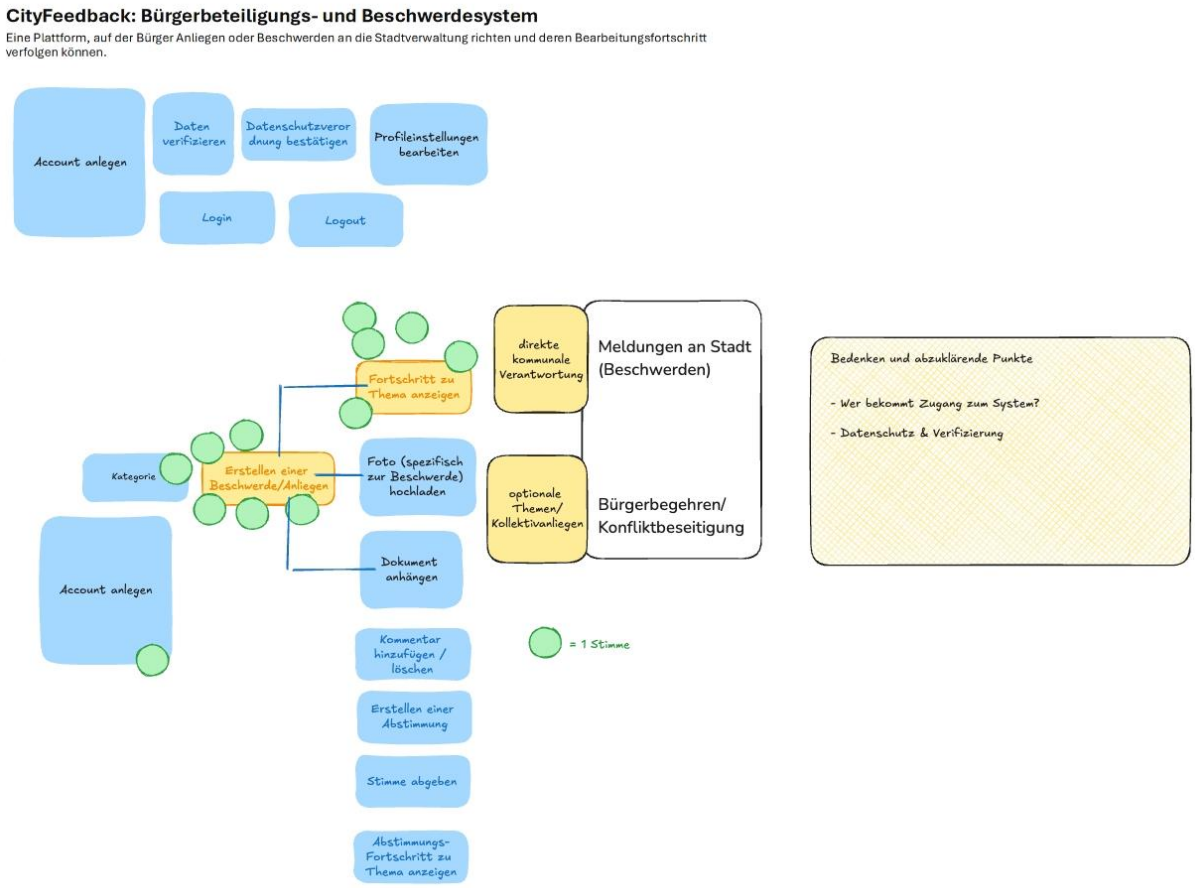
1. Event Storming

Im Rahmen einer kollaborativen Team-Session auf der Plattform Excalidraw, wurden die für das CityFeedback-Projekt relevanten Events identifiziert, und eine visuelle Darstellung inklusive der Reihenfolge und Zusammenhänge erarbeitet.
In der ersten Phase wurden wichtige Events erarbeitet, die in der folgenden Übersicht zu finden sind.

