

Architektur und Modellierung

Übersicht

Architektur

- Architektur?

- OpenSlides 3

- Zerlegung in Belange

- Services?

- Details

Services

- Übersicht

- Modelle

- Eventstore

- Aktionen

- Autoupdates

- Weiteres

Modellierungen

Clean Architecture

- ▶ SOLID auf Architekturebene: Komponenten anstatt Klassen
- ▶ „Ausführungsart“ irrelevant: SoA, Pluginarchitektur, ...
- ▶ **Seperation of Concerns**
- ▶ Grenzen zwischen Komponenten
- ▶ Einsatzzweck vermitteln

Clean Architecture

- ▶ SOLID auf Architekturebene: Komponenten anstatt Klassen
- ▶ „Ausführungsart“ irrelevant: SoA, Pluginarchitektur, ...
- ▶ **Seperation of Concerns**
- ▶ Grenzen zwischen Komponenten
- ▶ Einsatzzweck vermitteln
- ▶ Zwecke
 - ▶ Entscheidungen soweit verzögern wie Nötig
 - ▶ Unterstützen in Entwicklung und Wartung
 - ▶ Reduzierung der Folgekosten, Optionen offen halten
 - ▶ → OCP

SOLID auf Architekturebene

- ▶ **Common Closure Principle:** Gruppiere Klassen in Komponenten, die sich aus den selben Gründen zur selben Zeit ändern. Separiere Klassen, die dies nicht tun. (→ SRP)

SOLID auf Architekturebene

- ▶ **Common Closure Principle:** Gruppiere Klassen in Komponenten, die sich aus den selben Gründen zur selben Zeit ändern. Separiere Klassen, die dies nicht tun. (→ SRP)
- ▶ **Common Reuse Principle:** Zwinge Nutzer einer Komponente nicht von Dingen abzuhängen, die nicht benötigt werden. (→ ISP)

SOLID auf Architekturebene

- ▶ **Common Closure Principle:** Gruppiere Klassen in Komponenten, die sich aus den selben Gründen zur selben Zeit ändern. Separiere Klassen, die dies nicht tun. (→ SRP)
- ▶ **Common Reuse Principle:** Zwingt Nutzer einer Komponente nicht von Dingen abzuhängen, die nicht benötigt werden. (→ ISP)
- ▶ **Reuse/Release Equivalence Principle:** Der Grad von Wiederverwendung entspricht dem Grad des Releases. Anders: Klassen einer Komponente sollen kohärent sein.

SOLID auf Architekturebene

- ▶ **Common Closure Principle:** Gruppiere Klassen in Komponenten, die sich aus den selben Gründen zur selben Zeit ändern. Separiere Klassen, die dies nicht tun. (\rightarrow SRP)
- ▶ **Common Reuse Principle:** Zwingt Nutzer einer Komponente nicht von Dingen abzuhängen, die nicht benötigt werden. (\rightarrow ISP)
- ▶ **Reuse/Release Equivalence Principle:** Der Grad von Wiederverwendung entspricht dem Grad des Releases. Anders: Klassen einer Komponente sollen kohärent sein.
- ▶ **Acyclic Dependencies Principle:** Verbiete zyklische Abhängigkeiten zwischen Komponenten.

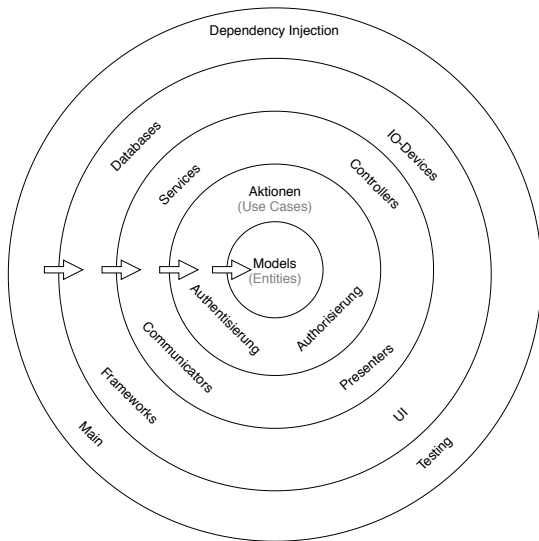
SOLID auf Architekturebene

- ▶ **Common Closure Principle:** Gruppiere Klassen in Komponenten, die sich aus den selben Gründen zur selben Zeit ändern. Separiere Klassen, die dies nicht tun. (\rightarrow SRP)
- ▶ **Common Reuse Principle:** Zwingt Nutzer einer Komponente nicht von Dingen abzuhängen, die nicht benötigt werden. (\rightarrow ISP)
- ▶ **Reuse/Release Equivalence Principle:** Der Grad von Wiederverwendung entspricht dem Grad des Releases. Anders: Klassen einer Komponente sollen kohärent sein.
- ▶ **Acyclic Dependencies Principle:** Verbiete zyklische Abhängigkeiten zwischen Komponenten.
- ▶ **Stable Dependency Principle** und **Stable Abstraction Principle.**

SOLID auf Architekturebene

- ▶ **Common Closure Principle:** Gruppiere Klassen in Komponenten, die sich aus den selben Gründen zur selben Zeit ändern. Separiere Klassen, die dies nicht tun. (→ SRP)
- ▶ **Common Reuse Principle:** Zwingt Nutzer einer Komponente nicht von Dingen abzuhängen, die nicht benötigt werden. (→ ISP)
- ▶ **Reuse/Release Equivalence Principle:** Der Grad von Wiederverwendung entspricht dem Grad des Releases. Anders: Klassen einer Komponente sollen kohärent sein.
- ▶ **Acyclic Dependencies Principle:** Verbiete zyklische Abhängigkeiten zwischen Komponenten.
- ▶ **Stable Dependency Principle** und **Stable Abstraction Principle.**
- ▶ **The Dependency Rule:** Quelltextabhängigkeiten zeigen immer in Richtung höherer Regeln (policies)

Clean Architecture: The Dependency Rule



The Dependency Rule: Quelltextabhängigkeiten zeigen immer in Richtung höherer Regeln (policies)

Details

- ▶ Die Datenbank ist ein Detail.
- ▶ Das Web ist ein Detail (IO-Device)
- ▶ Frameworks sind Details

Wieso Neuschreiben?

