzen DIN A4-Seite.

Orientieren Sie sich bei der Aufteilung bzw. dem Inhalt Ihrer Zusammenfassung an Kent Becks Artikel: `http://plg.uwaterloo.ca/~migod/research/beckOOPSLA.html`.

Inhaltsverzeichnis

1 Einleitung

In diesem Kapitel geben wir eine Einführung in die Ausarbeitung unserer Fallstudie. Wir erläutern unsere Motivation, warum wir uns für das Thema Lambda-Architektur entschieden haben, und stellen unsere Idee für die Analyse eines Parkhaussystems vor, das als Anwendungsfall für die Umsetzung der Lambda-Architektur dient. Anschließend werden wir die konkreten Ziele, die wir mit dieser Fallstudie erreichen wollen, sowie unser weiteres Vorgehen bei der Implementierung und Evaluierung definieren.

1.1 Motivation

Heutzutage werden immer größere Datenmengen verarbeitet. Moderne Anwendungen müssen daher in der Lage sein, große Datenströme effizient zu verarbeiten, zu speichern und bereitzustellen, ohne dabei Verlust an Leistung oder Zuverlässigkeit zu riskieren.

Ein zentrales Problem bei der Verarbeitung großer Datenmengen wird durch das CAP-Theorem beschrieben. Dieses Theorem besagt, dass ein verteiltes Datensystem immer drei grundlegende Eigenschaften berücksichtigen muss:

- **Consistency (Konsistenz)**: Jeder Lesevorgang liefert entweder den aktuellsten verfügbaren Wert oder schlägt fehl.
- **Availability (Verfügbarkeit)**: Jede Anfrage erhält garantiert eine Antwort, auch wenn einige Knoten des Systems ausgefallen sind.
- **Partition Tolerance (Partitionstoleranz)**: Das System funktioniert weiter, auch wenn Teile des Netzes ausfallen oder Kommunikationsprobleme auftreten.

Allerdings stellt das CAP-Theorem eine grundlegende Einschränkung dar: Ein System kann immer nur zwei dieser drei Eigenschaften gleichzeitig garantieren. In datenintensiven Anwendungen, die eine hohe Fehlertoleranz (Partition Tolerance) und Verfügbarkeit (Availability) erfordern, kann daher nicht sichergestellt werden, dass jede Abfrage immer die aktuellsten Daten in einer akzeptablen Zeit liefert. Dies stellt eine große Herausforderung für Systeme dar, die sowohl Echtzeitdatenverarbeitung als auch Langzeitanalysen ermöglichen sollen.

Im Rahmen der Vorlesung „Software-Architektur“ haben wir uns entschieden, eine Fallstudie zur Lambda-Architektur zu erstellen. Die Lambda-Architektur ist ein Architekturmuster, das speziell für Big-Data-Anwendungen entwickelt wurde, also für Systeme, die große Datenmengen effizient verarbeiten müssen. Ihr Ziel ist es, eine fehlertolerante und hochverfügbare Architektur bereitzustellen, die trotz der Einschränkungen des CAP-Theorems eine möglichst hohe Datenkonsistenz gewährleistet. Sie begegnet dem Problem des CAP-Theorems, indem sie zwei Verarbeitungswege kombiniert: den Batch Layer für konsistente Langzeitanalysen und den Speed Layer für schnelle Echtzeitabfragen.

In dieser Fallstudie wollen wir diese Eigenschaften der Lambda-Architektur analysieren und bewerten. Dazu simulieren wir einen realen Anwendungsfall, in dem sowohl die Langzeitanalyse großer Datenmengen als auch die schnelle Bereitstellung neuer Daten eine zentrale Rolle spielen.

Wir orientieren uns dabei an bestehenden Anwendungsgebieten der Lambda-Architektur. Diese Architektur ist insbesondere in Web- und IoT-Anwendungen weit verbreitet, da sie eine skalierbare Verarbeitung großer Datenmengen bei gleichzeitig schneller Verfügbarkeit aktueller Daten ermöglicht.

