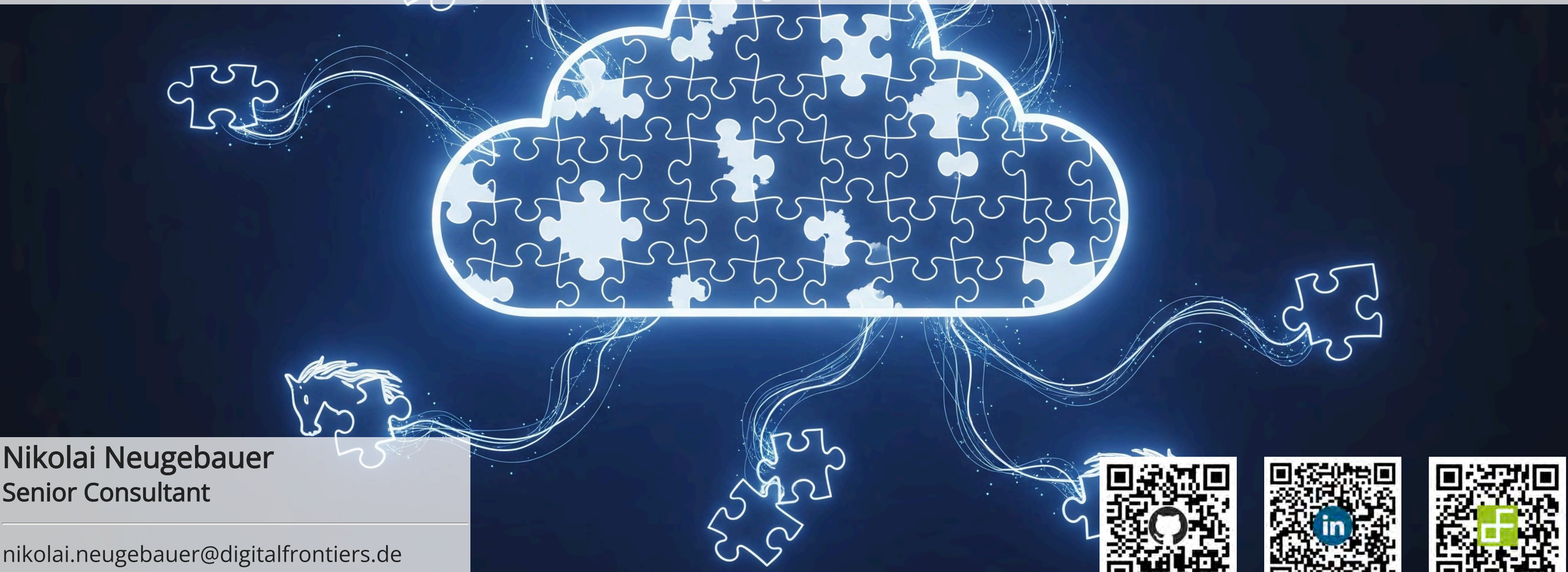
Jenseits von Dependency Injection

Aufgeräumte Fachlogik ohne Seiteneffekte



Nikolai Neugebauer
Senior Consultant

nikolai.neugebauer@digitalfrontiers.de
linkedin.com/in/nikolai-neugebauer
github.com/Nik101010

IT früher

Image by NASA on Wikimedia

IT heute

Image by Michael Pewny from Pixabay

Verteilte Daten

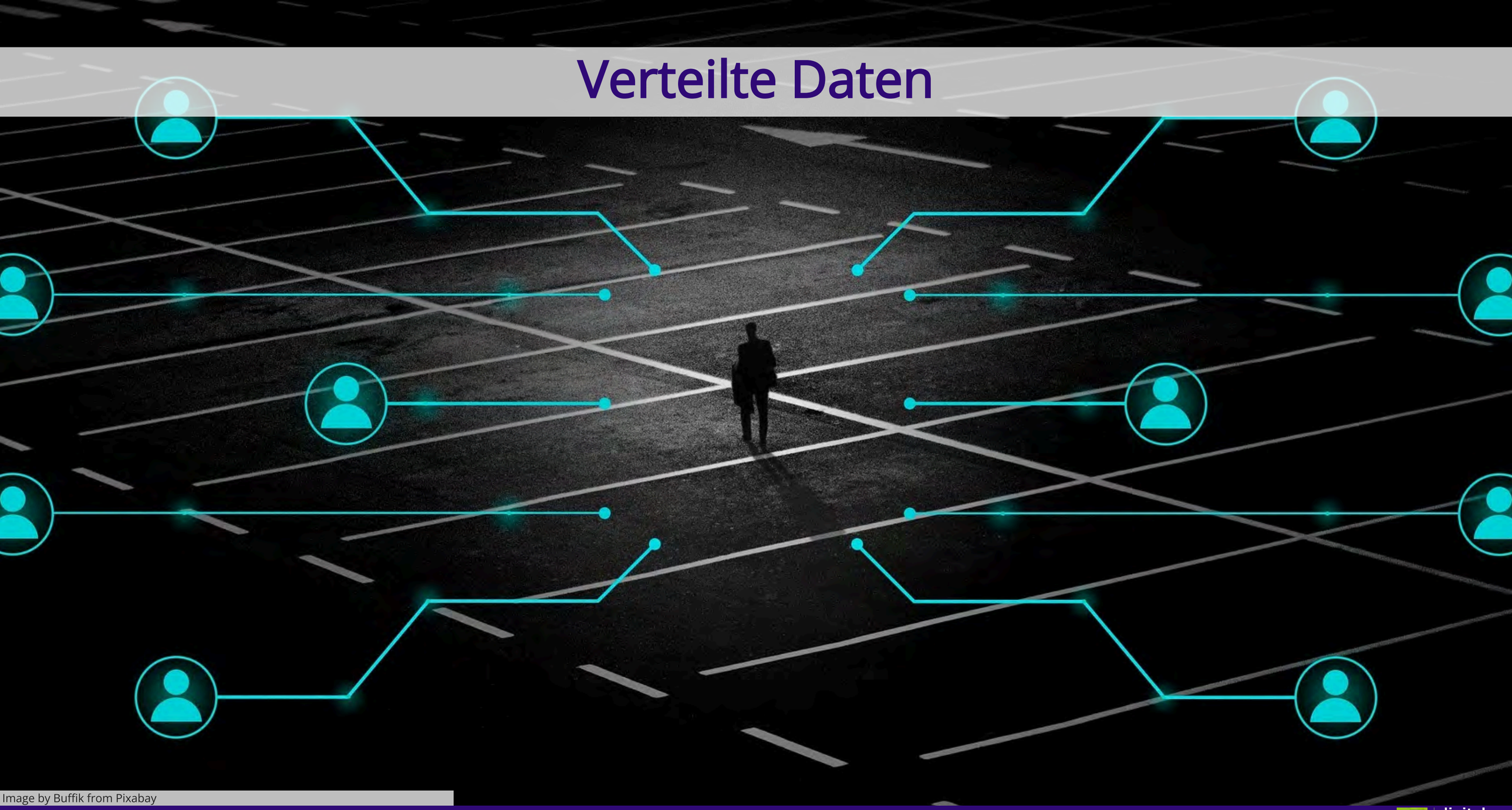


Image by Buffik from Pixabay

Verteilte Abhägigkeiten

Image by Kevin Schwarz from Pixabay

Netzwerke

Image by Tapani Hellman from Pixabay

Zero Downtime

Image by Romy from Pixabay

Häufige Deployments

Image by Steven Lynn from Unsplash

Hybride Zugriffe

Image by websubs from Pixabay

Und wie schreiben wir unseren Code?

Image by Camille Orgel from Pixabay

Code wird immer komplexer

Code wird immer komplexer

```java
@Service
public class MediaService {

    private final MediaRepository mediaRepository;
    private final ReaderService readerService;
    private final OutboundMessaging outboundMessaging;
    private final EmailService emailService;

  public MediaService(MediaRepository mediaRepository, ReaderService readerService, OutboundMessaging outbound
        this.mediaRepository = mediaRepository;
        this.readerService = readerService;
        this.outboundMessaging = outboundMessaging;
        this.emailService = emailService;
    }

    @Transactional
    public void borrowMedia(Long id, Long readerId) {
        Media media = mediaRepository.findById(id)
                .orElseThrow(() -> new MediaNotFoundException("mediaId not found: " + id));

        if (!media.isAvailable()) {
            throw new MediaNotAvailableException("media not available because of reasons, id: " + id);
        }

        if (
```

Code wird immer komplexer

```java
@Service
public class MediaService {

    private final MediaRepository mediaRepository;
    private final ReaderService readerService;
    private final OutboundMessaging outboundMessaging;
    private final EmailService emailService;

  public MediaService(MediaRepository mediaRepository, ReaderService readerService, OutboundMessaging outbound
        this.mediaRepository = mediaRepository;
        this.readerService = readerService;
        this.outboundMessaging = outboundMessaging;
        this.emailService = emailService;
    }

    @Transactional
    public void borrowMedia(Long id, Long readerId) {
        Media media = mediaRepository.findById(id)
                .orElseThrow(() -> new MediaNotFoundException("mediaId not found: " + id));

        if (!media.isAvailable()) {
            throw new MediaNotAvailableException("media not available because of reasons, id: " + id);
        }

```

Code wird immer komplexer

```
 7      private final EmailService emailService;
 8
 9    public MediaService(MediaRepository mediaRepository, ReaderService readerService, OutboundMessaging outbound
10          this.mediaRepository = mediaRepository;
11          this.readerService = readerService;
12          this.outboundMessaging = outboundMessaging;
13          this.emailService = emailService;
14      }
15
16      @Transactional
17      public void borrowMedia(Long id, Long readerId) {
18          Media media = mediaRepository.findById(id)
19                  .orElseThrow(() -> new MediaNotFoundException("mediaId not found: " + id));
20
21          if (!media.isAvailable()) {
22              throw new MediaNotAvailableException("media not available because of reasons, id: " + id);
23          }
24
25          if (media.getRentedTo() != null) {
26              throw new MediaRentedException("media already rented, id: " + id);
27          }
28
29          media.setRentedTo(readerId);
30          media.setAvailable(false);
```

Code wird immer komplexer

```
12          this.outboundMessaging = outboundMessaging;
13          this.emailService = emailService;
14      }
15
16      @Transactional
17      public void borrowMedia(Long id, Long readerId) {
18          Media media = mediaRepository.findById(id)
19                  .orElseThrow(() -> new MediaNotFoundException("mediaId not found: " + id));
20
21          if (!media.isAvailable()) {
22              throw new MediaNotAvailableException("media not available because of reasons, id: " + id);
23          }
24
25          if (media.getRentedTo() != null) {
26              throw new MediaRentedException("media already rented, id: " + id);
27          }
28
29          media.setRentedTo(readerId);
30          media.setAvailable(false);
31
32          switch (media.getMediaType()) {
33              case BOOK -> media.setExpectedReturnDate(LocalDate.now().plusDays(28));
34              case MAGAZINE -> media.setExpectedReturnDate(LocalDate.now().plusDays(7));
35          }
36
```

Code wird immer komplexer

```
21          if (!media.isAvailable()) {
22              throw new MediaNotAvailableException("media not available because of reasons, id: " + id);
23          }
24
25          if (media.getRentedTo() != null) {
26              throw new MediaRentedException("media already rented, id: " + id);
27          }
28
29          media.setRentedTo(readerId);
30          media.setAvailable(false);
31
32          switch (media.getMediaType()) {
33              case BOOK -> media.setExpectedReturnDate(LocalDate.now().plusDays(28));
34              case MAGAZINE -> media.setExpectedReturnDate(LocalDate.now().plusDays(7));
35          }
36
37          Media savedMedia = mediaRepository.save(media);
38
39          outboundMessaging.publishMediaLentMessage(
40                  savedMedia.getId(),
41                  readerId, savedMedia.getExpectedReturnDate()
42          );
43
44          Reader reader = readerService.loadReader(readerId);
45          emailService.publishEmail(
```

Code wird immer komplexer

```
29          media.setRentedTo(readerId);
30          media.setAvailable(false);
31
32          switch (media.getMediaType()) {
33              case BOOK -> media.setExpectedReturnDate(LocalDate.now().plusDays(28));
34              case MAGAZINE -> media.setExpectedReturnDate(LocalDate.now().plusDays(7));
35          }
36
37          Media savedMedia = mediaRepository.save(media);
38
39          outboundMessaging.publishMediaLentMessage(
40                  savedMedia.getId(),
41                  readerId, savedMedia.getExpectedReturnDate()
42          );
43
44          Reader reader = readerService.loadReader(readerId);
45          emailService.publishEmail(
46                  new MailTemplate.MediaLent(
47                          savedMedia.getAuthor(),
48                          savedMedia.getTitle(),
49                          savedMedia.getExpectedReturnDate(),
50                          reader.getName()
51                  ),
52                  reader.getEmailAddress()
```

