Implementierungsplan
Team 2

SEP WS 2021/22

Betreuer:

Prof. Dr. Christian Bachmaier

Projektphase	Leiter
Pflichtenheft	Johann Schicho
Entwurf	Stefanie Gürster
Feinspezifikation	Johannes Garstenauer
Implementierung	Thomas Kirz
Validierung	Sebastian Vogt

3. Dezember 2021

Inhaltsverzeichnis

1 Einleitung

Johann Schicho

Der Implementierungszeitraum erstreckt sich vom 03.12.2021 zum 14.01.2022. Dabei ist der Zeitraum in drei Milestones untergliedert, in welchen die Anwendung phasenweise entwickelt wird. Ein *Arbeitspaket* umfasst dabei immer die vertikale Implementierung eines Features. Das heißt, ein Facelet mit zugehörigem Backing-Bean, den benötigten Service- und Repositorymethoden und *DTOs*. Diese Vorgehensweise ermöglicht eine weitgehend unabhängige Arbeit zwischen den Teammitgliedern. ✓

Die Aufteilung wird im Folgenden genauer spezifiziert:

∀ hei: mal grob was in welchem MS (wie besprochen)

Milestone 1 dauert bis zum 10.12.2021. Hier werden zunächst Infrastrukturelemente, wie Datenbankverbindung, Logging, Config Reader und Crypto, implementiert. Anschließend werden grundlegende Arbeitspakete der Anwendung darauf aufbauend umgesetzt. Dazu gehören unter anderem Login, Homepage und Submission.

Milestone 2 dauert bis zum 17.12.2021. Implementiert werden weitere aufwendigere Arbeitspakete. Dazu gehören zum Beispiel Profilseite, Anlegen eines neuen Forums, Forumsübersicht, Registrierungs- und Administrationsseite und das Einreichen von Gutachten.

nicht so wichtig

evtl. tauschen?

Milestone 3 dauert bis zum 07.01.2022. In dieser Phase werden die letzten Arbeitspakete implementiert. Unter anderem das Impressum, Nutzerliste, automatischer E-Mail-Service und die Suchergebnisseite. Auch das Aussehen der Webanwendung wird durch CSS und Bootstrap vereinheitlicht.

∀ Feature-komplett, bis auf Hilfeinhalte

Vorläufige Abgabe ist am 14.01.2022. Diese umfasst nun das vorläufig vollständig implementierte System, das finale Handbuch und den Implementierungsbericht.

2 Arbeitspakete

2.1 Konventionen

Stefanie Gürster Um die Tabellen im folgenden übersichtlicher zu gestalten, werden einzelne Methoden immer nur dann verwendet, wenn nur einzelne Methoden implementiert werden. Sind alle Methoden zu implementieren, so sind diese nicht explizit genannt. ✓

Außerdem beinhaltet ein Arbeitspaket immer auch die dazugehörige Beschreibung im Handbuch.

↳ das wäre erst in der Woche nach MS2 geplant (aber technische Umsetzung des Aufnahmemechanismus vorher)

2.2 Milestone 1

Stefanie Gürster

Tabelle 1: Milestone 1

Arbeitspaket	Start	Ende	Aufwand	Bearbeiter:in	Artefakte
NewSubmisson	6.12.10 10:00 Uhr	8.12.2021 16:00 Uhr	10 h	Sebastian Vogt	**NewSessionBacking** **SubmissionService** .add() .addCoAuthor() **SubmissionRepository** .add() .addCoAuthor() **Submission** **ScientificForum** **PaperService** .add() **PaperRepository** .add() **Paper** **PDFValidator** **ForumNameExistsValidator**
Crypto	6.12.2021 17:00 Uhr	7.12.2021 14:00 Uhr	3 h	Sebastian Vogt	**Hashing**

Technische Gründe

Fortsetzung auf der nächsten Seite

z.B.
Wie soll so New Submission ausprobiert werden?

Tabelle 1: Milestone 1 (*Fortsetzung*)

Arbeitspaket	**Start**	**Ende**	**Aufwand**	**Bearbeiter:in**	**Artefakte**
Connection Pool	9.12.2021 8:00 Uhr	9.12.2021 19:00 Uhr	zu spät 1 Tag vor Abgabe 6 h	Sebastian Vogt	**ConnectionPool** **ConnectionState** **Transaction** **DataSourceUtil**
SQL Datenschema	6.12.2021 10:00 Uhr	6.12.2021 11:00 Uhr	0,5 h	Johann Schicho	createPaper.sql createReCoAuthored.sql createReIntrests.sql createReMemberOf.sql createReviewedBy.sql createReTopics.sql createReview.sql createScienceField.sql createScientificForum.sql createSubmission.sql createSystem.sql createUser.sql createVerification.sql
Pagination	6.12.2021 11:00 Uhr	6.12.2021 15:00 Uhr	2 h	Johann Schicho	pagination.xhtml **Pagination** **ResultListParameters** **DateSelect** **SubmissionState** **SortOrder** **DateTimeInFutureValidator**
Template: SortSearchColumn	6.12.2021 15:00 uhr	6.12.2021 16:00 Uhr	1 h	Johann Schicho	sortSearchColumn.xhtml

Warum so viele Einzeldateien?

bzw. schon fertig?

Fortsetzung auf der nächsten Seite

Tabelle 1: Milestone 1 (*Fortsetzung*)

Arbeitspaket	**Start**	**Ende**	**Aufwand**	**Bearbeiter:in**	**Artefakte**
Faclet: Welcome	6.12.2021 16:00 Uhr	7.12.2021 14:00 Uhr	5 h	Johann Schicho	welcome.xhtml **WelcomeBacking** **LoginService** **SystemSettings** **SystemSettingsRepository** .getSettings()

LifeTime	7.12.2021
	14:00Uhr
		8.12.2021
		15:00 Uhr
			7 h	Johann Schicho	**LifeTime**
					.startUp()
					.shutDown()

UserGetLists	8.12.2021 15:00 Uhr	9.12.2021 16:00Uhr	5 h	Johann Schicho	**UserService** .getList() **UserRepository** .getList()
LifeTime: Logger	6.12.2021 10:00 Uhr	6.12.2021 14:00 Uhr	3 h	Thomas Kirz	**LifeTimeListener** **LifeTime** .startUp()

Fortsetzung auf der nächsten Seite

Tabelle 1: Milestone 1 (*Fortsetzung*)

Arbeitspaket	Start	Ende	Aufwand	Bearbeiter:in	Artefakte
Homepage	6.12.2021 14:00 Uhr	9.12.2021 19:00 Uhr	13 h	Thomas Kirz	hompage.xhtml **HomePageBacking** **SubmissionService** .get() .canView() .getList() **SubmissionRepository** .get() .getList() **ScientificForumService** .get() **ScientificForumRepository** .get() **ScientificForum** **Privilege**
Message Bundle Producer	6.12.2021 17:00 Uhr	7.12.2021 10:00 Uhr	2 h	Thomas Kirz	**MessageBundleProducer**

Fortsetzung auf der nächsten Seite

rel. spät

Infrastrukt., Systemstart Config einlesen
Logger
CP
main template

als 1.

Tabelle 1: Milestone 1 (*Fortsetzung*)

Arbeitspaket	**Start**	**Ende**	**Aufwand**	**Bearbeiter:in**	**Artefakte**
Exceptions	7.12.2021 10:00 Uhr	7.10.2021 11:00 Uhr	0,5 h	Thomas Kirz	**IllegalAccessException** **IllegalFlowException** **ConfigNotReadableException** **DataNotCompleteException** **DataNotWrittenException** **DatasourceConfigurationException** **DatasourceNotFoundException** **DatasourceQueryFailedException** **EmailServiceFailedException** **EmailTransmissionFailedException** **InvalidFieldsException** **InvalidQueryParamsException** **KeyExistsException** **LogsNotWritableException** **NotFoundException**
Template: Main	6.12.2021 10:00 Uhr	7.12.2021 16:00 Uhr	2 h	Stefanie Gürster	main.xhtml

?. Gibt es schon oder?

