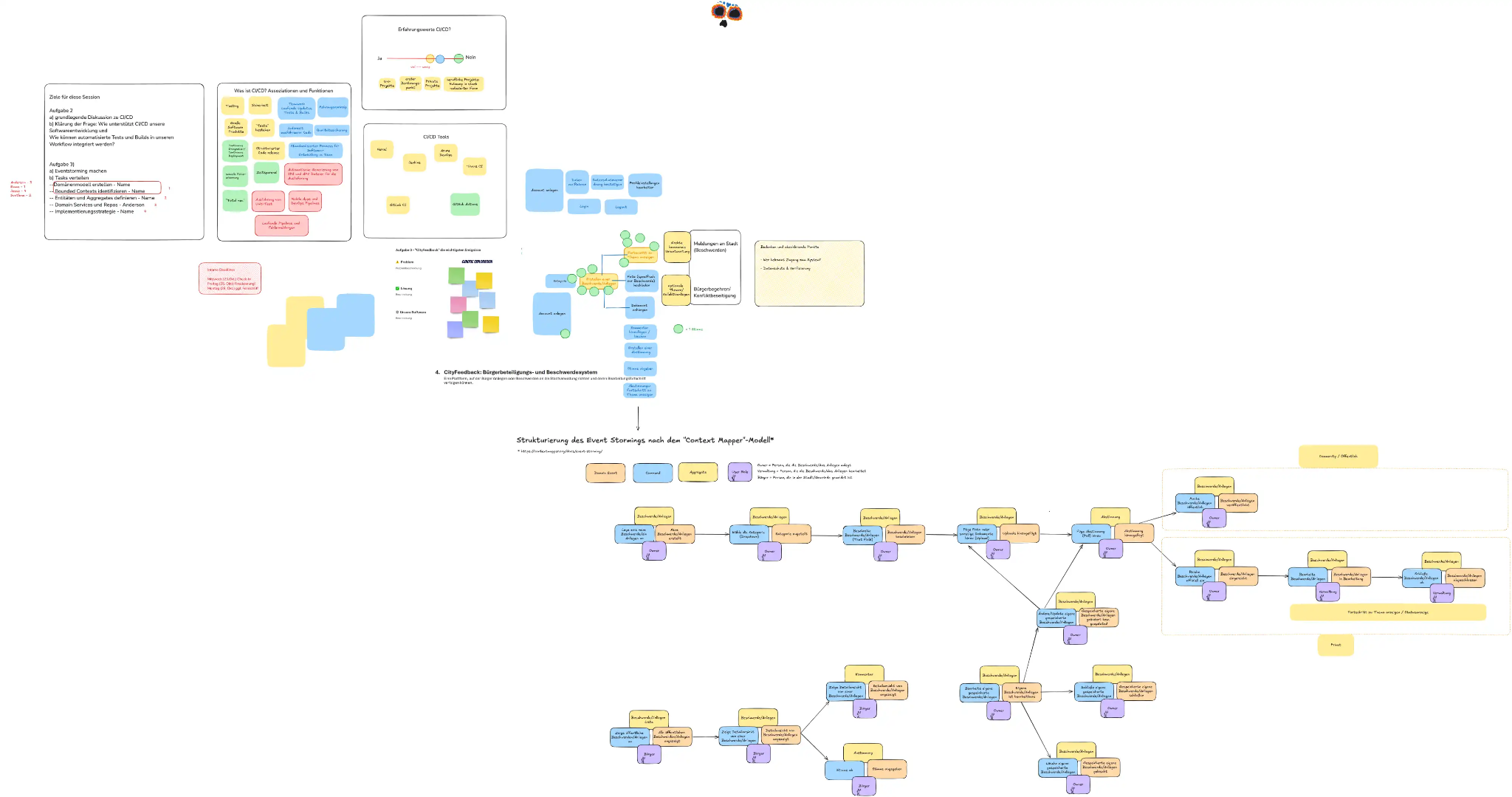
**1. Event Storming**

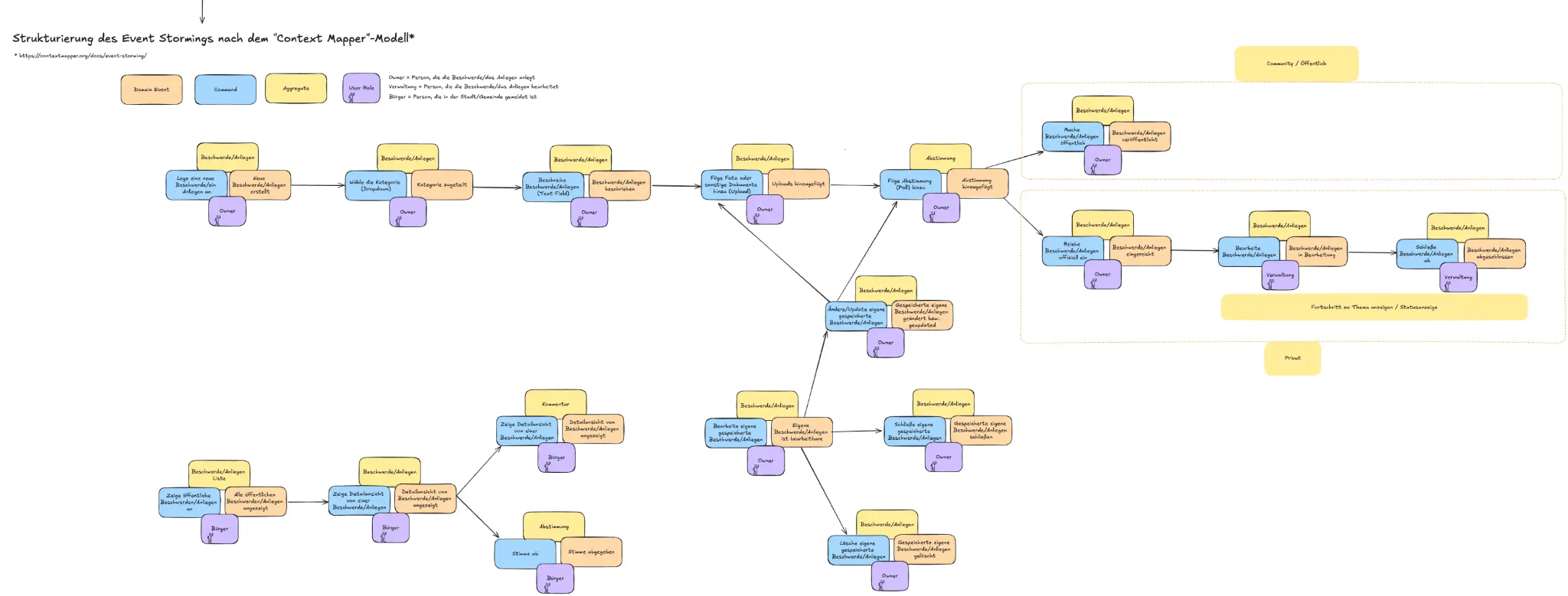
*Einleitung*

[Link zum Excalidraw-Dokument](https://excalidraw.com/#room=61ced5db800bda921629,Pr1lrzHT7A7dIUl5ZnzgnQ)

**1.1 Gesamtübersicht**

****

**1.2. Strukturierung nach Context-Mapper-Modell**

****

**4. Entitäten und Aggregates**

*Entitäten und Aggregates definieren: Definieren Sie die Entitäten und Aggregates innerhalb Ihrer Bounded Contexts, die aus den identifizierten Events und dem Domänenmodell abgeleitet werden. Bestimmen Sie, welche Attribute und Methoden sie haben sollten.*

**4.1 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:***

*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.