Code wird immer komplexer

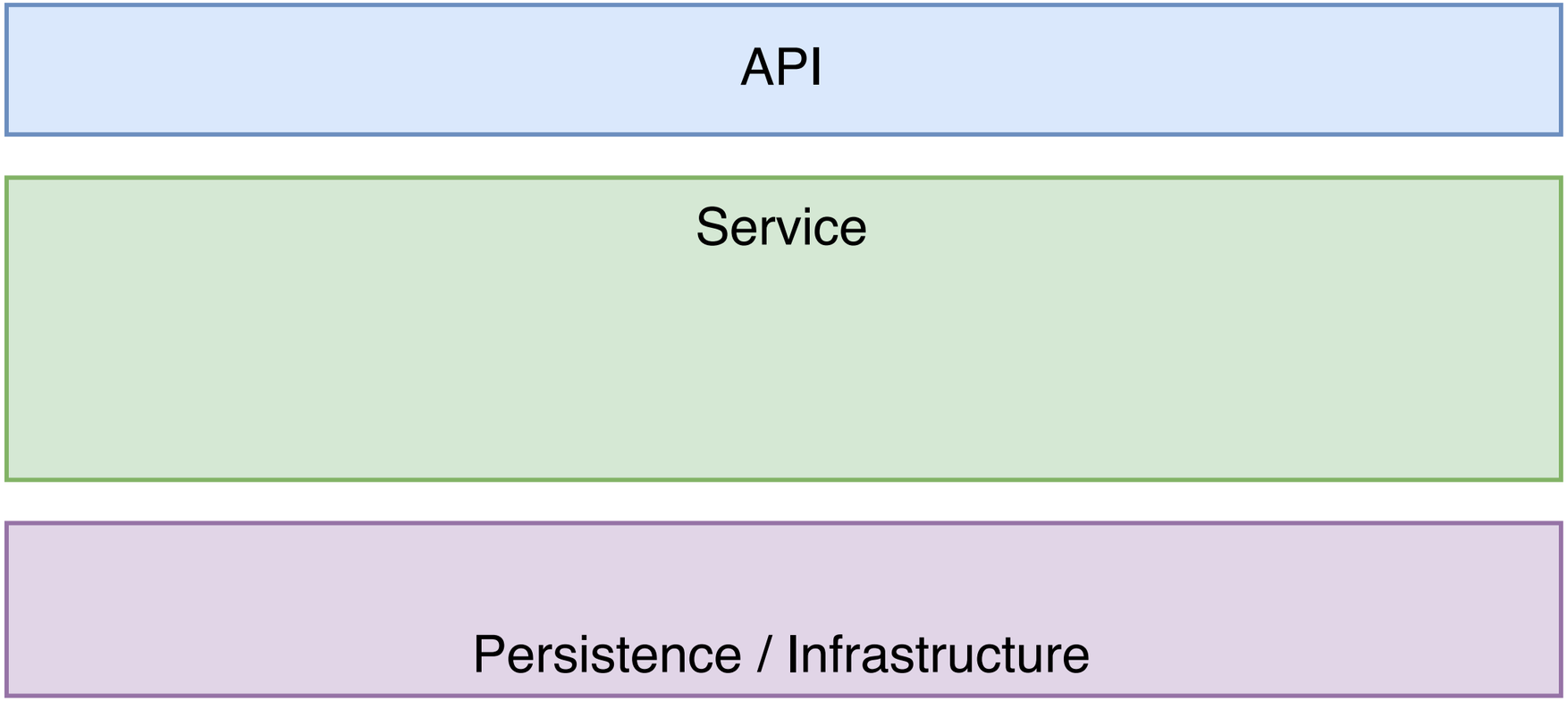
```
32          switch (media.getMediaType()) {
33              case BOOK -> media.setExpectedReturnDate(LocalDate.now().plusDays(28));
34              case MAGAZINE -> media.setExpectedReturnDate(LocalDate.now().plusDays(7));
35          }
36
37          Media savedMedia = mediaRepository.save(media);
38
39          outboundMessaging.publishMediaLentMessage(
40                  savedMedia.getId(),
41                  readerId, savedMedia.getExpectedReturnDate()
42          );
43
44          Reader reader = readerService.loadReader(readerId);
45          emailService.publishEmail(
46                  new MailTemplate.MediaLent(
47                          savedMedia.getAuthor(),
48                          savedMedia.getTitle(),
49                          savedMedia.getExpectedReturnDate(),
50                          reader.getName()
51                  ),
52                  reader.getEmailAddress()
53          );
54      }
55  }
```

Code wird immer komplexer

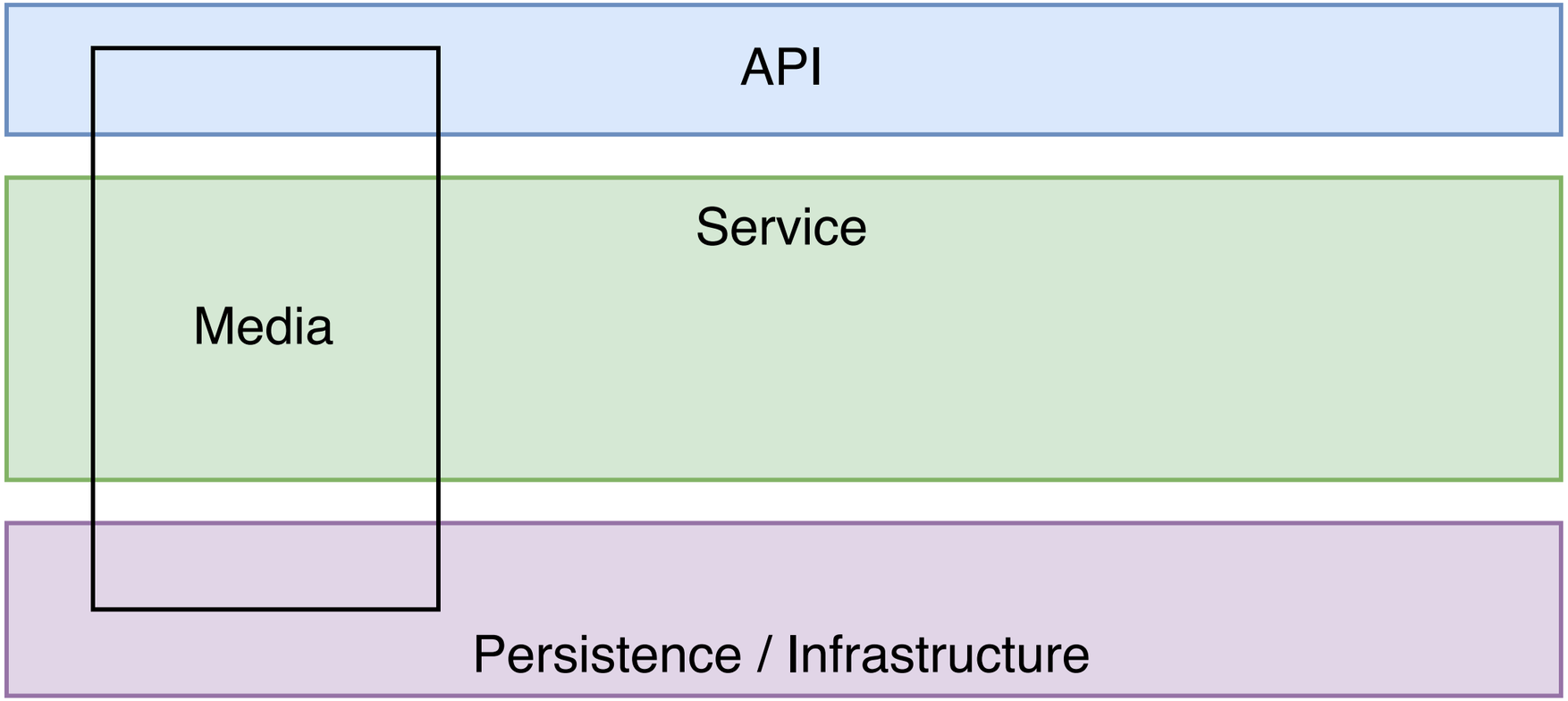
```
32          switch (media.getMediaType()) {
33              case BOOK -> media.setExpectedReturnDate(LocalDate.now().plusDays(28));
34              case MAGAZINE -> media.setExpectedReturnDate(LocalDate.now().plusDays(7));
35          }
36
37          Media savedMedia = mediaRepository.save(media);
38
39          outboundMessaging.publishMediaLentMessage(
40                  savedMedia.getId(),
41                  readerId, savedMedia.getExpectedReturnDate()
42          );
43
44          Reader reader = readerService.loadReader(readerId);
45          emailService.publishEmail(
46                  new MailTemplate.MediaLent(
47                          savedMedia.getAuthor(),
48                          savedMedia.getTitle(),
49                          savedMedia.getExpectedReturnDate(),
50                          reader.getName()
51                  ),
52                  reader.getEmailAddress()
53          );
54      }
55  }
```