Für unsere konkrete Fallstudie haben wir uns für eine IoT-Anwendung im Bereich der Parkhausanalyse entschieden. Ziel ist es, ein simuliertes Parkhaussystem zu entwickeln, das mithilfe von Kameras die Ein- und Ausfahrten in mehreren Parkhäusern erfasst und analysiert. Dabei sollen fahrzeugspezifische Informationen gespeichert und verarbeitet werden. Dieses Szenario eignet sich aus unserer Sicht besonders gut für die Lambda-Architektur, da es sowohl Langzeitanalysen (z.B. Belegungsmuster über mehrere Wochen) als auch Echtzeitabfragen (z.B. aktuelle Parkplatzverfügbarkeit) kombiniert.

1.2 Forschungsfragen

Mit dem Ziel unserer Fallstudie vor Augen können wir die folgenden Forschungsfragen definieren, die wir durch unsere Implementierung der Lambda-Architektur beantworten möchten:

- **RQ1:** Inwiefern lässt sich eine einfache, aber funktionsfähige Lambda-Architektur in einer kleinen, simulierten Umgebung implementieren?
- **RQ2:** Welche architektonischen Eigenschaften der Lambda-Architektur lassen sich in unserer Fallstudie identifizieren?
- **RQ3:** Welche Herausforderungen treten bei der Implementierung der Lambda-Architektur in einer kleinen, nicht skalierten Umgebung auf?

1.3 Forschungsansatz

Unser Ziel in dieser Fallstudie ist es, eine einfache, aber funktionsfähige Implementierung der Lambda-Architektur in einer simulierten Umgebung zu entwickeln. Der Fokus liegt dabei nicht auf der Entwicklung einer produktionsreifen Lösung für ein reales Parkhaussystem, sondern vielmehr auf der Schaffung eines praxisnahen Szenarios, das hilft, die Architektur besser zu verstehen und ihre Vor- und Nachteile zu evaluieren.

Um dieses Ziel zu erreichen, entwickeln wir eine simulierte Datenquelle, die künstliche Fahrzeugdaten generiert. Diese Daten durchlaufen die verschiedenen Schichten der Lambda-Architektur, so dass sowohl Echtzeit- als auch Batch-Verarbeitung getestet werden können.

Um die erste Forschungsfrage (RQ1) zu beantworten, haben wir uns für eine Implementierung auf Basis bewährter Open-Source-Technologien entschieden. Anstatt eine eigene Architektur von Grund auf neu zu entwickeln, verwenden wir existierende Werkzeuge, die bereits in der Praxis für die Lambda-Architektur eingesetzt werden. Die Auswahl basiert auf einer Literaturrecherche zu typischen Technologien für die drei Schichten der Lambda-Architektur. Eine detaillierte Erläuterung der Komponenten und Funktionsweisen der Lambda-Architektur erfolgt in Kapitel 2.

Der Speed Layer der Lambda-Architektur ist für die Echtzeitverarbeitung zuständig. Im Speed Layer müssen die Daten schnell verarbeitet werden. Für die Implementierung der Speed Layer wurde daher Apache Kafka gewählt. Kafka ermöglicht eine schnelle und zuverlässige Verarbeitung und Weiterleitung von Datenströmen.

Der Serving Layer übernimmt die Aggregation und Bereitstellung der verarbeiteten Daten. Er verbindet den Speed- und den Batch-Layer und bietet eine zusammenführende

Sicht auf die gespeicherten Informationen. Wir haben uns entschieden, den Serving Layer mit Apache Cassandra zu realisieren. Cassandra ist eine NoSQL-Datenbank, die speziell für schnelle Abfragen optimiert ist und in vielen realen Implementierungen der Lambda-Architektur für den Serving Layer verwendet wird.

Im Batch Layer werden alle historischen Rohdaten gespeichert und verarbeitet. In der klassischen Lambda-Architektur werden hier häufig verteilte Systeme wie Apache Hadoop oder Apache Spark eingesetzt, da diese für den Umgang mit großen Datenmengen optimiert sind. Da in unserer Fallstudie jedoch nicht mit so großen Datenmengen gearbeitet wird, verzichten wir auf eine skalierbare, verteilte Speicherung. Stattdessen verwenden wir auch hier Apache Cassandra, da es sowohl für schnelle Lesezugriffe als auch für die Langzeitspeicherung großer Datenmengen optimiert ist.

