



Eine für viele

# Multi-Tenancy-Software

Sascha Groß  
[sascha.gross@mathema.de](mailto:sascha.gross@mathema.de)

# Agenda

- Warum?
- Was?
- Wie?

# Meine Berührungspunkte mit mandantenfähiger Software

- Kreditvergabesystem einer Bank - 4,5 Jahre
  - Versicherungsvertrieb - 2 Jahre
  - Datenaustauschmiddleware für Bankdienstleister -  $\frac{3}{4}$  Jahre
  - Unternehmensportal - 7 Jahre
  - Backend für Banking für Bankdienstleister - 1 Jahr
  - Datenaustauschsystem für CO2 Fußabdrücke - 1 Jahr
- 
- **ca.  $\frac{2}{3}$  meines IT-Lebens**

# Definition Mandantenfähigkeit

<https://de.wikipedia.org/wiki/Mandantenfähigkeit>

Als mandantenfähig (auch mandantentauglich) wird Informationstechnik bezeichnet, die auf demselben Server oder demselben Software-System mehrere Mandanten, also Kunden oder Auftraggeber, bedienen kann, **ohne dass diese gegenseitigen Einblick in ihre Daten**, Benutzerverwaltung und Ähnliches haben. Ein IT-System, das dieser Eigenschaft genügt, bietet die Möglichkeit der disjunkten, mandantenorientierten Datenhaltung, Präsentation (GUI) und Konfiguration (Customizing). Jeder Kunde **kann nur seine Daten sehen und ändern**.

Ein System wird nicht mandantenfähig, indem man für jeden Mandanten eine eigene Instanz (Kopie) des Systems erstellt.

# Wer kann ein Mandant (Tenant) sein?

- Unternehmen
- Niederlassung
- Abteilung
- Organisationseinheit
- Behörde
- ...

## Keine Mandanten sind

- Benutzer
- Benutzergruppen

# Warum Mehrmandantenfähige Software?

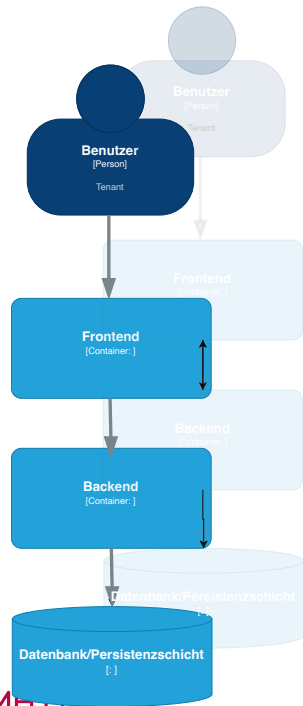
- Wiederverwendung von Software & Hardwareressourcen
  - Minimierung Betriebskosten
  - Minimierung Wartungsaufwand
  - Minimierung Erstellungskosten
- 
- **Wirtschaftliche und organisatorische Gründe**

# Scheinbare Nachteile

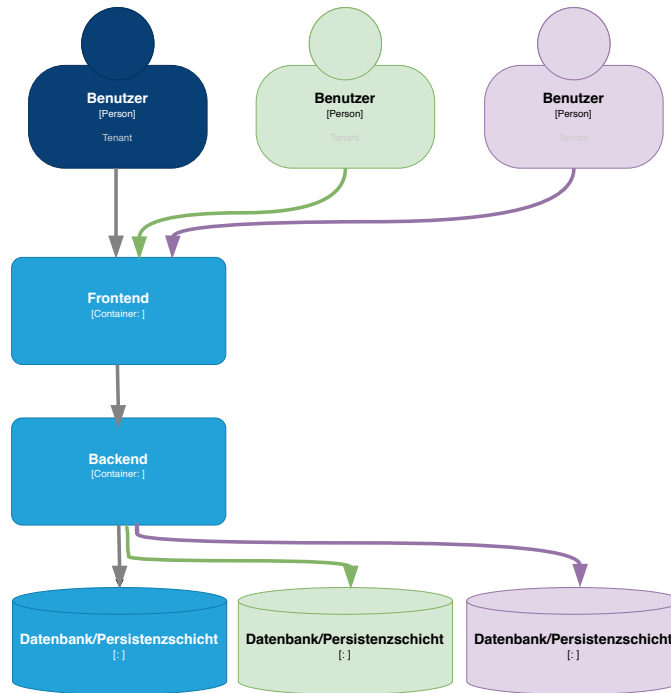
- Feature für einzelnen Mandanten
  - Lösung: Featuretoggle & Konfiguration
- Gleicher Softwarestand bei allen Mandanten
- Bugs bei allen Mandanten
- Ausfall bei allen Mandanten