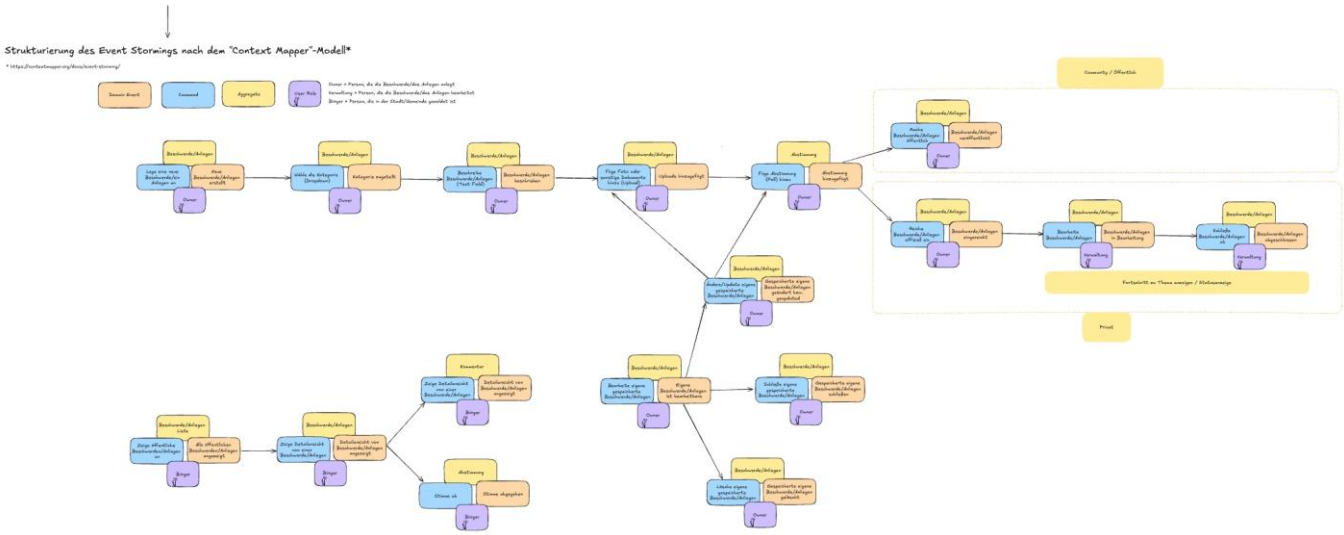
1.1 Gesamtübersicht - Events



Auszug aus Excalibur-Übersicht mit Abstimmung über welche Events wir als Teil des Projektes für relevant erachten

Aufbauend darauf haben wurden die Events in eine detailliertere visuelle Darstellung überführt, und dabei die Strukturierung nach dem Context-Mapper-Modell vorgenommen.

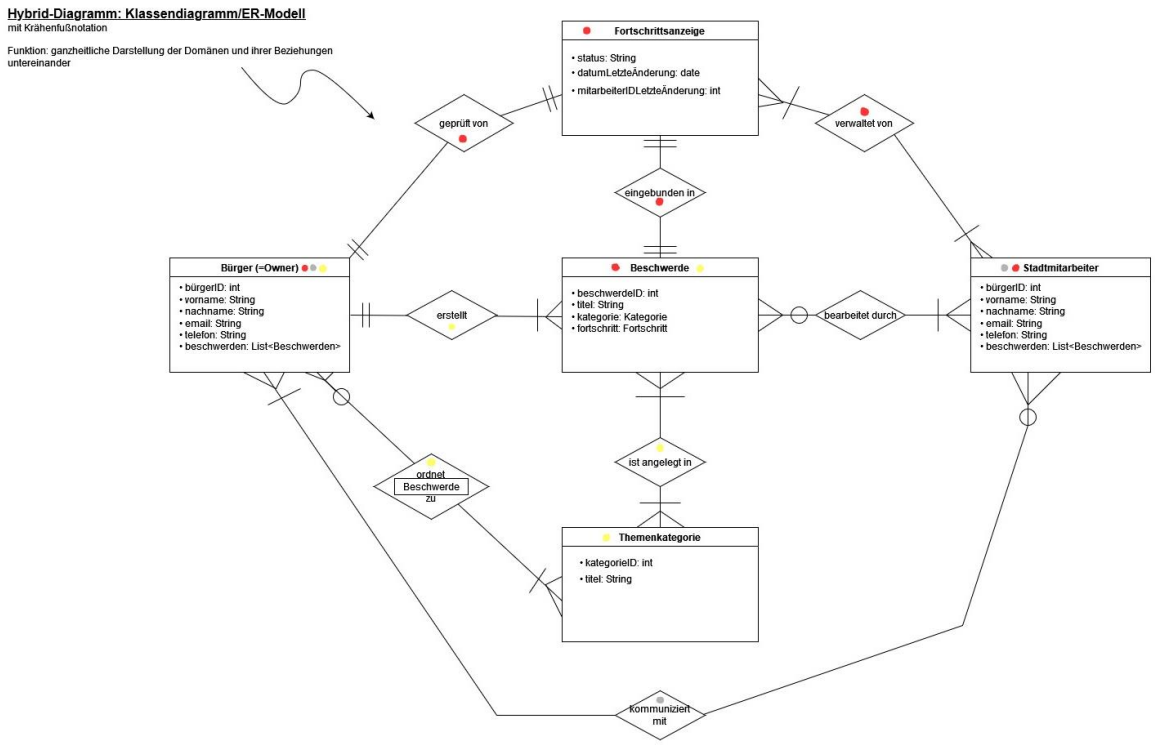
1.2. Strukturierung nach “Context-Mapper-Modell"