Schichtenarchitektur

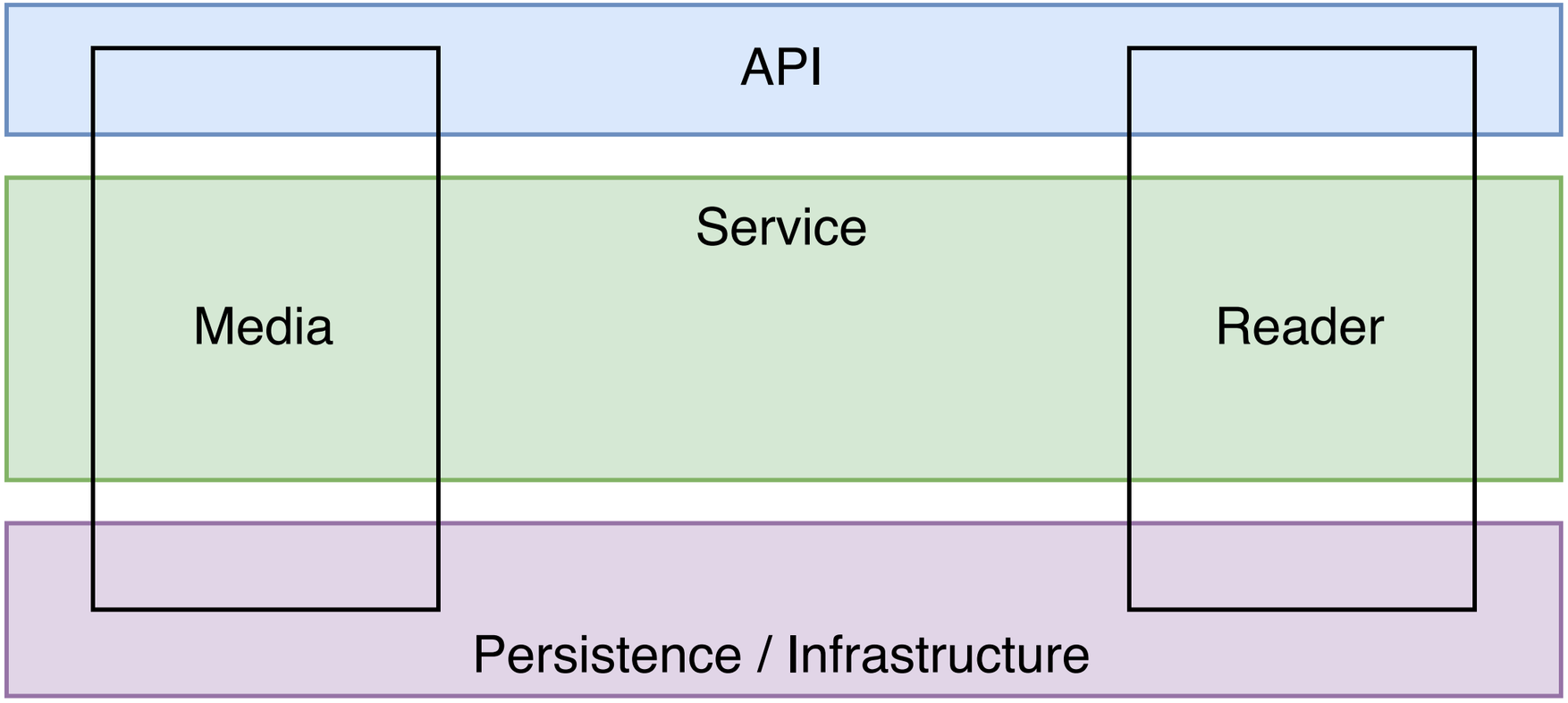
Schichtenarchitektur



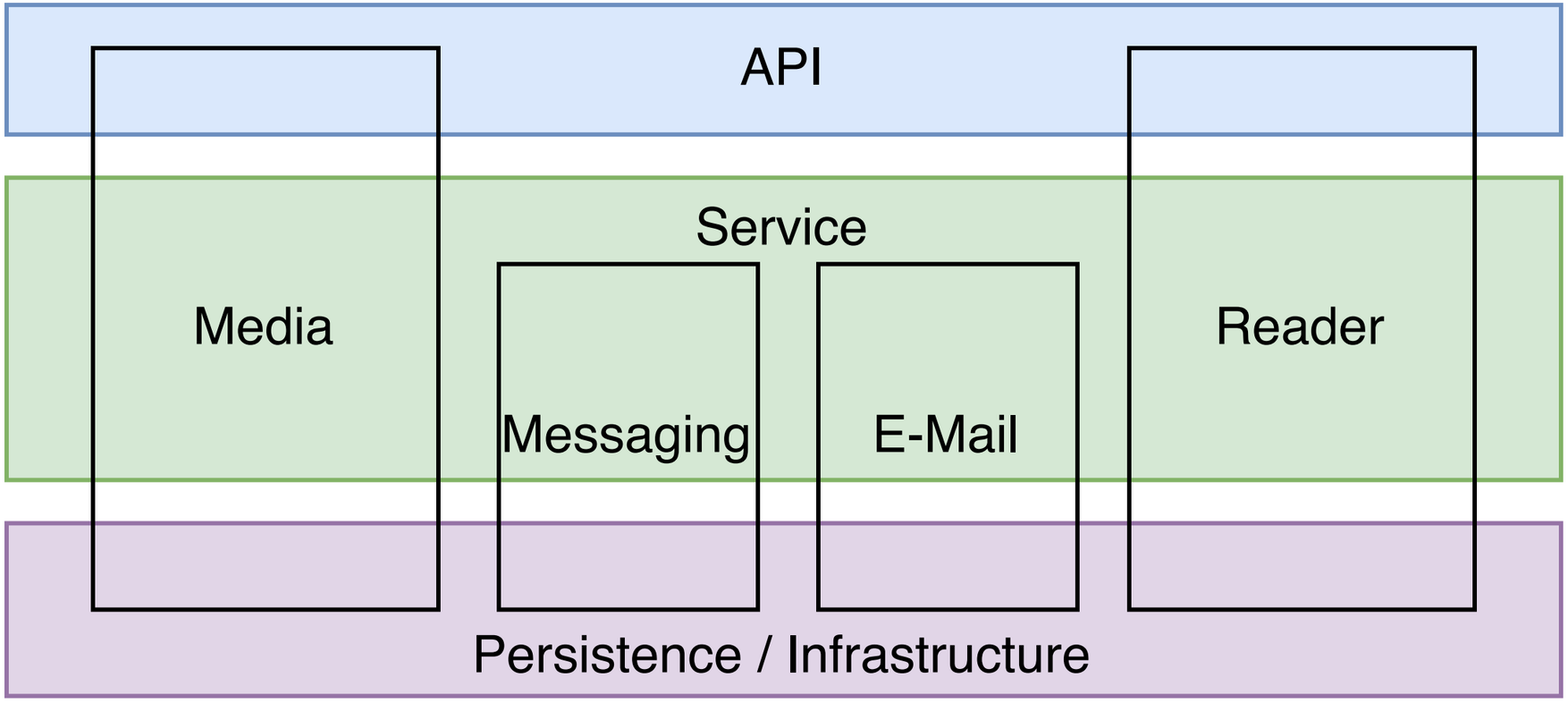
Schichtenarchitektur



Schichtenarchitektur

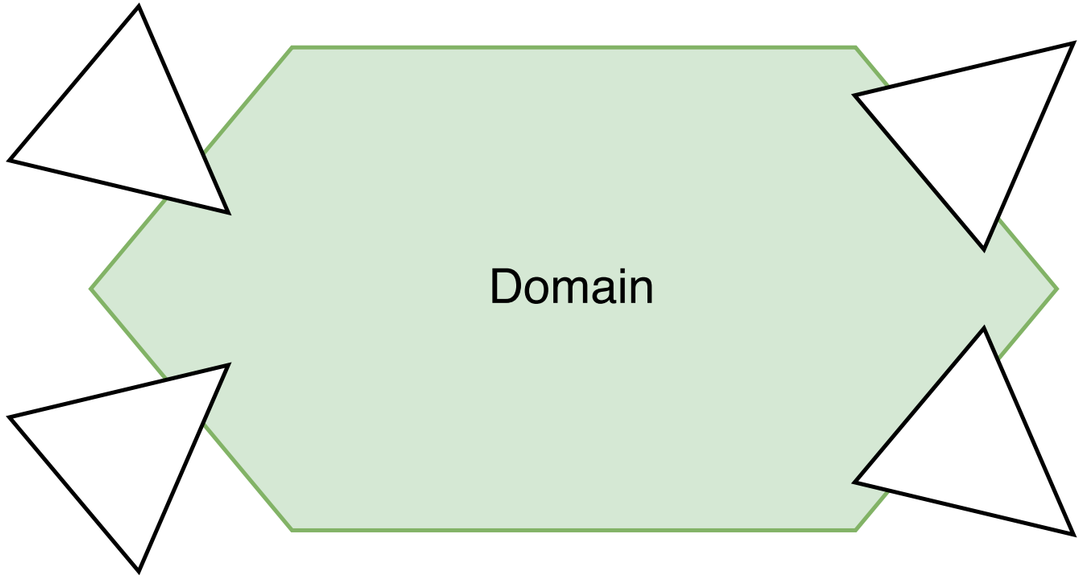


Schichtenarchitektur

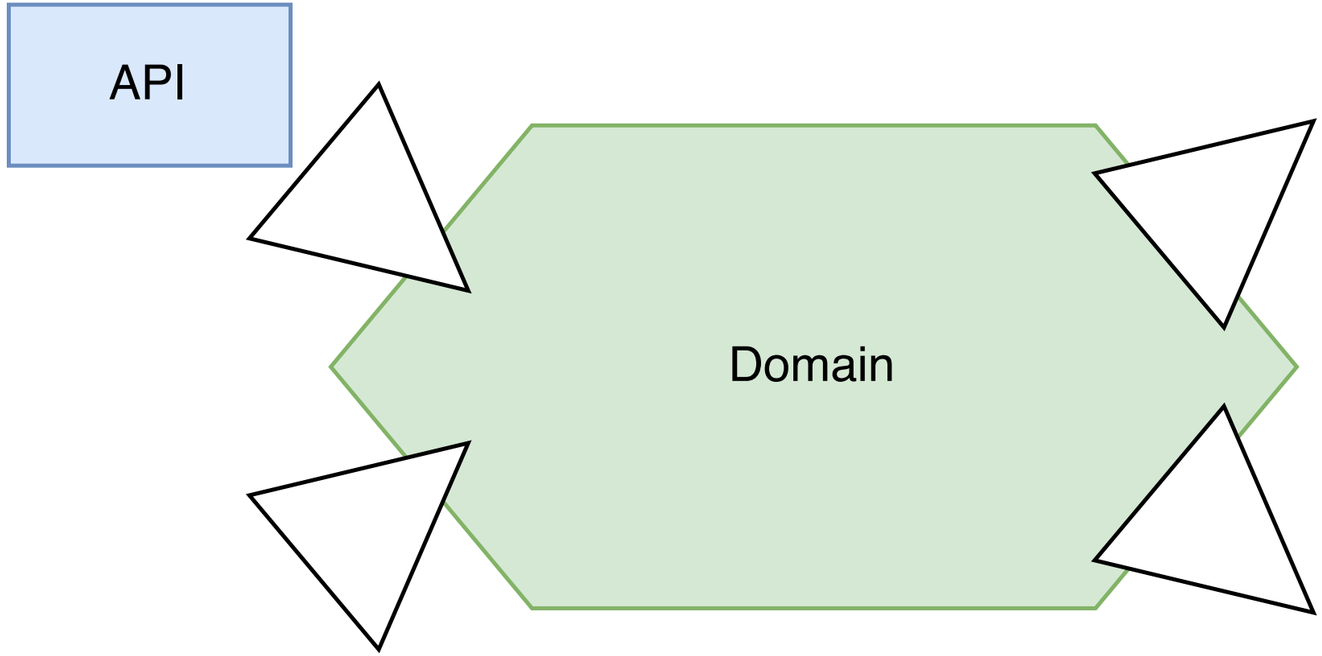


Hexagonale Architektur

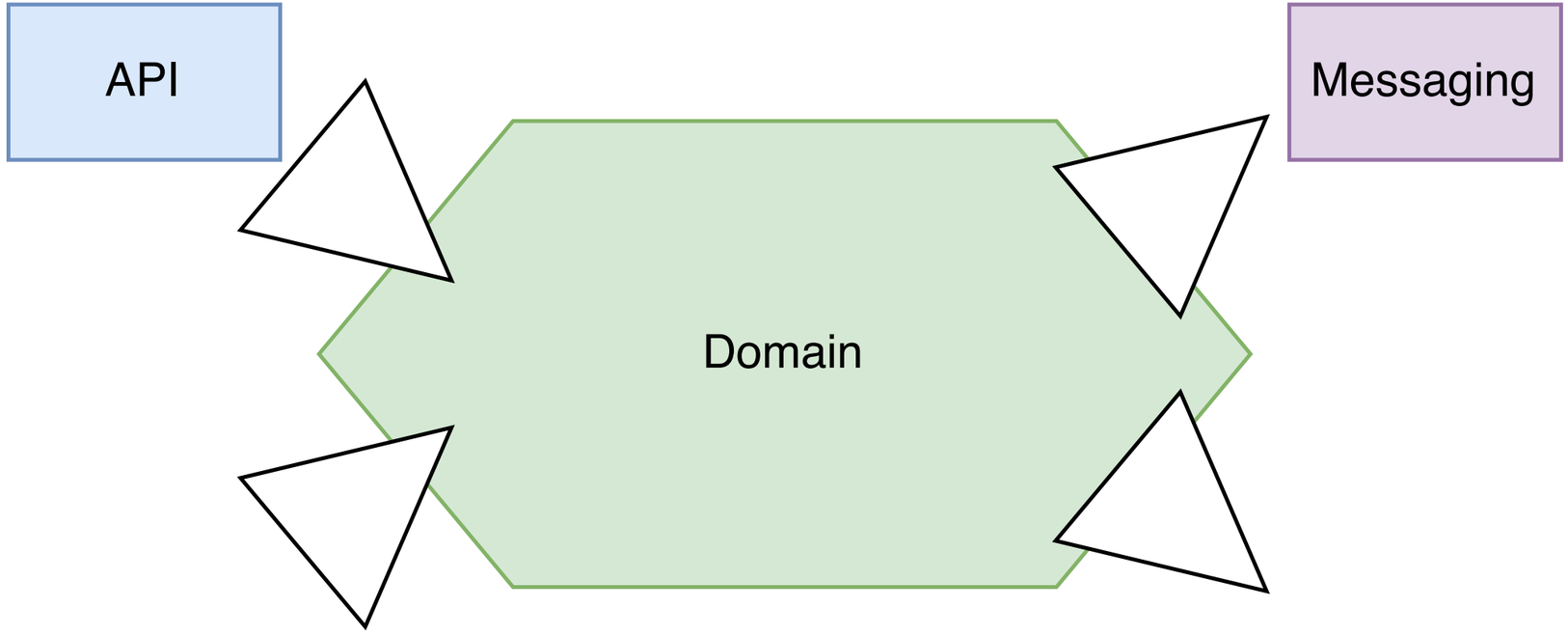
Hexagonale Architektur



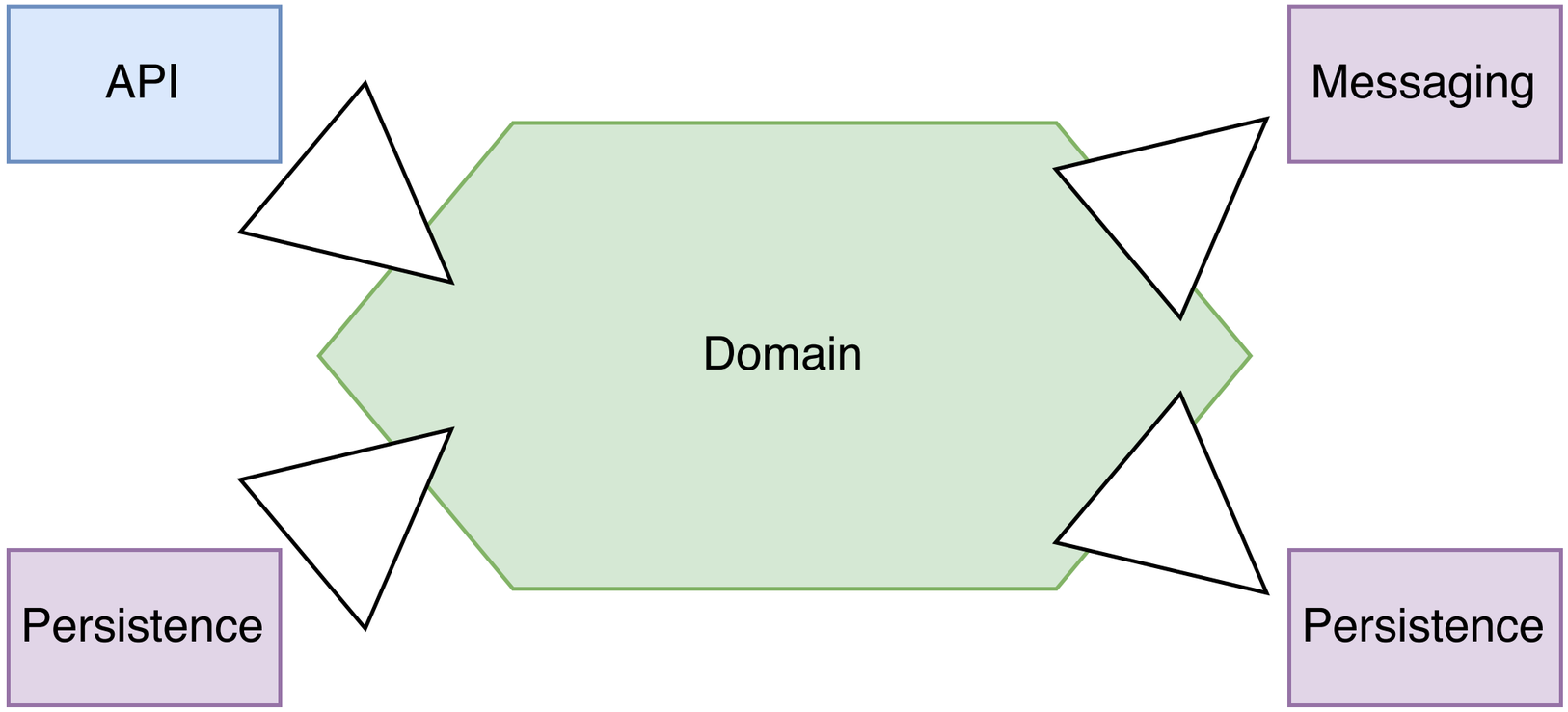
Hexagonale Architektur



Hexagonale Architektur



Hexagonale Architektur



Hexagonaler Code

```java
@Service
public class MediaService {

    private final MediaPersistencePort persistence;
    private final MediaMessagingPort messaging;
    private final MediaEmailPort emailPort;

    public MediaService(MediaPersistencePort persistence, MediaMessagingPort messaging, MediaEmailPort em
        this.persistence = persistence;
        this.messaging = messaging;
        this.emailPort = emailPort;
    }

    @Transactional
    public void borrowMedia(Long id, Long readerId) {
        Media media = persistence.findById(id)
                .orElseThrow(() -> new MediaNotFoundException("mediaId not found: " + id));

        if (!media.isAvailable()) {
            throw new MediaNotAvailableException("media not available because of reasons, id: " + id);
        }

        if (media.getLentTo() != null) {
            throw new MediaBorrowedException("media already borrowed, id: " + id);
```