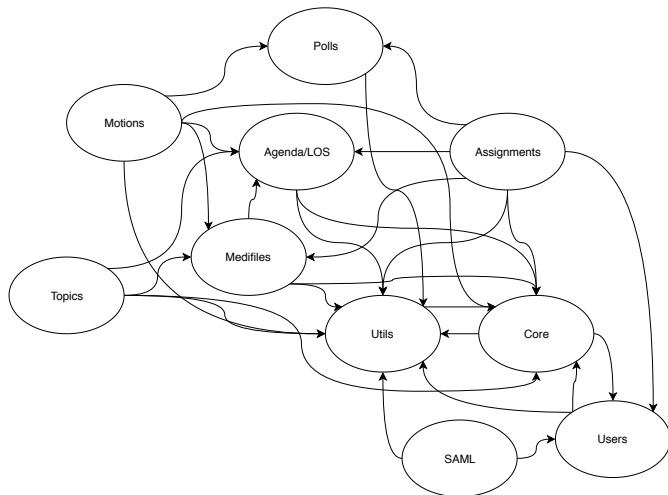
- ▶ Zu starke Kopplung in Django/DRF. Verstöße gegen die Abhängigkeitsregel, ...
- ▶ ORM bietet nicht die Möglichkeit zur Versionierung
- ▶ REST-Schnittstelle ist nicht für Aktionen geeignet
- ▶ Autoupdate zu groß
- ▶ Cache-Ansatz nicht skalierbar für neue Anforderungen

Wieso Neuschreiben?

- ▶ Zu starke Kopplung in Django/DRF. Verstöße gegen die Abhängigkeitsregel, ...
- ▶ ORM bietet nicht die Möglichkeit zur Versionierung
- ▶ REST-Schnittstelle ist nicht für Aktionen geeignet
- ▶ Autoupdate zu groß
- ▶ Cache-Ansatz nicht skalierbar für neue Anforderungen
- ▶ Was bleibt übrig?

Keine Microservices

- ▶ Architektur hält diese Entscheidung offen
- ▶ Trennen nach Apps führt zu zyklischen Abhängigkeiten
- ▶ Viele geteilte Belange; Transaktionen?



Separation of Concerns

Was sind unsere Belange?

1. Businessregeln
2. Authorisierung
3. Authentisierung
4. Persistenz (inkl. Versionierung)
5. Echtzeit-Updates

Separation of Concerns

Was sind unsere Belange?

1. Businessregeln
2. Authorisierung
3. Authentisierung
4. Persistenz (inkl. Versionierung)
5. Echtzeit-Updates

Geteilte Belange

1. Cache
2. Logging
3. Konfiguration der Software

Separation of Concerns

Was sind unsere Belange?

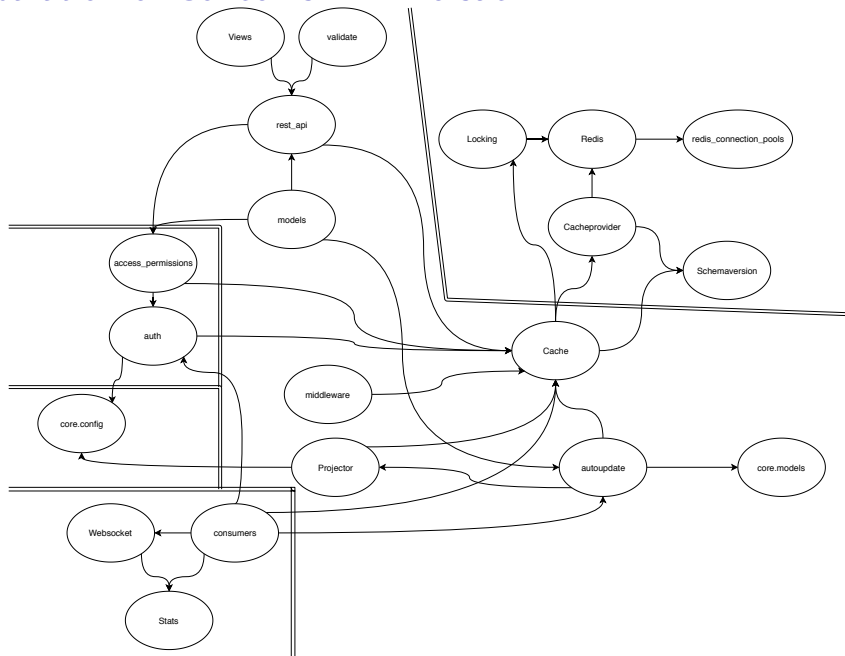
1. Businessregeln
2. Authorisierung
3. Authentisierung
4. Persistenz (inkl. Versionierung)
5. Echtzeit-Updates

Geteilte Belange

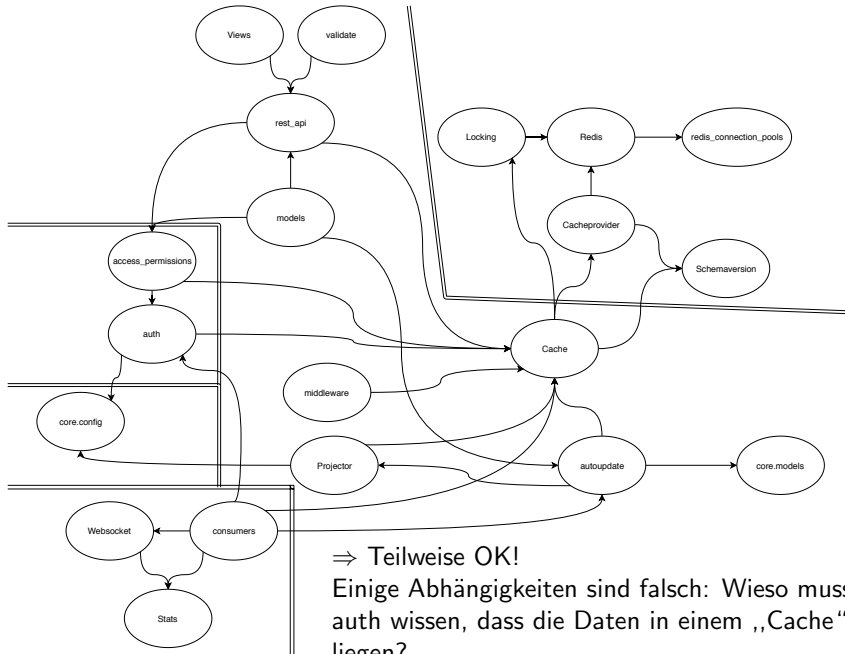
1. Cache
2. Logging
3. Konfiguration der Software

