



PERFORMANCE PROPHYLAXE

WIE BAUE ICH PERFORMANTE WEB-APPLIKATIONEN?

Claudia Maderthaner claudia.maderthaner@gepardec.com

AGENDA

- // Frontend Performance
- // Backend Performance
- // JPA/Hibernate Performance
- // Design for Performance



Premature optimization is the root of all evil (or at least the most of it) in programming.

The Art of Computer Programming
— Donald Knuth

HERANGEHENSWEISE

- // **Lesbaren** Code schreiben
- // Testen
- // Wenn notwendig, optimieren



FRONTEND-PERFORMANCE

ASSESSMENT TOOLS

// PageSpeed Insights

// WebPageTest

// Browser Tools

// Network Monitor

// Performance Monitor

CSS

- // CSS-Größe reduzieren
 - // Unnötiges CSS entfernen
 - // CSS in Module aufteilen
 - // CSS Minifizieren
- // CSS-Selektoren vereinfachen
 - // Styles nur auf die relevanten Elemente anwenden
 - // Redundanzen vermeiden
 - // `@import` vermeiden
 - // CSS im `<head>` laden

BEISPIELE

```
1 header h1.highlight {  
2   font-size: 2.4rem;  
3   font-weight: 600;  
4   color: #000000;  
5   margin-top: 0rem;  
6   margin-right: 0rem;  
7   margin-bottom: -0.4rem;  
8   margin-left: 0rem;  
9 }
```

```
1 header h1.highlight {  
2   font-size: 2.4rem;  
3   font-weight: 600;  
4   color: #000;  
5   margin: 0 0 -.4rem;  
6 }
```

BILDER

- // Geeignetes Bildformat wählen
 - // Rasterformate (JPEG, PNG, GIF)
 - // Vektorgrafiken (SVG)
- // Dateigröße der Bilder optimieren
- // Bilddimension definieren

- // Sprites
- // Lazy Loading

RESPONSIVE IMAGES

```

```

```

```

```

```

```

```

RESPONSE IMAGES

```
<picture>
  <source media="(max-width: 600px)"
    srcset="img/dog_narrow_200.jpg 200w, img/dog_narrow_400.jpg 400w,
      img/dog_narrow_560.jpg 560w"/>
  
</picture>
```

SCHRIFTEN

- // Anzahl der Schriftschnitte limitieren
- // Variable Fonts
- // Fallback Font

```
font-family: Roboto, Arial, sans-serif
```

<https://fonts.google.com/>

SCRIPTE

- // Kritische Ressourcen beim Laden priorisieren
- // Ausführung unkritischer Ressourcen aufschieben