Auszug aus Excalibur-Übersicht: Strukturierung des Event Stormings nach dem "Context Mapper-Modell"

2. Domänenmodell

Um die Domänen und die Beziehungen ganzheitlich darzustellen, wird ein hybrides Diagramm aus Klassen- und ER-Modell erstellt.



3. Bounded Contexts

Folgende Tabelle identifiziert die Bounded Contexts des Systems. Die Vorgänge und alle jeweils beteiligten Entities sind mit dem entsprechenden Farbsystem auch im obenstehenden Diagramm im Teil Domänenmodell vermerkt.

Vorgang	Akteure	Entities	Beschreibung
Kommunikation mit Bürger	Bürger, Stadtmitarbeiter	Bürger, Stadtmitarbeiter (,Beschwerde wg. Sachbezug)	Bürger und Stadtmitarbeiter stehen im bedarfsmäßigen Austausch miteinander. Die Kommunikation, etwa Rückfragen zum Beschwerdekontext, wird jedoch nicht über unser Portal vorgenommen, sondern der Einfachheit halber über Mail/Telefon, wie bei der Beschwerdeerstellung vom Bürger angegeben. Daher: keine Systemimplementierung.
Erstellung und Kategorisierung von Beschwerden	Bürger	Bürger, Beschwerde, Kategorie	Bürger können bei der Erstellung ihrer Beschwerde eine passende thematische Kategorie auswählen und somit ihr Anliegen Mitarbeitern besser zuordenbar machen. Die Kategorie kann bei Bedarf verändert werden, jedoch ausschließlich vom Owner.
Einsicht und Wartung Fortschrittsanzeige	Bürger, Stadtmitarbeiter	Bürger, Beschwerde, Fortschrittsanzeige	Bürger und Stadtmitarbeiter können die Statusanzeige zum Bearbeitungsfortschritt der Beschwerde betrachten und jederzeit prüfen. Eine Änderung der Fortschrittsanzeige erfolgt ausschließlich durch einen oder mehrere Stadtmitarbeiter.

4. Entitäten und Aggregates

Im nächsten Schritt wurden die Entitäten und Aggregate innerhalb der identifizierten Bounded Contexts definiert. Basierend auf den ermittelten Events und dem Domänenmodell wurden die relevanten Entitäten und Aggregate mit den entsprechenden Attributen und Methoden festgelegt.

4.1 Relevante Bounded Contexts

- ***Bürger-Kontext:***
  Bürger erstellt, reicht ein und verwaltet seine Beschwerden.
- ***Stadtverwaltungs-Kontext****:*
  (Stadt-)Mitarbeiter bearbeitet eingereichte Beschwerden, aktualisiert deren Fortschritt.
- ***Fortschrittsverfolgungs-Kontext****:*
  zur Verfolgung und Verwaltung des (Fortschritts-)Status von Beschwerden
- ***Kategorisierungs-Kontext****:*
  zur Verwaltung der Kategorien, die den Beschwerden zugeordnet werden können.