Die Anwendung wird in C# entwickelt, da diese Sprache eine gute Integration mit Apache Kafka und Apache Cassandra bietet und sich gut für die Implementierung unserer simulierten Umgebung eignet. Für die Verwaltung und Bereitstellung der verschiedenen Komponenten verwenden wir Docker. Diese Container-Technologie ermöglicht eine einfache Installation, Verwaltung und Reproduzierbarkeit unserer Umgebung, da alle benötigten Dienste in isolierten Containern laufen und unabhängig von der Konfiguration des Host-Systems sind.

2 Beispiele

2.1 Abkürzungen

Um Abkürzungen zu verwenden, muss über `\usepackage{acronym}` das benötigte Package geladen werden. Danach kann man lange Begriffe ganz bequem abkürzen:

So muss man nicht ständig Wireless Local Area Network (WLAN) ausschreiben, auch Transmission Control Protocol (TCP) lässt sich abkürzen. Würde man im Text WLAN oft verwenden, kann man sie, wie hier, nur als Abkürzung anzeigen lassen - oder bei Bedarf die Erklärung mitliefern (Wireless Local Area Network (WLAN)). Weiteres Beispiel könnte die Gang of Four (GoF) sein.

Weitere Informationen sind im Acronym-Manual zu finden.

2.2 Beispiel für BibLaTeX

BibLaTeX ist ein Package, das einem die Arbeit mit Zitaten bzw. Quellenangaben erleichtern kann. Mit JabRef (**??**) ist es möglich **.bib*-Dateien zu erstellen, in denen alle Angaben zu Autor, Buchtitel, Erscheinungsdatum usw. hinterlegt werden, welche zum passenden Zeitpunkt abgerufen werden können. Das Literaturverzeichnis wird mittels `\printbibliography` ausgegeben.

Im Allgemeinen wird im Literaturverzeichnis auch nur jene Literatur aufgenommen, die auch in der **.tex*-Datei referenziert wird. Danach ist es wichtig nicht nur mit *PdfLaTeX*, sondern auch mit *BibLaTeX* zu kompilieren, damit die zitierten Einträge in die verschiedenen Hilfsdateien aufgenommen werden können.

Einige Zitate

In diesem Satz könnten wir auf [**knuth:1976**] verweisen, ebenso auf das wichtige Werk [**dueck:trio**]. Wenn uns das nicht genug ist, sollten wir das anmerken, was in [**sommerville:1992**] geschrieben wurde. Im Zweifelsfall verweisen wir auf eine einzelne Seite, wie in [**bentley:1999**] zu finden.

Üblicherweise wird auch der Name des Autors bzw. der Autoren genannt, also beispielsweise bei einem Verweis auf **knuth:1976** [**knuth:1976**] oder auch bei mehreren Autoren **cormen:2001** [**cormen:2001**]. LaTeX stellt Mechanismen zur Verfügung, auch dies automatisiert zu erledigen.

2.3 Referenzierungen

Mit Referenzierungen kann ich ganz bequem auf Textpassagen, Kapitel, Sections oder Abbildungen im weiteren Text verweisen. Dies ist ein Verweis auf Abschnitt 2.3.1, der sich auf Abschnitt 2.3.1 befindet.

Auch ein Verweis auf Tabelle 2.1 auf Seite 7 ist möglich.

Man sollte beachten, dass man sein Dokument, wenn es Referenzierungen enthält, mehrmals kompiliert, da sonst manche Verweise nicht aufgelöst werden können.

2.3.1 Beispieltext

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

2.4 Dateien einbinden

Damit man nicht alle Einstellungen, Optionen, Packages und Texte, Abbildungen etc. in einer Datei unterbringen muss, werden zwei Befehle bereitgestellt, um externe **.tex*-Dateien einzubinden: `\include{PFAD}` und `\input{PFAD}`. Mit dem erstem Befehl wird eine neue Seite angelegt, danach kommen die Inhalte aus der angegebenen Datei; mit dem zweiten Befehl wird keine neue Seite angelegt – der Inhalt der angegebenen Datei wird direkt an die betroffene Stelle eingefügt.

