ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Projektarbeit

im Modul Software-Architektur
an der Hochschule für Technik und Wirtschaft des Saarlandes
im Studiengang Praktische Informatik
der Fakultät für Ingenieurwissenschaften

Cloud-Native Architekturen

vorgelegt von
Tristan Gläs und Carolin Becker

betreut und begutachtet von
Prof. Dr. Markus Esch

Saarbrücken, 22. März 2021

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, 22. März 2021

Tristan Gläs und Carolin Becker

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`.

- cloud native erklären - fallbeispiel - diskutieren

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth [5]

Danksagung

Hier können Sie Personen danken, die zum Erfolg der Arbeit beigetragen haben, beispielsweise Ihren Betreuern in der Firma, Ihren Professoren/Dozenten an der htw saar, Freunden, Familie usw.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 LaTeX installieren und einrichten . . . 1
  - 1.1.1 Unter Windows . . . 1
  - 1.1.2 Unter Linux . . . 1
- 1.2 Entwicklungsumgebungen . . . 1
- 1.3 Werkzeuge . . . 2
- 1.4 Struktur und Gebrauch der Vorlage . . . 2
  - 1.4.1 Struktur der Vorlage . . . 2
  - 1.4.2 Gebrauch der Vorlage . . . 3

2 Beispiele **5**
- 2.1 Abkürzungen . . . 5
- 2.2 Beispiel für BibLaTeX . . . 5
- 2.3 Referenzierungen . . . 5
  - 2.3.1 Beispieltext . . . 6
- 2.4 Dateien einbinden . . . 6
- 2.5 Tabellen . . . 6
  - 2.5.1 Einfache Tabelle . . . 6
  - 2.5.2 Erweiterte Tabellenbefehle . . . 7
- 2.6 Abbildungen . . . 8
  - 2.6.1 Wrapfigure . . . 8
  - 2.6.2 Subfigures . . . 9
  - 2.6.3 Qualitätsunterschiede . . . 9
- 2.7 Quellcode einbinden . . . 11
- 2.8 Mathematische Ausdrücke . . . 11
- 2.9 To-Do-Notes . . . 13

3 Einleitung **17**

4 Cloud Native **19**
- 4.1 Cloud Computing . . . 19
- 4.2 Definition Cloud Native . . . 19
- 4.3 Microservices . . . 20
- 4.4 Container . . . 20
- 4.5 Abgrenzungen . . . 21
- 4.6 Eigenschaften . . . 21
- 4.7 Design Prinzipien . . . 22
- 4.8 Vor- und Nachteile . . . 22
- 4.9 Einsatzgebiete . . . 22

5 Fallbeispiel **23**
- 5.1 Beschreibung . . . 23
- 5.2 Anforderungen . . . 23
  - 5.2.1 Nichtfunktionale Anforderungen . . . 23

5.2.2 Funktionale Anforderungen . . . 23
5.3 Architekturentwurf . . . 23
5.3.1 Microservices . . . 23
5.3.2 API-Gateway . . . 24
5.3.3 Kommunikation . . . 24
5.3.4 Client . . . 24
5.4 Implementierung des Prototyps . . . 24

6 Diskussion **27**
6.1 Vergleich mit Eigenschaften Cloud Nativ . . . 27
6.2 Microservices . . . 27
6.3 Tradeoffs . . . 27
6.4 Erweiterbarkeit . . . 27

7 Fazit **29**

Literatur **31**

Abbildungsverzeichnis **33**

Tabellenverzeichnis **33**

Listings **33**

Abkürzungsverzeichnis **35**

A Erster Abschnitt des Anhangs **39**

1 Einleitung

1.1 LaTeX installieren und einrichten

1.1.1 Unter Windows

Als LaTeX-Distribution unter Windows steht *MikTeX* zu Verfügung, die als freie Software im Internet erhältlich ist. *MikTeX* unterstützt Windows XP, Vista und Windows 7. Neben *MikTeX* wird noch ein PostScript-Interpreter benötigt, z.B. GhostScript, zu finden auf Chip.de.

Wichtig: Bei *MikTeX* unbedingt Vollinstallation auswählen, sonst sind eventuell benötigte Packages nicht vorhanden.

1.1.2 Unter Linux

Unter Linux existiert die LaTeX-Distribution *texlive*, die als aktuelle Version aus den Paketquellen geladen werden kann (unter Ubuntu mit `apt-get install texlive-full`). Auch hier ist ganz wichtig, die volle Distribution zu laden, damit alle Packages zur Verfügung stehen.

