



SymfonyLive
BERLIN 2023

Multi Tenancy – Yay or Nay?

Ein Talk von Simon Möller-Börkel

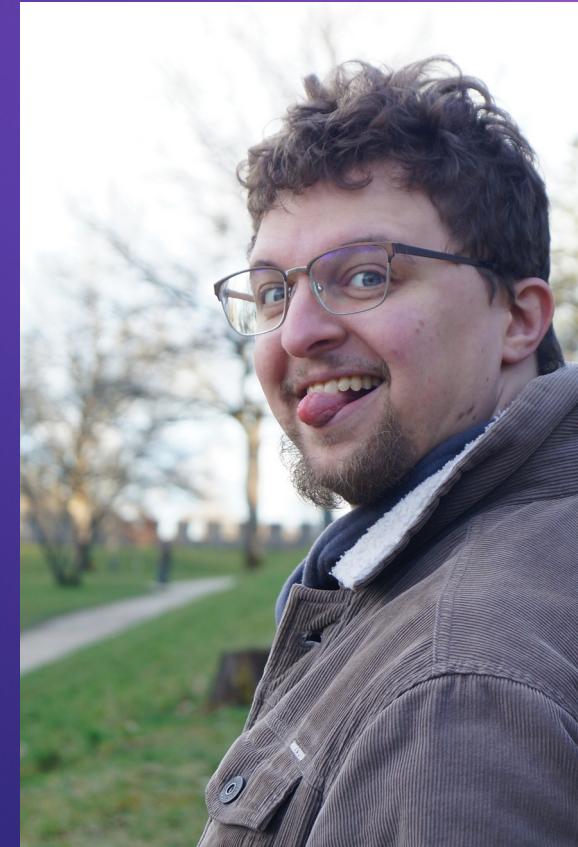
Was euch im Vortrag erwartet

- Was ist Multi Tenancy überhaupt?
- Welche Arten der Implementation gibt es und wann wähle ich welche?
- Wie implementiere ich die Ansätze in Symfony?

Simon Möller-Börkel
Software-Developer
aus dem Umland von Bonn



SiMoeBoe



Multi Tenancy?

- Multi Tenancy aka Mandantenfähigkeit
 - Eine Instanz der Software bedient mehrere Mandanten, ohne das diese voneinander wissen
 - Ein Mandant kann dabei sein
 - ein Kunde
 - ein Whitelabel-Produkt
 - eine Länderinstanz
 - ...

Vorteile von Mandantenfähigkeit

- Ressourcen- und Kostensparend, da prinzipiell eine physikalische Instanz beliebig viele Kunden bedienen kann
- Updates und Features stehen mit einem Rollout unmittelbar allen Mandanten zur Verfügung
- Hohe Skalierbarkeit möglich

Nachteile von Mandantenfähigkeit

- Höhere Komplexität in der Entwicklung
- Größere Gefahr eines Datenlecks
- Ausfallzeiten betreffen alle Mandanten
- Weniger Flexibilität bei Featurewünschen einzelner Mandanten
- Durch geteilte Ressourcen kann ein Mandant die Anwendung für alle Mandanten verlangsamen

Multi Tenancy: Yay or Nay?

- Hohe individuelle Anpassungen und Features?
- Wenige weitere Mandanten zu erwarten?

 Nay!

Multi Tenancy: Yay or Nay?

- Viele gleichartige Mandanten/Instanzen?
- Schnelles „time to market“ für alle Mandanten wichtig?

 Yay!

Mögliche Userstories

Als Product Owner will ich, dass ein Mandant ausschließlich auf seine eigenen Daten zugreifen kann.

Mögliche Userstories

Als Manager will ich einen Überblick über alle Dateien, um Statistiken und Suchen über alle Daten aller Mandanten hinweg nutzen zu können.

Mögliche Userstories

Als Support will ich Zugriff auf alle Daten aller Mandanten, um Support leisten zu können.

Mögliche Userstories

Als Jurist will ich, das Mandantendaten nicht nur logisch, sondern physikalisch getrennt sind.

Mögliche Userstories

Als User mehrerer Mandanten will ich ohne erneuten Login zwischen den Mandanten wechseln können.

Mögliche Userstories

Als Anbieter will ich, das Mandanten nicht wissen, dass sie auf einem Multimandantensystem betrieben werden.