Fortsetzung auf der nächsten Seite

Tabelle 1: Milestone 1 (*Fortsetzung*)

Arbeitspaket	**Start**	**Ende**	**Aufwand**	**Bearbeiter:in**	**Artefakte**

Submission ohne Gutachten	6.12.2021
	16:00 Uhr
		9.12.2021
		18:00 Uhr
			17 h	Stefanie Gürster	submission.xhtml
					SubmissionBacking
					.init()
					.onload()
					.preRenderViewListener
					.setState()
					.downloadPaper()
					.upLoadReview()
					.applyFilters()
					.releaseRevision()
					.getAuthorForPaper()
					.uploadPDF()
					.deleteSubmission()
					.getUploadedRevisionPDF()
					.setUploadesRevisionPDF()
					.getSubmission()
					.getScientificForum()
					.getCoAuthors()
					.getAuthor()
					.getPaperPagination()
					.getSessionInformation()
					.IsViewerSubmitter()
					.getSubmissionStates()
					.getDateSelects()
					.getNewestPaper()
					.loggedInUserIsEditor()
					.LoggedInUserReviewer()

Musterpaket

Fortsetzung auf der nächsten Seite

Tabelle 1: Milestone 1 (*Fortsetzung*)

Arbeitspaket	**Start**	**Ende**	**Aufwand**	**Bearbeiter:in**	**Artefakte**

Fortsetzung Submission ohne Gutachten				Stefanie Gürster	**UserService**
					.get()
					.getList()
					SubmissionService
					.remove()
					.change()
					.releaseReview()
					.removeCoAuthor()
					SubmissionRepository
					.remove()
					.change()
					.removeCoAuthor()
					PaperService
					.remove()
					.getList()
					.getLatest()
					.change()
					.reviewedBy()
					PaperRepository
					.get()
					.change()
					.getList()
					.getPDF()
					.setPDF()
					.getNewestPaperForSubmission()
					URLValidator
					AcceptanceStatus
					ReviewedBy
					ReviewedByRepository
					.get()

Fortsetzung auf der nächsten Seite

Schätzungen hier eher hoch?

Tabelle 1: Milestone 1 (*Fortsetzung*)

Arbeitspaket	**Start**	**Ende**	**Aufwand**	**Bearbeiter:in**	**Artefakte**
SessionInformation	6.12.2021 10:00 Uhr	6.12.2021 14:00 Uhr	2 h	Johannes Garstenauer	**SessionInformation** **User** **UserService** .get() **UserRepository** .get()
TrespassListener	6.12.2021 14:00 Uhr	6.12.2021 19:00 Uhr	4 h	Johannes Garstenauer	**TrespassListener**
UIMessage Generator	7.12.2021 10:00Uhr	7.12.2021 17:00 Uhr	5 h	Johannes Garstenauer	**UIMessageGenerator** **UIMessage** **MessageCategory**
Templates: Header, Footer	7.12.2021 17:00 Uhr	8.12.2021 16:00 Uhr zu spät	4 h	Johannes Garstenauer	footer.xhtml navigation.xhtml **ImageServlet** **FileDTo** **CustomizationService** .setLogo() .getLogo() **AvatarUtil**
Config Reader	8.12.2021 16:00 Uhr	9.12.2021 14:00 Uhr	3 h	Johannes Garstenauer	**ConfigPropagator** **ConfigReader**

kleine Arbeitspakete → unklar ob Start von abhängigen Paketen überlappend gut ist / gut gelingen wird

wenn ja wieviel?
10%

ist da "Luft" drin
z.B. für Nachbesserungen oder
absolute Herstellungszeiten?

2.2.1 Stundenanzahl

gut: Summe

Teammitglied	**Stundenanzahl**
Sebastian Vogt	19 h
Johann Schicho	20,5 h
Thomas Kirz	18,5 h
Stefanie Gürster	19 h
Johannes Garstenauer	18 h

2.3 Milestone 2

2.3.1 Arbeitpakete

Sebastian Vogt

Tabelle 2: Milestone 2

Arbeitspaket	**Start**	**Ende**	**Aufwand**	**Bearbeiter:in**	**Artefakte**
ScienceFields	13.12.2021 10:00	13.12.2021 20:00	5 h	Sebastian Vogt	**ScienceFieldService** **ScienceFieldRepository** **ScienceField**

Fortsetzung auf der nächsten Seite

Tabelle 2: Milestone 2 (*Fortsetzung*)

Arbeitspaket	Start	Ende	Aufwand	Bearbeiter:in	Artefakte

Profile	14.12.2021 08:00	16.12.2021 11:30	12 h eher hoch weil schon im Entwurfsheft/ Prototyp	Sebastian Vogt	profile.xhtml
					ProfileBacking
					AvatarValidator
					ScienceField
					UserService
					.getAvatar()
					.change()
					.setAvatar()
					.addScienceField()
					.removeScienceField()
					.remove()
					UserRepository
					.getAvatar()
					.change()
					.setAvatar()
					.addScienceField()
					.removeScienceField()
					.remove()
					SubmissionService
					.countSubmissions()
					SubmissionRepository
					.countSubmissions()

NewReview	13.12.2021 14:00	13.12.2021 18:00	4 h	Johann Schicho	newReview.xhtml **NewReviewBacking** **ReviewService** .add() **ReviewRepository** .add() .setPDF()

Fortsetzung auf der nächsten Seite

Tabelle 2: Milestone 2 (*Fortsetzung*)

Arbeitspaket	**Start**	**Ende**	**Aufwand**	**Bearbeiter:in**	**Artefakte**

Submission: Reviews	13.12.2021
	18:00
		15.12.2021
		15:00
			10 h	Johann Schicho	submission.xhtml
					Review
					SubmissionBacking
					.downloadReview()
					.releaseReview()
					.acceptReviewing()
					.declineReviewing()
					.getReviewerForReview()
					.getReviewPagination()
					.getReviewedBy()
					ReviewService
					.get()
					.change()
					.remove()
					.getList()
					.getFile()
					ReviewRepository
					.get()
					.change()
					.remove()
					.getList()
					.getPDF()

NewScientificForum	16.12.2021 10:00	16.12.2021 15:30	4 h	Johann Schicho	newScientificForum.xhtml **newScientificForumBacking** **ScientificForumService** .add() **ScientificForumRepository** .add()

Fortsetzung auf der nächsten Seite

Tabelle 2: Milestone 2 (*Fortsetzung*)

Arbeitspaket	**Start**	**Ende**	**Aufwand**	**Bearbeiter:in**	**Artefakte**
Registration besser überlappend	13.12.2021 14:00	14.12.2021 17:00	10 h eher hoch	Thomas Kirz	registration.xhtml **RegistrationBacking** **EmailAddressLayoutValidator** **EmailAddressUnoccupiedValidator** **PasswordValidator** **EmailAddressExistsValidator** **RegistrationService** .register() **UserService** .emailExists() **UserRepository** .add() .emailExists()
Verification	15.12.2021 13:00	16.12.2021 17:00	10 h	Thomas Kirz	verification.xhtml **VerificationBacking** **Verification** **UserService** .getVerification() **UserRepository** .getVerification() .setVerification()
Administration	13.12.2021 10:00	14.12.2021 14:00	6 h	Stefanie Gürster	administration.xhtml **AdministrationBacking** **SystemSettingsRepository** **CustomizationService**

Fortsetzung auf der nächsten Seite

Tabelle 2: Milestone 2 (*Fortsetzung*)

Arbeitspaket	**Start**	**Ende**	**Aufwand**	**Bearbeiter:in**	**Artefakte**

Toolbar	14.12.2021 15:00	16.12.2021 19:00	10 h	Stefanie Gürster	toolbar.xhtml
					ToolbarBacking
					SubmissionService
					.addReviewer()
					.removeReviewer()
					SubmissionRepository
					.addReviewer()
					.removeReviewer()
					ReviewedByRepository
					.get()

ScientificForum	13.12.2021 16:00	15.12.2021 18:00	15 h	Johannes Garstenauer	scientificForum.xhtml
					ScientificForumBacking
					ScientificForum
					ForumNameExistsValidator
					ScientificForumService
					.remove()
					.addEditor()
					.addScienceField()
					.removeEditor()
					.removeScienceField()
					.change()
					.getEditor()
					.exists()
					ScientificForumRepository
					.remove()
					.addEditor()
					.addScienceField()
					.removeEditor()
					.removeScienceField()
					.change()
					.exists()

Monsterpaket

Fortsetzung auf der nächsten Seite

Tabelle 2: Milestone 2 (*Fortsetzung*)