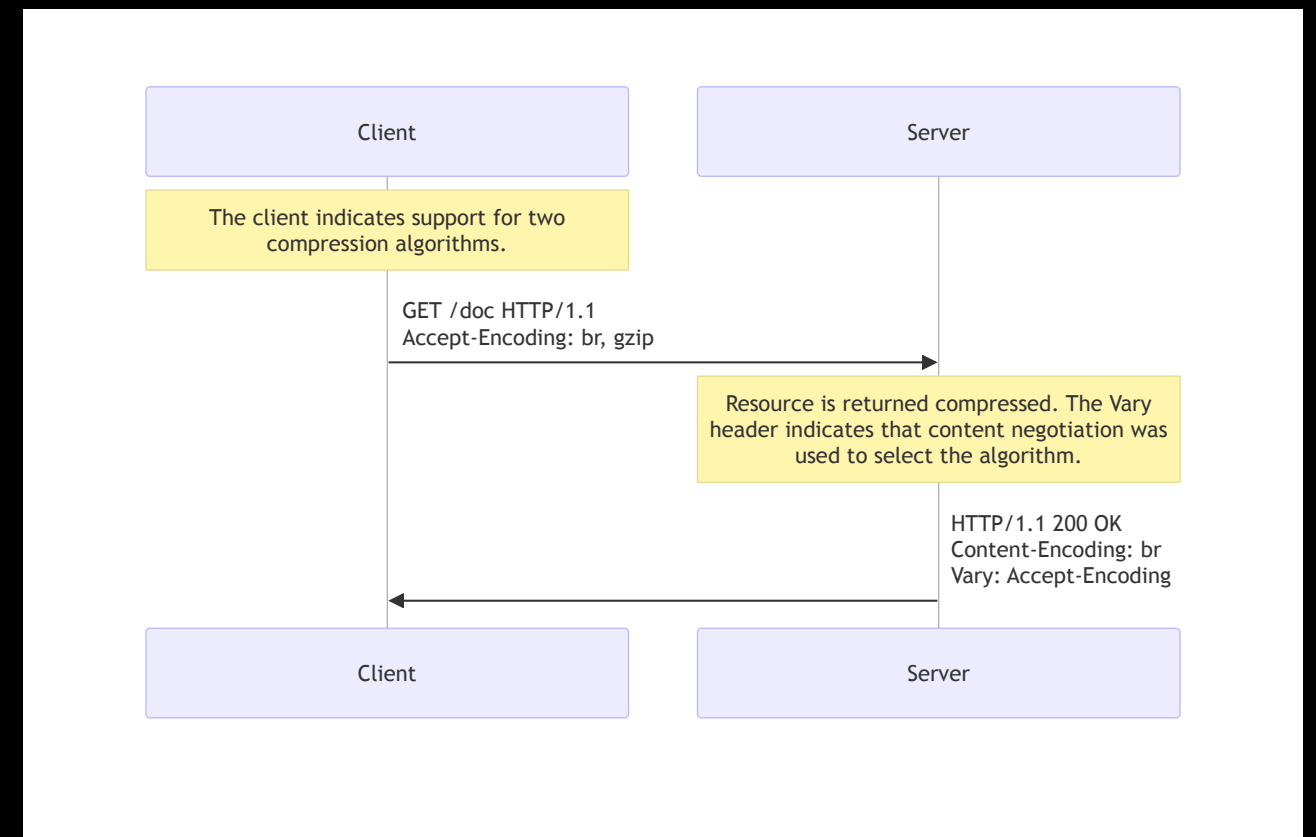
- // DOM Manipulation reduzieren
- // Änderungen als Batches ausführen
- // HTML-Struktur vereinfachen
- // Iterationen reduzieren
- // Asynchrone Ausführung
- // Web Workers

ANGULAR PERFORMANCE OPTIMIERUNG

- // trackBy in ngFor-Schleifen
- // Keine komplexen Bedingungen in ngIf
- // OnPush Detection Strategy
- // Reactive Programming
 - // RxJS
 - // Signals
- // Production Mode
- // Minifizierung
- // Tree-Shaking
- // AOT Compilation
- // Lazy Loading
- // Server-side Rendering (Angular Universal)

HTTP COMPRESSION

- // End-to-end Compression
- // Wird von allen aktuellen Servern und Browsern unterstützt
- // *Apache* `mode_deflate`
- // *Nginx* `ngx_http_gzip_module`
- // *IIS* `<httpCompression>`
- // Algorithmen: `gzip` oder `br`



HTTP CACHING

// Wiederverwendung von geladenen
Inhalten

```
Cache-Control: private
```

```
Cache-Control: max-age=3600
```

```
Expires: Tue, 28 Feb 2022 22:22:22 GMT
```

// Private Caches

// Browser Cache

// Shared Caches

// Proxies

// HTTPS unterbindet Proxy-Caching

// Managed Caches

// Reverse Proxies, CDNs, Service
Workers mit Cache API

HTTP/2

- // Kompression von Headers
- // Request Priorisierung
- // Multiplexing
- // Server Push

LINKS ZUM THEMA

- // MDN Web Docs - Performance
- // MDN Web Docs - Response Images
- // Can I Use

HEY GUYS LOOK

I'M A COFFEE BREAK!

HA HA GET IT?



SERIOUSLY THOUGH,
I'M REALLY HURT



BACKEND-PERFORMANCE

PDCA

// Plan

// Do

// Check

// Act/Adjust

HERANGEHENSWEISE

- // **Lesbaren** Code schreiben
- // Testen
- // Wenn notwendig, optimieren

PERFORMANCE MESSEN

// JDK Flight Recorder (JFR)

// JDK Mission Control (JMC)

// VirtualVM

// Java Microbenchmark Harness (JMH)

Post Mortem

// Thread Dump

// Heap Dump

LAUFZEITUMGEBUNG

// Aktuelle Versionen verwenden

// Java

// Libraries und Frameworks

// Ausreichende Ressourcen

// Memory

// CPU

DATENSTRUKTUREN

// Listen

// LinkedList

// ArrayList

// Arrays

// Stacks und Queues

// Trees

// Tables

// Graphs

// Heaps

SCHLEIFEN

for/while/do-while

// Initialisierung

// Schleifenbedingungen

// Implementierung

Vergleichbare Konstrukte

// Batches

// Jobs

// Migrationen



Achtung bei verschachtelten

Schleifen!

