Raus aus der Mocking-Hölle

Architektur-Refactoring für bessere Unit-Tests

Frank Scheffler
Senior Solution Architect

frank.scheffler@digitalfrontiers.de
github.com/maverick1601

Testgetriebene Entwicklung in der Theorie

```java
public class Calculator {

  public double divide(double dividend, double divisor) {
    if (divisor == 0.0) {
      throw new IllegalArgumentException("divide by zero");
    }
    return dividend / divisor;
  }
}
```

Testgetriebene Entwicklung in der Theorie

```java
public class CalculatorTest {

  private final Calculator calculator = new Calculator();

  @ParameterizedTest
  @CsvSource({
    "4.3, 2, 2.15",
    "-8, 8, -1",
    "-1, -1, 1.0",
  })
  public void divideByNonZero(double dividend, double divisor, double quotient) {
    var result = calculator.divide(dividend, divisor);

    assertThat(result).isEqualTo(quotient);
  }

  @Test
  public void divideByZero() {
    assertThatThrownBy(() -> calculator.divide(5.0, 0))
      .isInstanceOf(IllegalArgumentException.class);
  }
}
```

Testgetriebene Entwicklung in der Praxis

```
30      bookCopy = new BookCopy();
31      bookCopy.setBookCopyId(UUID.randomUUID());
32      book.getCopies().put(bookCopy.getBookCopyId(), bookCopy);
33    }
34
35    @Test
36    public void lentSuccessfully() {
37      when(bookRepository.findByIsbnAndCopies_BookCopyIdAndCopies_AvailableTrue(
38        book.getIsbn(),
39        bookCopy.getBookCopyId())
40      ).thenReturn(Optional.of(book));
41
42      subject.lendBook(book.getIsbn(), bookCopy.getBookCopyId(), reader);
43
44      ArgumentCaptor<Book> bookCaptor = ArgumentCaptor.forClass(Book.class);
45      verify(bookRepository).save(bookCaptor.capture());
46
47      assertThat(bookCaptor.getValue().getCopies())
48        .hasEntrySatisfying(bookCopy.getBookCopyId(), bc -> assertThat(bc.isAvailable()).isFalse());
49
50      ArgumentCaptor<MailTemplate> mailTemplateCaptor = ArgumentCaptor.forClass(MailTemplate.class);
51      verify(emailService).sendMailNotification(
52        eq(reader.email()),
53        eq("Book successfully rented"),
54        mailTemplateCaptor.capture()
```

Testgetriebene Entwicklung in der Praxis

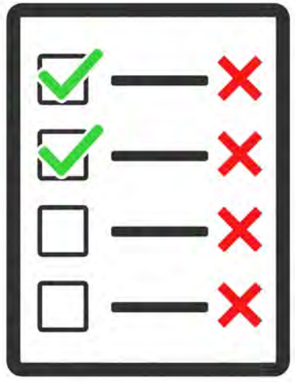
```
27      book.setAuthor("J.R.R. Tolkien");
28      book.setTitle("LOTR");
29
30      bookCopy = new BookCopy();
31      bookCopy.setBookCopyId(UUID.randomUUID());
32      book.getCopies().put(bookCopy.getBookCopyId(), bookCopy);
33    }
34
35    @Test
36    public void lentSuccessfully() {
37      when(bookRepository.findByIsbnAndCopies_BookCopyIdAndCopies_AvailableTrue(
38        book.getIsbn(),
39        bookCopy.getBookCopyId())
40      ).thenReturn(Optional.of(book));
41
42      subject.lendBook(book.getIsbn(), bookCopy.getBookCopyId(), reader);
43
44      ArgumentCaptor<Book> bookCaptor = ArgumentCaptor.forClass(Book.class);
45      verify(bookRepository).save(bookCaptor.capture());
46
47      assertThat(bookCaptor.getValue().getCopies())
48        .hasEntrySatisfying(bookCopy.getBookCopyId(), bc -> assertThat(bc.isAvailable()).isFalse());
49
50      ArgumentCaptor<MailTemplate> mailTemplateCaptor = ArgumentCaptor.forClass(MailTemplate.class);
```

Testgetriebene Entwicklung in der Praxis

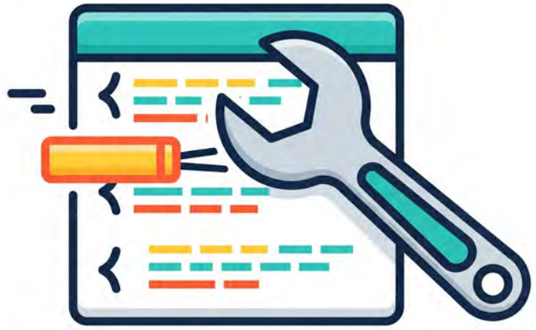
```
42      subject.lendBook(book.getIsbn(), bookCopy.getBookCopyId(), reader);
43
44      ArgumentCaptor<Book> bookCaptor = ArgumentCaptor.forClass(Book.class);
45      verify(bookRepository).save(bookCaptor.capture());
46
47      assertThat(bookCaptor.getValue().getCopies())
48        .hasEntrySatisfying(bookCopy.getBookCopyId(), bc -> assertThat(bc.isAvailable()).isFalse());
49
50      ArgumentCaptor<MailTemplate> mailTemplateCaptor = ArgumentCaptor.forClass(MailTemplate.class);
51      verify(emailService).sendMailNotification(
52        eq(reader.email()),
53        eq("Book successfully rented"),
54        mailTemplateCaptor.capture()
55      );
56
57      assertThat(mailTemplateCaptor.getValue())
58        .isInstanceOfSatisfying(MailTemplate.BookRented.class, bookRented -> {
59          assertThat(bookRented.book()).isSameAs(book);
60          assertThat(bookRented.dueAt()).isCloseTo(
61              Instant.now().plus(14, ChronoUnit.DAYS),
62              new TemporalUnitWithinOffset(5, ChronoUnit.SECONDS)
63          );
64        });
65    }
66  }
```

Wie wirkt sich das auf die Softwareentwicklung aus?