Mögliche Userstories

Als Sys-Admin will ich pro Mandant individuelle Backups erstellen.

Multi Tenancy Architekturvarianten

- Alle Daten in einer Datenbank/Storage
 - logisch voneinander getrennt
- Daten in verschiedenen Datenbanken/Storages
 - physikalisch getrennt

An was müssen wir immer denken?

- Aktuellen Tenant verwalten
- Tenant selektieren
 - bei Requests
 - bei Commands
- Zugriff auf Tenants nur für berechtigte User
- Switch zwischen Tenants ohne Neu-Login

```
#[ORM\Entity(
    repositoryClass: TenantRepository::class
)]
class Tenant
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 255)]
    private ?string $domain = null;

    #[ORM\Column(length: 255)]
    private ?string $name = null;

    #[ORM\Column(length: 255)]
    private ?string $database = null;

    #[ORM\ManyToMany(
        targetEntity: User::class,
        mappedBy: 'tenants'
    )]
    private Collection $users;
```

```
final class TenantManager
{
    private Tenant $currentTenant;

    public function __construct(
        private readonly TenantRepository $tenantRepository,
    ) {
    }

    public function getCurrentTenant(): Tenant
    {
        return $this->currentTenant;
    }

    private function setCurrentTenant(Tenant $tenant): void
    {
        $this->currentTenant = $tenant;
    }

    /** @throws UnknownTenantException */
    public function setCurrentTenantByDomain(string $domain): void
    {
        $tenant = $this->tenantRepository->findOneBy(['domain' => $domain]);