# Architekturen

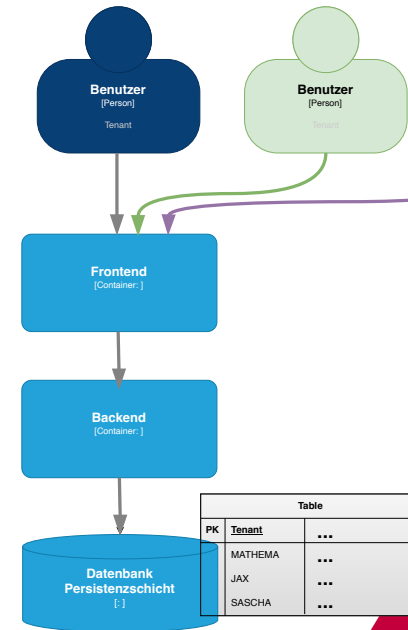
Silo model



Bridge model



Pool model





# Silo model

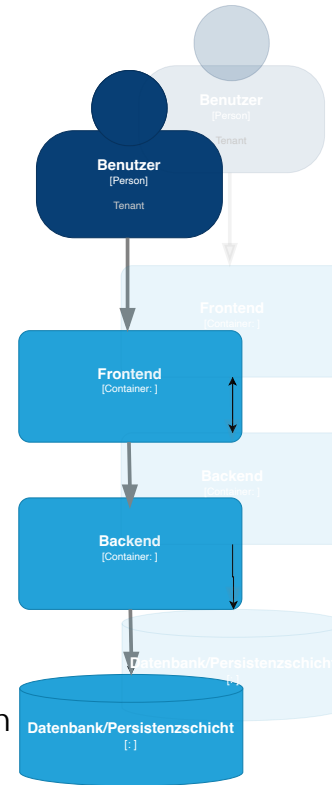
Getrennte Systeme pro Mandant

## Vorteile

- Nur ein Mandant, also *normale* Software
- Strikte Trennung von Daten (rechtliche Aspekte)
- Jeder Mandant hat eigene Instanzen
- Hohe Verfügbarkeit, da nur ein Mandant

## Nachteile

- *Copy & Paste* der Infrastruktur
- Verwaltungsaufwand/Überwachung der Ressourcen