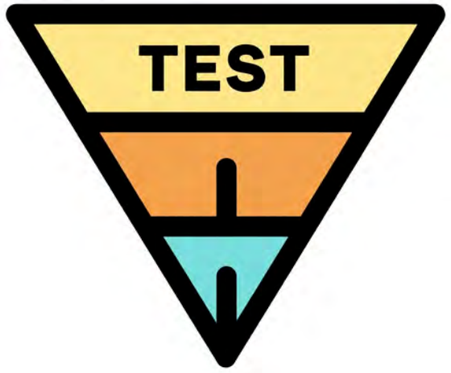
Abdeckung



Anpassbarkeit



Qualität



Warum landen wir eigentlich immer wieder in der Mocking-Hölle?

BookService

```java
@Service
public class BookService {

    private final BookRepository bookRepository;
    private final EmailService emailService;

    public BookService(BookRepository bookRepository, EmailService emailService) {
        this.bookRepository = bookRepository;
        this.emailService = emailService;
    }


    @Transactional
    public void lendBook(String isbn, UUID bookCopyId, Reader reader) {
        Book book = bookRepository
                .findByIsbnAndCopies_BookCopyIdAndCopies_AvailableTrue(isbn, bookCopyId)
                .orElseThrow(() -> new BookCopyNotAvailableException(bookCopyId));

        book.getCopies().get(bookCopyId).setAvailable(false);
        bookRepository.save(book);

        emailService.sendMailNotification(
                reader.email(),
                "Book successfully rented",
```

BookService

```java
    private final EmailService emailService;

    public BookService(BookRepository bookRepository, EmailService emailService) {
        this.bookRepository = bookRepository;
        this.emailService = emailService;
    }


    @Transactional
    public void lendBook(String isbn, UUID bookCopyId, Reader reader) {
        Book book = bookRepository
                .findByIsbnAndCopies_BookCopyIdAndCopies_AvailableTrue(isbn, bookCopyId)
                .orElseThrow(() -> new BookCopyNotAvailableException(bookCopyId));

        book.getCopies().get(bookCopyId).setAvailable(false);
        bookRepository.save(book);

        emailService.sendMailNotification(
                reader.email(),
                "Book successfully rented",
                new MailTemplate.BookRented(book, Instant.now().plus(14, ChronoUnit.DAYS))
        );
    }
}
```

BookService

```java
    private final EmailService emailService;

    public BookService(BookRepository bookRepository, EmailService emailService) {
        this.bookRepository = bookRepository;
        this.emailService = emailService;
    }


    @Transactional
    public void lendBook(String isbn, UUID bookCopyId, Reader reader) {
        Book book = bookRepository
                .findByIsbnAndCopies_BookCopyIdAndCopies_AvailableTrue(isbn, bookCopyId)
                .orElseThrow(() -> new BookCopyNotAvailableException(bookCopyId));

        book.getCopies().get(bookCopyId).setAvailable(false);
        bookRepository.save(book);

        emailService.sendMailNotification(
                reader.email(),
                "Book successfully rented",
                new MailTemplate.BookRented(book, Instant.now().plus(14, ChronoUnit.DAYS))
        );
    }
}
```

BookService

```
 5      private final EmailService emailService;
 6
 7      public BookService(BookRepository bookRepository, EmailService emailService) {
 8          this.bookRepository = bookRepository;
 9          this.emailService = emailService;
10      }
11
12
13      @Transactional
14      public void lendBook(String isbn, UUID bookCopyId, Reader reader) {
15          Book book = bookRepository
16                  .findByIsbnAndCopies_BookCopyIdAndCopies_AvailableTrue(isbn, bookCopyId)
17                  .orElseThrow(() -> new BookCopyNotAvailableException(bookCopyId));
18
19          book.getCopies().get(bookCopyId).setAvailable(false);
20          bookRepository.save(book);
21
22          emailService.sendMailNotification(
23                  reader.email(),
24                  "Book successfully rented",
25                  new MailTemplate.BookRented(book, Instant.now().plus(14, ChronoUnit.DAYS))
26          );
27      }
28  }
```

BookServiceTest

```java
@ExtendWith(MockitoExtension.class)
public class BookServiceTest {

  @Mock
  BookRepository bookRepository;

  @Mock
  EmailService emailService;

  @InjectMocks
  BookService subject;

  Reader reader;
  Book book;
  BookCopy bookCopy;

  @BeforeEach
  public void setup() {
    reader = new Reader(
      UUID.randomUUID(),
      "Ted Tester",
      "ted@test.com"
    );
```

BookServiceTest

```
11   BookService subject;
12
13   Reader reader;
14   Book book;
15   BookCopy bookCopy;
16
17   @BeforeEach
18   public void setup() {
19     reader = new Reader(
20       UUID.randomUUID(),
21       "Ted Tester",
22       "ted@test.com"
23     );
24
25     book = new Book();
26     book.setIsbn("978-3-16-148410-0");
27     book.setAuthor("J.R.R. Tolkien");
28     book.setTitle("LOTR");
29
30     bookCopy = new BookCopy();
31     bookCopy.setBookCopyId(UUID.randomUUID());
32     book.getCopies().put(bookCopy.getBookCopyId(), bookCopy);
33   }
34
35   @Test
```