1.2 Entwicklungsumgebungen

Hat man die passende Distribution installiert, bieten sich vielerlei Möglichkeiten an ein LaTeX-Projekt anzugehen oder einzelne Dokumente zu editieren. Unter Windows könnten dies folgende sein:

TeXnicCenter Umfangreiche Entwicklungsumgebung mit Projektorganisation und Autovervollständigung

TeXLipse Eclipse-Plugin, das alle Vorteile der Eclipseumgebung mit LaTeX verbindet

TeXmaker Einfacher LaTeX-Editor mit Pdf-Direktvorschau

Unter Linux stehen bereit:

Gummi Ebenfalls einfacher LaTeX-Editor mit Direktvorschau

TeXLipse Auch für Linux erhältlich

Kile Umfangreiche Entwicklungsumgebung, ähnlich wie TeXnicCenter

Nach der Installation muss die Entwicklungsumgebung eingerichtet werden; dazu finden sich viele Anleitungen im Internet, die genau erklären, welche Distribution auf welche Weise eingerichtet wird. Insbesondere sollte der PDF-Viewer festgelegt werden, damit bei Gummi und TeXmaker die Direktvorschau funktioniert. Manchmal kommt es vor, dass die Ausgabe nach dem Kompilieren Umlaute und Sonderzeichen nicht richtig darstellt. Unter Linux hängt dies mit den unterschiedlichen Zeichensätzen zusammen, die unterstützt werden. Um diese Vorlage zu verwenden ist es notwendig, den verwendeten Zeichensatz des Editors bzw. der Entwicklungsumgebung auf den in diesem Dokument verwendeten Zeichensatz umzustellen: UTF-8 ohne BOM (Byte Order Mark).

1.3 Werkzeuge

JabRef Ein Literaturverwaltungsprogramm, welches das *BibTeX*-Format einsetzt und mithilfe einer graphischen Oberfläche das Anlegen von Literaturverzeichnissen vereinfacht.

1.4 Struktur und Gebrauch der Vorlage

Die vorliegende Vorlage für Abschlussarbeiten besteht aus einer internen Struktur, die grundsätzlich nicht verändert werden sollte.

1.4.1 Struktur der Vorlage

htw-i-mst-config.tex Enthält alle zu ladenden Packages, Styleparameter für Hyperlinks, Codelistings und Literaturverzeichnis sowie globale Parameter für Tabellen und Beschriftungen. Im Besonderen befinden sich hier die Variablen für den eigenen Namen, Titel, Datum der Arbeit, den betreuenden Professor etc.

htw-i-mst-vorlage.tex Dies ist die Hauptdatei, in der alle notwendigen **.tex*-Dateien eingebunden werden, die zu dem Dokument gehören. Es empfiehlt sich die interne Struktur *nicht* zu verändern. Eigene Kapitel werden an der dafür markierten Stelle eingebunden.

Bibliography.bib Zentrale Datei für die Literaturangaben, welche man z.B. mit JabRef verwalten kann.

Chapters/ Ablageort für alle selbst angelegten Kapitel der Arbeit. Die Aufteilung in eigene Dateien erleichtert die Übersicht über den Quellcode.

Graphics/ Ablageort für alle im Dokument benötigten Grafikdateien. Gerne darf man hier Unterverzeichnisse zur besseren Strukturierung anlegen.

Examples/ Dieser Ordner enthält die in dieser Vorlage beigefügten LaTeX-Beispiele, welche vor der Abgabe der Arbeit selbstverständlich gelöscht werden sollten.

Frontbackmatter/ In diesem Ordner sind all jene Dateien abgelegt, die – außer dem Kerntext in *Chapters/* – die Gesamtheit der Abschlussarbeit ausmachen.

- **Titlepage.tex** Definiert die Titelseite der Abschlussarbeit. Diese Datei muss normalerweise nicht verändert werden.
- **Abbreviations.tex** Hier werden alle Abkürzungen hinterlegt, die im Dokument verwendet werden.
- **Abstract.tex** Eine kurze Zusammenfassung der Abschlussarbeit wird in diese Datei eingefügt.
- **Acknowledgements.tex** Dort finden Danksagungen ihren Platz.
- **ConfidentialityClause.tex** Beinhaltet den Sperrvermerk und ist nur zu verwenden, falls dies beispielsweise vom beteiligten Unternehmen gefordert wird.
- **Contents.tex** Enthält wichtige Eintragungen in die *Table-of-Contents*. Diese Datei muss normalerweise nicht geändert werden.
- **Declaration.tex** Enthält die Selbständigkeitserklärung. Diese Datei darf nicht geändert werden.