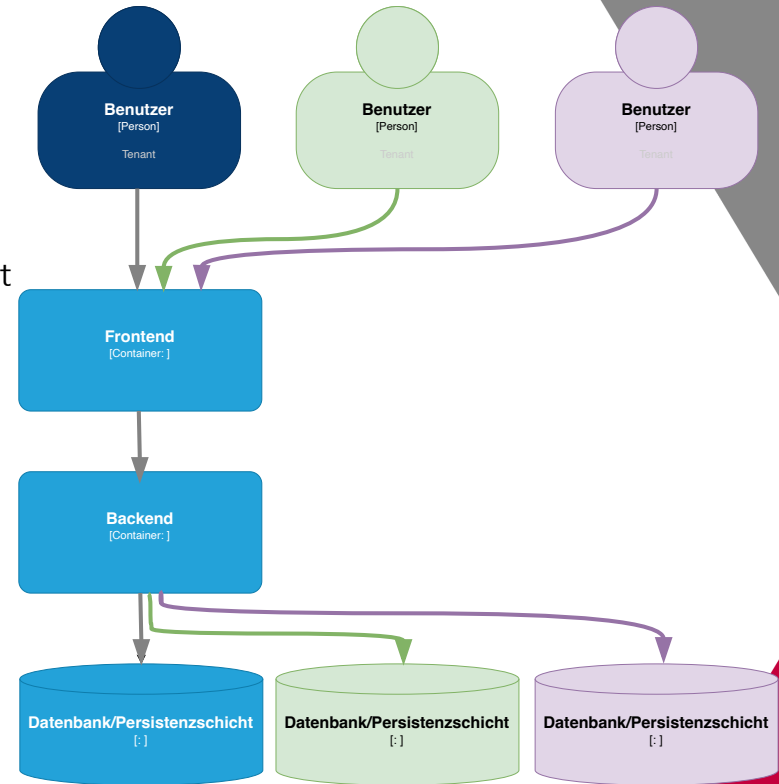


# Bridge model

- Gemeinsame Benutzung aller Ressourcen
- Gemeinsame Benutzung der Applikationsschicht
- Gemeinsame Benutzung der Datenhaltungsschicht
- Trennung der Datenhaltung über getrennte Datasoures/Datenbankschemata (in der gleichen Datenbank)

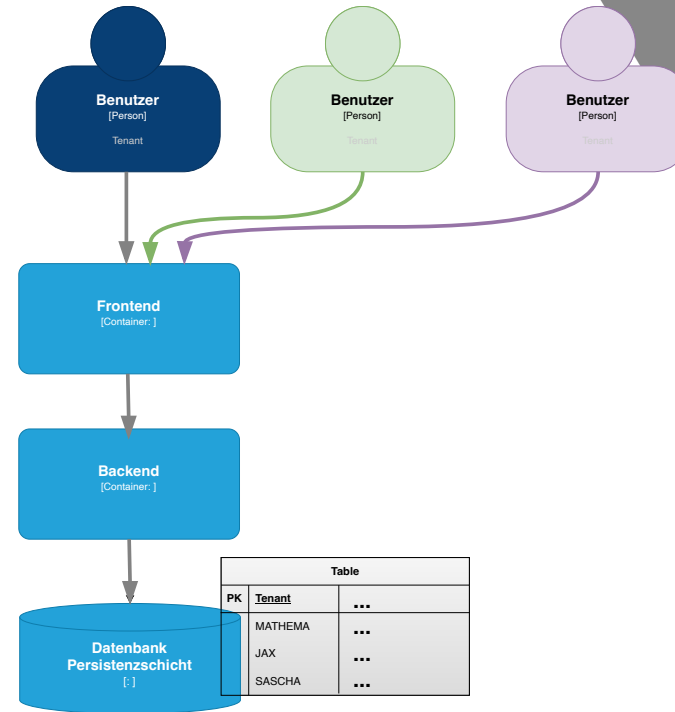
Nachteile:

- Getrennte Datenbankschemata und somit erhöhte Anzahl von Verbindungen



# Pool model

- Gemeinsame Benutzung der Infrastructure Ressourcen
- Gemeinsame Benutzung des Applikationsschicht
- Trennung der Datenhaltung über Diskriminator Spalte in Datenbank



# Technische Umsetzung - Mandantenerkennung

- HTTP Header
- Query Parameter `http://mein-super-service.de/super/?tenant= < tenant >`
- Url - Subdomain `http://< tenant >.mein-super-service.de`
- JSON Web Token (JWT)
- ...

# Datenbank

- Spalte **tenant** in Tabelle
- Row level Security (RLS)
  - Erlaubt Zugriff auf Zeile nur, wenn konfigurierte Bedingung erfüllt ist

# Datenbank - RLS - Postgres

```
1 CREATE TABLE IF NOT EXISTS asset
2 (
3   id      BIGINT NOT NULL,
4   tenant  VARCHAR(255) NOT NULL,
5   ...
6   PRIMARY KEY (id, tenant)
7 );
8
9 ALTER TABLE ONLY asset FORCE ROW LEVEL SECURITY;
10 ALTER TABLE asset ENABLE ROW LEVEL SECURITY;--do not forget to enable it
11
12 CREATE POLICY tenant_isolation_policy ON
13   asset USING (((tenant)::text = current_setting('app.current_tenant'::text)));
```

# Datenbankzugriff

```
1 set app.current_tenant = '< tenant >';
```



## Database Connection: PostgreSQL 35432

<jdbc:postgresql://localhost:35432/MULTITENANT>

Connection

**Properties**

Database Info

Data Types

### Connection Properties

Database Profile

Driver Properties

▼ PostgreSQL

Authentication

Delimited Identifiers

Qualifiers

Physical Connection

Transaction

Encoding

SQL Statements

Connection Hooks

Color and Border

### Database Connection Hooks

These properties specifies commands that will be sent to the database server directly after a successful connection.