Arbeitspaket	Start	Ende	Aufwand	Bearbeiter:in	Artefakte
ScientificForumList	16.12.2021 13:00	16.12.2021 18:00	5 h	Johannes Garstenauer	scientificForumList.xhtml **ScientificForumListBacking** **ScientificForumService** .getList() **ScientificForumRepository** .getList()

2.3.2 Stundenanzahl

✓

Teammitglied	Stundenanzahl
Sebastian Vogt	17 h
Johann Schicho	18 h
Thomas Kirz	20 h
Stefanie Gürster	16 h
Johannes Garstenauer	20 h

2.4 Milestone 3

Johannes Garstenauer

Tabelle 3: Milestone 3

Arbeitspaket	Start	Ende	Aufwand	Bearbeiter:in	Artefakte
Globale Suche	03.01.2022 10 Uhr	06.01.2022 14 Uhr	15 h	Sebastian Vogt	resultList.xhtml **ResultListBacking**

Fortsetzung auf der nächsten Seite

[illegible]

Tabelle 3: Milestone 3 (*Fortsetzung*)

Arbeitspaket	**Start**	**Ende**	**Aufwand**	**Bearbeiter:in**	**Artefakte**
Periodische Datenbankarbeiten	03.01.2022 10 Uhr	04.01.2022 18 Uhr	4 h	Johann Schicho	**PeriodicWorker** **DataSourceUtil** .cleanupVerficiations()
Initiale Administratorenkonfiguration	05.01.2022 10 Uhr	06.01.2022 14 Uhr	4 h	Johann Schicho	initialConfig.xhtml **InitialConfigBacking**
Fehlerseite	03.01.2022 10 Uhr	03.01.2022 18 Uhr	3 h	Thomas Kirz	errorPage.xhtml **ErrorPageBacking** **ErrorMessage**
Emailingfunktionalitäten	04.01.2022 12 Uhr	05.01.2022 14 Uhr	5 h	Thomas Kirz	**EmailUtil** **EmailSender**
Anlegung neuer Nutzer	05.01.2022 14 Uhr	06.01.2022 14 Uhr	5 h	Thomas Kirz	newUser.xhtml **NewUserBacking** **RegistrationService** .registerByAdmin()
Impressum	03.01.2022 10 Uhr	03.01.2022 18 Uhr	3 h	Stefanie Gürster	imprint.xhtml Resource Bundle 'message': Impressumstexte **ImprintBacking**
Cascading Style Sheets	04.01.2022 10 Uhr	06.01.2022 14 Uhr	10 h	Stefanie Gürster	Themes Verbesserung der Facelets mit CSS **Style.java**
Behandlung der ungeprüften Ausnahmen	03.01.2022 10 Uhr	04.01.2022 18 Uhr	7 h	Johannes Garstenauer	**UncheckedExceptionHandler** **UncheckedExceptionHandlerFactory**
Nutzerliste	05.01.2022 10 Uhr	06.01.2022 14 Uhr	4 h	Johannes Garstenauer	userList.xhtml **UserListBacking**

2.4.1 Stundenanzahl

Teammitglied	Stundenanzahl
Sebastian Vogt	15 h
Johann Schicho	8 h
Thomas Kirz	13 h
Stefanie Gürster	13 h
Johannes Garstenauer	11 h

2.5 Stundenanzahl aller Milestones

Teammitglied	Stundenanzahl
Sebastian Vogt	51 h
Johann Schicho	46,5 h
Thomas Kirz	51,5 h
Stefanie Gürster	48 h
Johannes Garstenauer	49 h

wenn Schätzungen einigermaßen passen (wovon ich ausgehe) gut aufgeteilt

Testing Fixing in den Paketen dabei, oder? Wer führt das durch?
Ersteller der Tests anderer als Implementierer?

wenn Ersteller der Tests gibt es dafür keine geplanten Aufwände ...

3 Abhängigkeiten

Thomas Kirz

Abbildung 1 zeigt alle Arbeitspakete mit ihrem Bearbeiter und dem geschätzten Zeitaufwand und stellt eine Übersicht über die Abhängigkeiten zwischen den Paketen dar. Es kann oft trotz einer Abhängigkeit zwischen zwei Arbeitspaketen gleichzeitig mit beiden begonnen werden, dies wird durch eine gestrichelte Linie gekennzeichnet.

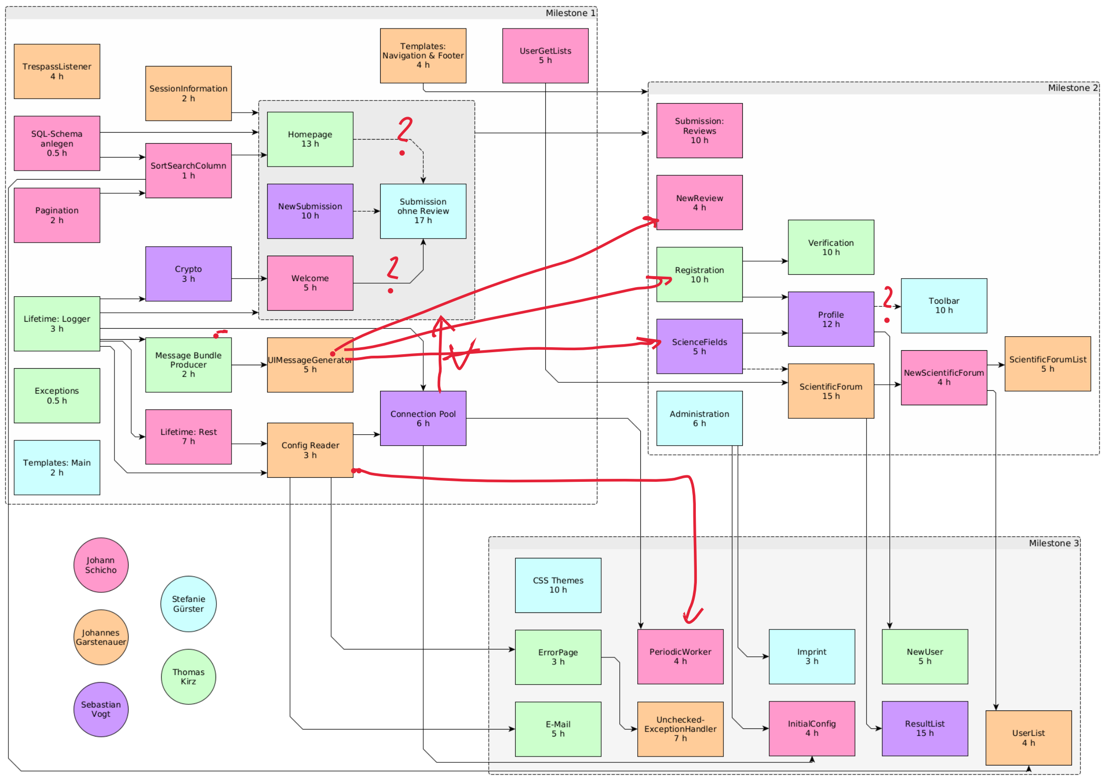

Abbildung 1: Abhängigkeiten der Arbeitspakete in einem PERT-Chart

4 Spezialgebiete

Johann Schicho

Bereits während der bisherigen Projektphasen haben sich Experten zu Spezialgebieten herauskristallisiert oder wurden für neue Spezialgebiete festgelegt, um Verantwortlichkeiten aufzuteilen und zu managen.

Die Spezialgebiete mit ihren Experten werden im Folgenden aufgelistet.

Spezialgebiet	Experte
Bootstrap	Stefanie Gürster
Composite Components	Johann Schicho
Config Datei	Johannes Garstenauer
Connection Pool	Sebastian Vogt
Crypto	Sebastian Vogt
CSS	Stefanie Gürster
Datei Upload/Download	Johannes Garstenauer
E-Mail Versendung	Thomas Kirz
Fehlerbehandlung	Johannes Garstenauer
Internationalisierung	Stefanie Gürster
großes Gebiet → Jakarta Server Faces	Stefanie Gürster
LaTeX	Thomas Kirz
Logging	Thomas Kirz
Testing JUnit → Mockito	Thomas Kirz
Pagination	Johann Schicho
Primefaces	Johann Schicho
Rollenprüfung	Johannes Garstenauer
RSA	Sebastian Vogt
Selenium	Sebastian Vogt
Servlets	Sebastian Vogt
SQL	Johann Schicho
URL Parameter	Sebastian Vogt

Schaut auf den 1. Blick so aus als ob für Arbeitspakete Spezialwissen gut genutzt wird

5 Tests

gut: TDD wie jetzt schon Testcode

Es folgen Beispiele für Whitebox-Tests für verschiedene Arbeitspakete aus allen Milestones. Sie sollen repräsentativ sein für bei der Implementierung entstehende Tests, sind aber nicht zwingend vollständige Test-Suites für die jeweiligen Klassen oder Arbeitspakete.