BookServiceTest

```
27      book.setAuthor("J.R.R. Tolkien");
28      book.setTitle("LOTR");
29
30      bookCopy = new BookCopy();
31      bookCopy.setBookCopyId(UUID.randomUUID());
32      book.getCopies().put(bookCopy.getBookCopyId(), bookCopy);
33    }
34
35    @Test
36    public void rentedSuccessfully() {
37      when(bookRepository.findByIsbnAndCopies_BookCopyIdAndCopies_AvailableTrue(
38        book.getIsbn(),
39        bookCopy.getBookCopyId())
40      ).thenReturn(Optional.of(book));
41
42      subject.lendBook(book.getIsbn(), bookCopy.getBookCopyId(), reader);
43
44      ArgumentCaptor<Book> bookCaptor = ArgumentCaptor.forClass(Book.class);
45      verify(bookRepository).save(bookCaptor.capture());
46
47      assertThat(bookCaptor.getValue().getCopies())
48        .hasEntrySatisfying(bookCopy.getBookCopyId(), bc -> assertThat(bc.isAvailable()).isFalse());
49
50      ArgumentCaptor<MailTemplate> mailTemplateCaptor = ArgumentCaptor.forClass(MailTemplate.class);
```

BookServiceTest

```
30      bookCopy = new BookCopy();
31      bookCopy.setBookCopyId(UUID.randomUUID());
32      book.getCopies().put(bookCopy.getBookCopyId(), bookCopy);
33    }
34
35    @Test
36    public void rentedSuccessfully() {
37      when(bookRepository.findByIsbnAndCopies_BookCopyIdAndCopies_AvailableTrue(
38        book.getIsbn(),
39        bookCopy.getBookCopyId())
40      ).thenReturn(Optional.of(book));
41
42      subject.lendBook(book.getIsbn(), bookCopy.getBookCopyId(), reader);
43
44      ArgumentCaptor<Book> bookCaptor = ArgumentCaptor.forClass(Book.class);
45      verify(bookRepository).save(bookCaptor.capture());
46
47      assertThat(bookCaptor.getValue().getCopies())
48        .hasEntrySatisfying(bookCopy.getBookCopyId(), bc -> assertThat(bc.isAvailable()).isFalse());
49
50      ArgumentCaptor<MailTemplate> mailTemplateCaptor = ArgumentCaptor.forClass(MailTemplate.class);
51      verify(emailService).sendMailNotification(
52        eq(reader.email()),
53        eq("Book successfully rented"),
54        mailTemplateCaptor.capture()
```

BookServiceTest

```
34
35    @Test
36    public void rentedSuccessfully() {
37      when(bookRepository.findByIsbnAndCopies_BookCopyIdAndCopies_AvailableTrue(
38        book.getIsbn(),
39        bookCopy.getBookCopyId())
40      ).thenReturn(Optional.of(book));
41
42      subject.lendBook(book.getIsbn(), bookCopy.getBookCopyId(), reader);
43
44      ArgumentCaptor<Book> bookCaptor = ArgumentCaptor.forClass(Book.class);
45      verify(bookRepository).save(bookCaptor.capture());
46
47      assertThat(bookCaptor.getValue().getCopies())
48        .hasEntrySatisfying(bookCopy.getBookCopyId(), bc -> assertThat(bc.isAvailable()).isFalse());
49
50      ArgumentCaptor<MailTemplate> mailTemplateCaptor = ArgumentCaptor.forClass(MailTemplate.class);
51      verify(emailService).sendMailNotification(
52        eq(reader.email()),
53        eq("Book successfully rented"),
54        mailTemplateCaptor.capture()
55      );
56
57      assertThat(mailTemplateCaptor.getValue())
58        .isInstanceOfSatisfying(MailTemplate.BookRented.class, bookRented -> {
```

BookServiceTest

```
43
44      ArgumentCaptor<Book> bookCaptor = ArgumentCaptor.forClass(Book.class);
45      verify(bookRepository).save(bookCaptor.capture());
46
47      assertThat(bookCaptor.getValue().getCopies())
48        .hasEntrySatisfying(bookCopy.getBookCopyId(), bc -> assertThat(bc.isAvailable()).isFalse());
49
50      ArgumentCaptor<MailTemplate> mailTemplateCaptor = ArgumentCaptor.forClass(MailTemplate.class);
51      verify(emailService).sendMailNotification(
52        eq(reader.email()),
53        eq("Book successfully rented"),
54        mailTemplateCaptor.capture()
55      );
56
57      assertThat(mailTemplateCaptor.getValue())
58        .isInstanceOfSatisfying(MailTemplate.BookRented.class, bookRented -> {
59          assertThat(bookRented.book()).isSameAs(book);
60          assertThat(bookRented.dueAt()).isCloseTo(
61              Instant.now().plus(14, ChronoUnit.DAYS),
62              new TemporalUnitWithinOffset(5, ChronoUnit.SECONDS)
63          );
64        });
65    }
66  }
```

Ist unser Test-Code eventuell gar nicht das ursächliche Problem?