4.2 Entitäten und ihre Attribute & Methoden

Entität: Bürger

Attribute	Methoden
• *bürgerID: int* • *vorname: String* • *nachname: String* • *email: String* • *telefon: String* • *beschwerden: List<Beschwerden>*	• *beschwerdeErstellen(titel: String, kategorie: Kategorie): Beschwerde* • *beschwerdeEinreichen(beschwerdeID: int): void* • *beschwerdeAnzeigen(beschwerdeID: int): Beschwerde* • *alleBeschwerdenAnzeigen(): List <Beschwerde>* • *Getter & Setter*

Entität: (Stadt-)Mitarbeiter

Attribute	Methoden
• *mitarbeiterID: int* • *vorname: String* • *nachname: String* • *email: String* • *telefon: String* • *beschwerden: List<Beschwerde>*	• *beschwerdeBearbeitungStarten(beschwerdeID): void* • *beschwerdeAnzeigen(beschwerdeID: int): Beschwerde* • *alleBeschwerdenAnzeigen(): List <Beschwerde>* • *Getter & Setter*

Nachtrag für Implementierung s. Abschnitt 6:
Superclass: Benutzer, Subclasses: Bürger, Mitarbeiter, Admin

Entität: Beschwerde

Attribute	Methoden
• *beschwerdeID: int* • *titel: String* • *kategorie: Kategorie* • *fortschritt: Fortschritt*	• *Getter & Setter*

Entität: Kategorie

Attribute	Methoden
• *kategorieID: int* • *titel: String*	• *Getter & Setter*

Entität: Fortschritt

Attribute	Methoden
• *status: String* • *datumLetzteÄnderung: date* • *mitarbeiterIDLetzteÄnderung: int*	• *zeigeStatus(): String* • *aktualisiereStatus(neuerStatus: String, mitarbeiterID: int, datum: date)* • *Getter & Setter*

Hinweis: Getter & Setter sind nicht explizit ausformuliert, aber vorgesehen. Ebenso die Konstruktoren.

4.3 Aggregates & Beziehungen:

- **Beschwerde ist Bestandteil eines Bürgers** (von ihm erstellt), kann aber auch unabhängig von ihm existieren (z.B. wenn Bürger nicht mehr im System ist, bleibt Beschwerde im System bestehen) Aggregate
  - Ein Bürger kann mehrere Beschwerden haben (1..*)
  - Eine Beschwerde gehört genau einem Bürger (1..1)

- **Beschwerde ist Bestandteil eines Stadtmitarbeiters** (von ihm bearbeitet), kann aber auch unabhängig von ihm existieren (z.B. wenn Mitarbeiter nicht mehr im System ist, bleibt Beschwerde im System bestehen) Aggregate
  - Ein Mitarbeiter kann mehrere Beschwerden bearbeiten (1..*)
  - Eine Beschwerde wird von genau einem Mitarbeiter (hauptverantwortlich) bearbeitet (1..1)