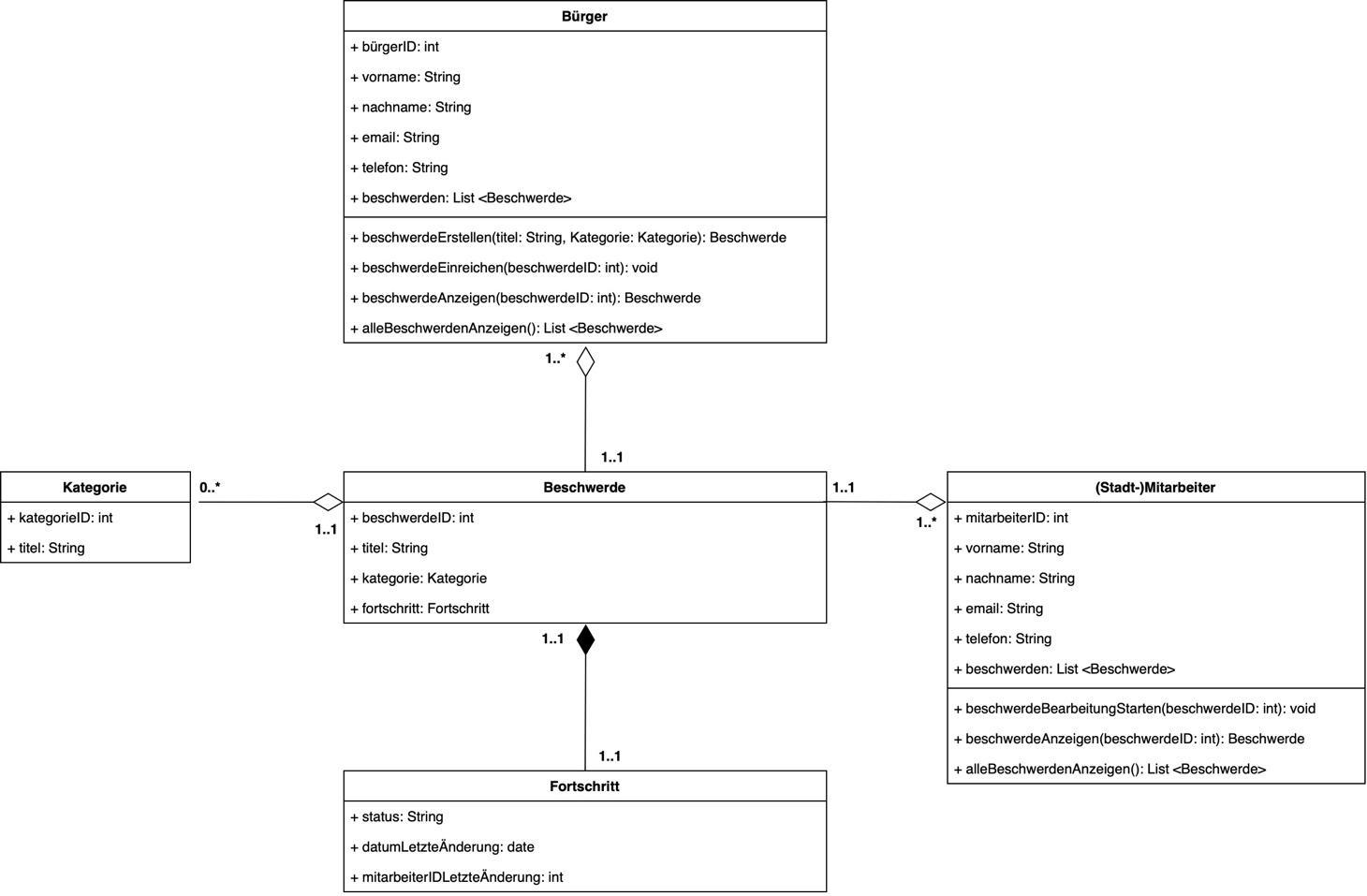
**4.3 Aggregates & Beziehungen:**

* **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**



**5. Domain Services und Repositories**

**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 werden folgende Schritte umgesetzt:

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**. Zur Speicherung der Daten wird **JSON** als Format verwendet, sodass die Entitäten in **JSON-Dateien** serialisiert und bei Bedarf wieder daraus geladen werden können. Da es sich um ein Uni-Projekt ohne Datenbank handelt, wird diese Methode gewählt, um die Persistenz der Daten zu gewährleisten, ohne eine komplexe Datenbank-Infrastruktur aufzubauen. Alternativ kann NoSQL in Betracht gezogen werden und wird ggf. eingesetzt, wenn sich eine höhere Flexibilität bei der Datenabfrage im Verlauf des Projektes aufzeigt.

**6.2. Beispiel Implementierungsstrategie für Beschwerde einreichen**

| 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 ...  }  } } |
| --- |