Wichtig: Der *Pfad* wird sinnigerweise *relativ* angegeben, wobei als Stammverzeichnis jenes Verzeichnis angesehen wird, in dem die **.tex*-Datei mit der *Document*-Umgebung abgelegt ist (in diesem Fall ist es *htwsaar-i-mst-config.tex*).

2.5 Tabellen

2.5.1 Einfache Tabelle

In LaTeX lassen sich Tabellen unterschiedlicher Ausprägung einfach erzeugen. Das allgemeine Format einer Tabelle sieht aus wie folgt:

Listing 2.1: Allgemeines Format

```
\begin{table}
        \caption{BESCHRIFTUNG}
        \begin{tabular}{FORMATIERUNG}
                TABELLENINHALT
        \end{tabular}
\end{table}
```

Eine Beispieltabelle (Tabelle 2.1) könnte also so aussehen:

Listing 2.2: Tabelle 2.1

```
\begin{table}
        \caption{Beispiel 1}
        \begin{tabular}{lrcr}
                \toprule
                \textbf{Name} & \textbf{Vorname} & \textbf{
                   Matrikelnummer} & \textbf{Lieblingsspeise}\\
                \midrule
                Jackson & Michael & 123456 & Erdbeereis \\
                Springsteen & Bruce & 234567 & Schwedisches Lakritz
                    \\
```

```
                Bach & Anna, Magdalena & 3456789 & Frankfurter
                    Kranz \\
                Schumann & Clara & 4567890 & Bisquitt\"ortchen \\
                \bottomrule
        \end{tabular}
        \label{tab:beispieltabelle1}
\end{table}
```

Mit `\caption{Beispiel 1}` bekommt unsere Tabelle eine Beschriftung am Tabellenkopf. `l|r|c|r` legt die Textausrichtung der einzelnen Spalten fest: `l` bedeutet linksausgerichtet, `r` rechtsausgerichtet und `c` zentriert. Durch `|` werden Spaltenlinien gezogen. `\toprule`, `\midrule` und `\bottomrule` erzeugen Kopf-, Mittel- und Abschlusslinie in der Tabelle. Als Spaltentrenner wird das `&` genutzt, Zeilentrenner ist der doppelte Backslash (`\\`). Am Ende kann die Tabelle auch mit einem Label versehen werden (`\label{tab:beispieltabelle1}`), über welches diese referenziert wird.

2.5.2 Erweiterte Tabellenbefehle

Um Tabellen in LaTeX flexibler zu gestalten gibt es weitere Befehle bzw. zusätzliche Pakete, die einem das Leben leichter machen (Tabelle 2.2). Hierzu ein weiteres Beispiel:

Listing 2.3: Tabelle 2.2

```
\begin{table}
        \centering
        \caption{Beispiel 2}
        \begin{tabular}{lll}
                \hline
                Author & Title & Year \\
                \hline
                \hline
                \multirow{3}{*}{Stanislav Lem} & Solaris & 1961 \\
                        & Roboterm\"archen & 1967 \\
                        & Der futurologische Kongress & 1971 \\
                \hline
                \multirow{3}{*}{Isaac Asimov} & Ich, der Robot &
                    1952 \\
                        & Der Tausendjahresplan & 1966 \\
                        & Doctor Schapirows Gehirn & 1988 \\
                \hline
        \end{tabular}
\label{tab:beispieltabelle2}
\end{table}
```

Tabelle 2.1: Beispiel 1

Name	Vorname	Matrikelnummer	Lieblingsspeise
Jackson	Michael	123456	Erdbeereis
Springsteen	Bruce	234567	Schwedisches Lakritz
Bach	Anna, Magdalena	3456789	Frankfurter Kranz
Schumann	Clara	4567890	Bisquittörtchen

Tabelle 2.2: So sollte man es nicht machen! Beispiel für einen schlechten Tabellenstil

Author	Title	Year
Stanislav Lem	Solaris	1961
	Robotermärchen	1967
	Der futurologische Kongress	1971
Isaac Asimov	Ich, der Robot	1952
	Der Tausendjahresplan	1966
	Doctor Schapirows Gehirn	1988

Mit `\centering` wird die Tabelle zentriert ausgerichtet, analoge Befehle für rechts- bzw. linksausrichtung sind z.B. `\raggedleft` und `\raggedright`.