Unterschiedlich was man getestet

5.1 Milestone 1

5.1.1 Arbeitspaket *Submission*

Thomas Kirz → Bearbeiterin Gürster

```java
@ExtendWith(MockitoExtension.class)
class SubmissionBackingTest {

    private static final int EXAMPLE_SUBMISSION_ID = 2;
    private static final int EXAMPLE_USER_ID = 6;

    @Mock SessionInformation sessionInformation;

    @Mock SubmissionService submissionService;

    @Mock UserService userService;

    @Mock PaperService paperService;

    @Mock ReviewService reviewService;
```

+ Beschreibung bzw. Doku/Javadoc der Tests

```java
    @InjectMocks SubmissionBacking submissionBacking;

    @Test
    void testInitAndOnLoad() {
        Submission sub = new Submission();
        sub.setId(EXAMPLE_SUBMISSION_ID);
        User user = new User();
        user.setId(EXAMPLE_USER_ID);

        when(sessionInformation.getUser()).thenReturn(user);
        when(submissionService.get(sub)).thenReturn(sub);

        SubmissionBacking submissionBacking = new SubmissionBacking();
        submissionBacking.init();
        submissionBacking.onLoad();

        verify(submissionService).get(sub);
        verify(userService).getList(sub, Privilege.AUTHOR);
        verify(paperService).getList(eq(sub), eq(user), any());
        verify(reviewService).getList(eq(sub), eq(user), any());
        verify(submissionService).getReviewedBy(eq(sub), eq(user));
        verify(paperService).getLatest(eq(sub), eq(user));
    }

    @Test
    void testInitAndOnLoadIllegal() {
        Submission sub = new Submission();
        sub.setId(EXAMPLE_SUBMISSION_ID);
        // User with no rights to view the submission
        User user = new User();
        user.setId(EXAMPLE_USER_ID + 1);

        when(sessionInformation.getUser()).thenReturn(user);
        when(submissionService.get(sub)).thenReturn(sub);

        submissionBacking.init();

        assertThrows(IllegalAccessException.class, () -> submissionBacking.onLoad());
    }
}
```

Listing 1: SubmissionBackingTest.java

```java
@ExtendWith(MockitoExtension.class)
class SubmissionServiceTest {

    // id of submission in mocked repository
    private static final int FIRST_SUBMISSION_ID = 0;
    private static final int EXAMPLE_REVIEW_VERSION = 3;
    private static final int EXAMPLE_USER_ID = 6;
    private static final String EXAMPLE_SUBMISSION_TITLE = "Submission title";

    static MockedStatic<SubmissionRepository> subRepo;
    static MockedStatic<ReviewRepository> reviewRepo;

    @BeforeAll
    static void mockRepositories() {
        Submission submissionFromRepo = new Submission();
        submissionFromRepo.setTitle(EXAMPLE_SUBMISSION_TITLE);
        submissionFromRepo.setAuthorId(EXAMPLE_USER_ID);
        User user = new User();
        user.setId(EXAMPLE_USER_ID);

        subRepo = mockStatic(SubmissionRepository.class);
        subRepo.when(() -> SubmissionRepository.get(
                eq(submissionFromRepo), any(Transaction.class)))
                .thenReturn(submissionFromRepo);
        subRepo.when(() -> SubmissionRepository.getList(
                eq(user), Privilege.AUTHOR, any(Transaction.class), any()))
                .thenReturn(new Submission[]{submissionFromRepo});

        reviewRepo = mockStatic(ReviewRepository.class);
    }

    @AfterAll
    static void closeRepositoryMocks() {
        subRepo.close();
        reviewRepo.close();
    }

    @Test
    void testGet() {
        Submission sub = new Submission();
        sub.setId(FIRST_SUBMISSION_ID);

        SubmissionService submissionService = new SubmissionService();
        Submission gotten = submissionService.get(sub);

        assertAll(
                () -> assertEquals(FIRST_SUBMISSION_ID, gotten.getId()),
                () -> assertEquals(EXAMPLE_SUBMISSION_TITLE, gotten.getTitle())
        );
    }

    @Test
    void testReleaseReview() {
        Submission submission = new Submission();
        submission.setId(FIRST_SUBMISSION_ID);
        Review review = new Review();
        review.setPaperVersion(EXAMPLE_REVIEW_VERSION);
        review.setSubmissionId(FIRST_SUBMISSION_ID);
        SubmissionService submissionService = new SubmissionService();

        submissionService.releaseReview(review, submission);

        reviewRepo.verify(() -> ReviewRepository.change(
                eq(review), any(Transaction.class)), times(1));
    }

    @Test
    void testGetOwnSubmissions() {
        User user = new User();
        user.setId(EXAMPLE_USER_ID);
        Submission submission = new Submission();
        submission.setId(FIRST_SUBMISSION_ID);

        SubmissionService submissionService = new SubmissionService();
        List<Submission> result = submissionService.getList(Privilege.AUTHOR, user, null);
        assertAll(
                () -> assertEquals(1, result.size()),
                () -> assertEquals(submission, result.get(0))
        );
    }

    @Test
    void testGetOwnSubmissionEmpty() {
        // different user to example user
        User user = new User();
        user.setId(EXAMPLE_USER_ID + 1);

        SubmissionService submissionService = new SubmissionService();
        List<Submission> result = submissionService.getList(Privilege.AUTHOR, user, null);
        assertEquals(0, result.size());
    }

}
```

Listing 2: SubmissionServiceTest.java

```java
class SubmissionRepositoryTest {

    private static final int EXAMPLE_SUBMISSION_ID_1 = 1;
    private static final int EXAMPLE_SUBMISSION_ID_2 = 2;
    private static int EXAMPLE_USER_ID;
    private static final String EXAMPLE_SUBMISSION_TITLE_1 = "Submission title";
    private static final String EXAMPLE_SUBMISSION_TITLE_2 = "Different title";

    @BeforeAll
    void addUser() throws Exception {
        Transaction transaction = new Transaction();
        UserRepository.add(new User(), transaction);
        EXAMPLE_USER_ID = UserRepository.getList(
                transaction, new ResultListParameters()).get(0).getId();
        transaction.commit();
    }

    @Test
    void testAddAndGetList() throws Exception {
        Transaction transaction = new Transaction();

        User user = new User();
        user.setId(EXAMPLE_USER_ID);

        Submission submission1 = new Submission();
        submission1.setId(EXAMPLE_SUBMISSION_ID_1);
        submission1.setAuthorId(EXAMPLE_USER_ID);
        submission1.setTitle(EXAMPLE_SUBMISSION_TITLE_1);

        Submission submission2 = new Submission();
        submission2.setId(EXAMPLE_SUBMISSION_ID_2);
        submission2.setEditorId(EXAMPLE_USER_ID);
        submission2.setTitle(EXAMPLE_SUBMISSION_TITLE_2);

        SubmissionRepository.add(submission1, transaction);
        SubmissionRepository.add(submission2, transaction);

        List<Submission> authoredList = SubmissionRepository
                .getList(user, Privilege.AUTHOR, transaction, null);
        List<Submission> editedList = SubmissionRepository
                .getList(user, Privilege.EDITOR, transaction, null);

        assertAll(
                () -> assertEquals(1, authoredList.size()),
                () -> assertEquals(EXAMPLE_SUBMISSION_TITLE_1, authoredList.get(0).getTitle()),
                () -> assertEquals(1, editedList.size()),
                () -> assertEquals(EXAMPLE_SUBMISSION_TITLE_2, editedList.get(0).getTitle())
        );
        transaction.abort();
    }

    @Test
    void testAddAndAbort() throws Exception {
        Transaction transaction = new Transaction();
        Submission submission = new Submission();
        submission.setId(EXAMPLE_SUBMISSION_ID_1);
        submission.setAuthorId(EXAMPLE_USER_ID);
        submission.setTitle(EXAMPLE_SUBMISSION_TITLE_1);
        SubmissionRepository.add(submission, transaction);
        transaction.abort();
        List<Submission> authoredList = SubmissionRepository.getList(
                new User(), Privilege.AUTHOR, transaction, null);
        assertEquals(0, authoredList.size());
    }
    // add two submissions and test countSubmissions()
    @Test
    void testCountSubmissions() throws Exception {
        User user = new User();
        user.setId(EXAMPLE_USER_ID);

        Submission submission1 = new Submission();
        submission1.setId(EXAMPLE_SUBMISSION_ID_1);
        submission1.setAuthorId(EXAMPLE_USER_ID);
        submission1.setTitle(EXAMPLE_SUBMISSION_TITLE_1);

        Submission submission2 = new Submission();
        submission2.setId(EXAMPLE_SUBMISSION_ID_2);
        submission2.setEditorId(EXAMPLE_USER_ID);
        submission2.setTitle(EXAMPLE_SUBMISSION_TITLE_2);

        Transaction transaction = new Transaction();

        SubmissionRepository.add(submission1, transaction);
        SubmissionRepository.add(submission2, transaction);

        assertEquals(1, SubmissionRepository.countSubmissions(user, transaction));

        transaction.abort();
    }

}
```