Colophon.tex Enthält einen Hinweis auf die Urheber dieser Vorlage. Diese Datei darf nicht geändert werden.

ListOfs.tex Enthält die Einträge für die Tabellen- und Abbildungsverzeichnisse etc. und muss gewöhnlich nicht verändert werden.

1.4.2 Gebrauch der Vorlage

Grundsätzlich ist nicht viel zu tun, um die Vorlage für Abschlussarbeiten zu verwenden. Man entpackt den Hauptordner in das gewünschte Verzeichnis und nutzt die Dateien so, wie in Abschnitt 1.4.1 beschrieben. Danach werden *alle* Dateien gespeichert und die Hauptdatei, *htwsaar-i-mst-vorlage.tex*, mehrfach kompiliert (LaTeX benötigt mehrere Durchgänge um z.B. Referenzen richtig zuzuordnen). Hat man Änderungen in *Bibliography.bib* bzw. *Bibliography.tex* vorgenommen oder neue Zitate z.B. mittels `\cite` eingefügt, muss erst mit *BibLaTeX* und anschließend mitdem entsprechenden LaTeX-nach-PDF-Compiler übersetzt werden.

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 (Abschnitt 1.3) 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 [4] verweisen, ebenso auf das wichtige Werk [3]. Wenn uns das nicht genug ist, sollten wir das anmerken, was in [6] geschrieben wurde. Im Zweifelsfall verweisen wir auf eine einzelne Seite, wie in [1, S. 112] zu finden.

Üblicherweise wird auch der Name des Autors bzw. der Autoren genannt, also beispielsweise bei einem Verweis auf Knuth [4] oder auch bei mehreren Autoren Cormen u. a. [2]. 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

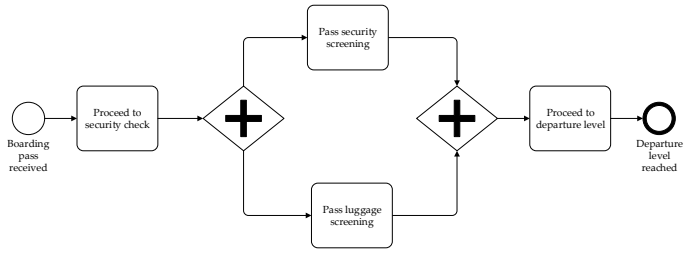
Es ist ebenso möglich, mehrere Abbildungen nebeneinander zu setzen, wie in Abbildung 2.3 zu sehen ist. Eine separate Referenzierung ist auch möglich: Abbildung 2.3a.

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

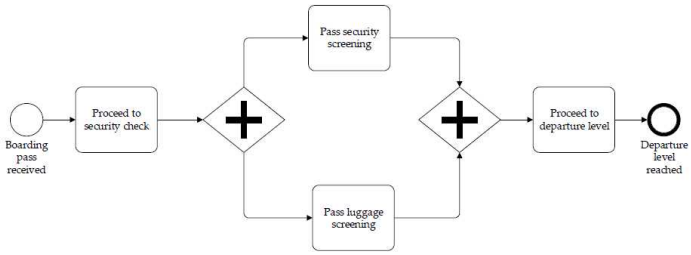
Abbildung 2.3: Mehrere Abbildungen nebeneinander