⇒ Teilen der Belange in Komponenten

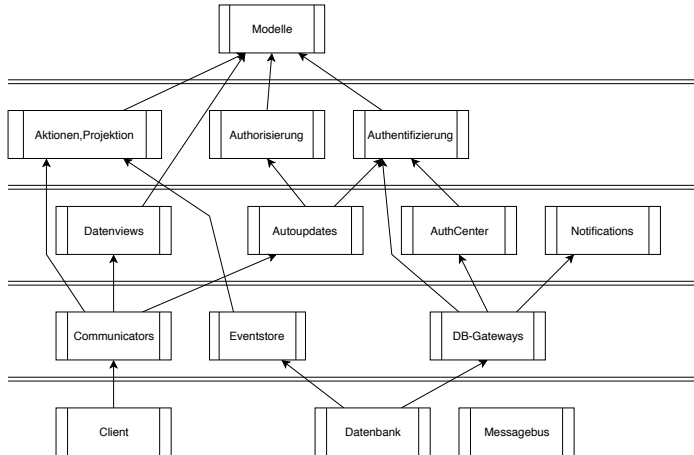
Separation of Concerns: Ein Versuch



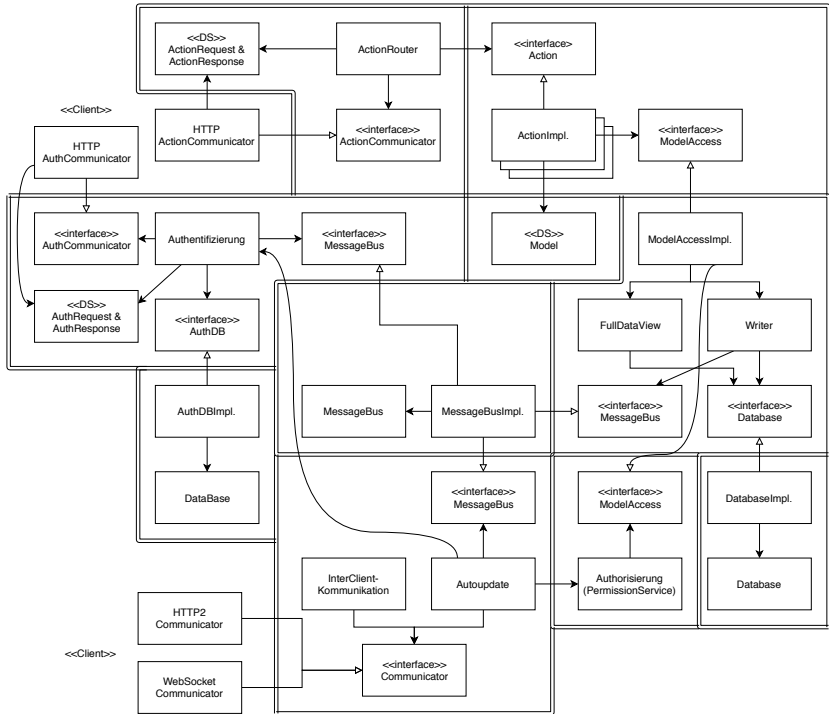
Separation of Concerns: Ein Versuch



Zerlegung in Komponenten



- ▶ Nicht vollständig (Pfeile sind Quelltextabhängigkeiten)
- ▶ Hierarchie: Oben: Businessregeln, Mitte: Verarbeitung, Unten: IO



Event sourcing

Was ist Event sourcing?

- ▶ Append-only Log an Änderungen (Events)
- ▶ Repräsentation des Systemzustandes
- ▶ Wiederherstellung durch Abspielen der Events
- ▶ Snapshots für beschleunigte Wiederherstellung
- ▶ siehe GIT

Event sourcing

Was ist Event sourcing?

- ▶ Append-only Log an Änderungen (Events)
- ▶ Repräsentation des Systemzustandes
- ▶ Wiederherstellung durch Abspielen der Events
- ▶ Snapshots für beschleunigte Wiederherstellung
- ▶ siehe GIT

Eigenschaften:

- ▶ Kann als „Persistenz mit Versionierung“ abstrahiert werden
- ▶ Bietet folgenden Vorteil: CRUD → CR
- ▶ Ermöglicht funktionale Programmierung für Aktionen: EVA
- ▶ Erinnerung: SSOT
- ▶ Eventstore ist Implementation der „Persistenz mit Versionierung“ mit Event sourcing

Wieso Services?

- ▶ Entkopplung auf *Quelltext-Level*, *Komponenten-Level*, *Service-Level*
- ▶ OS3: Misch aus Quelltext- und Komponenten-Level
- ▶ Erfahrung: Unterschiedlichste Anforderungen an verschiedene Teile des Systems

Wieso Services?

- ▶ Entkopplung auf *Quelltext-Level*, *Komponenten-Level*, *Service-Level*
- ▶ OS3: Misch aus Quelltext- und Komponenten-Level
- ▶ Erfahrung: Unterschiedlichste Anforderungen an verschiedene Teile des Systems
- ▶ SoA: Teilung nach Belangen (deren Anforderungen), nicht nach Komponenten
- ▶ Services selbst sollen auf Komponenten-Level sein

Wieso Services?

- ▶ Entkopplung auf *Quelltext-Level*, *Komponenten-Level*, *Service-Level*
- ▶ OS3: Misch aus Quelltext- und Komponenten-Level
- ▶ Erfahrung: Unterschiedlichste Anforderungen an verschiedene Teile des Systems
- ▶ SoA: Teilung nach Belangen (deren Anforderungen), nicht nach Komponenten
- ▶ Services selbst sollen auf Komponenten-Level sein
- ▶ Vorteile?
 - ▶ Wiederverwendbarkeit?
 - ▶ Unabhängigkeit?

Wieso Services?

- ▶ Entkopplung auf *Quelltext-Level*, *Komponenten-Level*, *Service-Level*
- ▶ OS3: Misch aus Quelltext- und Komponenten-Level
- ▶ Erfahrung: Unterschiedlichste Anforderungen an verschiedene Teile des Systems
- ▶ SoA: Teilung nach Belangen (deren Anforderungen), nicht nach Komponenten
- ▶ Services selbst sollen auf Komponenten-Level sein
- ▶ Vorteile?
 - ▶ Wiederverwendbarkeit?
 - ▶ Unabhängigkeit?
 - ▶ Testbarkeit!
 - ▶ Anpassungsfähigkeit! (Ressourcenmanagement)
 - ▶ Stärkste Trennung der Belange!