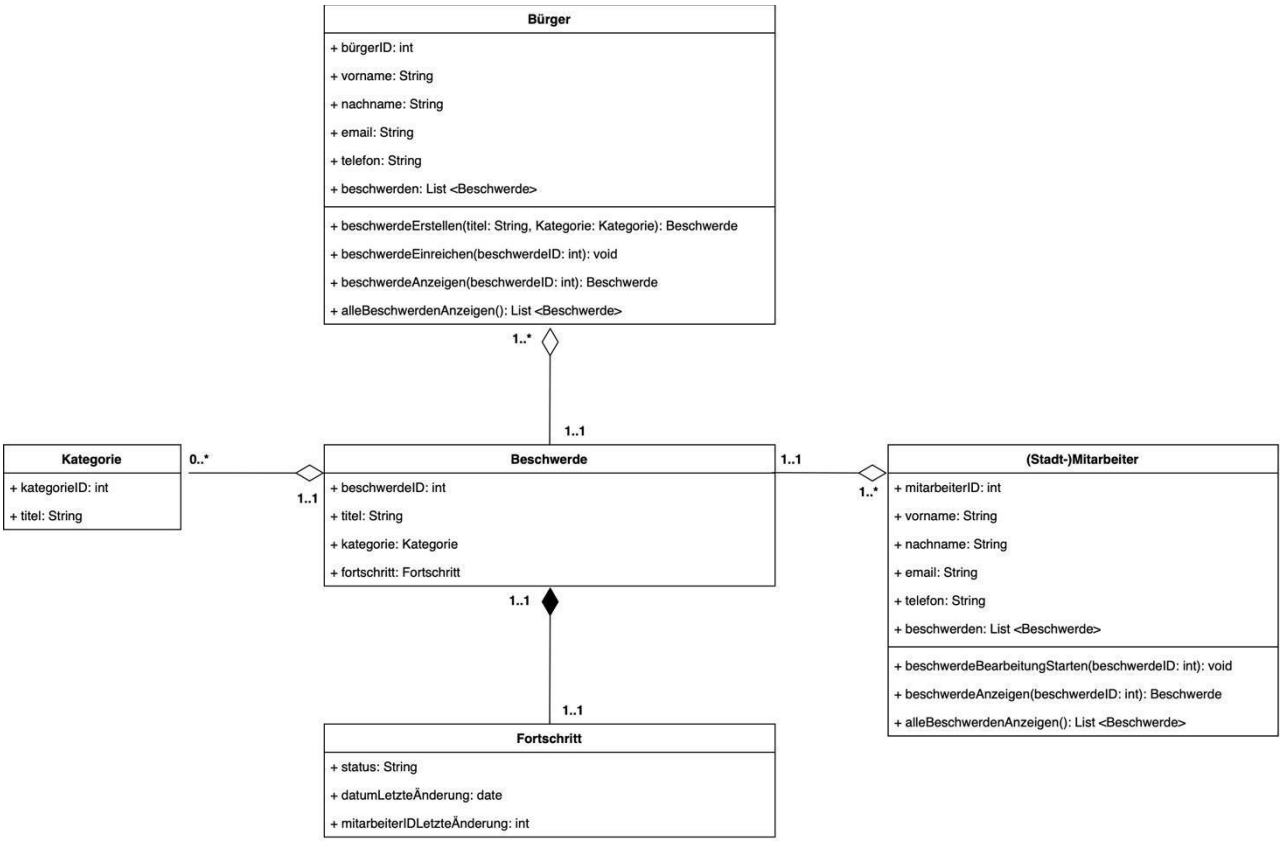
- **Fortschritt ist Bestandteil einer Beschwerde**, und existiert nur in Verbindung mit ihr (wenn Beschwerde gelöscht wird, wird auch Fortschritt gelöscht) Komposition

  - Eine Beschwerde hat genau einen Fortschritt (1..1)
  - Ein Fortschritt existiert nur in Verbindung mit einer Beschwerde (1..1)
- **Kategorie ist Bestandteil einer Beschwerde** (beschreibt Art der Beschwerde), existiert jedoch auch unabhängig von ihr (z.B. wenn die Beschwerde gelöscht wird, bleibt die Kategorie weiterhin verfügbar). □ Aggregate
  - Eine Beschwerde hat genau eine Kategorie (1..1)
  - Eine Kategorie kann vielen Beschwerden zugeordnet sein (0..*)

Ziel: Ein detailliertes Verständnis der Datenstruktur und der Logik in Ihrer Anwendung.
Output: Erstellen Sie ein Dokument, das jede Entität und ihr zugehöriges Aggregate beschreibt, einschließlich ihrer Beziehungen.

4.4 Visualisierung per UML

Die folgende Grafik zeigt jede Entität und ihre zugehörigen Aggregate einschließlich ihrer Beziehungen. Diese Darstellung basiert auf einem UML-Diagramm, das eine verdichtete Version des im Abschnitt 2 gezeigten Hybrid-Diagramms des Domänenmodells ist.



5. Domain Services und Repositories

Dieser Abschnitt beschreibt die Domain Services und Repositories, die die zentralen Geschäftslogiken verwalten und den Zugriff auf die Datenebene ermöglichen.

5.1 Domain Services

- **AuthentifizierungsService:** ist verantwortlich für die Verwaltung der Benutzeranmeldung und -registrierung sowie für die Überprüfung der Identität von Benutzern. Er sorgt dafür, dass nur autorisierte Benutzer auf bestimmte Funktionen zugreifen können.
- **BeschwerdeService:** Verwaltet domänenübergreifende Logik zur Bearbeitung und Verwaltung von Beschwerden.
- **StatusAnzeigeService:** Dieser Service stellt die Logik bereit, um den aktuellen Status einer Beschwerde an Benutzer anzuzeigen oder einen bestehenden Status zu bearbeiten.
- **BenachrichtigungsService:** ist dafür zuständig, Benutzer über relevante Ereignisse in Bezug auf ihre Beschwerden zu informieren, wie z.B. Statusänderungen oder neue Kommentare.