Listing 3: SubmissionRepositoryTest.java

5.1.2 Arbeitspaket *Welcome*

Johann Schicho

```java
@ExtendWith(MockitoExtension.class)
class LoginServiceTest {

    private static final String SECURE_EXAMPLE_PASSWORD = "Affe123!";
    private static final String VERY_RANDOM_SALT_BASE64 = "QWZmZW4gc2luZCBzdXBlcg==";
    // PBKDF2WithHmacSHA256, 128 bit, 75000 iterations.
    private static final String SECURE_PASSWORD_HASH_BASE64 = "YXirI3Xsv1hhne/qI+um6Q==";
```

↑ Wo kommt Wert her?
Von eigener Implementierung die getestet werden soll?

bzw. Wichtig bei crypto-Tests

```java
    @Test
    void testLoginSuccess() {
        User UserJustLogin = new User();
        UserJustLogin.setId(1);
        UserJustLogin.setPasswordNotHashed(SECURE_EXAMPLE_PASSWORD);

        User UserInDBWithID1 = new User();
        UserInDBWithID1.setId(1);
        UserInDBWithID1.setPasswordHashed(SECURE_PASSWORD_HASH_BASE64);

        LoginService loginService = new LoginService();

        try (MockedStatic<UserRepository> mockedStatic = mockStatic(UserRepository.class)) {
            mockedStatic.when(() -> UserRepository.get(eq(UserJustLogin), any())).thenReturn(UserInDBWithID1);

            assertEquals(UserInDBWithID1, loginService.login(UserJustLogin));
        }
    }

    @Test
    void testLoginFail() {
        User UserJustLogin = new User();
        UserJustLogin.setId(1);
        UserJustLogin.setPasswordNotHashed("incorrectPassword");

        User UserInDBWithID1 = new User();
        UserInDBWithID1.setId(1);
        UserInDBWithID1.setPasswordHashed(SECURE_PASSWORD_HASH_BASE64);

        LoginService loginService = new LoginService();

        try (MockedStatic<UserRepository> mockedStatic = mockStatic(UserRepository.class)) {
            mockedStatic.when(() -> UserRepository.get(eq(UserJustLogin), any())).thenReturn(UserInDBWithID1);

            assertNull(loginService.login(UserJustLogin));
        }
    }
}
```

Listing 4: LoginServiceTest.java

5.1.3 Arbeitspaket *UserGetLists*

Johann Schicho

```
@ExtendWith(MockitoExtension.class)
public class UserRepositoryGetListTest {

    private static final User user1 = new User();
    private static final User user2 = new User();
    private static final User user3 = new User();
    private static final User user4 = new User();
    private static final User user5 = new User();
    private static final User user6 = new User();
    private static final User user7 = new User();

    private static List<User> userTestData;

    @BeforeAll
    static void init() {
        user1.setFirstName("Anton");
        user2.setFirstName("Bertha");
        user3.setFirstName("Cäsar");
        user4.setFirstName("Dora");
        user5.setFirstName("Emil");
        user6.setFirstName("Friedrich");
        user7.setFirstName("Gustav");

        user1.setLastName("Grimm");
        user2.setLastName("Fischer");
        user3.setLastName("Eckert");
        user4.setLastName("Decker");
        user5.setLastName("Claas");
        user6.setLastName("Becker");
        user7.setLastName("Ackermann");

        userTestData = Arrays.asList(user1, user2, user3, user4, user5, user6, user7);

        // Give every user an ID.
        int i = 1;
        for (User user : userTestData) {
            user.setId(i);
            i++;
        }

        user1.setAdmin(true);
    }

    @Test
    void testGetListSortByFirstname() throws NoSuchFieldException, IllegalAccessException, DataNotCompleteException {
        ResultListParameters params = new ResultListParameters();
        params.setPageNo(1);
        params.setSortColumn("firstname");
        params.setSortOrder(SortOrder.ASCENDING);

        Transaction mockTransaction = mock(Transaction.class);
        Connection mockConnection = mock(Connection.class);
        Statement mockStmt = mock(Statement.class);
        ResultSet rst = mock(ResultSet.class);
```

man könnte auch echte Objekte
z.B. Connection aus CP hernehmen, dann
würde das auch mitgetestet

```
        try {
            // Mock the connection to get a mocked statement.
            when(mockConnection.createStatement()).thenReturn(mockStmt);

            // Mock the statement to get a mocked ResultSet.
            when(mockStmt.executeQuery(any())).thenReturn(rst);

            // Finally, mock the ResultSet to return the test values
            // for all 7 entries of the testdata in the order we expect.
            final int[] i = {0};
            when(rst.next()).thenAnswer((Answer<Boolean>) invocation -> {
                if (i[0] < 7) {
                    i[0]++;
                    return true;
                } else {
                    return false;
                }
            });

            // All return values are based on the number of previous rst.next() calls.
            when(rst.getInt("id")).thenAnswer(I -> userTestData.get(i[0]).getId());
            when(rst.getString("email_address")).thenAnswer(I -> userTestData.get(i[0]).getEmailAddress());
            when(rst.getString("is_administrator")).thenAnswer(I -> userTestData.get(i[0]).isAdmin());
            when(rst.getString("firstname")).thenAnswer(I -> userTestData.get(i[0]).getFirstName());
            when(rst.getString("lastname")).thenAnswer(I -> userTestData.get(i[0]).getLastName());
            when(rst.getString("title")).thenAnswer(I -> userTestData.get(i[0]).getTitle());
            when(rst.getString("employer")).thenAnswer(I -> userTestData.get(i[0]).getEmployer());
            when(rst.getString("birthdate")).thenAnswer(I -> userTestData.get(i[0]).getDateOfBirth());
            when(rst.getString("is_registered")).thenAnswer(I -> userTestData.get(i[0]).isRegistered());
            when(rst.getString("password_hash")).thenAnswer(I -> userTestData.get(i[0]).getPasswordHashed());
            when(rst.getString("password_salt")).thenAnswer(I -> userTestData.get(i[0]).getPasswordSalt());

        } catch (SQLException e) {
            e.printStackTrace();
        }

        // get the modified connection from transaction.
        when(mockTransaction.getConnection()).thenReturn(mockConnection);

        assertEquals(userTestData, UserRepository.getList(mockTransaction, params));
    }
}
```

Listing 5: UserRepositoryGetListTest.java

5.1.4 Arbeitspaket *ConnectionPool*

Sebastian Vogt → Bearbeiter Vogt

```
@ExtendWith(MockitoExtension.class)
class TransactionTest {

    @Test
    void testCommitCommitsOnConnection() throws NoSuchFieldException,
            IllegalAccessException, SQLException {
```

```
        Transaction transaction = new Transaction();
        Connection mockConnection = Mockito.mock(Connection.class);

        Field connectionField
                = Transaction.class.getDeclaredField("connection");
        connectionField.setAccessible(true);
        connectionField.set(transaction, mockConnection);

        transaction.commit();
        Mockito.verify(mockConnection).commit();
    }

    @Test
    void testAbortAbortsOnConnection() throws NoSuchFieldException,
            IllegalAccessException, SQLException {
        Transaction transaction = new Transaction();
        Connection mockConnection = Mockito.mock(Connection.class);

        Field connectionField
                = Transaction.class.getDeclaredField("connection");
        connectionField.setAccessible(true);
        connectionField.set(transaction, mockConnection);

        transaction.abort();
        Mockito.verify(mockConnection).rollback();
    }

}
```