Hexagonaler Code

```java
    private final MediaMessagingPort messaging;
    private final MediaEmailPort emailPort;

    public MediaService(MediaPersistencePort persistence, MediaMessagingPort messaging, MediaEmailPort em
        this.persistence = persistence;
        this.messaging = messaging;
        this.emailPort = emailPort;
    }

    @Transactional
    public void borrowMedia(Long id, Long readerId) {
        Media media = persistence.findById(id)
                .orElseThrow(() -> new MediaNotFoundException("mediaId not found: " + id));

        if (!media.isAvailable()) {
            throw new MediaNotAvailableException("media not available because of reasons, id: " + id);
        }

        if (media.getLentTo() != null) {
            throw new MediaBorrowedException("media already borrowed, id: " + id);
        }

        media.setLentTo(readerId);
        media.setAvailable(false);
```

Hexagonaler Code

```
15      public void borrowMedia(Long id, Long readerId) {
16          Media media = persistence.findById(id)
17                  .orElseThrow(() -> new MediaNotFoundException("mediaId not found: " + id));
18
19          if (!media.isAvailable()) {
20              throw new MediaNotAvailableException("media not available because of reasons, id: " + id);
21          }
22
23          if (media.getLentTo() != null) {
24              throw new MediaBorrowedException("media already borrowed, id: " + id);
25          }
26
27          media.setLentTo(readerId);
28          media.setAvailable(false);
29
30          switch (media.getMediaType()) {
31              case BOOK -> media.setExpectedReturnDate(LocalDate.now().plusDays(28));
32              case MAGAZINE -> media.setExpectedReturnDate(LocalDate.now().plusDays(7));
33          }
34
35          Media savedMedia = persistence.save(media);
36
37          messaging.publishMediaLentMessage(
38                  savedMedia.getId(),
39                  readerId, savedMedia.getExpectedReturnDate()
```

Hexagonaler Code

```
26
27          media.setLentTo(readerId);
28          media.setAvailable(false);
29
30          switch (media.getMediaType()) {
31              case BOOK -> media.setExpectedReturnDate(LocalDate.now().plusDays(28));
32              case MAGAZINE -> media.setExpectedReturnDate(LocalDate.now().plusDays(7));
33          }
34
35          Media savedMedia = persistence.save(media);
36
37          messaging.publishMediaLentMessage(
38                  savedMedia.getId(),
39                  readerId, savedMedia.getExpectedReturnDate()
40          );
41
42          emailPort.mediaLentEmail(
43                  savedMedia.getAuthor(),
44                  savedMedia.getTitle(),
45                  savedMedia.getExpectedReturnDate(),
46                  readerId
47          );
48      }
49  }
```

Hexagonaler Code

```
26
27          media.setLentTo(readerId);
28          media.setAvailable(false);
29
30          switch (media.getMediaType()) {
31              case BOOK -> media.setExpectedReturnDate(LocalDate.now().plusDays(28));
32              case MAGAZINE -> media.setExpectedReturnDate(LocalDate.now().plusDays(7));
33          }
34
35          Media savedMedia = persistence.save(media);
36
37          messaging.publishMediaLentMessage(
38                  savedMedia.getId(),
39                  readerId, savedMedia.getExpectedReturnDate()
40          );
41
42          emailPort.mediaLentEmail(
43                  savedMedia.getAuthor(),
44                  savedMedia.getTitle(),
45                  savedMedia.getExpectedReturnDate(),
46                  readerId
47          );
48      }
49  }
```

Wir verzichten auf Kopplung!

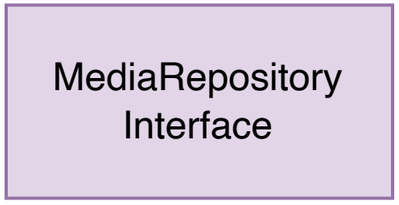
Wir verzichten auf Kopplung!

Wir verzichten auf Kopplung!

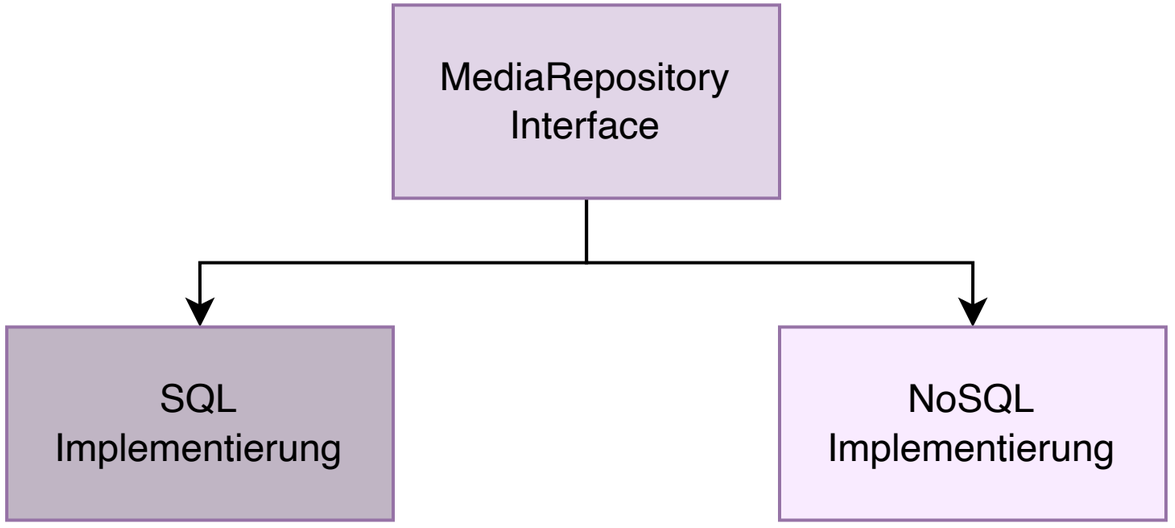
Ausprägungen von Kopplung

Kopplung von Interface / Implementierung

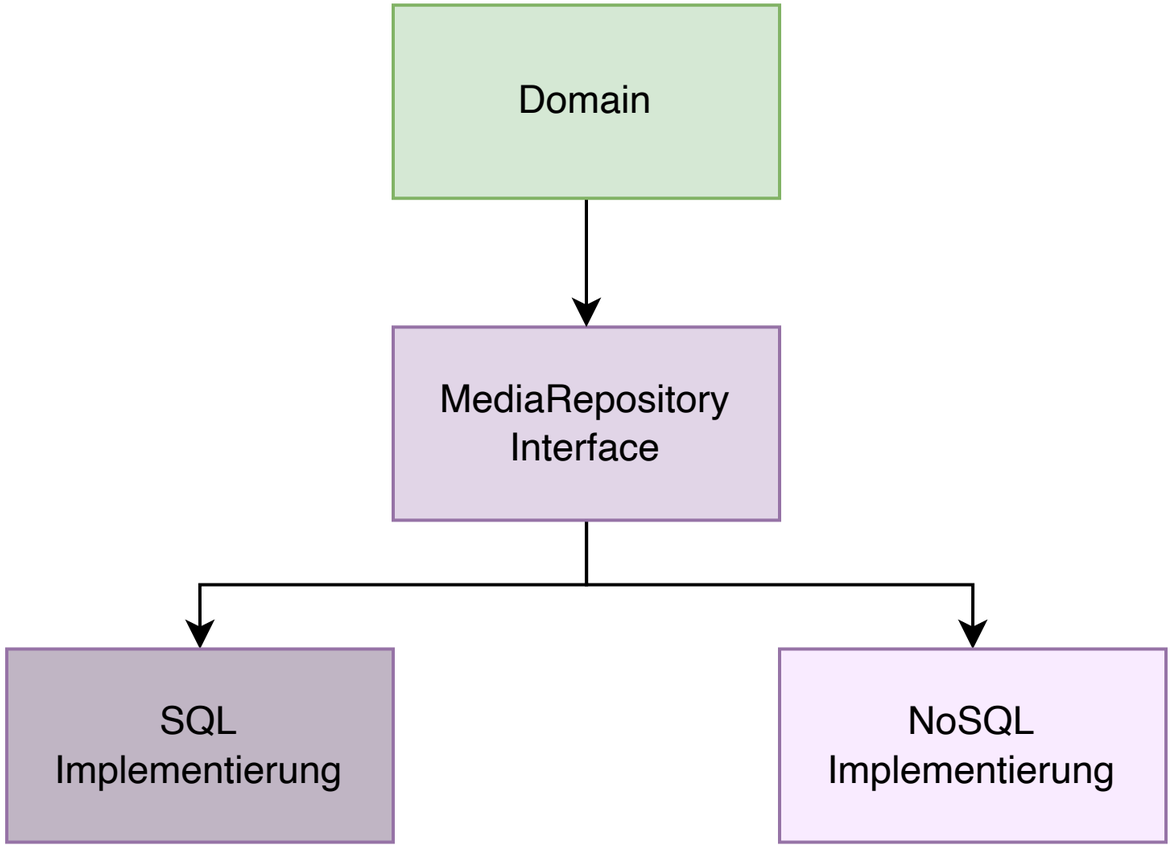
Kopplung von Interface / Implementierung



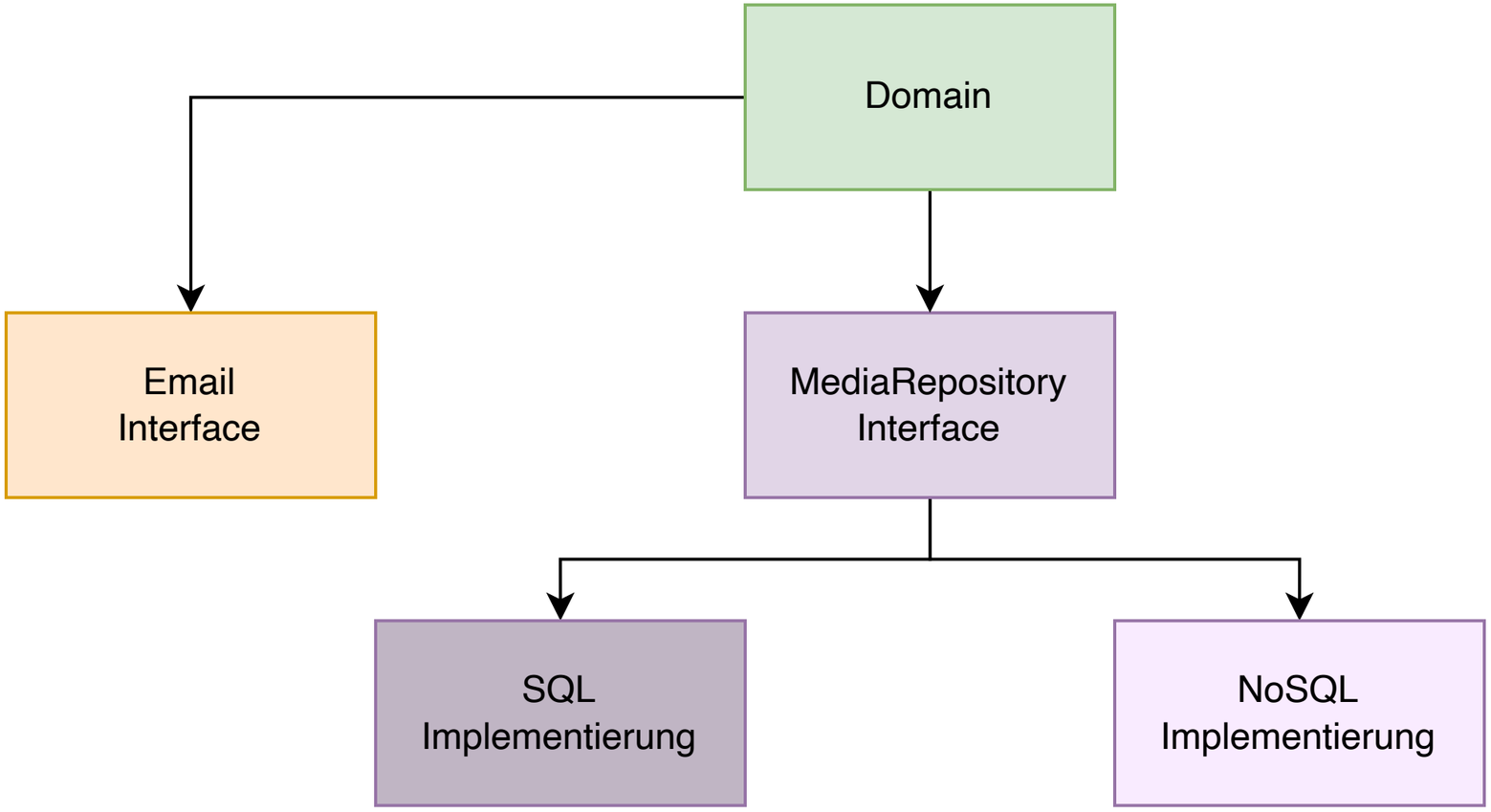
Kopplung von Interface / Implementierung