OBJECT POOLING

- // Wiederverwendung von Objekten
- // Verringert Overhead der Objekterzeugung
- // Reduziert Aufwände für Garbage Collection

Vorteile

- // Performance
- // Ressourcenmanagement
- // Skalierung

Nachteile

- // Komplexität
- // Speicherverbrauch
- // Ressourcenverbrauch
- // Synchronisation

BEISPIEL OBJECT POOL

STRINGS

- // String Pooling
- // Compact Strings (Java 9)
- // String Concatenation

BEISPIELE STRINGS

THREADS

- // Synchronisation
 - // Race conditions
 - // Deadlocks
- // Immutable Objects
- // Thread Pools

CACHING

// Zwischenspeicher für häufig abgerufene Daten

Vorteile

- // bessere Performance
- // reduzierte Last
- // verbesserte Skalierbarkeit

Arten

- // In-Memory-Cache
- // Verteilter Cache (Redis, Infinispan)

Cache-Strategien

- // Write-Through
- // Write-Back
- // Read-Through

Herausforderungen

- // Cache-Größe
- // Cache-Invalidierung
- // Cache-Trefferquote

BEISPIEL GUAVA CACHE (IN-MEMORY-CACHE)

<https://github.com/google/guava/wiki/cachesexplained>

```
LoadingCache<String, String> cache = new CacheBuilder.newBuilder()
    .maximumSize(100)
    .expireAfterWrite(10, TimeUnit.SECONDS)
    .build(
        new CacheLoader<String, String>() {
            public String load(String key) {
                return loadValue(key);
            }
        }
    );

String read1 = cache.get("key1"); // Wert wird neu geladen
String read2 = cache.get("key1"); // Wert wird aus Cache zurückgegeben
```

LOGGING

Vorteile

- // Fehlerdiagnose
- // Monitoring und Analyse

Nachteile

- // I/O-Operationen
- // String-Verarbeitung
- // Synchronisation
- // Ressourcenverbrauch

Herausforderungen

- // Verständliche Log-Messages formulieren
- // Sicherstellen, dass gelogged wird
- // "Unsichtbare" Fehler loggen

LOGGING OPTIMIERUNG

- // An Umgebung angepasstes Log-Level
- // Menge der Log-Daten
- // Überprüfung des Log-Levels
- // Effiziente String-Verarbeitung
- // Caching häufiger Log-Messages
- // Asynchrones Logging
- // Geeignetes Log-Framework

FEHLERBEHANDLUNG

Nachteile

- // Erstellung von Stack Traces
- // Objekt-Erstellung
- // Kontrollfluss-Änderung