        if ($tenant === null) {
            throw new UnknownTenantException($domain);
        }

        $this->setCurrentTenant($tenant);
    }
}
```

```
#[AsEventListener]
public function setTenantByRequest(RequestEvent $requestEvent): void
{
    $this->tenantManager->setCurrentTenantByDomain($requestEvent->getRequest()->getHttpHost());

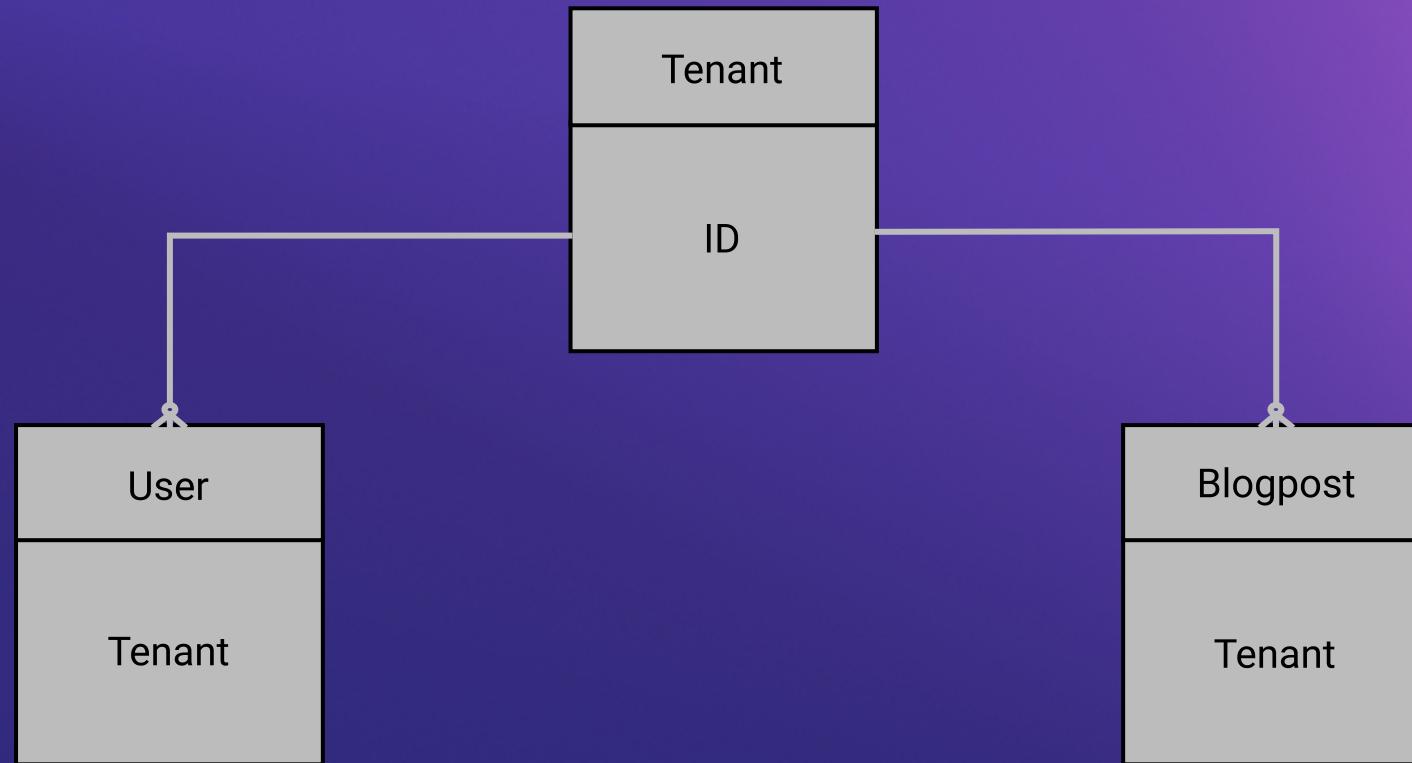
    $user = $this->security->getUser();
    if ($user instanceof User) {
        if (!$user->getTenants()->contains($this->tenantManager->getCurrentTenant())) {
            $this->security->logout(false);
            throw new AuthenticationCredentialsNotFoundException();
        }
    }
}
```

```
#[AsEventListener]
public function setTenantByCLI(ConsoleCommandEvent $commandEvent): void
{
    $input = $commandEvent->getInput();
    if (!$input->hasOption('tenant')) {
        $commandEvent->getCommand()->addOption('tenant', null, InputOption::VALUE_REQUIRED, 'Tenant ID or Domain');
        $input->bind($commandEvent->getCommand()->getDefinition());
    }

    if ($input->hasOption('tenant') && $input->getOption('tenant') !== null) {
        $this->tenantManager->setCurrentTenantByDomain($input->getOption('tenant'));
    }
}
```

```
framework:  
  session:  
    handler_id: null  
    cookie_secure: auto  
    cookie_samesite: lax  
    cookie_domain: 'app.localhost'  
    storage_factory_id: session.storage.factory.native
```