Details

... sind für die Architektur irrelevant! Jedoch für die Erstimplementation:

Details

... sind für die Architektur irrelevant! Jedoch für die Erstimplementation:

- ▶ Bedachte Wahl von Technologien: Jede eingesetzte Technologie muss **verstanden** und **beherrscht** werden können

Details

...sind für die Architektur irrelevant! Jedoch für die Erstimplementation:

- ▶ Bedachte Wahl von Technologien: Jede eingesetzte Technologie muss **verstanden** und **beherrscht** werden können
- ▶ Je größer die Vielzahl an Protokollen, Datenbanken, Caches,..., je komplexer wird die Software

Details

...sind für die Architektur irrelevant! Jedoch für die Erstimplementation:

- ▶ Bedachte Wahl von Technologien: Jede eingesetzte Technologie muss **verstanden** und **beherrscht** werden können
- ▶ Je größer die Vielzahl an Protokollen, Datenbanken, Caches,..., je komplexer wird die Software
- ▶ Kapseln von Technologien: Unser Code ist unabhängig von diesen!

Details

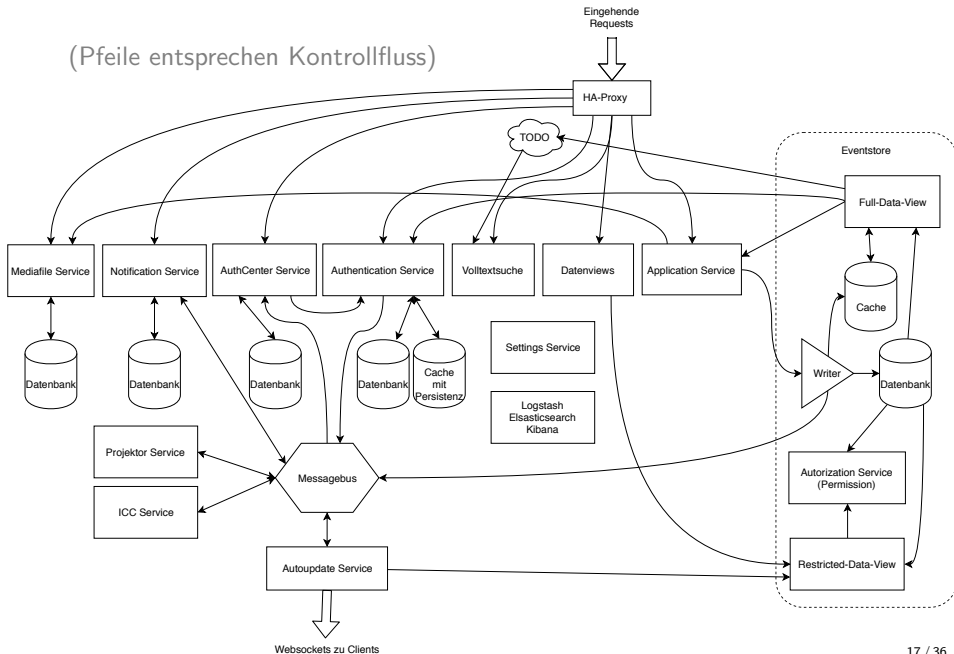
...sind für die Architektur irrelevant! Jedoch für die Erstimplementation:

- ▶ Bedachte Wahl von Technologien: Jede eingesetzte Technologie muss **verstanden** und **beherrscht** werden können
- ▶ Je größer die Vielzahl an Protokollen, Datenbanken, Caches,..., je komplexer wird die Software
- ▶ Kapseln von Technologien: Unser Code ist unabhängig von diesen!
- ▶ Vorgabe: Postgresql für jegliche Persistenz
Achtung: Das bedeutet nicht, dass keine Abstraktion stattfindet (→ Testen)

Services

Übersicht

(Pfeile entsprechen Kontrollfluss)



Modelle

- ▶ Ansammlung an Key-Value-Paaren

Modelle

- ▶ Ansammlung an Key-Value-Paaren
- ▶ Orientiert an OS3

Modelle

- ▶ Ansammlung an Key-Value-Paaren
- ▶ Orientiert an OS3
- ▶ Dokumente, Verschachtelung möglich, aber nicht erwünscht
 - ▶ Autoupdates und Events sind auf Key-Basis
 - ▶ Keine Notwendigkeit der Denormalisierung durch Verschachtelung

Modelle

- ▶ Ansammlung an Key-Value-Paaren
- ▶ Orientiert an OS3
- ▶ Dokumente, Verschachtelung möglich, aber nicht erwünscht
 - ▶ Autoupdates und Events sind auf Key-Basis
 - ▶ Keine Notwendigkeit der Denormalisierung durch Verschachtelung
- ▶ Referenzen werden in beiden Modellen gespeichert: Vereinfachtes Lesen

Modelle

- ▶ Ansammlung an Key-Value-Paaren
- ▶ Orientiert an OS3
- ▶ Dokumente, Verschachtelung möglich, aber nicht erwünscht
 - ▶ Autoupdates und Events sind auf Key-Basis
 - ▶ Keine Notwendigkeit der Denormalisierung durch Verschachtelung
- ▶ Referenzen werden in beiden Modellen gespeichert: Vereinfachtes Lesen
- ▶ IDs: Nur Zahlen?

Modelle

- ▶ Ansammlung an Key-Value-Paaren
- ▶ Orientiert an OS3
- ▶ Dokumente, Verschachtelung möglich, aber nicht erwünscht
 - ▶ Autoupdates und Events sind auf Key-Basis
 - ▶ Keine Notwendigkeit der Denormalisierung durch Verschachtelung
- ▶ Referenzen werden in beiden Modellen gespeichert: Vereinfachtes Lesen
- ▶ IDs: Nur Zahlen?
- ▶ Keys bestehen aus [a-z_]+

Modelle

- ▶ Ansammlung an Key-Value-Paaren
- ▶ Orientiert an OS3
- ▶ Dokumente, Verschachtelung möglich, aber nicht erwünscht
 - ▶ Autoupdates und Events sind auf Key-Basis
 - ▶ Keine Notwendigkeit der Denormalisierung durch Verschachtelung
- ▶ Referenzen werden in beiden Modellen gespeichert: Vereinfachtes Lesen
- ▶ IDs: Nur Zahlen?
- ▶ Keys bestehen aus [a-z_]+
- ▶ Struktur innerhalb der Keys durch : (`amendment_paragraphs:3, meta:position`)