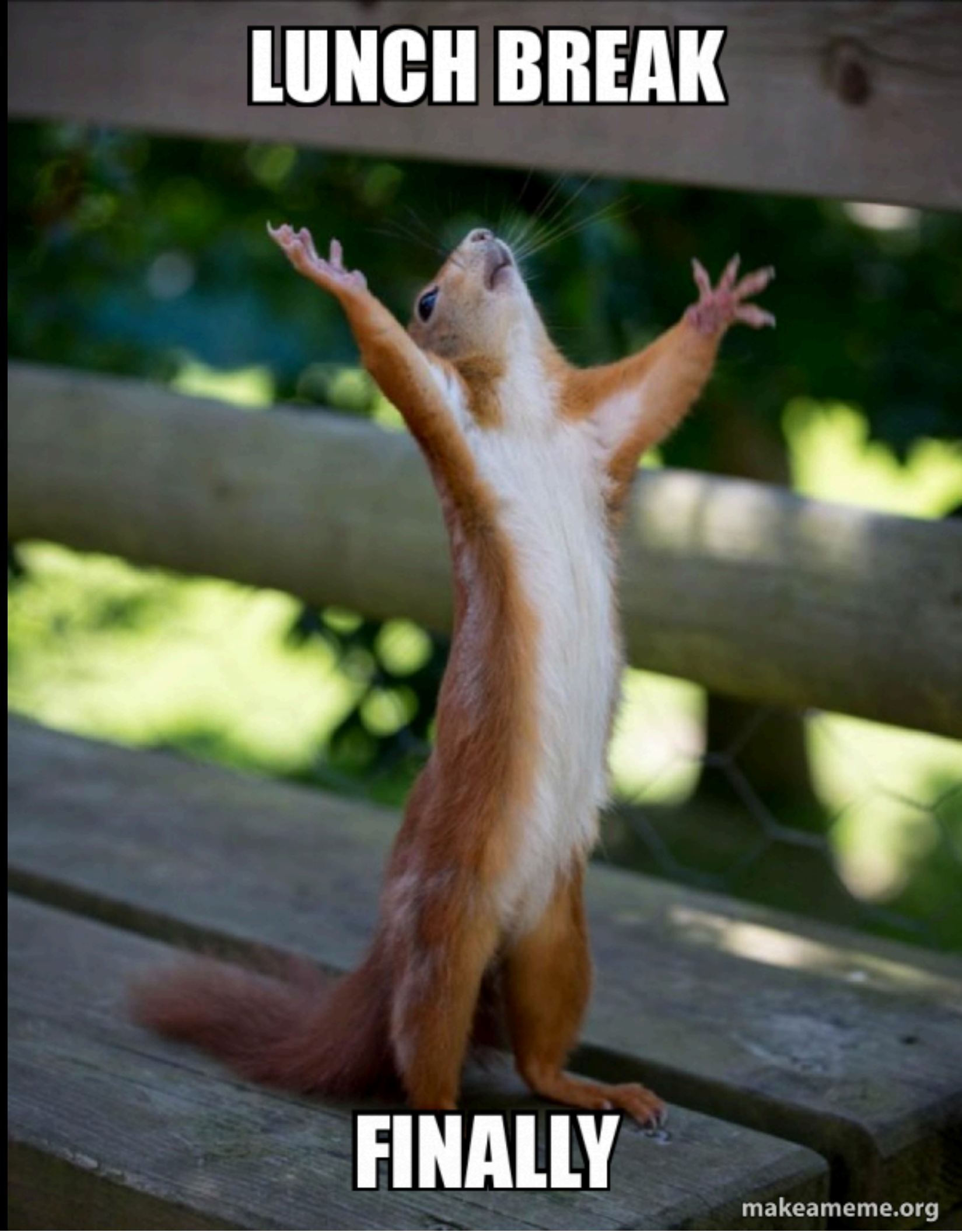
Optimierung

- // Exceptions nur in Ausnahmefällen
- // Frühzeitiges Behandeln von Exceptions
- // Spezifische Exceptions verwenden
- // Geschäftslogik nicht über Exceptions abbilden

BÜCHER UND LINKS ZUM THEMA

- // Scott Oaks: Java Performance, 2nd Edition, O'Reilly, 2020
- // Edward Lavieri Jr: High Performance with Java, Packt Publishing, 2024
- // [Java Almanach](#)

LUNCH BREAK



FINALLY



JPA/HIBERNATE PERFORMANCE

DATABASE DESIGN

- // Datenbankschema
 - // Normalisiert vs. Denormalisiert
 - // Views
- // Indizes
- // Query Optimierung

BEISPIEL DENORMALISIERTES SCHEMA

BookID	Author	Title	Publisher	Price	Genre
1	M. Mustermann	Java Performance	Book Co.	40.00	Non-Fiction
2	K. Anderer	Java - eine Odyssee	BoPub	25.00	Fiction
3	M. Mustermann	Java Concurrency	BoPub	32.00	Non-Fiction
4	K. Anderer	Java EE - Nitty Gritty	Book Co.	80.00	Non-Fiction

BEISPIEL NORMALISIERTES SCHEMA

```
SELECT * FROM Books;
```

BEISPIEL NORMALISIERTES SCHEMA

BookID	AuthorID	Title	PublisherID	Price	Genre
1	1	Java Performance	1	40.00	Non-Fiction
2	2	Java - eine Odyssee	2	25.00	Fiction
3	1	Java Concurrency	2	32.00	Non-Fiction
4	2	Java EE - Nitty Gritty	1	80.00	Non-Fiction

AuthorID	Author
1	M. Mustermann
2	K. Anderer

PublisherID	Publisher
1	Book Co.
2	BoPub

BEISPIEL DENORMALISIERTES SCHEMA

```
SELECT * FROM Books b
  JOIN Authors a ON b.AuthorID = a.AuthorID
  JOIN Publishers p ON b.PublisherID = p.PublisherID;
```