Was benötigen wir eigentlich wirklich im Produktiv-Code?

```java
public record BookModel(
        String isbn,
        String author,
        String title,
        Map<UUID, Boolean> copies
) {
    public BookModel withCopyAvailability(UUID bookCopyId, Boolean availability) {
    // ...
    }
}

public record LendBookCopyCommand(
        String isbn,
        UUID bookCopyId,
        Reader reader
) {}

public BookModel lend(BookModel bookModel, LendBookCopyCommand command) {
        Boolean availability = bookModel.copies().get(command.bookCopyId());

        if (availability == null) {
            throw new BookCopyNotFoundException(command.bookCopyId());
        }

        if (
```

Was benötigen wir eigentlich wirklich im Produktiv-Code?

```
        String isbn,
        String author,
        String title,
        Map<UUID, Boolean> copies
) {
    public BookModel withCopyAvailability(UUID bookCopyId, Boolean availability) {
    // ...
    }
}

public record LendBookCopyCommand(
        String isbn,
        UUID bookCopyId,
        Reader reader
) {}

public BookModel lend(BookModel bookModel, LendBookCopyCommand command) {
        Boolean availability = bookModel.copies().get(command.bookCopyId());

        if (availability == null) {
            throw new BookCopyNotFoundException(command.bookCopyId());
        }

        if (!availability) {
            throw new BookCopyNotAvailableException(command.bookCopyId());
```

Was benötigen wir eigentlich wirklich im Produktiv-Code?

```
    public BookModel withCopyAvailability(UUID bookCopyId, Boolean availability) {
    // ...
    }
}

public record LendBookCopyCommand(
        String isbn,
        UUID bookCopyId,
        Reader reader
) {}

public BookModel lend(BookModel bookModel, LendBookCopyCommand command) {
        Boolean availability = bookModel.copies().get(command.bookCopyId());

        if (availability == null) {
            throw new BookCopyNotFoundException(command.bookCopyId());
        }

        if (!availability) {
            throw new BookCopyNotAvailableException(command.bookCopyId());
        }

        return bookModel.withCopyAvailability(command.bookCopyId(), false);
}
```

Was benötigen wir eigentlich wirklich im Produktiv-Code?

```
    public BookModel withCopyAvailability(UUID bookCopyId, Boolean availability) {
    // ...
    }
}

public record LendBookCopyCommand(
        String isbn,
        UUID bookCopyId,
        Reader reader
) {}

public BookModel lend(BookModel bookModel, LendBookCopyCommand command) {
        Boolean availability = bookModel.copies().get(command.bookCopyId());

        if (availability == null) {
            throw new BookCopyNotFoundException(command.bookCopyId());
        }

        if (!availability) {
            throw new BookCopyNotAvailableException(command.bookCopyId());
        }

        return bookModel.withCopyAvailability(command.bookCopyId(), false);
}
```

Was benötigen wir eigentlich wirklich im Produktiv-Code?

```
    public BookModel withCopyAvailability(UUID bookCopyId, Boolean availability) {
    // ...
    }
}

public record LendBookCopyCommand(
        String isbn,
        UUID bookCopyId,
        Reader reader
) {}

public BookModel lend(BookModel bookModel, LendBookCopyCommand command) {
        Boolean availability = bookModel.copies().get(command.bookCopyId());

        if (availability == null) {
            throw new BookCopyNotFoundException(command.bookCopyId());
        }

        if (!availability) {
            throw new BookCopyNotAvailableException(command.bookCopyId());
        }

        return bookModel.withCopyAvailability(command.bookCopyId(), false);
}
```

BookLendingTest

```
public class BookLendingTest {
    BookLending subject = new BookLending();

    Reader reader = new Reader(
            UUID.randomUUID(),
            "Ted Tester",
            "tester@ted.com"
    );

    @Test
    public void lentSuccessfully() {
        var bookCopyId = UUID.randomUUID();
        var bookModel = new BookModel(
                "978-0008471286",
                "J.R.R. Tolkien",
                "Lord of the Rings",
                Map.of(bookCopyId, true)
        );
        var command = new LendBookCopyCommand("978-0008471286", bookCopyId, reader);

        var updatedModel = subject.lend(bookModel, command);

        assertThat(updatedModel.copies().get(bookCopyId)).isFalse();
    }
}
```

BookLendingTest

```
2     BookLending subject = new BookLending();
3
4     Reader reader = new Reader(
5             UUID.randomUUID(),
6             "Ted Tester",
7             "tester@ted.com"
8     );
9
10    @Test
11    public void lentSuccessfully() {
12        var bookCopyId = UUID.randomUUID();
13        var bookModel = new BookModel(
14                "978-0008471286",
15                "J.R.R. Tolkien",
16                "Lord of the Rings",
17                Map.of(bookCopyId, true)
18        );
19        var command = new LendBookCopyCommand("978-0008471286", bookCopyId, reader);
20
21        var updatedModel = subject.lend(bookModel, command);
22
23        assertThat(updatedModel.copies().get(bookCopyId)).isFalse();
24    }
25 }
```

BookLendingTest

```
2     BookLending subject = new BookLending();
3
4     Reader reader = new Reader(
5             UUID.randomUUID(),
6             "Ted Tester",
7             "tester@ted.com"
8     );
9
10    @Test
11    public void lentSuccessfully() {
12        var bookCopyId = UUID.randomUUID();
13        var bookModel = new BookModel(
14                "978-0008471286",
15                "J.R.R. Tolkien",
16                "Lord of the Rings",
17                Map.of(bookCopyId, true)
18        );
19        var command = new LendBookCopyCommand("978-0008471286", bookCopyId, reader);
20
21        var updatedModel = subject.lend(bookModel, command);
22
23        assertThat(updatedModel.copies().get(bookCopyId)).isFalse();
24    }
25 }
```

Woher wissen wir jetzt, ob eine E-Mail verschickt werden soll?