Modelle

- ▶ Ansammlung an Key-Value-Paaren
- ▶ Orientiert an OS3
- ▶ Dokumente, Verschachtelung möglich, aber nicht erwünscht
 - ▶ Autoupdates und Events sind auf Key-Basis
 - ▶ Keine Notwendigkeit der Denormalisierung durch Verschachtelung
- ▶ Referenzen werden in beiden Modellen gespeichert: Vereinfachtes Lesen
- ▶ IDs: Nur Zahlen?
- ▶ Keys bestehen aus [a-z_]+
- ▶ Struktur innerhalb der Keys durch : (`amendment_paragraphs:3`, `meta:position`)
- ▶ Reservierter meta-Key-Prefix: Position des Modells, gelöscht,... (später)

Modelle

- ▶ Ansammlung an Key-Value-Paaren
- ▶ Orientiert an OS3
- ▶ Dokumente, Verschachtelung möglich, aber nicht erwünscht
 - ▶ Autoupdates und Events sind auf Key-Basis
 - ▶ Keine Notwendigkeit der Denormalisierung durch Verschachtelung
- ▶ Referenzen werden in beiden Modellen gespeichert: Vereinfachtes Lesen
- ▶ IDs: Nur Zahlen?
- ▶ Keys bestehen aus [a-z_]+
- ▶ Struktur innerhalb der Keys durch : (amendment_paragraphs:3, meta:position)
- ▶ Reservierter meta-Key-Prefix: Position des Modells, gelöscht,... (später)
- ▶ Modelle sind Collections zugeordnet: motions, projector, motion-state,...

Modelle

- ▶ Ansammlung an Key-Value-Paaren
- ▶ Orientiert an OS3
- ▶ Dokumente, Verschachtelung möglich, aber nicht erwünscht
 - ▶ Autoupdates und Events sind auf Key-Basis
 - ▶ Keine Notwendigkeit der Denormalisierung durch Verschachtelung
- ▶ Referenzen werden in beiden Modellen gespeichert: Vereinfachtes Lesen
- ▶ IDs: Nur Zahlen?
- ▶ Keys bestehen aus [a-z_]+
- ▶ Struktur innerhalb der Keys durch : (amendment_paragraphs:3, meta:position)
- ▶ Reservierter meta-Key-Prefix: Position des Modells, gelöscht,... (später)
- ▶ Modelle sind Collections zugeordnet: motions, projector, motion-state,...
- ▶ *fqid* (full-qualified-id): <collection>/<id>, z.B.:
projector-message/42

Modelle

- ▶ Ansammlung an Key-Value-Paaren
- ▶ Orientiert an OS3
- ▶ Dokumente, Verschachtelung möglich, aber nicht erwünscht
 - ▶ Autoupdates und Events sind auf Key-Basis
 - ▶ Keine Notwendigkeit der Denormalisierung durch Verschachtelung
- ▶ Referenzen werden in beiden Modellen gespeichert: Vereinfachtes Lesen
- ▶ IDs: Nur Zahlen?
- ▶ Keys bestehen aus [a-z_]+
- ▶ Struktur innerhalb der Keys durch : (amendment_paragraphs:3, meta:position)
- ▶ Reservierter meta-Key-Prefix: Position des Modells, gelöscht,... (später)
- ▶ Modelle sind Collections zugeordnet: motions, projector, motion-state,...
- ▶ *fqid* (full-qualified-id): <collection>/<id>, z.B.:
projector-message/42
- ▶ *fkey* (full-qualified-key): <fqid>/<key>, z.B.:
projector-message/42/message

Modelle

- ▶ Ansammlung an Key-Value-Paaren
- ▶ Orientiert an OS3
- ▶ Dokumente, Verschachtelung möglich, aber nicht erwünscht
 - ▶ Autoupdates und Events sind auf Key-Basis
 - ▶ Keine Notwendigkeit der Denormalisierung durch Verschachtelung
- ▶ Referenzen werden in beiden Modellen gespeichert: Vereinfachtes Lesen
- ▶ IDs: Nur Zahlen?
- ▶ Keys bestehen aus [a-z_]+
- ▶ Struktur innerhalb der Keys durch : (amendment_paragraphs:3, meta:position)
- ▶ Reservierter meta-Key-Prefix: Position des Modells, gelöscht,... (später)
- ▶ Modelle sind Collections zugeordnet: motions, projector, motion-state,...
- ▶ *fqid* (full-qualified-id): <collection>/<id>, z.B.: projector-message/42
- ▶ *fqkey* (full-qualified-key): <fqid>/<key>, z.B.: projector-message/42/message
- ▶ String-values: HTML-Strings bekommen eine Maximallänge (konfigurierbar, TODO: Wo?)

Eventstore

AKA. Transaction log, Incremental Backup,...

Datenfelder von Events

1. Event-Id: Intern.
2. Position: Totale Ordnung, bündelt zusammengehörige Events.
3. fqid: Jedes Event ist einem Modell zugeordnet
4. Eventtyp
 - ▶ Create
 - ▶ UpdateKeys
 - ▶ DeleteKeys
 - ▶ Delete
 - ▶ Restore
 - ▶ Noop (Intern)
5. Eventdaten als partielles Modell
6. Schemaversion
7. Zuordnung von Zeitstempel, Beschreibung,... zu Position
8. Benötigt: Zuordnung Veranstaltung \leftrightarrow Modell?

Eventstore

Methoden

- ▶ Schreiben von Events
 - ▶
- ▶ Single-Writer: Keine Synchronisation nötig, ist atomar
- ▶ Konsistenz der Events wird abgesichert.
- ▶ OCC: „Lazy“ Isolation
- ▶ → ACID

Beispiel: Eventstore write

```
write(request: WriteRequest): WriteResponse;
```

```
Interface WriteRequest{  
  data: {  
    <fqid>: {position: Position; type: Create|Restore; model: Model;}  
    | { position: Position; type: Delete;};  
    <fqkey>: {position: Position; type: Update; value: any;}  
    | { position: Position; type: DeleteKey;};  
  };  
  general_keys: {  
    <CollectionKey>: Position; // TODO: je nach dem wie IDs aussehen (nur  
    // Zahlen?) ist CollectionKey nicht eindeutig von fqid zu unterscheiden  
    <fqid>: Position;  
    <fqkey>: Position;  
  };  
}
```

```
Interface WriteResponse{  
  changed_models: {<fqid>: Position};  
  current_position: Position;  
}
```

Beispiel: Eventstore write

Wirft Exceptions:

- ▶ `ModelDoesNotExist(fqid)`
- ▶ `ModelTooOld(fqid)`
- ▶ `KeyTooOld(fqkey)`
- ▶ `RequestTooOld()`
- ▶ `ModelExists(fqid)`; Kann bei Create auftreten, wenn die ID bekannt ist.