Eine weitere Form der Tabellen ist das Package *tabularx*, das variable Spaltenbreiten unterstützt, und *booktabs*, welches mit horizontalen Linien besser arbeiten kann.

2.6 Abbildungen

LaTeX unterstützt generell die Formate **.jpeg*, **.png* und **.pdf*. Handelt es sich z.B. um Strichgrafiken oder skalierbare Farbflächen, sollte **.pdf* die erste Wahl sein, da sich in diesem Format Vektorgrafiken ohne Qualitätsverlust darstellen bzw. skalieren lassen.

Abbildung 2.1: Erstes Bild, Völklinger Hütte

2.6.1 Wrapfigure

Abbildung 2.1 ist zwar ganz nett anzusehen, aber vielleicht sähe es eleganter aus, wenn die Abbildung von unserem Textabschnitt umflossen werden würde. Diese Art von Abbildungen sollte jedoch sparsam und mit großer Sorgfalt eingesetzt werden, da es zu unschönen Darstellungen kommen kann. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an.

Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Abbildung 2.2: Völklinger Hütte, *.jpg

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

2.6.2 Subfigures

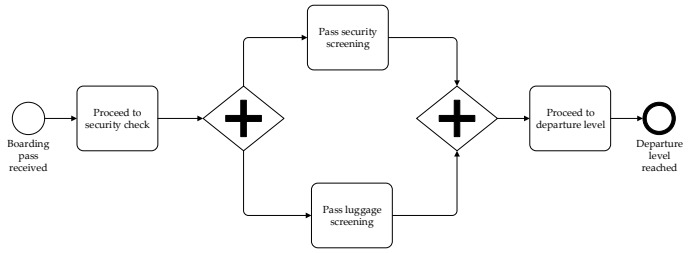
Es ist ebenso möglich, mehrere Abbildungen nebeneinander zu setzen, wie in Abbildung 2.3 zu sehen ist. Eine separate Referenzierung ist auch möglich: Abbildung 2.3a.

(a) Erstes ... (b) ... und zweites Bild

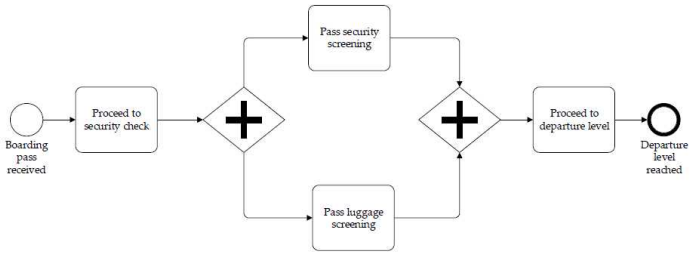
Abbildung 2.3: Mehrere Abbildungen nebeneinander

2.6.3 Qualitätsunterschiede

Leider haben die unterschiedlichen Grafikformate bedingt durch die unterschiedlichen Kompressionsverfahren einige Schwächen, insbesondere die Umwandlung in das *JPG*-Format erzeugt unangenehme Artefakte im Bild. Abbildung 2.4 zeigt die Unterschiede zwischen *PDF-Format* und *JPG-Format* im Vergleich.