BookModel (vor dem Ausleihvorgang)

```
{
  "isbn": "978-0008471286",
  "author": "J.R.R. Tolkien",
  "title": "Lord of the Rings",
  "copies": {
    "e8f8641f-9a49-4aab-8c45-8a3482727f09": true
  }
}
```

BookModel (nach dem Ausleihvorgang)

```
{
  "isbn": "978-0008471286",
  "author": "J.R.R. Tolkien",
  "title": "Lord of the Rings",
  "copies": {
    "e8f8641f-9a49-4aab-8c45-8a3482727f09": false
  }
}
```

Änderungen müssen sichtbar gemacht werden

```java
public class BookLending {

    @CommandHandling
    public void lend(
        BookModel bookModel,
        LendBookCopyCommand command,
        CommandEventPublisher<BookModel> publisher
    ) {
        Boolean availability = bookModel.copies().get(command.bookCopyId());

        if (availability == null) {
            throw new BookCopyNotFoundException(command.bookCopyId());
        }

        if (!availability) {
            throw new BookCopyNotAvailableException(command.bookCopyId());
        }

        publisher.publish(
                new BookCopyLentEvent(
                        command.isbn(),
                        command.bookCopyId(),
                        command.reader(),
                        Instant.now().plus(14, ChronoUnit.DAYS)
```

Änderungen müssen sichtbar gemacht werden

```
10
11          if (availability == null) {
12              throw new BookCopyNotFoundException(command.bookCopyId());
13          }
14
15          if (!availability) {
16              throw new BookCopyNotAvailableException(command.bookCopyId());
17          }
18
19          publisher.publish(
20                  new BookCopyLentEvent(
21                          command.isbn(),
22                          command.bookCopyId(),
23                          command.reader(),
24                          Instant.now().plus(14, ChronoUnit.DAYS)
25                  )
26          );
27      }
28
29      @StateRebuilding
30      public BookModel on(BookModel bookModel, BookCopyLentEvent event) {
31          return bookModel.withCopyAvailability(event.bookCopyId(), false);
32      }
33  }
```

BookLendingTest (Given-When-Then)

```java
public class BookLendingTest {

    Reader reader = new Reader(
            UUID.randomUUID(),
            "Ted Tester",
            "test@ted.com"
    );

    CommandHandlingTestFixture fixture;

    @Test
    public void lentSuccessfully() {
        var bookCopyId = UUID.randomUUID();
        fixture
                .given(new BookCatalogedEvent("978-0008471286", "J.R.R. Tolkien", "Lord of the Rings"))
                .andGiven(new BookCopyPurchasedEvent("978-0008471286", bookCopyId))

                .when(new LendBookCopyCommand("978-0008471286", bookCopyId, reader))

                .expectSingleEventType(BookCopyLentEvent.class);
    }
}
```

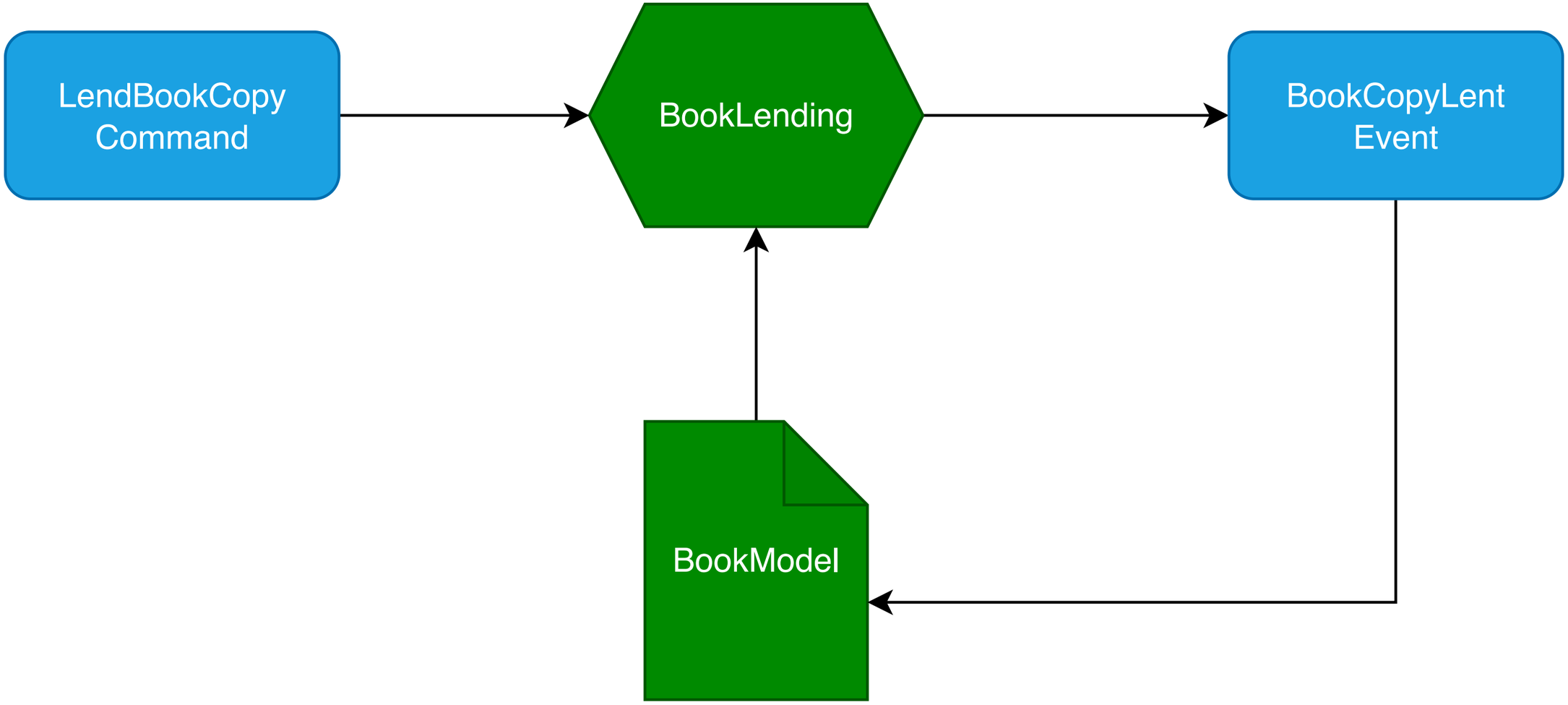
BookLendingTest (Given-When-Then)

```java
public class BookLendingTest {

    Reader reader = new Reader(
            UUID.randomUUID(),
            "Ted Tester",
            "test@ted.com"
    );

    CommandHandlingTestFixture fixture;

    @Test
    public void lentSuccessfully() {
        var bookCopyId = UUID.randomUUID();
        fixture
                .given(new BookCatalogedEvent("978-0008471286", "J.R.R. Tolkien", "Lord of the Rings"))
                .andGiven(new BookCopyPurchasedEvent("978-0008471286", bookCopyId))

                .when(new LendBookCopyCommand("978-0008471286", bookCopyId, reader))

                .expectSingleEventType(BookCopyLentEvent.class);
    }
}
```