Beispiel: Eventstore write

Wirft Exceptions:

- ▶ `ModelDoesNotExist(fqid)`
- ▶ `ModelTooOld(fqid)`
- ▶ `KeyTooOld(fqkey)`
- ▶ `RequestTooOld()`
- ▶ `ModelExists(fqid)`; Kann bei Create auftreten, wenn die ID bekannt ist.

Weiteres:

- ▶ Besonderheit: ID muss bei Create bekannt sein. Der Aufrufer muss sich darum kümmern.
- ▶ Vorteil: Relations können bei Create angelegt werden

Beispiel: Eventstore write

Wirft Exceptions:

- ▶ `ModelDoesNotExist(fqid)`
- ▶ `ModelTooOld(fqid)`
- ▶ `KeyTooOld(fqkey)`
- ▶ `RequestTooOld()`
- ▶ `ModelExists(fqid)`; Kann bei Create auftreten, wenn die ID bekannt ist.

Weiteres:

- ▶ Besonderheit: ID muss bei Create bekannt sein. Der Aufrufer muss sich darum kümmern.
- ▶ Vorteil: Relations können bei Create angelegt werden
- ▶ Single Writer: Keine Synchronisation nötig

Beispiel: Eventstore write

Wirft Exceptions:

- ▶ `ModelDoesNotExist(fqid)`
- ▶ `ModelTooOld(fqid)`
- ▶ `KeyTooOld(fqkey)`
- ▶ `RequestTooOld()`
- ▶ `ModelExists(fqid)`; Kann bei Create auftreten, wenn die ID bekannt ist.

Weiteres:

- ▶ Besonderheit: ID muss bei Create bekannt sein. Der Aufrufer muss sich darum kümmern.
- ▶ Vorteil: Relations können bei Create angelegt werden
- ▶ Single Writer: Keine Synchronisation nötig
- ▶ OCC: Eine Datenstruktur hält die letzten X geänderten `fqids/fqkeys`

Beispiel: Eventstore write

Wirft Exceptions:

- ▶ `ModelDoesNotExist(fqid)`
- ▶ `ModelTooOld(fqid)`
- ▶ `KeyTooOld(fqkey)`
- ▶ `RequestTooOld()`
- ▶ `ModelExists(fqid)`; Kann bei Create auftreten, wenn die ID bekannt ist.

Weiteres:

- ▶ Besonderheit: ID muss bei Create bekannt sein. Der Aufrufer muss sich darum kümmern.
- ▶ Vorteil: Relations können bei Create angelegt werden
- ▶ Single Writer: Keine Synchronisation nötig
- ▶ OCC: Eine Datenstruktur hält die letzten X geänderten `fqids/fqkeys`
- ▶ Beispiel: Create mit unique Key

- ▶ Elemente lassen sich wiederherstellen
- ▶ DSGVO: Wir (als Betreiber) können in die Datenbank des Eventstores eingreifen und z.B. Nutzer entfernen (d.h. nicht das Nutzer-Model löschen, sondern Daten unkenntlich machen)
- ▶ Wiederherstellung benötigt Logik für alle Referenzen
 - ▶ Existieren diese noch?
 - ▶ Falls Ja, sind diese valide? → Beide Modelle anpassen (doppelte Referenzen)
 - ▶ Falls Nein, ist das valide?
 - ▶ Falls nicht valide: Nutzer bekommt "Create"-Form zum Bearbeiten

Aktionen

- ▶ Atomar
- ▶ Hat Bezeichner. Z.B.: `motions/reset-state`, `user/add_group`,...
- ▶ Input: `data: Object` und `user_id: ID`
- ▶ Output: `events: Event[]`
- ▶ Aktionen teilen sich in `validate` und `execute`
- ▶ Client kann mehrere Aktionen schicken. Ausführungsmodi:
 - ▶ Sukzessive bearbeiten; Nach Validierung sofort ausführen; Beim ersten Fehler abbrechen
(→ Operationen, die aufeinander aufbauen, aber in Summe nicht atomar sind)
 - ▶ Sukzessive bearbeiten; Nach Validierung sofort ausführen; Alle Fehler melden
(→ unabhängige Operationen, z.B. Bulk-Operationen)
 - ▶ Erst alle Validieren, dann alle Ausführen
(→ Transaktion; Single-Writer versichert atomare Ausführung)
- ▶ Autoupdate in Response

Autoupdates

- ▶ *Fokussierte Modelle* (Detail/Liste)
- ▶ Client abonniert fokussierte Modelle inkl. benötigter Keys (ID implizit) und Relationen

```
interface ModelDescriptor {  
    <key>: ModelDescriptor | null;  
    // null -> Nur der Wert des Keys  
}
```

```
interface ModelRequest {  
    ids: number | number[] | null;  
    // Ein spezifisches Modell, mehrere spezifische Modelle  
    // oder alle Modelle (siehe assembly_id)  
    assembly_id: number | null; // Benötigt für ids=null  
    collection: string;  
    keys: ModelDescriptor;  
}
```

Autoupdates: Beispiel

```
modelRequest = {  
  ids: 4,  
  collection: 'motion-workflow',  
  keys: {  
    name: null,  
    states_id: null,  
    first_state_id: {  
      collection: 'motion-state',  
      keys: {  
        name: null,  
        css_class: null,  
        next_states_id: {  
          name: null  
        }  
      }  
    }  
  }  
}
```

```
ExampleDataFromServer = {  
  'motion-workflow': {  
    4: {  
      name: 'Example Workflow',  
      states_id: [1,2,3,4,5,6],  
      first_state_id: 1,  
    }  
  },  
  'motion-state': {  
    1: {  
      id: 1,  
      name: 'My first state',  
      css_class: 'lightblue',  
      next_states_id: [2, 3]  
    },  
    2: {  
      id: 2,  
      name: 'Accept'  
    },  
    3: {  
      id: 2,  
      name: 'Deny'  
    }  
  }  
}
```