Listing 6: TransactionTest.java

```
@ExtendWith(MockitoExtension.class)
class ConnectionPoolTest {

    @Test
    void testGetConnectionUninitialized() {
        ConnectionPool pool = ConnectionPool.getInstance();
        assertThrows(IllegalStateException.class, pool::getConnection);
    }

    @Test
    void testGetConnectionOpen() throws SQLException {
        ConnectionPool.init();
        ConnectionPool pool = ConnectionPool.getInstance();
        Connection conn = pool.getConnection();
        assertFalse(conn.isClosed());
    }

    @Test
    void testReleaseForeignConnection() {
        ConnectionPool.init();
        ConnectionPool pool = ConnectionPool.getInstance();
        assertThrows(IllegalArgumentException.class, () -> {
            pool.releaseConnection(Mockito.mock(Connection.class));
        });
    }

    @Test
    void testShutdownDeInitializes() {
        ConnectionPool.init();
        ConnectionPool.shutDown();
        ConnectionPool pool = ConnectionPool.getInstance();
        assertThrows(IllegalStateException.class, pool::getConnection);
    }

}
```

Listing 7: ConnectionPoolTest.java

5.1.5 Arbeitspaket *Crypto*

Stefanie Gürster

```java
public class HashingTest {

    private static final String PASSWORD1 = "winter45sommer";
    private static final String SALT1 = "idkfjADGdkds945803jdkl==";
    private static final String HASH1 = "VlE2L0tuG/djzLbVz0r6OA==";

    private static final String PASSWORD2 = "basti3schi";
    private static final String SALT2 = "i845jdlkfjDkjf34ljsdSS==";
    private static final String HASH2 = "Yz7+fC9F2UuxC1dsOI4OGg==";

    @Test
    void testCorrectHashing() {
        String calculatedHash1 = Hashing.hashWithGivenSalt(PASSWORD1, SALT1);
        String calculatedHash2 = Hashing.hashWithGivenSalt(PASSWORD2, SALT2);
        assertAll(
                () -> assertEquals(HASH1, calculatedHash1),
                () -> assertEquals(HASH2, calculatedHash2)
        );
    }

}
```

Listing 8: HashingTest.java

5.1.6 Arbeitspaket *NewSubmission*

Johannes Garstenauer

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

    // The required repository.
    private static MockedStatic<ScientificForumRepository> forumRepoMock;
    private static MockedStatic<UserRepository> userRepoMock;

    // The required service.
    private static ScientificForumService forumService;

    // The required DTOs and their values
    private static ScientificForum forum;
    private static final Integer EXAMPLE_FORUM_ID = 1;
    private static final String EXAMPLE_FORUM_NAME = "Advances in Neural Information " +
            "Processing Systems";
    private static final String EXAMPLE_FORUM_NAME_ALTERNATIVE =
            "Proceedings of the IEEE International Conference on Computer Vision";
    private static final String EXAMPLE_FORUM_DESCRIPTION = "Lorem ipsum dolor sit";

    private static final Integer EXAMPLE_USER_ID_EDITOR = 17;
    private static final Integer EXAMPLE_USER_ID_EDITOR_ALTERNATIVE = 18;
    private static final Integer EXAMPLE_USER_ID_NOT_AN_EDITOR = 16;


    @BeforeAll
    static void init() {

        // Mock repository and initialize services.
        forumRepoMock = mockStatic(ScientificForumRepository.class);
        userRepoMock = mockStatic(UserRepository.class);
        forumService = new ScientificForumService();

        // Mock get to return a forum with a name if the id is correct.
        forumRepoMock.when(() -> ScientificForumRepository
            .get(eq(forum), any(Transaction.class))).thenReturn(forum);
}

@BeforeEach
void resetDTOS() {

    // Reset the dto's value's after each test.
    forum = new ScientificForum();
    forum.setId(EXAMPLE_FORUM_ID);
    forum.setName(EXAMPLE_FORUM_NAME);
}

@AfterAll
static void closeRepositoryMocks() {

    // Close the mocks.
    forumRepoMock.close();
    userRepoMock.close();
}

@Test
void testGet() {
    forumService.add(forum, null, null);

    // Request the forum with name from the forum with just an id.
    ScientificForum result = forumService.get(forum);
    assertAll(
            () -> assertEquals(EXAMPLE_FORUM_ID, result.getId()),
            () -> assertEquals(EXAMPLE_FORUM_NAME, result.getName())
    );
}

@Test
void testChange() {

    // A forum containing an old name.
    forum.setDescription(EXAMPLE_FORUM_DESCRIPTION);
    forumService.add(forum, null, null);

    // Create a forum with a new name but the same id.
    ScientificForum newForum = new ScientificForum();
    newForum.setId(EXAMPLE_FORUM_ID);
    newForum.setName(EXAMPLE_FORUM_NAME_ALTERNATIVE);
    newForum.setDescription(EXAMPLE_FORUM_DESCRIPTION);

    // Change the forum to contain the new name.
    forumService.change(newForum);

    // Mock the repository to return the new forum.
    forumRepoMock.when(() -> ScientificForumRepository
            .get(eq(forum), any(Transaction.class))).thenReturn(newForum);

    // Request the changed forum.
    ScientificForum result = forumService.get(forum);

    assertAll(
            () -> assertEquals(EXAMPLE_FORUM_ID, result.getId()),
            () -> assertEquals(EXAMPLE_FORUM_NAME_ALTERNATIVE, result.getName()),
            () -> assertEquals(EXAMPLE_FORUM_DESCRIPTION, result.getDescription())
    );
}

@Test
void testGetEditors() {

    // Create two editors of a forum and one editor of a different forum and the respective forums.
    User editor = new User();
    editor.setId(EXAMPLE_USER_ID_EDITOR);
    User anotherEditor = new User();
```

```
        anotherEditor.setId(EXAMPLE_USER_ID_EDITOR_ALTERNATIVE);
        User notAnEditor = new User();
        notAnEditor.setId(EXAMPLE_USER_ID_NOT_AN_EDITOR);

        // Create another forum for the extra editor.
        ScientificForum anotherForum = new ScientificForum();
        anotherForum.setId(EXAMPLE_FORUM_ID + 1);

        // Add the editors to their forums.
        forumService.addEditor(editor, forum);
        forumService.addEditor(anotherEditor, forum);
        forumService.addEditor(notAnEditor, anotherForum);

        // Mock the underlying repository function.
        List<User> tempEditorsList = Arrays.asList(anotherEditor, editor);
        userRepoMock.when(() -> UserRepository.getList(any(Transaction.class),
                eq(forum))).thenReturn(tempEditorsList);

        List<User> editors = forumService.getEditors(forum);
        assertAll(

                // Assert that the user editor is contained somewhere within
                // the list of editors
                () -> assertTrue(() -> {
                    for (User user : editors) {
                        if (user.getId().equals(editor.getId())) {
                            return true;
                        }
                    }
                    return false;
                }),

                // Assert that the user anotherEditor is contained somewhere within
                // the list of editors
                () -> assertTrue(() -> {
                    for (User user : editors) {
                        if (user.getId().equals(anotherEditor.getId())) {
                            return true;
                        }
                    }
                    return false;
                }),

                // Assert that the user notAnEditor is not contained somewhere within
                // the list of editors
                () -> assertTrue(() -> {
                    for (User user : editors) {
                        if (user.getId().equals(notAnEditor.getId())) {
                            return false;
                        }
                    }
                    return true;
                })
        );
    }
}
```

Listing 9: ScientificForumServiceTest.java

5.1.7 Arbeitspaket *NewSubmission*

Johannes Garstenauer

```
@ExtendWith(MockitoExtension.class)
public class PaperServiceTest {
```

```java
// The required repository.
private static MockedStatic<PaperRepository> paperRepoMocked;

// The required service.
private static PaperService paperService;

// The required DTOs and their values
private static Paper paper;
private static final Integer EXAMPLE_SUBMISSION_ID = 1;
private static final Integer EXAMPLE_VERSION_NUMBER = 10;
private static FileDTO fileDTO;
private static final byte[] EXAMPLE_PDF = new byte[]{};


@BeforeAll
static void init() {

    // Mock repositories and initialize services.
    paperRepoMocked = mockStatic(PaperRepository.class);
    paperService = new PaperService();