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

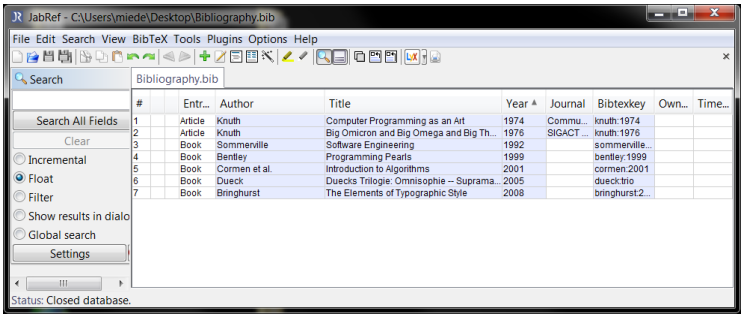


Abbildung 2.5: *PNG*-Format

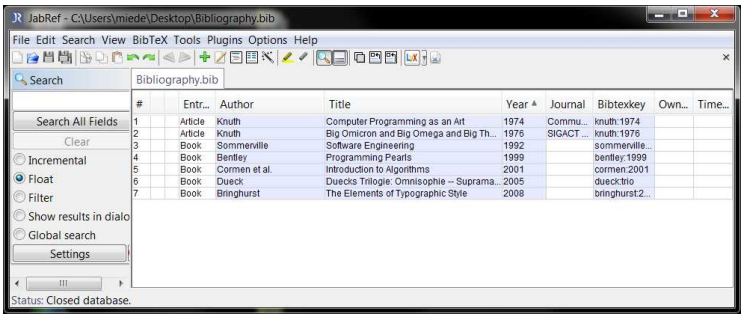


Abbildung 2.6: *JPG*-Format

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:

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

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.

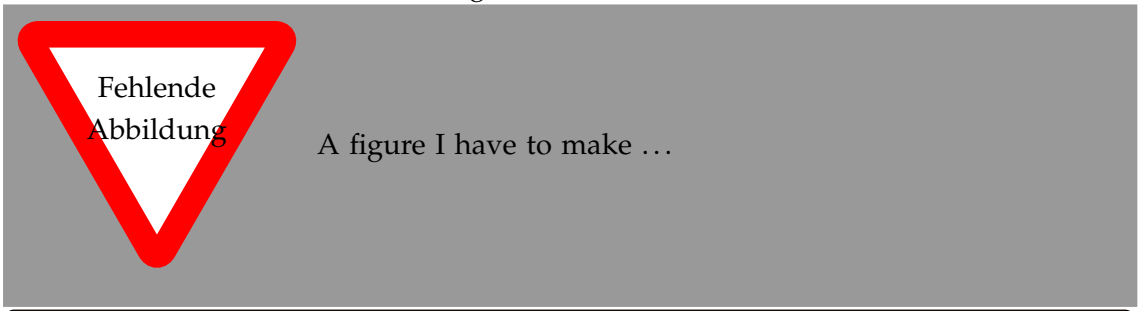
Todonote that is only shown in the margin and not in the list of todos.

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.

Liste der noch zu erledigenden Punkte

Plain todonotes. 13
Plain todonotes. 13
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 13
A note with no line back to the text. 13
A short entry in the list of todos 13
Abbildung: A figure I have to make 13
1: Erste Nummer... 13
2: Zweite Nummer... 13

3 Einleitung

Die Weiterentwicklung und der Ausbau des Cloud Computing Bereiches, die durch die steigenden Zahlen von Benutzern und Daten schnell vorangetrieben werden, benötigt auch entsprechende Softwaresysteme, die mit den bereitgestellten Resourcen effektiv umgehen können und auf die Cloud Umgebung abgestimmt sind. Solche Architekturen nennt man Cloud-Native Architekturen oder kurz einfach nur Cloud-Native. Es ist ein relativ neuer Ansatz, der eine Mischung aus bestehenden und neuen Konzepten des Softwarearchitekturdesigns bildet. Cloud-Native ist ein stetig wachsender Bereich, der immer mehr an popularität gewinnt. Die Cloud Native Computing Foundation ist eine Organisation, die eine Platform für Informationen rund um den Bereich Cloud-Native bietet.

https://www.cncf.io/blog/2020/08/14/state-of-cloud-native-development/

4 Cloud Native

In diesem Kapitel gehen wir auf die Definition von Cloud-Native Architekturen ein, grenzen sie von anderen Absätzen ab und betrachten wichtige Eigenschaften und Design-Prinzipien. Abschließend beschäftigen wir uns mit den Vor-und Nachteilen und den typischen Einsatzgebieten.

4.1 Cloud Computing

Bevor wir uns mit Cloud-Native auseinandersetzten können, müssen wir uns zuerst mit dem Cloud Computing beschäftigen, da es nämlich die Basis für diese Architekturen bildet und sie maßgeblich beeinflusst. Die NIST Definition von Cloud Computing enthält die wichtigsten Merkmalen.

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models. TODO

Entscheidend für Cloud-Native ist nun die schnelle Bereitstellung von Resourcen, denn dies eröffnet neue Möglichkeiten hinsichtlich der Skalierbarkeit und hat dadurch einen starken Einfluss auf die Architekturen.

4.2 Definition Cloud Native

Was genau Cloud-Native ist und wie man es definieren kann ist schwierig, da der Begriff noch relativ neu ist. Eine erste Version einer Definition kommt von der Cloud Native Computing Foundation, einer Organisation, die als Vorreiter in Sachen Cloud-Native gilt.