Autoupdates

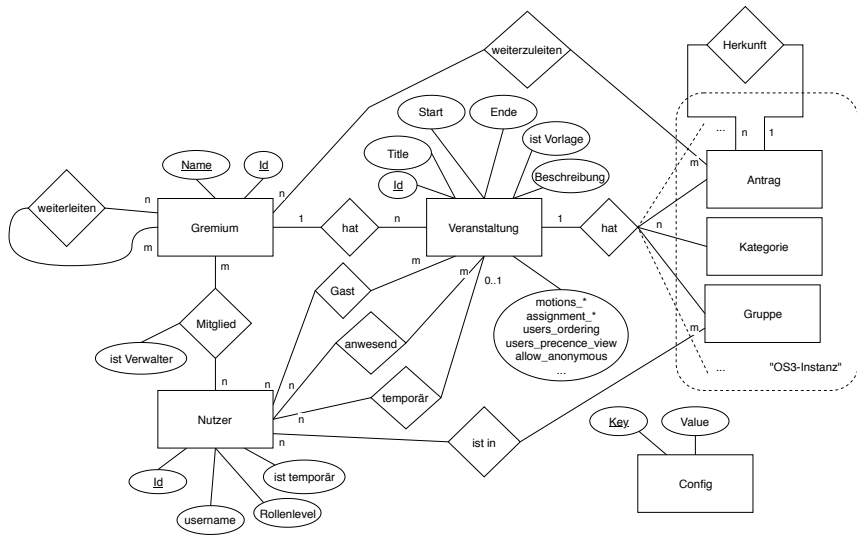
- ▶ Jede Subscription hat ein Handle zum Beenden
- ▶ Kein Caching (vorerst)
- ▶ Client übernimmt Flusssteuerung; Server kann Autoupdates akkumulieren
- ▶ TODO: Vorsätzliche Akkumulierung?
- ▶ TODO: Explizites vs. Implizites Abonnieren im Client

Weiteres

- ▶ Offlinemodus: Explizit anstatt Implizit
- ▶ Presenters
- ▶ Mediafiles: Speichern in einer Datenbank. Extra Service zum Ausliefern
- ▶ Im- und Export von Veranstaltungen
- ▶ Volltextsuche: ?
- ▶ Migrationen: ?
- ▶ Einstellungen
- ▶ Logging

Modellierungen

ER-Diagramm für Organisationsebene



Rollen auf Organisationsebene

► Rollenlevel:

0. **Keine Berechtigungen** (Standard): Ein Nutzer darf die Gremien sehen, in der er Mitglied ist, oder in einer Veranstaltung Gast
1. **Nutzer-Manager**: Darf Nutzer verwalten (erstellen, temporär zu fest, bearbeiten, Löschen, Passwort managen, sperren)
2. **Organisations-Manager**: Nutzer-Manager + Darf alle Gremien sehen, erstellen, löschen, Nutzer zuweisen, Nutzer als Verwalter im Gremium ernennen und die Weiterleitungsstruktur bearbeiten
3. **Superadmin**: Wie Organisations-Manager, hat jedoch in allen Gremien und Veranstaltungen alle Rechte (transitiv)

Rollen auf Organisationsebene

- ▶ Rollenlevel:

0. **Keine Berechtigungen** (Standard): Ein Nutzer darf die Gremien sehen, in der er Mitglied ist, oder in einer Veranstaltung Gast
1. **Nutzer-Manager**: Darf Nutzer verwalten (erstellen, temporär zu fest, bearbeiten, Löschen, Passwort managen, sperren)
2. **Organisations-Manager**: Nutzer-Manager + Darf alle Gremien sehen, erstellen, löschen, Nutzer zuweisen, Nutzer als Verwalter im Gremium ernennen und die Weiterleitungsstruktur bearbeiten
3. **Superadmin**: Wie Organisations-Manager, hat jedoch in allen Gremien und Veranstaltungen alle Rechte (transitiv)

- ▶ Jeder darf eigene Rolle ändern, aber nur abstufen

Rollen auf Organisationsebene

- ▶ Rollenlevel:
 0. **Keine Berechtigungen** (Standard): Ein Nutzer darf die Gremien sehen, in der er Mitglied ist, oder in einer Veranstaltung Gast
 1. **Nutzer-Manager**: Darf Nutzer verwalten (erstellen, temporär zu fest, bearbeiten, Löschen, Passwort managen, sperren)
 2. **Organisations-Manager**: Nutzer-Manager + Darf alle Gremien sehen, erstellen, löschen, Nutzer zuweisen, Nutzer als Verwalter im Gremium ernennen und die Weiterleitungsstruktur bearbeiten
 3. **Superadmin**: Wie Organisations-Manager, hat jedoch in allen Gremien und Veranstaltungen alle Rechte (transitiv)
- ▶ Jeder darf eigene Rolle ändern, aber nur abstufen
- ▶ Ab Rollenlevel $L \geq 1$ darf ein Nutzer andere Rollenlevel setzen, aber nur auf maximal L

Rollen auf Organisationsebene

- ▶ Rollenlevel:
 0. **Keine Berechtigungen** (Standard): Ein Nutzer darf die Gremien sehen, in der er Mitglied ist, oder in einer Veranstaltung Gast
 1. **Nutzer-Manager**: Darf Nutzer verwalten (erstellen, temporär zu fest, bearbeiten, Löschen, Passwort managen, sperren)
 2. **Organisations-Manager**: Nutzer-Manager + Darf alle Gremien sehen, erstellen, löschen, Nutzer zuweisen, Nutzer als Verwalter im Gremium ernennen und die Weiterleitungsstruktur bearbeiten
 3. **Superadmin**: Wie Organisations-Manager, hat jedoch in allen Gremien und Veranstaltungen alle Rechte (transitiv)
- ▶ Jeder darf eigene Rolle ändern, aber nur abstufen
- ▶ Ab Rollenlevel $L \geq 1$ darf ein Nutzer andere Rollenlevel setzen, aber nur auf maximal L
- ▶ Es existiert immer ein Superadmin

Rollen in Gremien

- ▶ *Mitglieder* sind dem Gremium zugeordnete Nutzer
- ▶ Mitglieder sehen Veranstaltungen, die Anonymous erlauben, der Nutzer Gast ist, oder der Nutzer in mindestens einer Gruppe ist
- ▶ Mitglieder können Verwalter sein
- ▶ Superadmins auf Organisationsebene sind implizit Mitglied und Verwalter
- ▶ Verwalter können Veranstaltungen verwalten: Erstellen, Löschen, Kopieren, Vorlage auswählen, Importieren, Exportieren, Eigenschaften der Veranstaltung ändern