    // Mock get to return a paper or file if the ids are correct.
    paperRepoMocked.when(() -> PaperRepository.get(eq(paper),
            any(Transaction.class))).thenReturn(paper);
    paperRepoMocked.when(() -> PaperRepository.getPDF(eq(paper),
            any(Transaction.class))).thenReturn(fileDTO);
}

@BeforeEach
void resetDTOS() {

    // Reset the dto's value's after each test.
    paper = new Paper();
    paper.setSubmissionId(EXAMPLE_SUBMISSION_ID);
    paper.setVersionNumber(EXAMPLE_VERSION_NUMBER);
    fileDTO = new FileDTO();
    fileDTO.setFile(EXAMPLE_PDF);
}

@AfterAll
static void close() {

    // Close the mocks
    paperRepoMocked.close();
}

@Test
void testGet() {
    paperService.add(fileDTO, paper);

    Paper gotten = paperService.get(paper);
    assertAll(
            () -> assertEquals(EXAMPLE_SUBMISSION_ID, gotten.getSubmissionId()),
            () -> assertEquals(EXAMPLE_VERSION_NUMBER, gotten.getVersionNumber())
    );
}

@Test
void testAdd() {
    paperService.add(fileDTO, paper);

    Paper gotten = paperService.get(paper);
    FileDTO gottenFile = paperService.getFile(paper);
    assertAll(
            () -> assertEquals(EXAMPLE_SUBMISSION_ID, gotten.getSubmissionId()),
            () -> assertEquals(EXAMPLE_VERSION_NUMBER, gotten.getVersionNumber()),
            () -> assertEquals(EXAMPLE_PDF, gottenFile.getFile())
    );
}
```

```
    @Test
    void testChange() {

        // A paper containing an old visibility.
        paper.setVisible(true);
        paperService.add(fileDTO, paper);

        // Create a paper with a new visibility but the same ids.
        Paper newPaper = new Paper();
        newPaper.setSubmissionId(EXAMPLE_SUBMISSION_ID);
        newPaper.setVersionNumber(EXAMPLE_VERSION_NUMBER);
        newPaper.setVisible(false);

        // Change the paper to contain the new comment.
        paperService.change(newPaper);

        // Mock the repository request to return the new paper.
        paperRepoMocked.when(() -> PaperRepository
                .get(eq(paper), any(Transaction.class))).thenReturn(newPaper);

        // Request the changed paper.
        Paper result = paperService.get(paper);
        assertAll(
                () -> assertEquals(EXAMPLE_VERSION_NUMBER, result.getVersionNumber()),
                () -> assertEquals(EXAMPLE_SUBMISSION_ID, result.getSubmissionId()),
                () -> assertFalse(result.isVisible())
        );
    }
}
```

Listing 10: PaperServiceTest.java

5.2 Milestone 2

5.2.1 Arbeitspaket *Registration*

Thomas Kirz

```
class PasswordValidatorTest {

    @Test
    void testValidate() {
        PasswordValidator validator = new PasswordValidator();

        // test strings adhering to the password policy:
        // they contains uppercase and lowercase letters, numbers and special characters
        // and they are between 8 and 100 characters long
        String validPassword1 = "12345AaB'";
        String validPassword2 = "??345XöB;";
        String validPassword3 = validPassword2.repeat(4);
        assertAll(
                () -> assertDoesNotThrow(() -> validator.validate(null, null, validPassword1)),
                () -> assertDoesNotThrow(() -> validator.validate(null, null, validPassword2)),
                () -> assertDoesNotThrow(() -> validator.validate(null, null, validPassword3))
        );
    }

    @Test
    void testValidate_invalid() {
        PasswordValidator validator = new PasswordValidator();

        // test strings that do not adhere to the password policy:
        String validPassword1 = "1aA;'";
        String validPassword2 = "12345678";
        String validPassword3 = "Aa;" + "0".repeat(98);
        assertAll(
                () -> assertThrows(ValidatorException.class,
                        () -> validator.validate(null, null, validPassword1)),
                () -> assertThrows(ValidatorException.class,
                        () -> validator.validate(null, null, validPassword2)),
                () -> assertThrows(ValidatorException.class,
                        () -> validator.validate(null, null, validPassword3))
        );
    }
}
```

Listing 11: PasswordValidatorTest.java

```
class UserRepositoryTest {

    private static final String EXAMPLE_EMAIL_ADDRESS = "john.doe@foo.bar";

    @Test
    void testAddEmailExists() throws Exception {
        User user = new User();
        user.setEmailAddress(EXAMPLE_EMAIL_ADDRESS);

        Transaction transaction = new Transaction();
        UserRepository.add(user, transaction);
        assertTrue(UserRepository.emailExists(user, transaction));
        transaction.abort();
    }

    @Test
    void testEmailDoesNotExist() throws Exception {
        User user = new User();
        user.setEmailAddress(EXAMPLE_EMAIL_ADDRESS);

        Transaction transaction = new Transaction();
        UserRepository.remove(user, transaction);
        assertFalse(UserRepository.emailExists(user, transaction));
        transaction.abort();
    }
}
```

Listing 12: RegistrationServiceTest.java

```
class UserRepositoryTest {

    private static final String EXAMPLE_EMAIL_ADDRESS = "john.doe@foo.bar";

    @Test
    void testAddEmailExists() throws Exception {
        User user = new User();
        user.setEmailAddress(EXAMPLE_EMAIL_ADDRESS);

        Transaction transaction = new Transaction();
        UserRepository.add(user, transaction);
        assertTrue(UserRepository.emailExists(user, transaction));
        transaction.abort();
    }

    @Test
    void testEmailDoesNotExist() throws Exception {
        User user = new User();
        user.setEmailAddress(EXAMPLE_EMAIL_ADDRESS);

        Transaction transaction = new Transaction();
        UserRepository.remove(user, transaction);
        assertFalse(UserRepository.emailExists(user, transaction));
        transaction.abort();
    }
}
```

Listing 13: UserRepositoryTest.java

5.2.2 Arbeitspaket *Profile*

Johann Schicho

```
@ExtendWith(MockitoExtension.class)
public class ProfileBackingTest {

    @Mock
    SessionInformation currentSession;

    @InjectMocks
    ProfileBacking profileBackingOwner = new ProfileBacking();


    @Test
    void testHasOwnerEditRights() {
        User profileOwner = new User();
        profileOwner.setId(123456);
        profileBackingOwner.setUser(profileOwner);

        when(currentSession.getUser()).thenReturn(profileOwner);

        assertTrue(profileBackingOwner.hasViewerEditRights());
    }

    @Test
    void testHasAdminEditRights() {
        User profileOwner = new User();
        profileOwner.setId(123456);
        profileBackingOwner.setUser(profileOwner);

        User admin = new User();
        admin.setId(1);
        admin.setAdmin(true);

        when(currentSession.getUser()).thenReturn(admin);

        assertTrue(profileBackingOwner.hasViewerEditRights());
    }

    @Test
    void testHasEditorEditRights() {
        User profileOwner = new User();
        profileOwner.setId(123456);
        profileBackingOwner.setUser(profileOwner);

        User editor = new User();
        editor.setId(2);

        when(currentSession.getUser()).thenReturn(editor);

        assertFalse(profileBackingOwner.hasViewerEditRights());
    }
}
```

Listing 14: ProfileBackingTest.java

5.2.3 Arbeitspaket *Administration*

```java
class SystemSettingsRepositoryTest {

   private static SystemSettings systemSettings;

   private static Transaction transaction;

   @BeforeAll
   static void initSystemSettings() {
      transaction = new Transaction();
      systemSettings = new SystemSettings();
      systemSettings.setImprint("imprint");
      systemSettings.setCompanyName("company");
      systemSettings.setHeadlineWelcomePage("welcome");
      systemSettings.setMessageWelcomePage("hello world");
      systemSettings.setStyle("dark.css");
   }

   static boolean allFieldsTheSame(SystemSettings systemSettings1,
                           SystemSettings systemSettings2) {
      return
            systemSettings1.getImprint()
                  .equals(systemSettings2.getImprint())
                  && systemSettings1.getCompanyName()
                  .equals(systemSettings2.getCompanyName())
                  && systemSettings1.getHeadlineWelcomePage()
                  .equals(systemSettings2.getHeadlineWelcomePage())
                  && systemSettings1.getMessageWelcomePage()
                  .equals(systemSettings2.getMessageWelcomePage())
                  && systemSettings1.getStyle()
                  .equals(systemSettings2.getStyle());
   }