CNCF Cloud Native Definition v1.0

Cloud native Technologien ermöglichen es Unternehmen, skalierbare Anwendungen in modernen, dynamischen Umgebungen zu implementieren und zu betreiben. Dies können öffentliche, private und Hybrid-Clouds sein. Best Practices, wie Container, Service-Meshs, Microservices, immutable Infrastruktur und deklarative APIs, unterstützen diesen Ansatz.

Die zugrundeliegenden Techniken ermöglichen die Umsetzung von entkoppelten Systemen, die belastbar, handhabbar und beobachtbar sind. Kombiniert mit einer robusten Automatisierung können Softwareentwickler mit geringem Aufwand flexibel und schnell auf Änderungen reagieren.

Die Cloud Native Computing Foundation fördert die Akzeptanz dieser Paradigmen durch die Ausgestaltung eines Open Source Ökosystems aus herstellerneutralen Projekten. Wir demokratisieren modernste und innovative Softwareentwicklungs-Patterns, um diese Innovationen für alle zugänglich zu machen.

TODO wichtigsten punkte nennen

4.3 Microservices

definition warum braucht man das im cloud native bereich Eigenschaften, Vorteile - um diese microservices zu verwalten werden ...

- Microservices sind in Cloud Native Systemen ein beliebter Architekturstil
- Software aus kleinen Services bzw. Module, die unabhängig von einander sind
- Service kann unabhängig von anderen Services entwickelt und bereitgestellt werden, ohne die Funktionsfähigkeit anderer Services zu beeinträchtigen
- Eigenständigkeit: besitzt jede Komponentenservice seinen eigenen Code sowie seine eigene Implementierung
- jede einzelne Komponente ist für eine Reihe von Funktionen spezialisiert fokussiert sich auf die Lösung eines bestimmten Problems
- Wenn ein einzelner Service z.B. hinsichtlich Code zu komplex wird kann er in kleinere Services unterteilt werden
- Services kommunizieren über definierte APIs (application programming interface)
- Durch die unabhängigen Services kann jeder einzelne Service der Anwendung einfach nach Bedarf skaliert werden

-Flexibilität
-kleine Codebasis
-Fehlerisolation
-einfache Bereitstellung

4.4 Container

definition warum braucht man das im cloud native bereich Eigenschaften, Vorteile Docker, Kubernetes

- Standard-Software-Einheit, die den Code und seine Abhängigkeiten verpackt -> Anwendung schnell und zuverlässig von einer Umgebung zur anderen ausgeführt werden kann
- Bieten schlanke und unveränderliche Infrastruktur für die Paketierung und Bereitstellung von Anwendungen
- Cloud native Anwendungen nutzen Container für ein einheitliches Betriebssystem, denn durch den geringen Verwaltungsaufwand und die hohe Dichte können viele Container auf demselben virtuellen Computer gehostet werden

Vorteile:
- Weniger Speicherplatz: Virtualisierung auf Betriebssystemebene, mehrere Container direkt auf dem Kernel des Betriebssystems ausgeführt werden -> starten schneller
- Isolation: laufen von anderen Anwendungen isoliert
- Überall ausführbar/Portabilität: da die Softwarepakete alle Elemente enthalten, die zur Ausführung in beliebten Umgebungen erforderlich sind
- Schnelle Skalierbarkeit: können schnell gestartet und angehalten werden -> Ressourcen hoch- und herunterskaliert werden können

Docker: Beliebtes Open-Source-Containerformat zur Automatisierung der Bereitstellung von Anwendungen die z.B. in der Cloud ausgeführt werden können

Kubernetes: Open-Source-Orchestrierungssystem zur Automatisierung der Verwaltung, Platzierung, Skalierung und des Routings von Containern

4.5 Abgrenzungen

gegenüber monolith Definition, kurz: Eigenschaften anhand von Bild

- gesamte Architektur muss skaliert werden, wenn in einem Prozess ein Fehler auftritt
- Hinzufügen und Verbessern von Funktionen mit zunehmender Codebasis komplexer -> erschwert Umsetzung neuer Konzepte
- Erhöhtes Risiko der Anwendungsverfügbarkeit -> viele Abhängige und eng miteinander verbundene Prozesse -> einzelner Prozessausfall erhöht

gegenüber virtualisierung
kurz: Definition, Vergleich/Abgrenzung anhand von Bild

Virtualisierung der gesamten Hardware bzw. Abstraktion der physischen Hardware, die einen Server in viele Server wandelt

4.6 Eigenschaften

Als nächstes betrachten wir einige typischen Eigenschaften von Cloud-Native Architekturen. Im nächsten Abschnitt gehen wir dann darauf ein aus welchen Design Prinzipien hervorgehen.