Rollen in Gremien

- ▶ *Mitglieder* sind dem Gremium zugeordnete Nutzer
- ▶ Mitglieder sehen Veranstaltungen, die Anonymous erlauben, der Nutzer Gast ist, oder der Nutzer in mindestens einer Gruppe ist
- ▶ Mitglieder können Verwalter sein
- ▶ Superadmins auf Organisationsebene sind implizit Mitglied und Verwalter
- ▶ Verwalter können Veranstaltungen verwalten: Erstellen, Löschen, Kopieren, Vorlage auswählen, Importieren, Exportieren, Eigenschaften der Veranstaltung ändern
- ▶ Nutzer, die Gäste aber keine Mitglieder sind, sehen Gremien nicht
- ▶ Im Dashboard gibt es eine Übersicht von Veranstaltungen, in denen ein Nutzer Gast ist

Rollen in Gremien

- ▶ *Mitglieder* sind dem Gremium zugeordnete Nutzer
- ▶ Mitglieder sehen Veranstaltungen, die Anonymous erlauben, der Nutzer Gast ist, oder der Nutzer in mindestens einer Gruppe ist
- ▶ Mitglieder können Verwalter sein
- ▶ Superadmins auf Organisationsebene sind implizit Mitglied und Verwalter
- ▶ Verwalter können Veranstaltungen verwalten: Erstellen, Löschen, Kopieren, Vorlage auswählen, Importieren, Exportieren, Eigenschaften der Veranstaltung ändern
- ▶ Nutzer, die Gäste aber keine Mitglieder sind, sehen Gremien nicht
- ▶ Im Dashboard gibt es eine Übersicht von Veranstaltungen, in denen ein Nutzer Gast ist
- ▶ Veranstaltungen können optionalen eindeutigen Identifikator erhalten (Gäste können darüber zugreifen)

Berechtigungen in Veranstaltungen

- ▶ Rechte auf Basis der Gruppenzuweisung
- ▶ Es gibt eine Gästegruppe für Gäste ($\hat{=}$ Default in OS3)
- ▶ Es gibt eine Superadmingruppe
- ▶ Superadmins auf Organisationsebene sind implizit in der Superadmingruppe
- ▶ Gremienverwalter sind implizit in der Superadmingruppe
- ▶ Nutzer mit `users.can_manage` können Nutzer Gruppen zuweisen und Gruppenberechtigungen einstellen
- ▶ Nutzer bekommen neues Feld "Funktion", damit die Gruppen nur für die Rechtezuweisungen benutzt werden

URLs

- ▶ Veranstaltungen:
`org.openslides.com/<assembly_id>/`
- ▶ Veranstaltungen mit eindeutigem Identifikator:
`org.openslides.com/<identifier>/`
- ▶ Serverkonfiguration (HaProxy, optional):
`<identifier>.org.openslides.com/`

URLs

- ▶ Veranstaltungen:
`org.openslides.com/<assembly_id>/`
- ▶ Veranstaltungen mit eindeutigem Identifikator:
`org.openslides.com/<identifier>/`
- ▶ Serverkonfiguration (HaProxy, optional):
`<identifier>.org.openslides.com/`
- ▶ Dashboard (als main landing page) unter `org.openslides.com/`
- ▶ Spezielle URLs: `/dashboard/`, `/users/`, `/committee/`, `/auth/`
- ▶ Spezielle Urls können nicht als Identifikatoren verwendet werden

Anonymous

- ▶ Anonymous \leftrightarrow Gast
- ▶ Kann pro Veranstaltung aktiviert werden
- ▶ Sind implizit in der Gästegruppe
- ▶ Anonymous sehen nicht das Dashboard und Gremien
- ▶ Müssen Veranstaltungs-URL kennen

Temporäre Nutzer

- ▶ Szenario:
 - ▶ Nutzergruppe der Veranstaltung nicht bekannt
 - ▶ Anwesende Nutzer sollen in OpenSlides teilnehmen
 - ▶ Einen systemweiten Nutzer anzulegen ist keine Option

Temporäre Nutzer

- ▶ Szenario:
 - ▶ Nutzergruppe der Veranstaltung nicht bekannt
 - ▶ Anwesende Nutzer sollen in OpenSlides teilnehmen
 - ▶ Einen systemweiten Nutzer anzulegen ist keine Option
- ▶ `users.can_manage_temporary_user`

Temporäre Nutzer

- ▶ Szenario:
 - ▶ Nutzergruppe der Veranstaltung nicht bekannt
 - ▶ Anwesende Nutzer sollen in OpenSlides teilnehmen
 - ▶ Einen systemweiten Nutzer anzulegen ist keine Option
- ▶ `users.can_manage_temporary_user`
- ▶ Im System sind dies normale Nutzer, haben jedoch eine `assembly_id` (temporär := `assembly_id != null`)

Temporäre Nutzer

- ▶ Szenario:
 - ▶ Nutzergruppe der Veranstaltung nicht bekannt
 - ▶ Anwesende Nutzer sollen in OpenSlides teilnehmen
 - ▶ Einen systemweiten Nutzer anzulegen ist keine Option
- ▶ `users.can_manage_temporary_user`
- ▶ Im System sind dies normale Nutzer, haben jedoch eine `assembly_id` (temporär := `assembly_id != null`)
- ▶ Temporäre Nutzer können sich einloggen, wenn Passwort vergeben (optional!)

Temporäre Nutzer

- ▶ Szenario:
 - ▶ Nutzergruppe der Veranstaltung nicht bekannt
 - ▶ Anwesende Nutzer sollen in OpenSlides teilnehmen
 - ▶ Einen systemweiten Nutzer anzulegen ist keine Option
- ▶ `users.can_manage_temporary_user`
- ▶ Im System sind dies normale Nutzer, haben jedoch eine `assembly_id` (temporär := `assembly_id != null`)
- ▶ Temporäre Nutzer können sich einloggen, wenn Passwort vergeben (optional!)
- ▶ Temporäre Nutzer können gelöscht werden.

Temporäre Nutzer

- ▶ Szenario:
 - ▶ Nutzergruppe der Veranstaltung nicht bekannt
 - ▶ Anwesende Nutzer sollen in OpenSlides teilnehmen
 - ▶ Einen systemweiten Nutzer anzulegen ist keine Option
- ▶ `users.can_manage_temporary_user`
- ▶ Im System sind dies normale Nutzer, haben jedoch eine `assembly_id` (temporär := `assembly_id != null`)
- ▶ Temporäre Nutzer können sich einloggen, wenn Passwort vergeben (optional!)
- ▶ Temporäre Nutzer können gelöscht werden.
- ▶ In zentraler Nutzerverwaltung: Vereinigung temporärer Nutzer und umwandlung in systemweiten Nutzer (`assembly_id=null`)