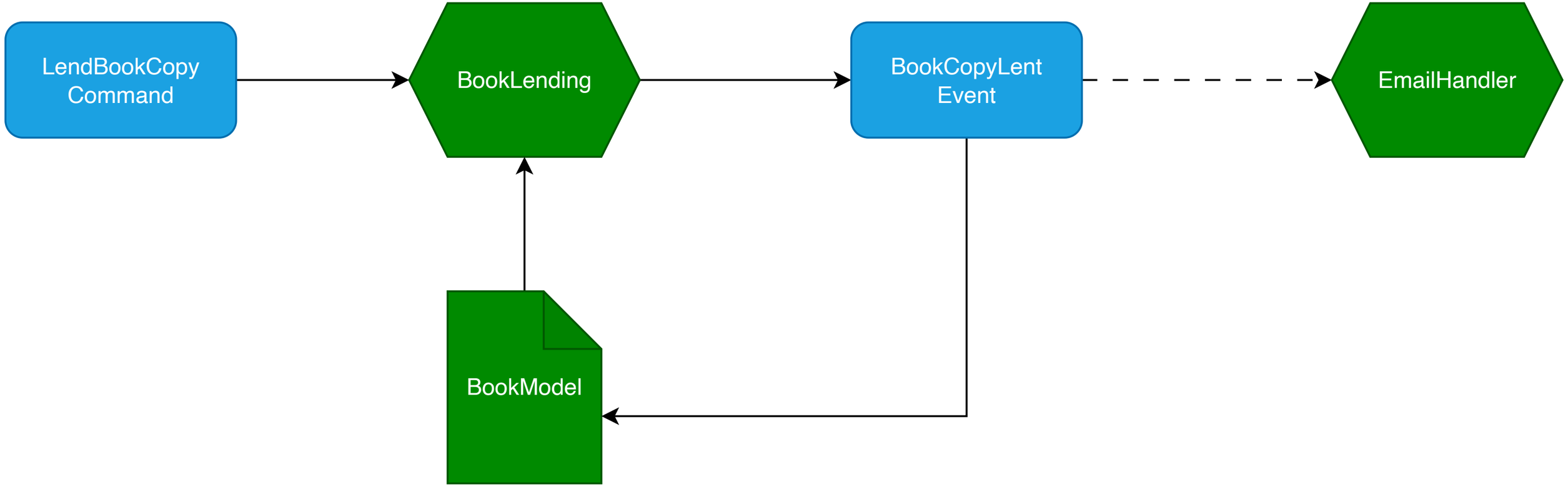
BookLendingTest (Given-When-Then)

```java
public class BookLendingTest {

    Reader reader = new Reader(
            UUID.randomUUID(),
            "Ted Tester",
            "test@ted.com"
    );

    CommandHandlingTestFixture fixture;

    @Test
    public void lentSuccessfully() {
        var bookCopyId = UUID.randomUUID();
        fixture
                .given(new BookCatalogedEvent("978-0008471286", "J.R.R. Tolkien", "Lord of the Rings"))
                .andGiven(new BookCopyPurchasedEvent("978-0008471286", bookCopyId))

                .when(new LendBookCopyCommand("978-0008471286", bookCopyId, reader))

                .expectSingleEventType(BookCopyLentEvent.class);
    }
}
```

Wer kümmert sich denn jetzt um den E-Mail Versand?