(a) *PDF*-Format



(b) *JPG*-Format

Abbildung 2.4: Beide Formate im Vergleich

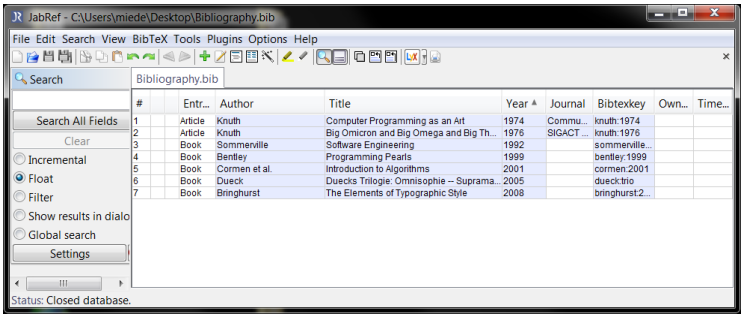


Abbildung 2.5: *PNG*-Format

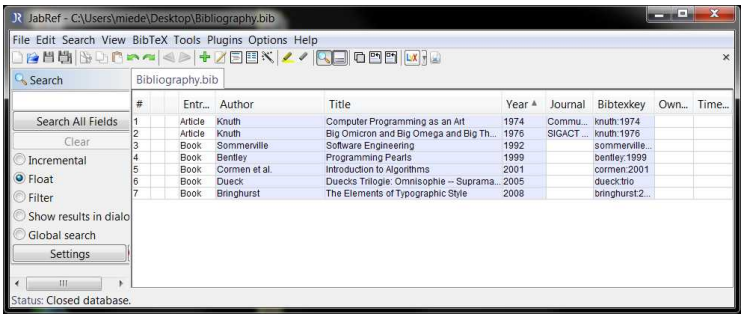


Abbildung 2.6: *JPG*-Format

Wenn eine **.pdf*-Datei nicht infrage kommt, beispielsweise bei Screenshots, ist unbedingt das *PNG-Format* vorzuziehen. Den Unterschied machen Abbildung 2.5 und Abbildung 2.6 deutlich.

„Faustregeln“ im Umgang mit Abbildungen:

- Diagramme bzw. alles, was Linien usw. enthält: *PDF* (im Vektorformat).
- Screenshots bzw. alles, was größere gleichfarbige Flächen enthält: *PNG*.
- Der Rest (in der Regel Fotos): *JPEG*.

2.7 Quellcode einbinden

Das Package *lstlisting* ermöglicht es, Quellcode ansprechend in das Dokument einzubinden. Man kann Quellcode einzeilig einbinden mittels `\lstinline|Quellcode|`. Dabei ist darauf zu achten, dass der Befehl einmal mit { } und einmal mit | | aufgerufen werden kann, je nachdem, welche Zeichen im angegebenen Quelltext genutzt werden. Es ist auch möglich eine eigene Umgebung für Quelltext zu schaffen:

Listing 2.4: Erstes Listing

```
private Umgebung(int i, int k)
{
        System.out.println("Eine Funktion mit " + i + "und" + k ".");
}
```

Wer Quelltext aus externen Dateien einbinden möchte, geht wie folgt vor:

Listing 2.5: Externer Quellcode

```
public class HalloWelt {
    public static void main(String[] args) {
        System.out.println("Hallo Welt!");
    }
}
```

Wie genau der Quellcode formatiert und gefärbt ist, ist in *htwsaar.i.mst.config.tex* hinterlegt, wobei fü verschiedene Sprachen auch eigene Styles angelegt werden können (hier z.B. für Java).

2.8 Mathematische Ausdrücke

Mathematische Ausdrücke sind eine kleine Kunst für sich. Am allereinfachsten kann man eine Formel, wie $a + b = c$ in den Fließtext einbinden, wobei LaTeX die Höhe der Ausdrücke der Zeile anpasst, wie hier zu sehen $\sum_{y=0}^{x} a$. In einer Umgebung sieht das schon anders aus:

$$\sum_{y=0}^{x} a \tag{2.1}$$

Griechische Buchstaben:

$$\alpha\beta\gamma\delta\epsilon\varepsilon\zeta\eta\theta\iota\kappa\lambda\mu\nu\xi\pi\omega\rho\varrho\sigma\tau\upsilon\phi\varphi\chi\psi\omega \tag{2.2}$$

Brüche:

$$Ergebnis = \frac{a}{b} \tag{2.3}$$

$$\frac{\sin \alpha^2 + \cos \alpha^2}{1} = 1 \tag{2.4}$$

$$\frac{-9x}{\frac{2y}{3z+2}} \tag{2.5}$$

Text innerhalb von Formeln:

$$\sum_{y=1}^{n} y = \frac{n * (n+1)}{2} \quad \text{Gaußsche Summenformel} \tag{2.6}$$