Single Database



```

class TenantAwareFilter extends SQLFilter
{
    public function addFilterConstraint(ClassMetadata $targetEntity, $targetTableAlias): string
    {
        if (!in_array(IsTenantSpecificEntity::class, $targetEntity->getReflectionClass()->getTraitNames())) {
            return '';
        }

        if ($this->getParameter('tenant') === '') {
            throw new NoTenantSpecifiedException();
        }

        return $targetTableAlias . '.tenant_id = ' . $this->getParameter('tenant');
    }
}

```

```

private function setCurrentTenant(Tenant $tenant): void
{
    $this->currentTenant = $tenant;

    $this->entityManager->getFilters()
        ->getFilter('tenant_aware')
        ->setParameter('tenant', $tenant->getId());
}

```

```

doctrine:
    orm:
        filters:
            tenant_aware:
                class: App\Tenant\TenantAwareFilter
                # We activate the filter with no specific
                # tenant to avoid queries without setting
                # the current tenant
                enabled: true
                parameters:
                    tenant: null

```

Geteilte Datenhaltung - Pro

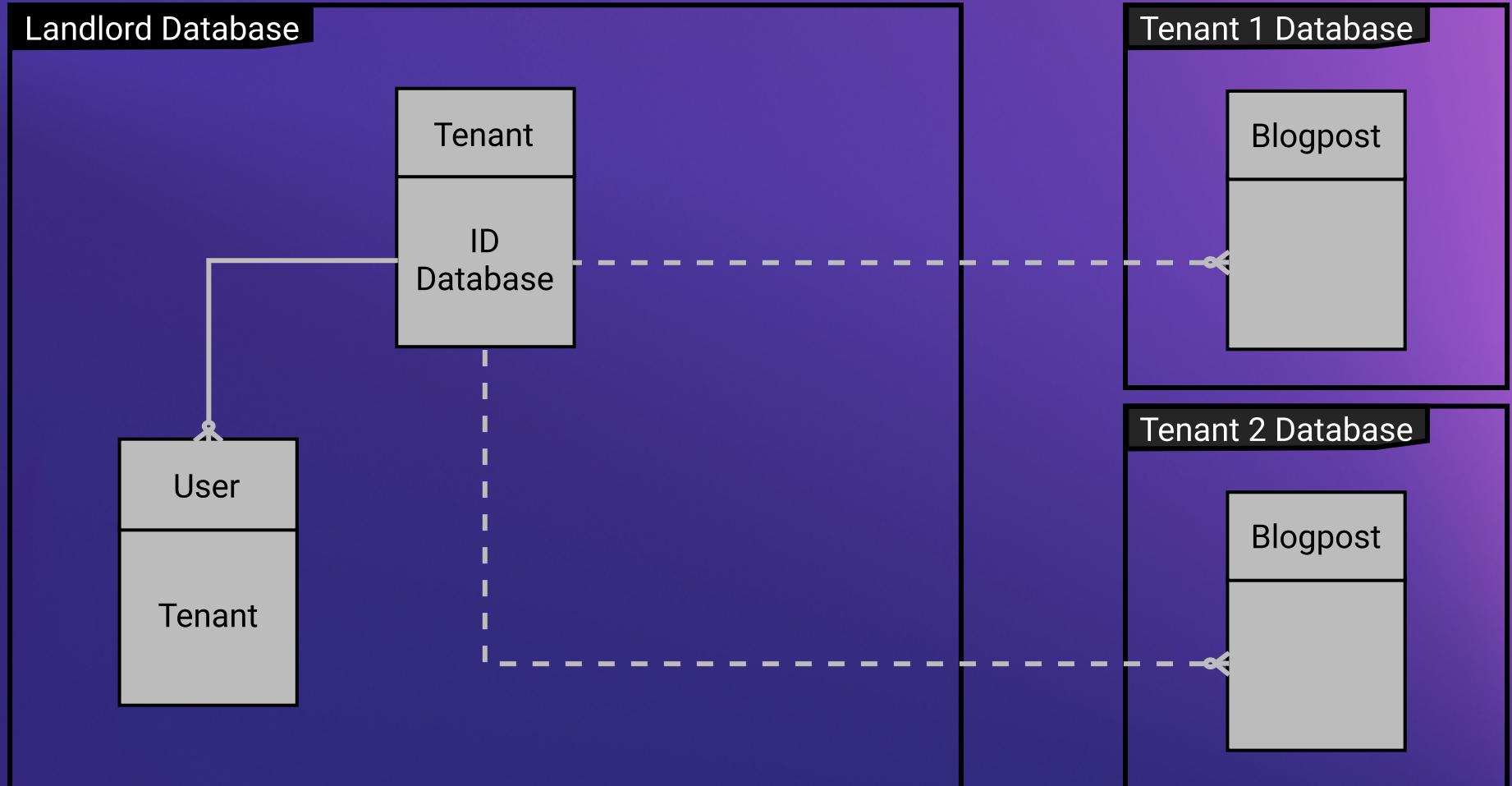
- Bekanntes Grundsetup
- Migrationen bleiben einfach
- Nur eine DB Instanz notwendig
- Leichte Möglichkeit, alle Daten aller Mandanten zusammenzuführen

Geteilte Datenhaltung - Contra

- Für jeden Query wird ein Scope bzw eine ,WHERE'-Clausel benötigt
 - erhöhte Gefahr, dass Mandanten ihnen fremde Daten sehen
 - Je nachdem schwierige Integration bei Fremdpaketen mit eigenen Tabellen
- Schwierigere Backups