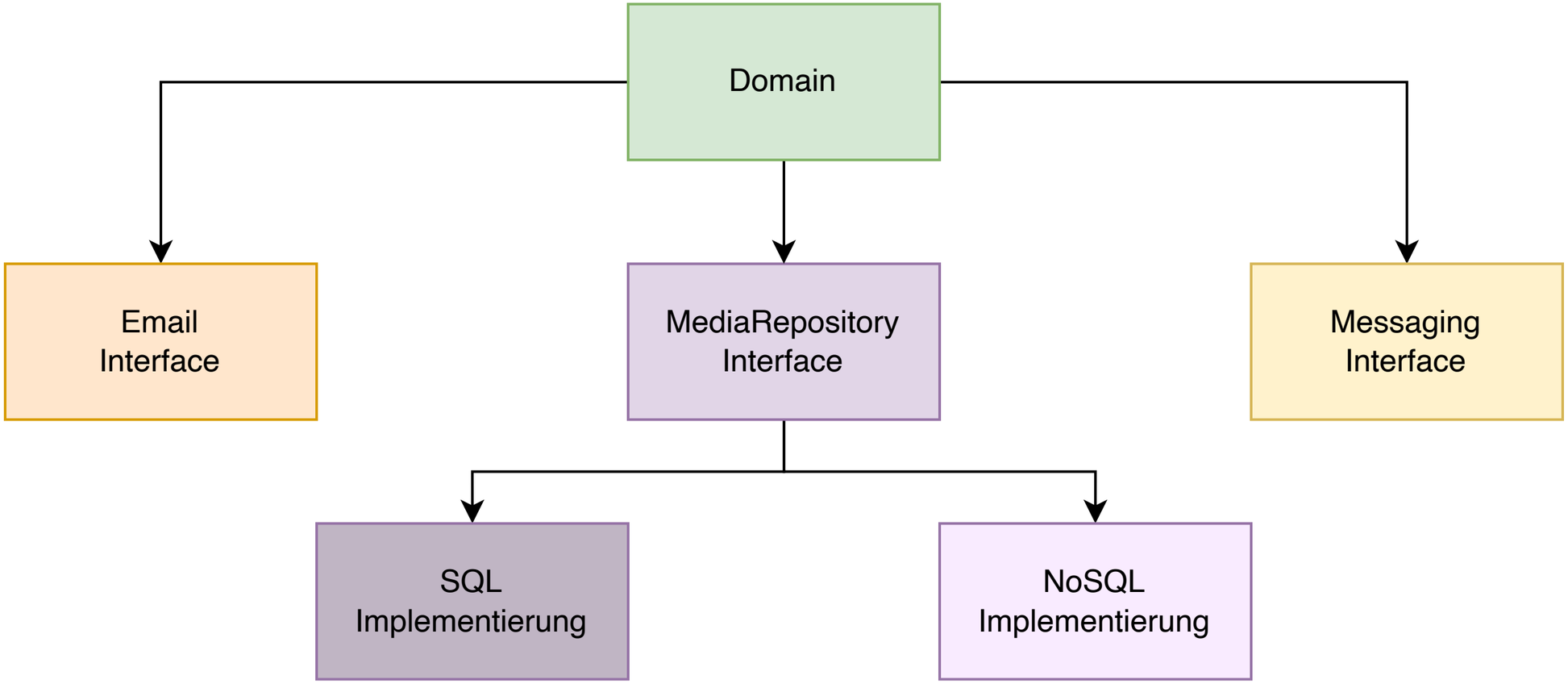
Kopplung von Interface / Implementierung



Kopplung von Interface / Implementierung



Kopplung von Interface / Implementierung



Kopplung an 1 Datenmodell

Kopplung an 1 Datenmodell

- Wie finden wir heraus, wie viele Medien ein Leser ausgeliehen hat?

Kopplung an 1 Datenmodell

- Wie finden wir heraus, wie viele Medien ein Leser ausgeliehen hat?
- Woher bekommen wir die E-Mailadresse des Users?

Kopplung an 1 Datenmodell

- Wie finden wir heraus, wie viele Medien ein Leser ausgeliehen hat?
- Woher bekommen wir die E-Mailadresse des Users?
- Woher wissen wir wie viele Leser gerade Bücher ausgeliehen haben?

Kopplung an 1 Datenmodell

- Wie finden wir heraus, wie viele Medien ein Leser ausgeliehen hat?
- Woher bekommen wir die E-Mailadresse des Users?
- Woher wissen wir wie viele Leser gerade Bücher ausgeliehen haben?
- Wie führen wir ein Audit Log?

Kopplung an 1 Datenmodell

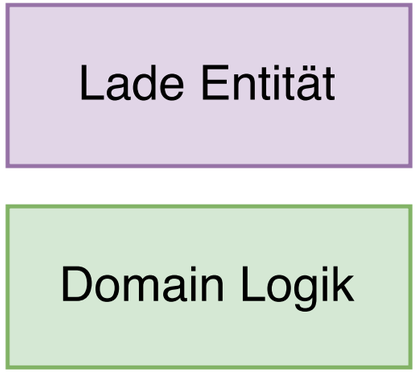
- Wie finden wir heraus, wie viele Medien ein Leser ausgeliehen hat?
- Woher bekommen wir die E-Mailadresse des Users?
- Woher wissen wir wie viele Leser gerade Bücher ausgeliehen haben?
- Wie führen wir ein Audit Log?

Gehört das alles in ein großes Modell?