Events können beliebige Aktionen auslösen



EmailHandler

```java
@Component
public class EmailHandler {

    private final MailApi mailApi;

    public EmailHandler(MailApi mailApi) {
        this.mailApi = mailApi;
    }

    @EventHandling("mail")
    public void on(BookCopyLentEvent event) {
        mailApi.sendMail(
                "no-reply@tdd.com",
                event.reader().email(),
                "Buch erfolgreich ausgeliehen: " + event.isbn(),
                "Bitte beachten Sie die Rückgabe bis spätestens: " + event.dueAt()
        );
    }
}
```

EmailHandler

```java
@Component
public class EmailHandler {

    private final MailApi mailApi;

    public EmailHandler(MailApi mailApi) {
        this.mailApi = mailApi;
    }

    @EventHandling("mail")
    public void on(BookCopyLentEvent event) {
        mailApi.sendMail(
                "no-reply@tdd.com",
                event.reader().email(),
                "Buch erfolgreich ausgeliehen: " + event.isbn(),
                "Bitte beachten Sie die Rückgabe bis spätestens: " + event.dueAt()
        );
    }
}
```

EmailHandler

```java
@Component
public class EmailHandler {

    private final MailApi mailApi;

    public EmailHandler(MailApi mailApi) {
        this.mailApi = mailApi;
    }

    @EventHandling("mail")
    public void on(BookCopyLentEvent event) {
        mailApi.sendMail(
                "no-reply@tdd.com",
                event.reader().email(),
                "Buch erfolgreich ausgeliehen: " + event.isbn(),
                "Bitte beachten Sie die Rückgabe bis spätestens: " + event.dueAt()
        );
    }
}
```

EmailHandlerTest

```java
@ExtendWith(MockitoExtension.class)
public class EmailHandlerTest {

    @InjectMocks
    private EmailHandler subject;

    @Mock
    private MailApi mailApi;

    @Test
    public void bookCopyLentMailSent() {
        var reader = new Reader(
                UUID.randomUUID(),
                "Ted Tester",
                "ted@test.com"
        );
        var bookCopyId = UUID.randomUUID();
        var dueAt = Instant.now().plus(5, ChronoUnit.DAYS);

        subject.on(
                new BookCopyLentEvent(
                        "978-0008471286",
                        bookCopyId,
                        reader,
```

EmailHandlerTest

```
3
4       @InjectMocks
5       private EmailHandler subject;
6
7       @Mock
8       private MailApi mailApi;
9
10      @Test
11      public void bookCopyLentMailSent() {
12          var reader = new Reader(
13                  UUID.randomUUID(),
14                  "Ted Tester",
15                  "ted@test.com"
16          );
17          var bookCopyId = UUID.randomUUID();
18          var dueAt = Instant.now().plus(5, ChronoUnit.DAYS);
19
20          subject.on(
21                  new BookCopyLentEvent(
22                          "978-0008471286",
23                          bookCopyId,
24                          reader,
25                          dueAt
26                  )
27          );
```

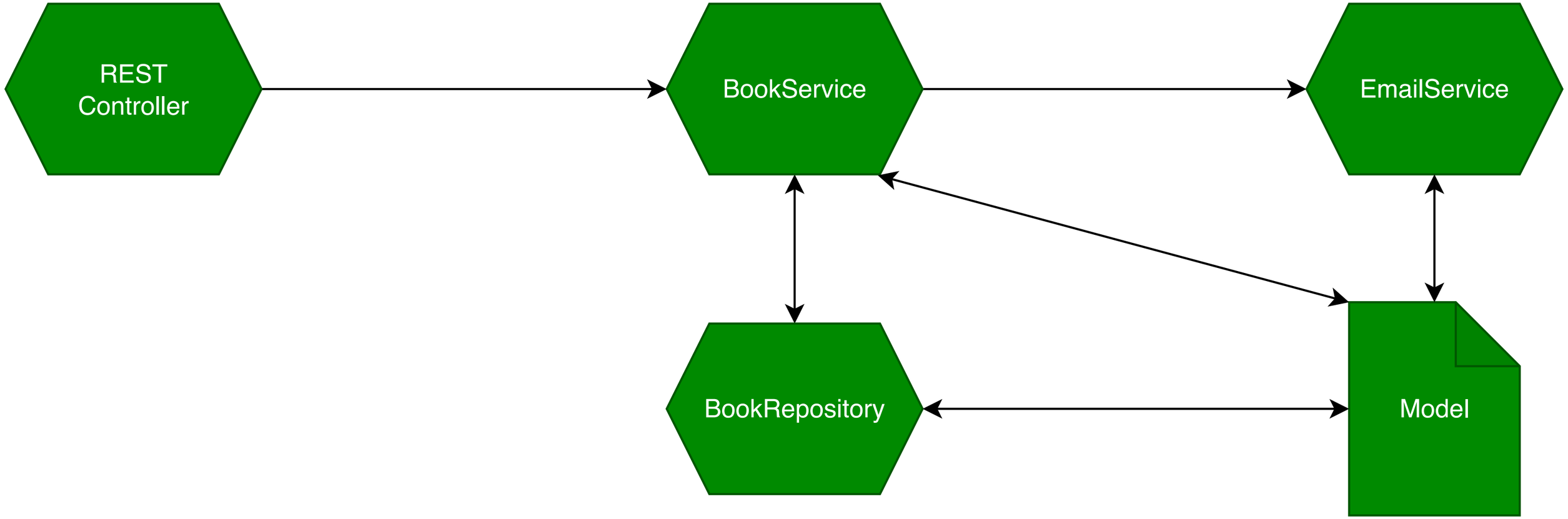
EmailHandlerTest

```
12          var reader = new Reader(
13                  UUID.randomUUID(),
14                  "Ted Tester",
15                  "ted@test.com"
16          );
17          var bookCopyId = UUID.randomUUID();
18          var dueAt = Instant.now().plus(5, ChronoUnit.DAYS);
19
20          subject.on(
21                  new BookCopyLentEvent(
22                          "978-0008471286",
23                          bookCopyId,
24                          reader,
25                          dueAt
26                  )
27          );
28
29          verify(mailApi).sendMail(
30                  "no-reply@tdd.com",
31                  reader.email(),
32                  "Buch erfolgreich ausgeliehen: 978-0008471286",
33                  "Bitte beachten Sie die Rückgabe bis spätestens: " + dueAt
34          );
35      }
```