Mögliche Userstories

- ✓ Als Product Owner will ich, dass ein Mandant ausschließlich auf seine eigenen Daten zugreifen kann
- ✓ Als Manager will ich einen Überblick über alle Dateien, um Statistiken und Suchen über alle Daten aller Mandanten hinweg nutzen zu können
- ✓ Als Support will ich Zugriff auf alle Daten aller Mandanten, um Support leisten zu können
- ✗ Als Jurist will ich, dass Mandantendaten nicht nur logisch, sondern physikalisch getrennt sind
- ! Als User mehrerer Mandanten will ich ohne erneuten Login zwischen den Mandanten wechseln können
- ! Als Anbieter will ich, dass Mandanten nicht wissen, dass sie auf einem Multimandantensystem betrieben werden
- ✗ Als Sys-Admin will ich pro Mandant individuelle Backups erstellen



```
class TenantDatabaseConnectionWrapper extends Connection
{
    public function __construct(
        #[SensitiveParameter] private array $params,
        Driver $driver,
        ?Configuration $config = null,
        ?EventManager $eventManager = null
    ) {
        parent::__construct($params, $driver, $config, $eventManager);
    }

    public function selectDatabase(Tenant $tenant): void
    {
        if ($this->isConnected()) {
            $this->close();
        }

        $this->params['dbname'] = $tenant->getDatabase();
        parent::__construct(
            $this->params, $this->_driver,
            $this->_config, $this->_eventManager);
    }
}
```

```
private function setCurrentTenant(Tenant $tenant): void
{
    $this->currentTenant = $tenant;

    $connection = $this->entityManager->getConnection();
    if (!$connection instanceof TenantDatabaseConnectionWrapper) {
        throw new NoTenantDatabaseConnectionException();
    }

    $connection->selectDatabase($tenant);
}
```

```
doctrine:  
    dbal:  
        default_connection: default  
        connections:  
            landlord:  
                url: '%env(resolve:DATABASE_LANDLORD_URL)%'  
            default:  
                url: '%env(resolve:DATABASE_TENANT_URL)%'  
                wrapper_class: App\Tenant\TenantDatabaseConnectionWrapper  
    orm:  
        default_entity_manager: default  
        entity_managers:  
            landlord:  
                connection: landlord  
                mappings:  
                    App:  
                        is_bundle: false  
                        dir: '%kernel.project_dir%/src/Entity/Landlord'  
                        prefix: App\Entity\Landlord  
                        alias: App  
            default:  
                connection: default  
                mappings:  
                    App:  
                        is_bundle: false  
                        dir: '%kernel.project_dir%/src/Entity/Tenant'  
                        prefix: App\Entity\Tenant  
                        alias: App
```

```
security:  
    providers:  
        app_user_provider:  
            entity:  
                class: App\Entity\Landlord\User  
                property: email  
                manager_name: landlord
```

```
// migrations/landlord.php  
return [  
    'migrations_paths' => [  
        'DoctrineMigrations\Landlord' => './migrations/Landlord',  
    ],  
  
    'connection' => 'landlord',  
    'em' => 'landlord',
```

```
class TenantFixtures extends Fixture implements FixtureGroupInterface  
{  
    public static function getGroups(): array  
    {  
        return ['landlord'];  
    }}
```

Getrennte Datenhaltung - Pro

- Einfache Queries
- Klare Schnittstelle, an der Daten segmentiert werden
- Vergleichsweise schwer, auf mandantenfremde Daten zuzugreifen
- Drittakete können Out-of-the-Box verwendet werden
- Separate Backups pro Mandant möglich

Getrennte Datenhaltung - Contra

- Migrations und potentielle andere Commands müssen pro Mandant ausgeführt werden
- Keine einfache Möglichkeit, alle Daten zusammengeführt anzuzeigen oder auszuwerten
- Höherer Verwaltungsaufwand/Support/Pflege, da potentiell mehr als ein System gepflegt werden muss
- Höhere Kosten durch höhere Ressourcen

Mögliche Userstories

- ✓ Als Product Owner will ich, dass ein Mandant ausschließlich auf seine eigenen Daten zugreifen kann
- ✗ Als Manager will ich einen Überblick über alle Dateien, um Statistiken und Suchen über alle Daten aller Mandanten hinweg nutzen zu können
- ✓ Als Support will ich Zugriff auf alle Daten aller Mandanten, um Support leisten zu können
- ✓ Als Jurist will ich, dass Mandantendaten nicht nur logisch, sondern physikalisch getrennt sind
- ! Als User mehrerer Mandanten will ich ohne erneuten Login zwischen den Mandanten wechseln können
- ! Als Anbieter will ich, dass Mandanten nicht wissen, dass sie auf einem Multimandantensystem betrieben werden
- ✓ Als Sys-Admin will ich pro Mandant individuelle Backups erstellen

Fazit

- Multi-Tenancy-Systeme können dabei helfen, Ressourcen zu sparen
- Welche Architektur gewählt wird, hängt vom Einsatzzweck und den Use-Cases ab
- Symfony bietet dabei einfache Stellschrauben, um eine Anwendung als Multi-Tenancy-Anwendung zu bauen

Alle Klarheiten beseitigt?

Danke fürs zuhören :-)



github.com/SiMoeBoe/multi-tenancy-yon-talk