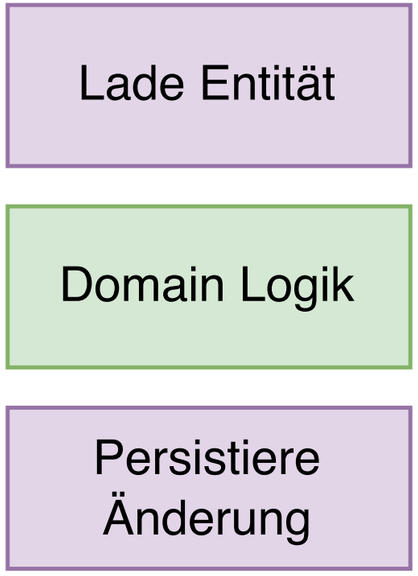
Laufzeit / Transaktionskopplung

Laufzeit / Transaktionskopplung

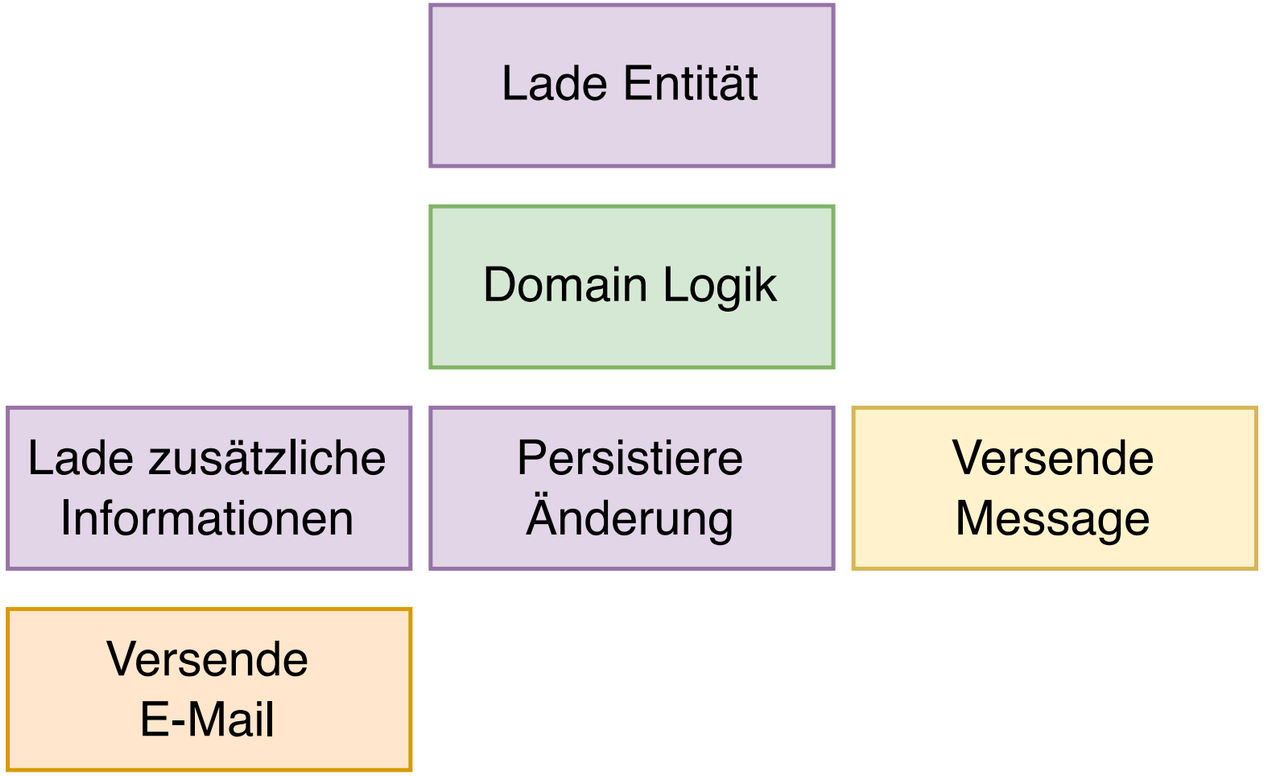
Laufzeit / Transaktionskopplung



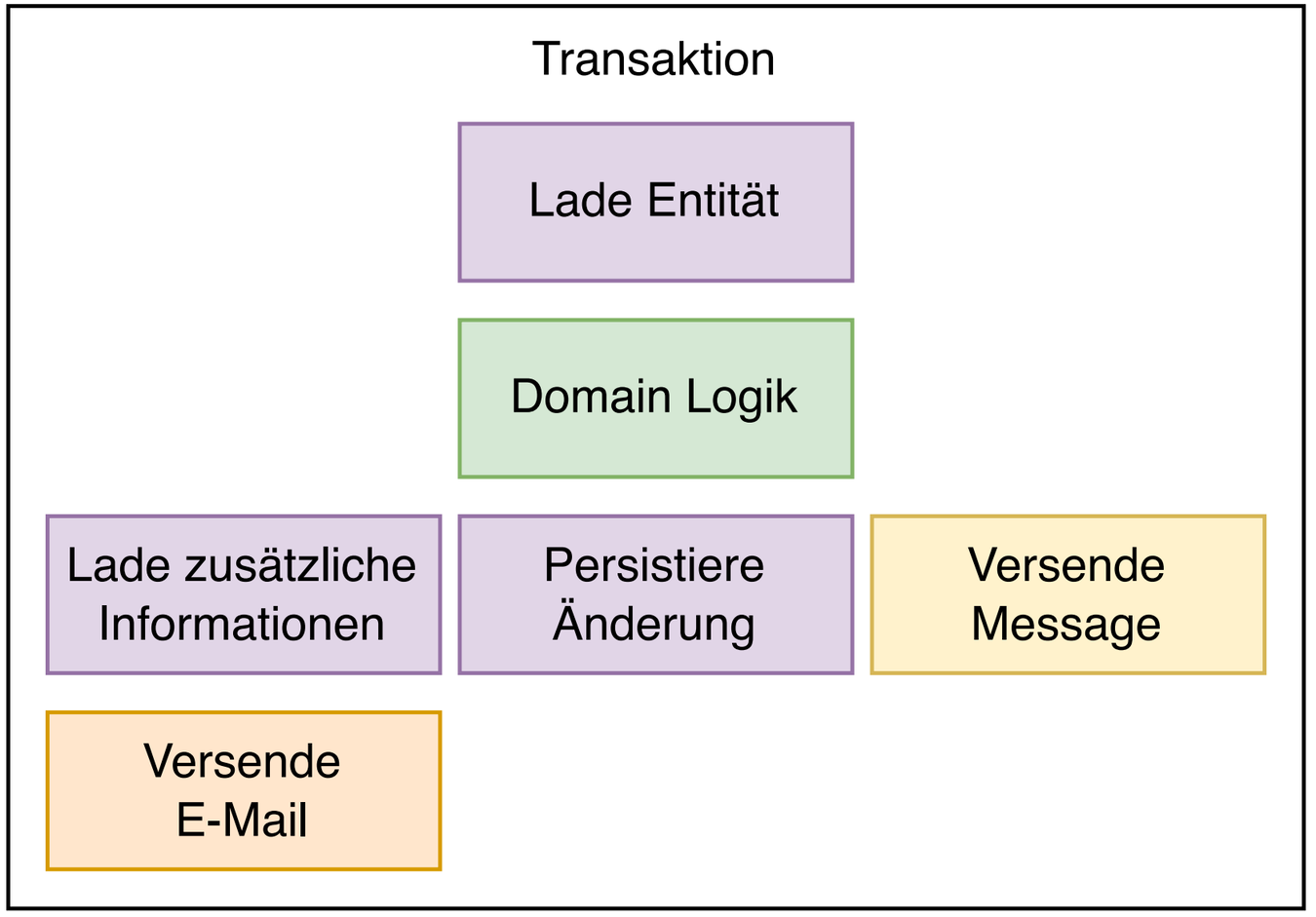
Laufzeit / Transaktionskopplung



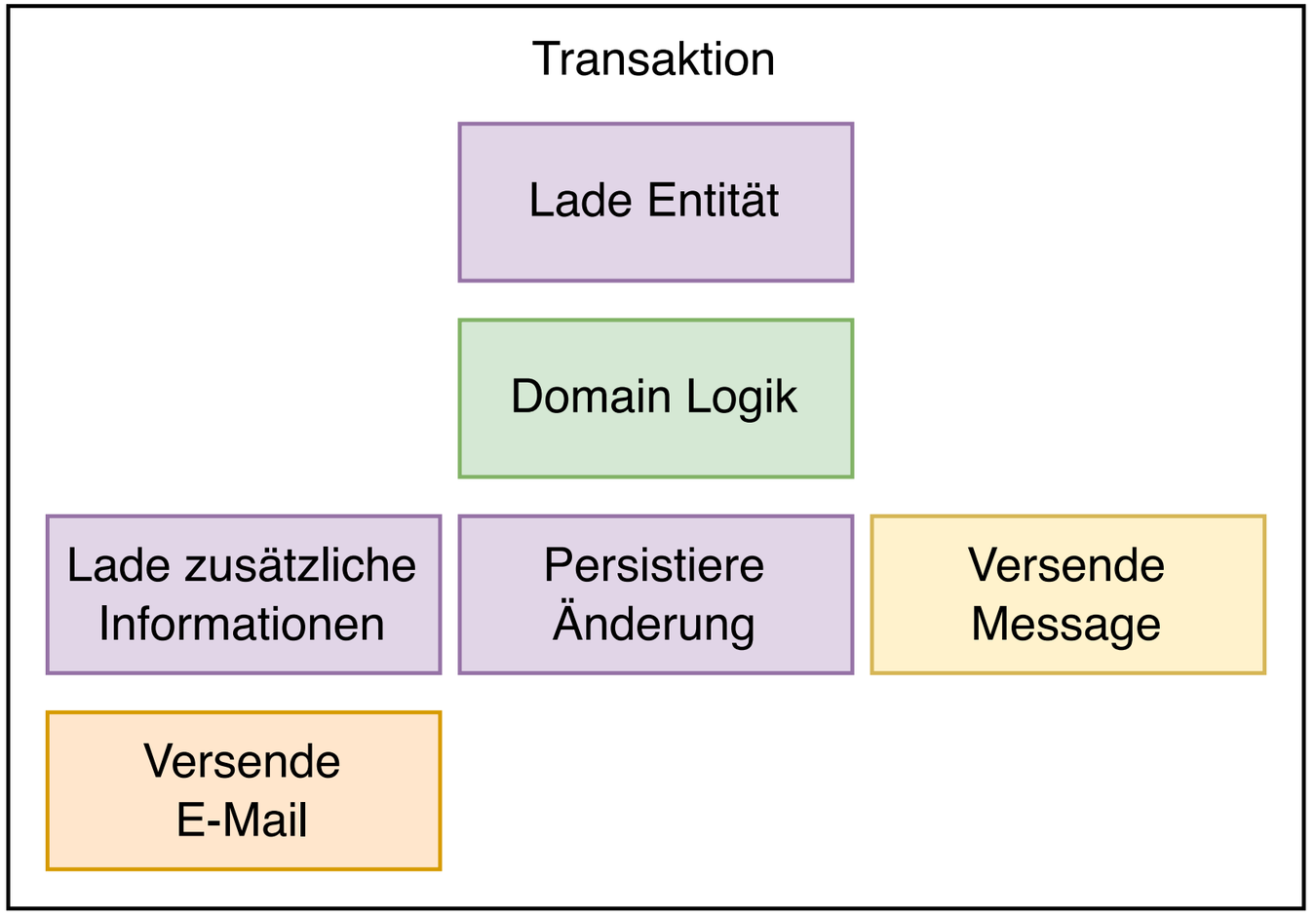
Laufzeit / Transaktionskopplung



Laufzeit / Transaktionskopplung



Laufzeit / Transaktionskopplung

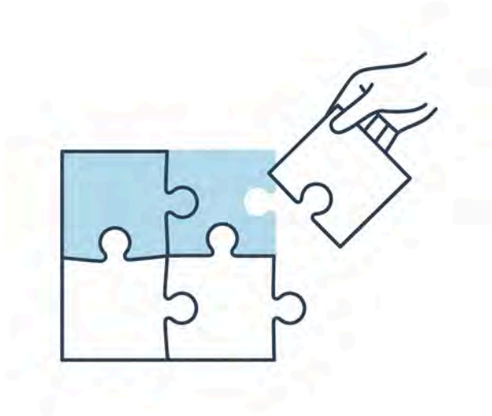


Wer hat schonmal eine E-Mail zurückgerufen?

Ausprägungen von Kopplung

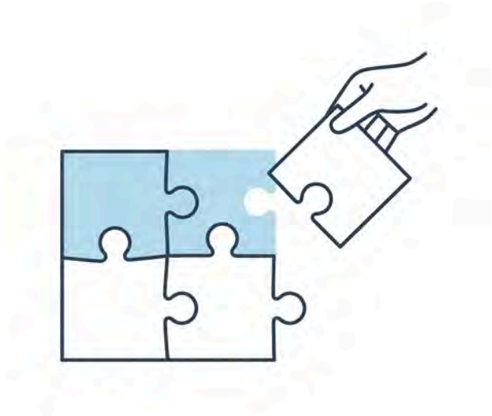
Ausprägungen von Kopplung

Interface / Implementierung



Ausprägungen von Kopplung

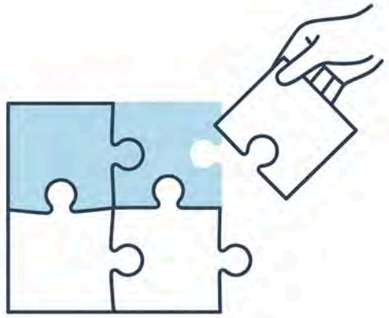
Interface / Implementierung



Datenmodell

Ausprägungen von Kopplung

Interface / Implementierung

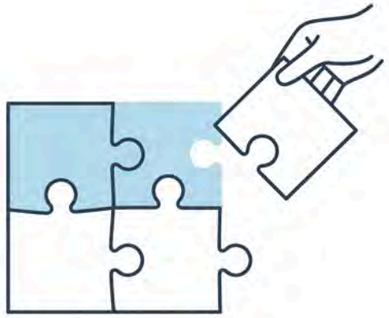


Datenmodell

Laufzeit / Transaktion

Ausprägungen von Kopplung

Interface / Implementierung



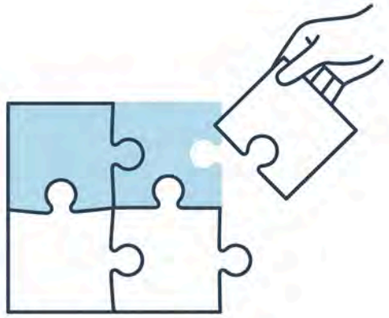
🟡 / ✅

Datenmodell

Laufzeit / Transaktion

Ausprägungen von Kopplung

Interface / Implementierung



🟡 / ✅

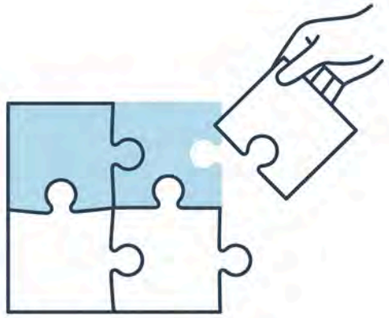
Datenmodell

⛔

Laufzeit / Transaktion

Ausprägungen von Kopplung

Interface / Implementierung



🟡 / ✅

Datenmodell

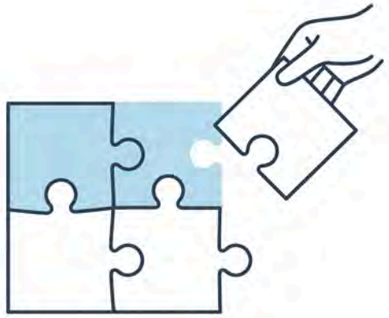
⛔

Laufzeit / Transaktion

⛔

Ausprägungen von Kopplung

Interface / Implementierung



🟡 / ✅

Datenmodell

⛔

Laufzeit / Transaktion

⛔

Dependency Injection ist keine Silver Bullet

Die reine Fachlogik

```java
public record Media(
        Long id,
        String isbn,
        String author,
        String title,
        MediaType type,
        boolean available,
        Long lentTo,
        LocalDate expectedReturnDate
) {}

@Service
public class MediaService {

    public Media borrowMedia(Media media, Long readerId) {
        if (!media.available()) {
            throw new MediaNotAvailableException("media not available because of reasons, id: " + media.i
        }

        if (media.lentTo() != null) {
            throw new MediaBorrowedException("media already borrowed, id: " + media.id);
        }
```

Die reine Fachlogik

```java
public record Media(
        Long id,
        String isbn,
        String author,
        String title,
        MediaType type,
        boolean available,
        Long lentTo,
        LocalDate expectedReturnDate
) {}

@Service
public class MediaService {

    public Media borrowMedia(Media media, Long readerId) {
        if (!media.available()) {
            throw new MediaNotAvailableException("media not available because of reasons, id: " + media.i
        }

        if (media.lentTo() != null) {
            throw new MediaBorrowedException("media already borrowed, id: " + media.id);
        }
```

Die reine Fachlogik

```
6           MediaType type,
7           boolean available,
8           Long lentTo,
9           LocalDate expectedReturnDate
10  ) {}
11
12  @Service
13  public class MediaService {
14
15      public Media borrowMedia(Media media, Long readerId) {
16          if (!media.available()) {
17              throw new MediaNotAvailableException("media not available because of reasons, id: " + media.i
18          }
19
20          if (media.lentTo() != null) {
21              throw new MediaBorrowedException("media already borrowed, id: " + media.id);
22          }
23
24          LocalDate expectedReturnDate = switch (media.type) {
25              case BOOK -> LocalDate.now().plusDays(28);
26              case MAGAZINE -> LocalDate.now().plusDays(7);
27          };
28
```

Die reine Fachlogik

```
10 ) {}
11
12 @Service
13 public class MediaService {
14
15     public Media borrowMedia(Media media, Long readerId) {
16         if (!media.available()) {
17             throw new MediaNotAvailableException("media not available because of reasons, id: " + media.i
18         }
19
20         if (media.lentTo() != null) {
21             throw new MediaBorrowedException("media already borrowed, id: " + media.id);
22         }
23
24         LocalDate expectedReturnDate = switch (media.type) {
25             case BOOK -> LocalDate.now().plusDays(28);
26             case MAGAZINE -> LocalDate.now().plusDays(7);
27         };
28
29         return media.withLending(readerId, expectedReturnDate);
30     }
31 }
```

Die reine Fachlogik

```
10 ) {}
11
12 @Service
13 public class MediaService {
14
15     public Media borrowMedia(Media media, Long readerId) {
16         if (!media.available()) {
17             throw new MediaNotAvailableException("media not available because of reasons, id: " + media.i
18         }
19
20         if (media.lentTo() != null) {
21             throw new MediaBorrowedException("media already borrowed, id: " + media.id);
22         }
23
24         LocalDate expectedReturnDate = switch (media.type) {
25             case BOOK -> LocalDate.now().plusDays(28);
26             case MAGAZINE -> LocalDate.now().plusDays(7);
27         };
28
29         return media.withLending(readerId, expectedReturnDate);
30     }
31 }
```

Die reine Fachlogik

```
10 ) {}
11
12 @Service
13 public class MediaService {
14
15     public Media borrowMedia(Media media, Long readerId) {
16         if (!media.available()) {
17             throw new MediaNotAvailableException("media not available because of reasons, id: " + media.i
18         }
19
20         if (media.lentTo() != null) {
21             throw new MediaBorrowedException("media already borrowed, id: " + media.id);
22         }
23
24         LocalDate expectedReturnDate = switch (media.type) {
25             case BOOK -> LocalDate.now().plusDays(28);
26             case MAGAZINE -> LocalDate.now().plusDays(7);
27         };
28
29         return media.withLending(readerId, expectedReturnDate);
30     }
31 }
```

Die reine Fachlogik

```java
public record Media(
        Long id,
        String isbn,
        String author,
        String title,
        MediaType type,
        boolean available,
        Long lentTo,
        LocalDate expectedReturnDate
) {}

@Service
public class MediaService {

    public Media borrowMedia(Media media, Long readerId) {
        if (!media.available()) {
            throw new MediaNotAvailableException("media not available because of reasons, id: " + media.i
        }

        if (media.lentTo() != null) {
            throw new MediaBorrowedException("media already borrowed, id: " + media.id);
        }
```

Und wer versendet jetzt die E-Mail?