1.) Globale Ebene Cloud-Native Architekturen sind oft für eine globale Ebene ausgelegt. Das impliziert z.B. das Daten und Dienste mehrfach deployed und Daten von verschiedenen Quellen synchronisiert werden müssen.

2.) Skalierbarkeit Die entstehenden Architekturen sind skalierbar und können eine sehr große Menge von Benutzern unterstützen. Dies ist besonders in Kombination mit der globalen Ebene, wenn man Sychronisation und Konsistenz betrachtet, eine große Herausforderung.

3.) "Built on the assumption that infrastructure is fluid and failure is constant"Die Annahme ïnfrastructure is fluid"bedeutet, dass die unterliegenden Strukturen (Hardware) der nicht konstant sind. So können z.B. Recheneinheiten (CPUs) hinzukommen oder wegfallen. Diese Annahme resultiert aus der Verwendung von Cloud Technologien und bildet die Basis für das Entwerfen von skalierbaren Architekturen. Die zweite Annahme, dass Fehler konstant auftreten, ergibt sich ebenfalls aus der Verwendung von Cloud Technologien, denn wenn eine große Anzahl von Hardwarekomponenten verwendet wird steigt die Wahrscheinlichkeit von einem Ausfall. Die Architektur muss also die Möglichkeit von Hardwarefehlern miteinbeziehen. Anders kann man dies auch Wiederstandsfähigkeit bezeichnen.

4.) Verbesserungen und Tests verlaufen unscheinbar Die Architekturen sind so entworfen, dass Systeme, ohne Verlust von Verfügbarkeit, geupdatet und getestet werden können.

5.) Sicherheit Sicherheit spielt eine wichtige Rolle in Cloud-Native Architekturen. Die meisten Systeme bestehen aus vielen kleinen Teilen, für welche Zugriff auf andere Teile

und Autorisierung/Authentifizierung von Benutzern geregelt werden muss.

4.7 Design Prinzipien

Google ist einer der großen Vertreter, wenn es um Cloud Technologien geht. Mit einer abgewandelten Version der Twelve-Factor-App hat Google eine Liste von Design Prinzipien erstellt, die dabei helfen die Eigenschaften von Cloud-Native Architekturen zu realisieren. Anzumerken ist, dass die Twelve-Factor-App Prinzipien nicht nur für Cloud-Native Applikationen sind, sondern auch für andere Zwecke verwendet werden können.

1.) Codebase 2.) Dependencies 3.) Configuration 4.) Backing services 5.) Build, release, run 6.) Processes 7.) Port Binding 8.) Concurrency 9.) Disposability 10.) Environment parity 11.) Logs 12.) Admin processes

4.8 Vor- und Nachteile

In diesem Abschnitt nennen und erklären wir einige Vor- und Nachteile von Cloud-Native Architekturen. Wir beginnen mit den Vorteilen.

1. Skalierbarkeit/Elastizität 2. Zuverlässigkeit/Wiederstandsfähigkeit 3. Änderbarkeit 4. Übertragbarkeit 5. Erweiterbarkeit 6. CNCF

Noch was schreiben dann nachteile

1. Komplexität 2. Neuer Ansatz/Technologie

4.9 Einsatzgebiete

Cloud-Native Architekturen werden derzeit meistens für Systeme benutzt, die entweder mit vielen Daten und/oder mit einer großen Anzahl von Benutzern umgehen müssen. Also generell Systeme, die ein hohes Maß an Skalierbarkeit fordern. Besonders in den Bereichen Streaming und Big Data werden häufig Cloud-Native Architekturen verwendet. Beispiele sind der Streaming-Dienst von Netflix sowie die Cloud Platformen von Goolge (Goolge Cloud Platform) und Amazon (AWS). In den Fällen von Google und Amazon werden Platformen angeboten die es wiederum ermöglichen Cloud-Native Applikation zu entwicklen. Anzumerken ist, dass viele Unternehmen eine Migration ihrer Dienste in die Cloud vorgenommen haben, da die Möglichkeiten für Cloud basierte Systeme erst im letzten Jahrzent wirklich zu einer Option wurde.