5.2 Repositories

- **BenutzerRepository:** Dient zur Speicherung und Verwaltung von Benutzerinformationen, wie z.B. Anmeldedaten, E-Mails etc..
- **AuthentifizierungsRepository:** Speichert und prüft Authentifizierungsinformationen, wie gehashte Passwörter und Token.
- **BeschwerdeRepository:** Dieses Repository ist für die Verwaltung und den Zugriff auf Beschwerden verantwortlich. Es bietet Methoden, um Beschwerden aus der Datenbank abzurufen, zu speichern oder zu aktualisieren.
- **StatusRepository:** Dieses Repository verwaltet den Status einer Beschwerde und speichert den Statusverlauf.

5.3 Übersicht der Repositories und Methoden

- **BenutzerRepository**
  - *findBenutzerById(Number id): Benutzer*
  - *indBenutzerByName(String name): Benutzer[]*
  - *saveBenutzer(Benutzer: benutzer): void*

- **AuthentifizierungsRepository**
  - *tokenValidation(String token): Boolean*

    - *benutzerAnmeldung(String email, String passwort, String Rolle): string*
    - *benutzerRegistrierung(Benutzer: benutzer): Number*
- **BeschwerdeRepository**
    - *findBeschwerdeById(Number id): Beschwerde*
    - *findBeschwerdeByTitleAndDescription(String text): Beschwerde[]*
    - *saveBeschwerde(Beschwerde beschwerde): void*
- **StatusRepository**
    - *findCurrentStatus(Number beschwerdeId): string*
    - *updateStatus(Number beschwerdeId, String status): void*

6. Implementierungsstrategie

6.1. Herangehensweise

Im Rahmen der Implementierung wird folgende strukturierte Herangehensweise verfolgt:

1. Zunächst wird eine Superklasse **Benutzer** als Java-Klasse erstellt, welche die gemeinsamen Attribute und Methoden aus Punkt 4.2. für alle Benutzerrollen enthält. Aufbauend darauf werden die Subklassen **Bürger**, **Mitarbeiter** und **Admin** abgeleitet, die jeweils zusätzliche, spezifische Methoden und Attribute besitzen.
2. Jeder Bereich der Geschäftslogik bzw. des **Domain Services** wird durch eine dedizierte **Service-Klasse** repräsentiert. Diese interagiert über **Repositories** mit der Datenschicht, um **Entitäten** zu verwalten und zu persistieren.
3. Jedes **Repository** wird als Interface definiert und bietet Methoden zum Speichern und Abrufen von **Entitäten**.

6.2. Beispiel Implementierungsstrategie für Beschwerde einreichen

Nachdem nun die allgemeine Implementierungsstrategie beschrieben wurde, folgt im nächsten Schritt ein Beispiel zur konkreten Umsetzung am Beispiel des Service zum Einreichen von Beschwerden. Hierbei wird im Code aus folgendem Grund die englische Sprache verwendet: Die Attribute unter 4.2. sind auf Deutsch benannt, um die Fachterminologie im Kontext des Projekts verständlich zu halten. Im Code wird jedoch Englisch verwendet, da dies in der Programmierung als internationale Standardsprache gilt und die Wartbarkeit sowie Zusammenarbeit erleichtert.

```
public class ComplaintService {
    private ComplaintRepository complaintRepository;

    // Constructor to pass the repository
    public ComplaintService(ComplaintRepository complaintRepository) {
        this.complaintRepository = complaintRepository;
    }

    // Method for submitting a new complaint
    public void submitComplaint(...) {
        Complaint complaint = new Complaint(...); // Create new complaint
        complaintRepository.saveComplaint(complaint); // Save complaint to repository
    }

    // Method for updating an existing complaint
    public void updateComplaint(...) {
        Complaint complaint = complaintRepository.findComplaintById(complaintId);
        if (complaint != null) {
            // Update complaint details ...
        }
    }
}
```