Any problems occurring while executing these commands are reported in the **Tools->Debug Window**.

☒ Run SQL at Connect:

```
set app.current_tenant = 'mathema';
```

☒ Run SQL at Disconnect:



# Datenbank - RLS - Applikationsseitig

```
1  public Connection getConnection() throws SQLException {  
2      Connection connection = super.getConnection();  
3      try (Statement sql = connection.createStatement()) {  
4          sql.execute("SET app.current_tenant = '" + tenantService.getCurrentTenant() + "'");  
5      }  
6      return connection;  
7  }
```



# Java - TenantService

```
1  public class TenantService {  
2      ThreadLocal<String> TLS_TENANT = new InheritableThreadLocal<>();  
3  
4      void setCurrentTenant(String tenant) {  
5          TLS_TENANT.set(tenant);  
6      }  
7      void remove() {  
8          TLS_TENANT.remove();  
9      }  
10  
11     public String getCurrentTenant() {  
12         return TLS_TENANT.get();  
13     }  
14 }
```

# Java - TenantFilter - JaxRS

```
1  @Provider
2  public class TenantFilter implements ContainerRequestFilter {
3
4      @Inject
5      private TenantService tenantService;
6
7      @Override
8      public void filter(ContainerRequestContext requestContext) {
9          var headers = requestContext.getHeaders();
10         var tenantList = headers.get("tenant");
11         if (tenantList == null) {
12             tenantList = Collections.emptyList();
13         }
14
15         var tenant = tenantList.stream().findFirst();
16
17         if (tenant.isPresent()) {
18             tenantService.setCurrentTenant(tenant.get());
19         } else {
20             tenantService.setCurrentTenant("default");
21         }
22     }
23 }
```

# Batch - Dunkelverarbeitung

```
1      private static final List<String> TENANTS =
2          List.of("mathema", "campus", "jax", "sascha");
3
4      public void start() {
5          TENANTS.forEach(this::doIt);
6      }
7
8      private void doIt(String tenant) {
9          try {
10             tenantService.setCurrentTenant(tenant);
11
12             doIt();
13         } finally {
14             tenantService.remove();
15         }
16     }
17
18     private void doIt() {
19         ...
20     }
```

# Annotation bei JPA (EclipseLink)

```
1  @Entity
2  @org.eclipse.persistence.annotations.Multitenant
3      (value = SINGLE_TABLE | TABLE_PER_TENANT)
4  @org.eclipse.persistence.annotations.TenantDiscriminatorColumn(name = "tenant")
5  public class ...
```

# Annotation bei JPA (Hibernate)

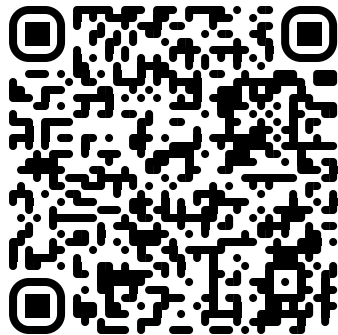
```
1  @Entity
2  public class ...
3
4      @org.hibernate.annotations.TenantId
5      private String tenant
```

```
1  interface org.hibernate.context.spi.CurrentTenantIdentifierResolver<T> {
2      T      resolveCurrentTenantIdentifier()
3      ...
}
```



Eine für viele - Multi-Tenancy-Software

# Dankeschön / Fragen?



<https://github.com/saschagr/multitenant-service>

Sascha Groß

[sascha.gross@mathema.de](mailto:sascha.gross@mathema.de)