5 Fallbeispiel

In diesem Kapitel stellen wir, eine anhand eines selbst ausgedachten Anwendungsfalls, eine Cloud-Native Architektur vor, die wir prototypisch implementiert haben.

5.1 Beschreibung

In unserem Beispiel haben wir einen Nachrichtendienst (ChatApp) entworfen. Benutzer können sich registieren, anmelden und Nachrichten an andere Benutzer senden.

5.2 Anforderungen

5.2.1 Nichtfunktionale Anforderungen

1. Skalierbarkeit Das System muss mit einer großen Zahl von Benutzern, die das System gleichzeit verwenden, umgehen können.

2. Verfügbarkeit Das System muss zu jeder Zeit verfügbar sein.

3. Sicherheit Das System muss Berechtigungen überprüfen können (z.B. darf Benutzer X die Nachrichten lesen) und das System sollte sich gegen übliche Cyberangriffe (z.B. DDoS) schützen können.

4. Änderbarkeit/Erweiterbarkeit Das System muss so entworfen werden, dass weitere Komponenten (z.B. Hochladen von Bildern und Videos) einfach hinzugefügt werden können.

5.2.2 Funktionale Anforderungen

1. Registierung Ein Benutzer kann sich mit seiner E-Mail Adresse und einem Passwort im System registieren.

2. Ein- und Ausloggen Ein Benutzer kann sich mit seiner E-Mail Adresse und seinem Passwort im System anmelden und danach die Funkionen nutzen bis er sich abmeldet.

3. Nachrichten schreiben/lesen Ein Benutzer kann Nachrichten an andere Benutzer senden und Nachrichten, die an ihn gesendet worden sind, abrufen.

5.3 Architekturentwurf

5.3.1 Microservices

Die Architektur basiert auf Microservices, die einem Client, über ein API-Gateway, ihre Funktionen zur Verfügung stellen. Die Aufteilung der Microservies ergibt sich aus Annahme, dass die Funkionalitäten, die sie bereitstellen unterschiedlich oft genutzt werden: Anzahl der Registrierungen < Anzahl der Logins/Logout < Anzahl der geschriebenen Nachrichten.

jeder service eigene datenbank

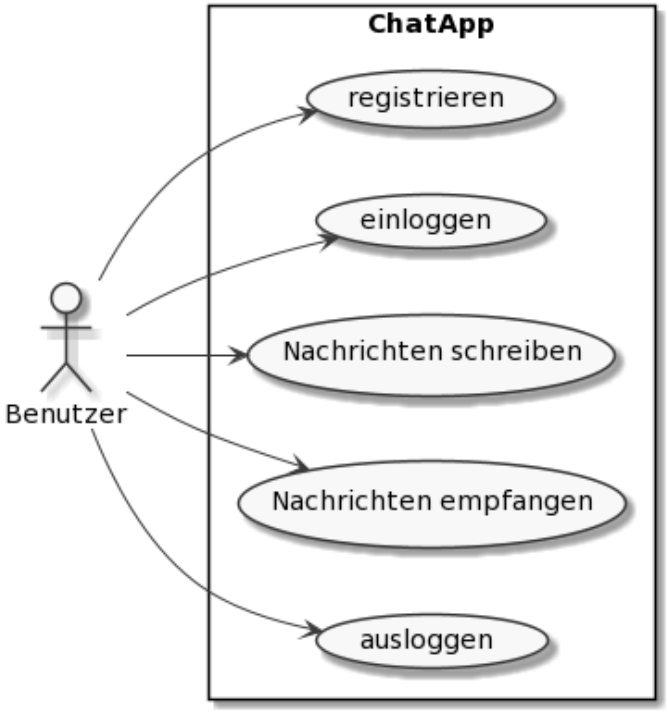


Abbildung 5.1: Use-Case Diagramm

5.3.2 API-Gateway

load balancing schutz vor ddos

5.3.3 Kommunikation

5.3.4 Client

5.4 Implementierung des Prototyps

java react http datenbank stove pipe einzelne programme die individuell gestartet werden können
auch genutzte Technologien aufzeigen