Hoch- bzw. Tiefstellungen:

$$x_{i,j}^2 \tag{2.7}$$

$${x_{i,j}}^2 \tag{2.8}$$

$$x_{n_0} \tag{2.9}$$

Matrizen: Matrizen werden innerhalb der mathematischen Umgebung als wiederum neue Umgebung eingebunden. Wie bei Tabellen auch werden Zeilen durch \\ und Spalten durch & getrennt.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{2.10}$$

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} \tag{2.11}$$

Fallunterscheidung:

$$f(x) = \begin{cases} 0, & \text{falls } x < 0 \\ 1, & \text{falls } x \geq 0 \end{cases} \tag{2.12}$$

2.9 To-Do-Notes

Um bei einer längeren Arbeit nicht den Überblick zu verlieren, an welcher Stelle es nötig ist weiter zu arbeiten, bietet es sich an, kleine Notizen einzufügen. Das Package *todonotes* stellt eine elegante Lösung bereit, um differenziert und vielfarbig jene Abschnitte zu kennzeichnen, die einer weiteren Bearbeitung bedürfen.

Beispiel für To-Do-Notes

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: Dies ist ein Blindtext? oder Huardest gefburn? Kjift? mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Les- barkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschie- dene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein.Fremdsprachige Texte wie Lorem ipsum dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld.

Plain todonotes.

Plain todonotes.

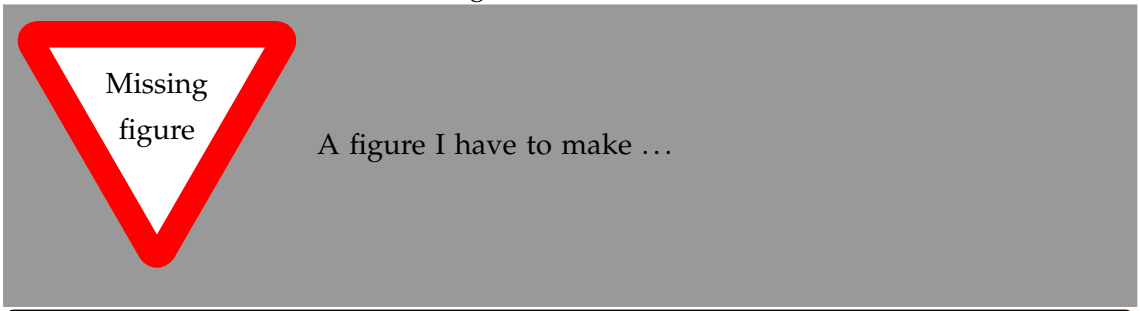
Todonote that is only shown in the margin and not in the list of todos.

A very long todonote that certainly will fill more than a single line in the list of todos. Just to make sure let's add some more text …

Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: Dies ist ein Blindtext? oder Huardest gefburn? Kjift? mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie Lorem ipsum dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

A note with no line back to the text.

A very long todonote that certainly will fill more than a single line in the list of todos …



1: Erste Nummer...

2: Zweite Nummer...

Nachfolgend wird noch eine Liste aller To-Dos auf einer separaten Seite ausgegeben.

Todo list

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

WLAN Wireless Local Area Network

TCP Transmission Control Protocol

GoF Gang of Four

Anhang

A Erster Abschnitt des Anhangs

In den Anhang gehören „Hintergrundinformationen“, also weiterführende Information, ausführliche Listings, Graphen, Diagramme oder Tabellen, die den Haupttext mit detaillierten Informationen ergänzen.

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Kolophon

Dieses Dokument wurde mit der LaTeX-Vorlage für Abschlussarbeiten an der htw saar im Bereich Informatik/Mechatronik-Sensortechnik erstellt (Version 2.25, August 2024). Die Vorlage wurde von Yves Hary und André Miede entwickelt (mit freundlicher Unterstützung von Thomas Kretschmer, Helmut G. Folz und Martina Lehser). Daten: (F)10.95 – (B)426.79135pt – (H)688.5567pt