Die Änderung

Die Änderung

Media (vor dem Ausleihvorgang)

```
{
  "id": 42,
  "isbn": "DE13 37 42 21",
  "author": "J.R.R. Tolkien",
  "title": "LOTR",
  "type": "BOOK",
  "readerId": null,
  "expectedReturnDate": null
}
```

Die Änderung

Media (vor dem Ausleihvorgang)

```
{
  "id": 42,
  "isbn": "DE13 37 42 21",
  "author": "J.R.R. Tolkien",
  "title": "LOTR",
  "type": "BOOK",
  "readerId": null,
  "expectedReturnDate": null
}
```

Media (nach dem Ausleihvorgang)

```
{
  "id": 42,
  "isbn": "DE13 37 42 21",
  "author": "J.R.R. Tolkien",
  "title": "LOTR",
  "type": "BOOK",
  "readerId": 21,
  "expectedReturnDate": "2025-07-28"
}
```

Die Änderung als First Class Citizen

```java
public record Media(
            Long id,
            String isbn,
            String author,
            String title,
            MediaType type,
            boolean available,
            Long lentTo,
            LocalDate expectedReturnDate
    ) {
        public Media withLending(Long readerId, LocalDate expectedReturnDate) {
            return new Media(
                    this.id,
                    this.isbn,
                    this.author,
                    this.title,
                    this.type,
                    false,
                    readerId,
                    expectedReturnDate
            );
        }
    }

```

Die Änderung als First Class Citizen

```
35      public record MediaLentEvent(
36              Long mediaId,
37              Long readerId,
38              LocalDate expectedReturnDate
39      ) {}
40
41  @CommandHandlerConfiguration
42  public class MediaLending {
43
44      @CommandHandling
45      public void handle(
46              Media media,
47              BorrowMediaCommand command,
48              CommandEventPublisher<Media> eventPublisher
49      ) {
50          if (!media.available()) {
51              throw new MediaNotAvailableException("media not available because of reasons, id: " + media.i
52          }
53
54          if (media.lentTo() != null) {
55              throw new MediaBorrowedException("media already borrowed, id: " + media.id());
56          }
57
58          LocalDate expectedReturnDate = switch (media.type()) {
59              case BOOK -> LocalDate.now().plusDays(28);
```

Die Änderung als First Class Citizen

```
15                      this.author,
16                      this.title,
17                      this.type,
18                      false,
19                      readerId,
20                      expectedReturnDate
21              );
22          }
23      }
24
25      public record BorrowMediaCommand(
26              Long mediaId,
27              Long readerId
28      ) implements Command {
29          @Override
30          public String getSubject() {
31              return "/media/"+mediaId;
32          }
33      }
34
35      public record MediaLentEvent(
36              Long mediaId,
37              Long readerId,
38              LocalDate expectedReturnDate
```

Die Änderung als First Class Citizen

```
44      @CommandHandling
45      public void handle(
46              Media media,
47              BorrowMediaCommand command,
48              CommandEventPublisher<Media> eventPublisher
49      ) {
50          if (!media.available()) {
51              throw new MediaNotAvailableException("media not available because of reasons, id: " + media.i
52          }
53
54          if (media.lentTo() != null) {
55              throw new MediaBorrowedException("media already borrowed, id: " + media.id());
56          }
57
58          LocalDate expectedReturnDate = switch (media.type()) {
59              case BOOK -> LocalDate.now().plusDays(28);
60              case MAGAZINE -> LocalDate.now().plusDays(7);
61          };
62
63          eventPublisher.publish(
64                  new MediaLentEvent(
65                          media.id(),
66                          command.readerId(),
67                          expectedReturnDate
```

Die Änderung als First Class Citizen

```
54          if (media.lentTo() != null) {
55              throw new MediaBorrowedException("media already borrowed, id: " + media.id());
56          }
57
58          LocalDate expectedReturnDate = switch (media.type()) {
59              case BOOK -> LocalDate.now().plusDays(28);
60              case MAGAZINE -> LocalDate.now().plusDays(7);
61          };
62
63          eventPublisher.publish(
64                  new MediaLentEvent(
65                          media.id(),
66                          command.readerId(),
67                          expectedReturnDate
68                  )
69          );
70      }
71
72      @StateRebuilding
73      public Media on(
74              Media media,
75              MediaLentEvent event
76      ) {
77          return media.withLending(event.readerId(), event.expectedReturnDate());
78      }
```

Die Änderung als First Class Citizen

```
56          }
57
58          LocalDate expectedReturnDate = switch (media.type()) {
59              case BOOK -> LocalDate.now().plusDays(28);
60              case MAGAZINE -> LocalDate.now().plusDays(7);
61          };
62
63          eventPublisher.publish(
64                  new MediaLentEvent(
65                          media.id(),
66                          command.readerId(),
67                          expectedReturnDate
68                  )
69          );
70      }
71
72      @StateRebuilding
73      public Media on(
74              Media media,
75              MediaLentEvent event
76      ) {
77          return media.withLending(event.readerId(), event.expectedReturnDate());
78      }
79  }
```

Und wie versenden wir jetzt die E-Mail?

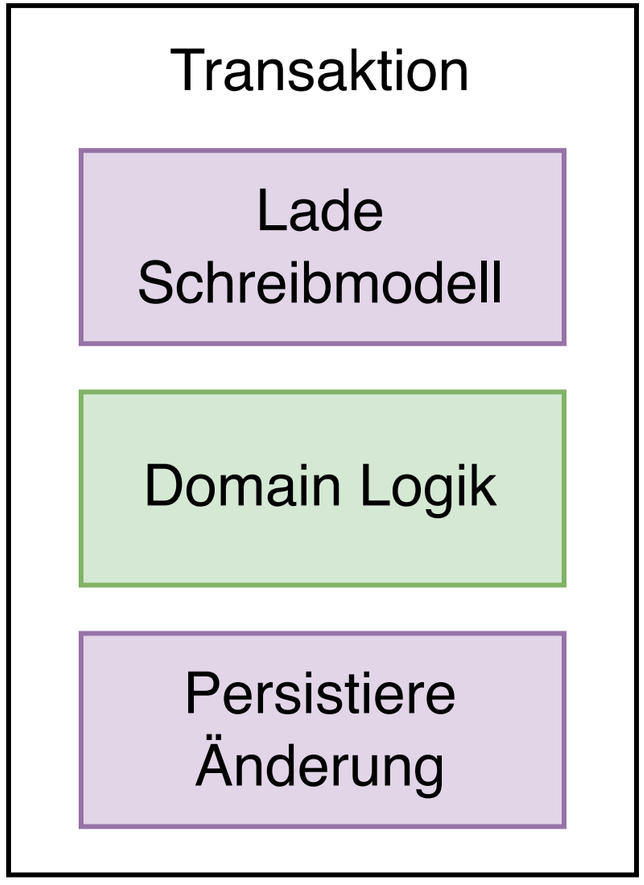
Und wie versenden wir jetzt die E-Mail?

```java
@Component
public class EmailHandler {

    private final ReaderProjection readerProjection;
    private final MediaProjection mediaProjection;
    private final MailApi mailApi;

    public EmailHandler(ReaderProjection readerProjection, MediaProjection mediaProjection, MailApi mailA
        this.readerProjection = readerProjection;
        this.mediaProjection = mediaProjection;
        this.mailApi = mailApi;
    }

    @EventHandling
    public void on(MediaLentEvent event) {
        String emailAddress = readerProjection.lookupEmailAddress(event.readerId());
        MediaDetails mediaDetails = mediaProjection.getMediaDetails(event.mediaId());

        mailApi.publishMail(
                new MailTemplate.MediaLent(
                        mediaDetails.author(),
                        mediaDetails.title(),
                        event.expectedReturnDate()
                ),
```

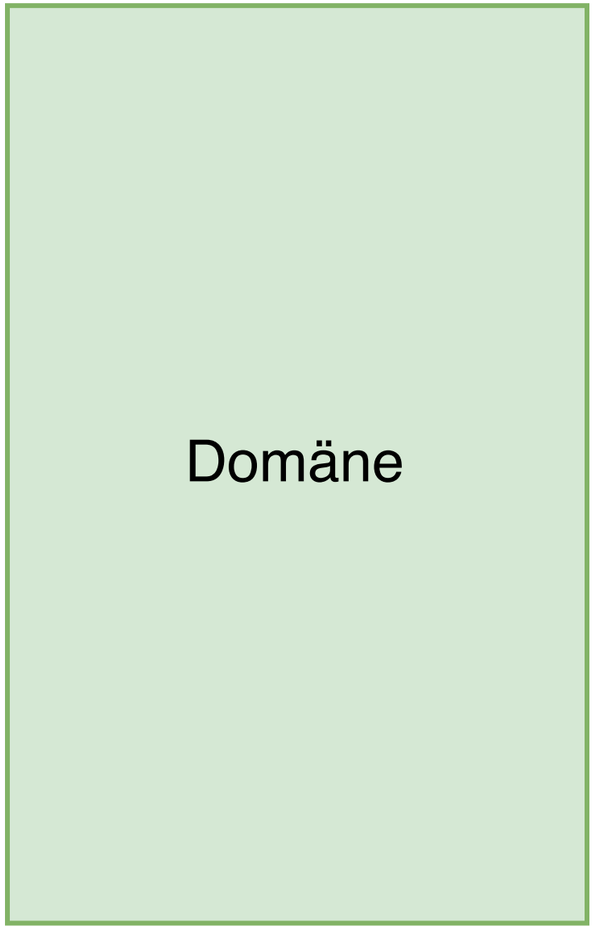
Und wie versenden wir jetzt die E-Mail?

```java
@Component
public class EmailHandler {

    private final ReaderProjection readerProjection;
    private final MediaProjection mediaProjection;
    private final MailApi mailApi;

    public EmailHandler(ReaderProjection readerProjection, MediaProjection mediaProjection, MailApi mailA
        this.readerProjection = readerProjection;
        this.mediaProjection = mediaProjection;
        this.mailApi = mailApi;
    }

    @EventHandling
    public void on(MediaLentEvent event) {
        String emailAddress = readerProjection.lookupEmailAddress(event.readerId());
        MediaDetails mediaDetails = mediaProjection.getMediaDetails(event.mediaId());

        mailApi.publishMail(
                new MailTemplate.MediaLent(
                        mediaDetails.author(),
                        mediaDetails.title(),
                        event.expectedReturnDate()
                ),
```