INDIZES

```
CREATE INDEX idx_authors_authorid ON Authors (AuthorID);
```

QUERY OPTIMIERUNG

// Vermeide SELECT *, führe die notwendigen Spalten explizit an

```
SELECT Title, Price FROM Books;
```

// Schränke Abfragen so gut wie möglich ein

```
SELECT Title, Price FROM Books WHERE Genre = 'Fiction';
```

// Begrenze die Anzahl der Ergebnisse

```
SELECT Title, Price FROM Books WHERE Genre = 'Fiction' LIMIT 10;
```

HIBERNATE OPTIMIERUNG

- // Query Optimierung
- // Caching
- // Batch Processing
- // Datenbank Indizierung
- // Connection Pooling
- // Fetch Strategien
- // Transaktionsmanagement

DATASOURCE KONFIGURATION

```
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/yourdatabase</property>  
<property name="hibernate.connection.username">yourusername</property>  
<property name="hibernate.connection.password">yourpassword</property>  
<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>  
<property name="hibernate.show_sql">false</property>
```

DATASOURCE KONFIGURATION

```
<property name="hibernate.transaction.factory_class">  
    org.hibernate.transaction.JTATransactionFactory  
</property>  
<property name="hibernate.transaction.manager_lookup_class">  
    org.hibernate.transaction.JBossTransactionManagerLookup  
</property>
```

DATASOURCE KONFIGURATION

```
<property name="hibernate.c3p0.min_size">5</property>  
<property name="hibernate.c3p0.max_size">20</property>  
<property name="hibernate.c3p0.timeout">300</property>  
<property name="hibernate.c3p0.max_statements">50</property>  
<property name="hibernate.c3p0.idle_test_period">3000</property>
```

DATASOURCE KONFIGURATION

```
<property name="hibernate.cache.use_second_level_cache">true</property>  
<property name="hibernate.cache.region.factory_class">  
    org.hibernate.cache.ehcache.EhCacheRegionFactory  
</property>  
<property name="hibernate.default_batch_fetch_size">16</property>
```

QUERY OPTIMIERUNG

// Projections

// FetchType

// @ManyToMany mit Set

QUERY CACHING

1st Level Cache

- // immer aktiv

- // enthält alle Entities, die gemanaged werden

2nd Level Cache

- // session-unabhängig

BATCH PROCESSING

DATENBANK INDIZIERUNG

```
@Table(indexes = @Index(name="idx_msg_text", columnList = "text"))
```


CONNECTION POOLING

// HikariCP

// Apache Commons DBCP

// C3P0

HIBERNATE STATISTIKEN

```
<property name="hibernate.generate_statistics" value="true" />  
<property name="hibernate.log_slow_query" value="1" />
```

REFERENZEN

// Hibernate Performance Tuning

Take a little



COFFEE BREAK

lovethispic.com

DESIGN FOR PERFORMANCE

INFRASTRUKTUR

// CPU

// Memory

// Garbage Collector

ARCHITEKTUR

- // Skalierbarkeit
- // Asynchrone Kommunikation
- // Application Gateway
- // Backend for Frontend

FEHLERTOLERANZ

// Load Balancing

// Stateless

// Sticky Sessions

// Failover

// Datenbank

// JMS/ESB

// Circuit Breaker

// Timeout Handling

PERFORMANCE TESTS

// JMeter

// Gatling

// Zipkin

MONITORING

- // Performance Logging
- // Kibana
- // Prometheus

IMMER AM PULS DER ZEIT

- // Aktuelle Versionen verwenden
- // Neue Entwicklungen verfolgen
- // DevMode in Produktion deaktivieren

A still from the movie Toy Story showing Woody and Buzz Lightyear. Woody is on the left, looking concerned. Buzz is on the right, looking excited and gesturing with his hands. The background is a simple room with a door and a window.

QUESTIONS

QUESTIONS EVERYWHERE

makeameme.org



WAITING FOR THE MEETING TO END

KONTAKT

Claudia Maderthaner
Gepardec IT Services GmbH

- // Wien: Ernst-Melchior-Gasse 24, 1020 Wien
- // Linz: Europaplatz 4, 4020 Linz

WEB www.gepardec.com

*gepard*ec
simplify your business