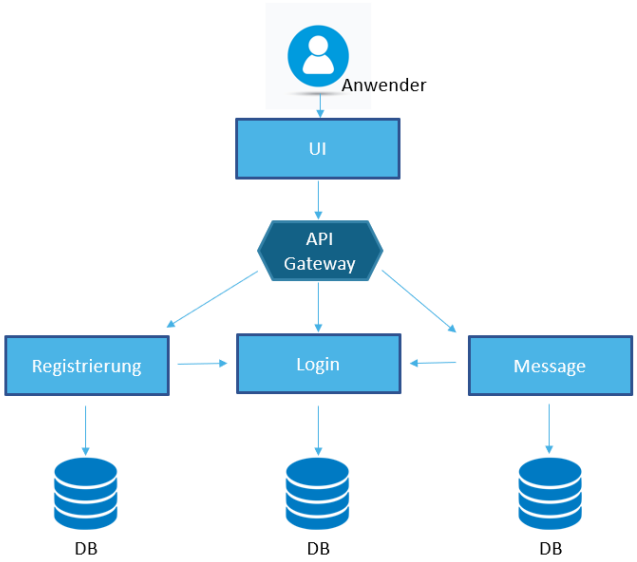


Abbildung 5.2: Architekturentwurf

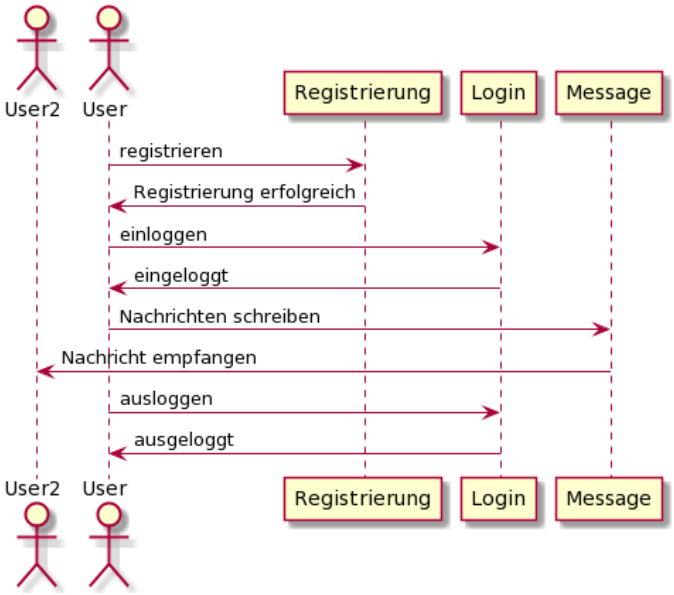


Abbildung 5.3: Sequenzdiagramm

6 Diskussion

In diesem Kapitel diskutieren wird die entworfe Architekturn und den dazugehörigen Prototypen.

6.1 Vergleich mit Eigenschaften Cloud Nativ

skalierbarkeit infrastructure is fluid Updates und Tests verlaufen unscheinbar Sicherheit ist ein Teil der Architektur Globale Ebene

6.2 Microservices

Docker, Kubernetes

6.3 Tradeoffs

sicherheit skalierbarkeit
flexibilität komplexität

6.4 Erweiterbarkeit

7 Fazit

ob prototyp cloud native konform ist warum warum nicht
war gudd awa han ke bock das nochmol zu mache danke

Literatur

[1] Jon Bentley. *Programming Pearls*. Addison–Wesley, 1999.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest und Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[3] Gunter Dueck. *Duecks Trilogie: Omnisophie – Supramanie – Topothesie*. `http://www.omnisophie.com`. Springer, 2005.

[4] Donald E. Knuth. „Big Omicron and Big Omega and Big Theta“. In: *SIGACT News* 8.2 (1976), S. 18–24.

[5] Donald E. Knuth. „Computer Programming as an Art“. In: *Communications of the ACM* 17.12 (1974), S. 667–673.

[6] Ian Sommerville. *Software Engineering*. Boston, MA, USA: Addison-Wesley, 1992.

Abbildungsverzeichnis

2.1 Erstes Bild, Völklinger Hütte 8
2.2 Völklinger Hütte, *.jpg 9
2.3 Mehrere Abbildungen nebeneinander 9
2.4 Beide Formate im Vergleich 10
2.5 *PNG*-Format 10
2.6 *JPG*-Format 10

5.1 Use-Case Diagramm 24
5.2 Architekturentwurf 25
5.3 Sequenzdiagramm 25

Tabellenverzeichnis

2.1 Beispiel 1 7
2.2 So sollte man es nicht machen! Beispiel für einen schlechten Tabellenstil . 8

Listings

2.1 Allgemeines Format 6
2.2 Tabelle 2.1 6
2.3 Tabelle 2.2 7
2.4 Erstes Listing 11
2.5 Externer Quellcode 11

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 2.1). 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