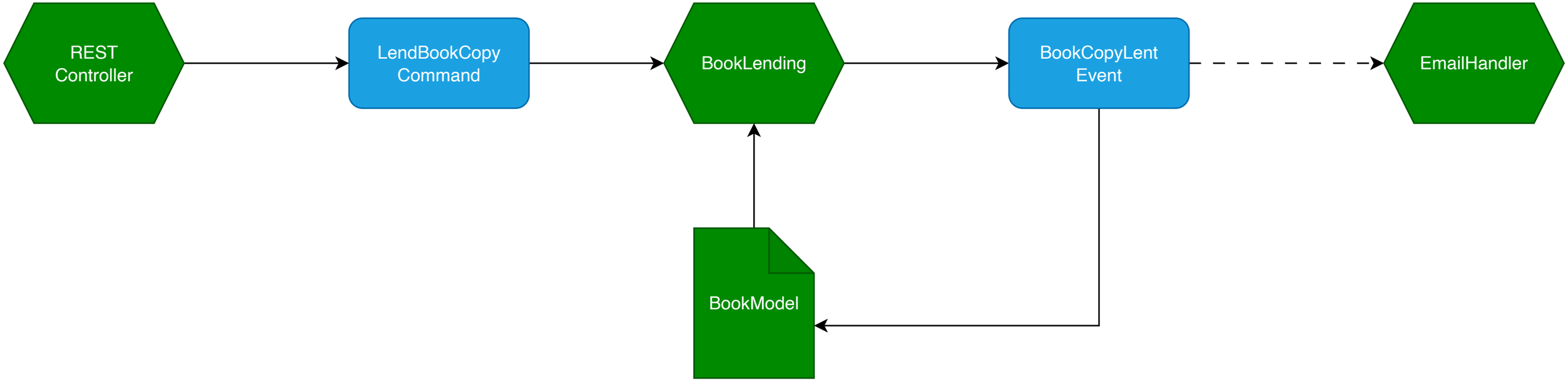
EmailHandlerTest

```
13              UUID.randomUUID(),
14              "Ted Tester",
15              "ted@test.com"
16      );
17      var bookCopyId = UUID.randomUUID();
18      var dueAt = Instant.now().plus(5, ChronoUnit.DAYS);
19
20      subject.on(
21              new BookCopyLentEvent(
22                      "978-0008471286",
23                      bookCopyId,
24                      reader,
25                      dueAt
26              )
27      );
28
29      verify(mailApi).sendMail(
30              "no-reply@tdd.com",
31              reader.email(),
32              "Buch erfolgreich ausgeliehen: 978-0008471286",
33              "Bitte beachten Sie die Rückgabe bis spätestens: " + dueAt
34      );
35  }
36 }
```

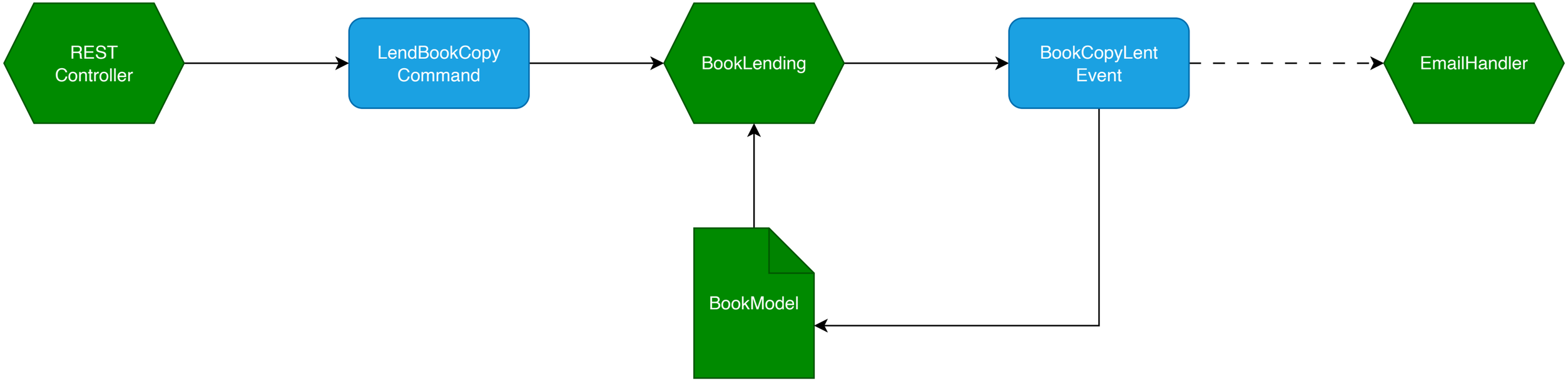
Fazit


REST
Controller
BookService
EmailService
BookRepository
Model

Fazit


REST Controller
LendBookCopy Command
BookLending
BookCopyLent Event
EmailHandler
BookModel

Fazit



Und das nennt sich übrigens CQRS:
Command Query Responsibility Segregation!

Was sind eure Erfahrungen mit leichtgewichtigen und stabilen Tests?

Sample Code

Frank Scheffler

OpenCQRS