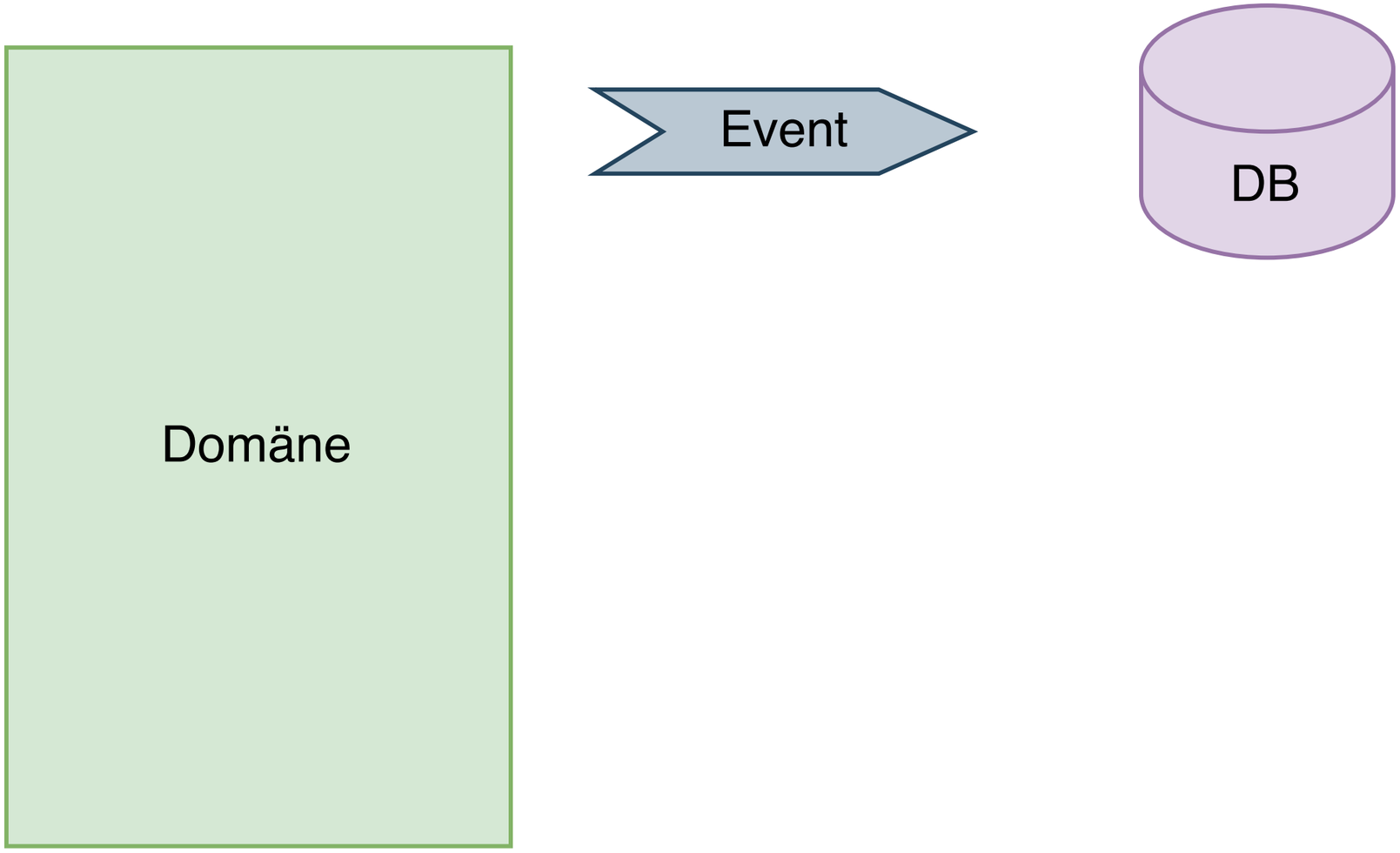
Die minimale Transaktion



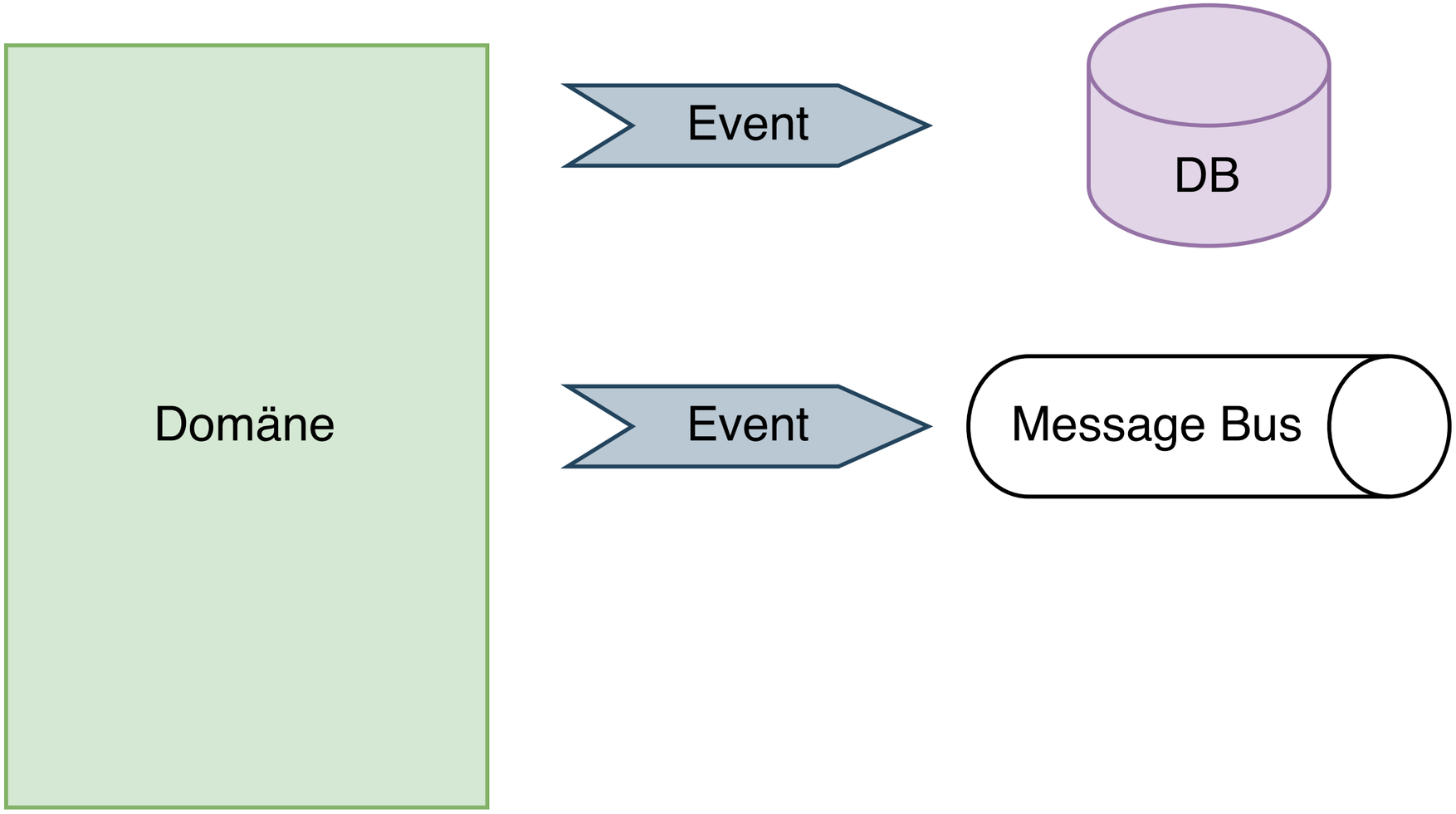
Lesemodelle



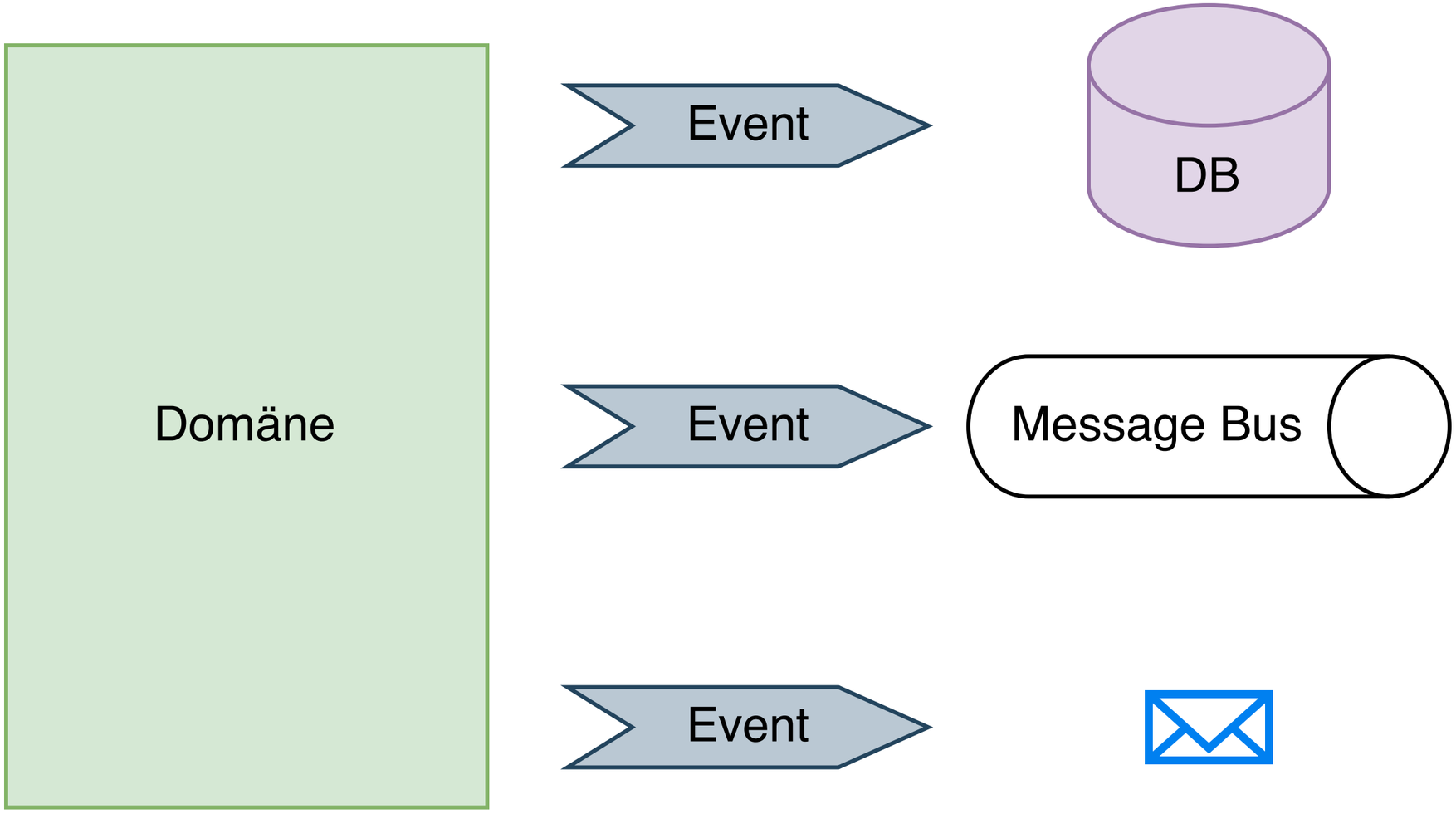
Lesemodelle



Lesemodelle



Lesemodelle



Command Query Responsibility Segregation (CQRS)

Command Query Responsibility Segregation (CQRS)

Schreibmodell (Commands)

- Führt Business-Logik aus
- Erzeugt Events
- Ändert den Zustand, durch Events

Command Query Responsibility Segregation (CQRS)

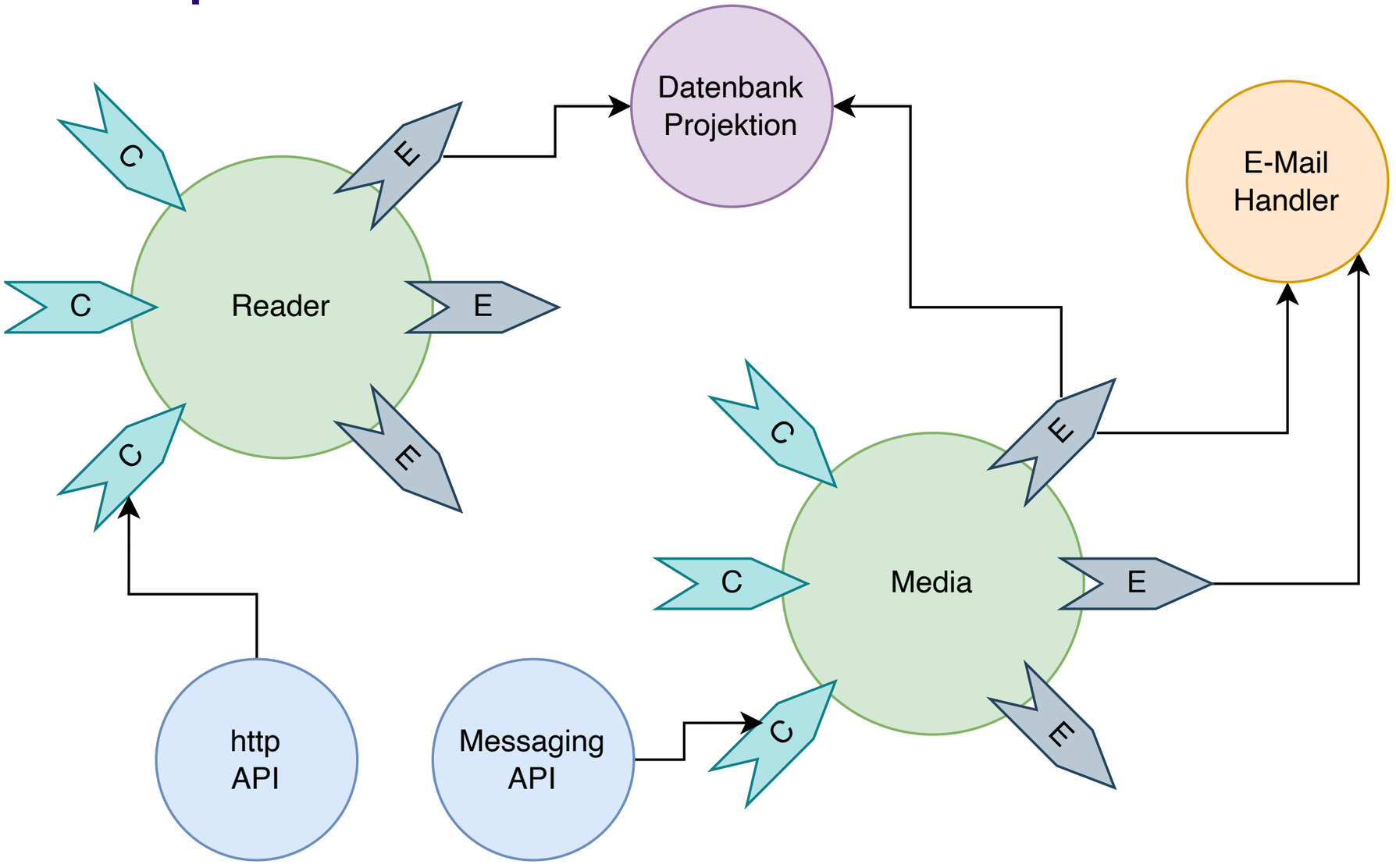
Schreibmodell (Commands)

- Führt Business-Logik aus
- Erzeugt Events
- Ändert den Zustand, durch Events

Lesemodelle (Queries)

- Konsumieren Events, um Seiteneffekte auszulösen
- Nutzen projizierten Zustand
- Optimiert für einen Anwendungsfall

Ports and Adapters mal anders



Dependency Injection

Dependency Injection

⛔

- Über Modulgrenzen hinweg

Dependency Injection

- Über Modulgrenzen hinweg
- Verteilte Transaktionen

Dependency Injection

- Über Modulgrenzen hinweg
- Verteilte Transaktionen
- Vermischung von Fachlogik mit Seiteneffekten

Dependency Injection

✅

- Entkopplung von Interface und Implementierung

⛔

- Über Modulgrenzen hinweg
- Verteilte Transaktionen
- Vermischung von Fachlogik mit Seiteneffekten

Dependency Injection

- Entkopplung von Interface und Implementierung
- Abstraktion von 3rd Party Services

- Über Modulgrenzen hinweg
- Verteilte Transaktionen
- Vermischung von Fachlogik mit Seiteneffekten

Dependency Injection

- Entkopplung von Interface und Implementierung
- Abstraktion von 3rd Party Services
- Innerhalb von Modulen

- Über Modulgrenzen hinweg
- Verteilte Transaktionen
- Vermischung von Fachlogik mit Seiteneffekten

Dependency Injection

✅

- Entkopplung von Interface und Implementierung
- Abstraktion von 3rd Party Services
- Innerhalb von Modulen

⛔

- Über Modulgrenzen hinweg
- Verteilte Transaktionen
- Vermischung von Fachlogik mit Seiteneffekten

CQRS gehört in jeden Entwicklerwerkzeugkasten

Wofür verwendet ihr Dependency Injection?

Jenseits von Dependency Injection

Aufgeräumte Fachlogik ohne Seiteneffekte