   @Test
   void testUpdateAndGet() throws DataNotWrittenException {
      SystemSettingsRepository.updateSettings(systemSettings, transaction);
      SystemSettings savedSettings = SystemSettingsRepository
            .getSettings(transaction);
      assertTrue(allFieldsTheSame(systemSettings, savedSettings));
   }

   @Test
   void testUpdateWithNull() {
      String oldImprint = systemSettings.getImprint();
      systemSettings.setImprint(null);
      assertThrows(InvalidFieldsException.class, () -> {
         SystemSettingsRepository.updateSettings(systemSettings,
               transaction);
      });
      systemSettings.setImprint(oldImprint);
   }

   @Test
   void testGetAndSetLogo() throws DataNotWrittenException {
      FileDTO file = new FileDTO();
      file.setFile(new byte[]{});
      SystemSettingsRepository.setLogo(file, transaction);
      FileDTO returnedFile = SystemSettingsRepository.getLogo(transaction);
      assertAll(
            () -> {
               assertNotNull(returnedFile);
            },
            () -> {
               assertEquals(0, returnedFile.getFile().length);
            }
      );
   }
```

```java
    @Test
    void testSetLogoWithNull() {
        FileDTO file = new FileDTO();
        assertThrows(InvalidFieldsException.class, () ->
                SystemSettingsRepository.setLogo(file, transaction));
    }

    @AfterAll
    static void rollbackChanges() {
        transaction.abort();
    }

}
```

Listing 15: SystemSettingsRepositoryTest.java

5.2.4 Arbeitspaket *Science Field*

Stefanie Gürster

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

    private static final String NAME_MATH_SF = "Math";

    static MockedStatic<ScienceFieldRepository> mockedStatic;

    @BeforeAll
    static void mockRepository() {
        mockedStatic = mockStatic(ScienceFieldRepository.class);
    }

    @AfterAll
    static void closeMock() {
        mockedStatic.close();
    }

    @Test
    void testAddScienceField() {
        ScienceFieldService scienceFieldService = new ScienceFieldService();
        ScienceField mathField = new ScienceField();
        mathField.setName(NAME_MATH_SF);
        scienceFieldService.add(mathField);

        assertEquals(mathField, scienceFieldService.get(NAME_MATH_SF));
    }

    @Test
    void testRemoveScienceField() {
        ScienceFieldService scienceFieldService = new ScienceFieldService();
        ScienceField mathField = new ScienceField();
        mathField.setName(NAME_MATH_SF);
        scienceFieldService.add(mathField);

        assertEquals(mathField, scienceFieldService.get(NAME_MATH_SF));

        scienceFieldService.remove(mathField);

        assertNull(scienceFieldService.get(NAME_MATH_SF));
    }

    @Test
    void testGetScienceFieldsOfUser() {
        User user = new User();
        user.setId(1);
        ScienceField mathField = new ScienceField();
        mathField.setName("Math");
        ScienceField csField = new ScienceField();
        csField.setName("Computer Science");

        ResultListParameters params = new ResultListParameters();

        List<ScienceField> userScienceFields = Arrays.asList(mathField, csField);

        mockedStatic.when(() -> ScienceFieldRepository.getList(eq(user), any(), eq(params)))
        .thenReturn(userScienceFields);

        assertEquals(userScienceFields, ScienceFieldService.getList(user, params));
    }
}
```

Listing 16: ScienceFieldServiceTest.java

5.3 Milestone 3

5.3.1 Arbeitspaket *ErrorPage*

Thomas Kirz

```
class ErrorPageBackingTest {

    @Mock ConfigPropagator configPropagator;

    @Test
    void testErrorPageBacking() {
        ErrorPageBacking errorPageBacking = new ErrorPageBacking();
        ErrorMessage errorMessage = new ErrorMessage("test", "test");
        errorPageBacking.setErrorMessage(errorMessage);
        assertEquals(errorMessage, errorPageBacking.getErrorMessage());
    }

    @Test
    void testDevelopmentModeOn() {
        ErrorPageBacking errorPageBacking = new ErrorPageBacking();
        when(configPropagator.getProperty("DEBUG_AND_TEST_MODE")).thenReturn("true");
        assertTrue(errorPageBacking.isDevelopmentMode());
    }

    @Test
    void testDevelopmentModeOff() {
        ErrorPageBacking errorPageBacking = new ErrorPageBacking();
        when(configPropagator.getProperty("DEBUG")).thenReturn("false");
        assertFalse(errorPageBacking.isDevelopmentMode());

    }

}
```

Listing 17: ErrorPageBackingTest.java

```
class ErrorMessageTest {

    private static final String EXAMPLE_MESSAGE = "ERROR: This is an example error message";
    private static final String EXAMPLE_STACKTRACE = "java.lang.Exception: This is an example stacktrace";

    @Test
    void testErrorMessage() {
        ErrorMessage errorMessage = new ErrorMessage(EXAMPLE_MESSAGE, EXAMPLE_STACKTRACE);
        assertAll(
                () -> assertEquals(EXAMPLE_MESSAGE, errorMessage.getMessage()),
                () -> assertEquals(EXAMPLE_STACKTRACE, errorMessage.getStackTrace()),
                () -> assertEquals(MessageCategory.FATAL, errorMessage.getCategory())
        );
    }

}
```

Listing 18: ErrorMessageTest.java

5.3.2 Arbeitspaket *Email*

Johann Schicho

```
public class EmailUtilTest {

    private static final String emailSubjectCCnoBody
        = "mailto:info@example.com?subject=important&cc=cc@example.com";
    private static final String emailSubjectCCBody
        = "mailto:info@example.com?subject=important&body=Important%20Mail%0D%0ARegards&cc=cc@example.com";
    private static final String emailOnlyAddress = "mailto:info@example.com";
    private static final String emailMultipleRecipients = "mailto:info@example.com,service@example.com";

    private static final String emailAddressRecipient = "info@example.com";
    private static final String emailAddressRecipient2 = "service@example.com";
    private static final String emailAddressCC = "cc@example.com";
    private static final String subject = "important";
    private static final String body = "Important Mail\nRegards";

    @Test
    void testGenerateMailToLinkNoBody() {
        String mailto = EmailUtil.generateMailToLink(new String[]{emailAddressRecipient},
            new String[]{emailAddressCC}, subject, null);
        assertEquals(emailSubjectCCnoBody, mailto);
    }

    @Test
    void testGenerateMailToLinkWithBody() {
        String mailto = EmailUtil.generateMailToLink(new String[]{emailAddressRecipient},
            new String[]{emailAddressCC}, subject, body);
        assertEquals(emailSubjectCCBody, mailto);
    }

    @Test
    void testGenerateMailToEmptyEmail() {
        String mailto = EmailUtil.generateMailToLink(new String[]{emailAddressRecipient}, null,null,null);
        assertEquals(emailOnlyAddress, mailto);
    }

    @Test
    void testGenerateMailToMultipleRecipients(){
        String mailto = EmailUtil.generateMailToLink(new String[]{emailAddressRecipient, emailAddressRecipient2},
            null, null, null);
        assertEquals(emailMultipleRecipients, mailto);
    }
}
```

Listing 19: EmailUtilTest.java

5.3.3 Arbeitspaket *NewUser*

Sebastian Vogt

```
@ExtendWith(MockitoExtension.class)
```

```java
class NewUserBackingTest {

    static NewUserBacking backing = new NewUserBacking();

    @Test
    void testSaveUser() throws NoSuchFieldException, IllegalAccessException {
        RegistrationService mockService
                = Mockito.mock(RegistrationService.class);

        User dummyUser = new User();

        Field serviceField
                = NewUserBacking.class.getDeclaredField("registrationService");
        serviceField.setAccessible(true);
        serviceField.set(backing, mockService);

        Field userField
                = NewUserBacking.class.getDeclaredField("newUser");
        userField.setAccessible(true);
        userField.set(backing, dummyUser);

        backing.saveUser();

        Mockito.verify(mockService).registerByAdmin(dummyUser);
    }

}
```

Listing 20: NewUserBackingTest.java

5.3.4 Arbeitspaket *UserList*

Stefanie Gürster

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

    private final UserService userService = new UserService();

    static MockedStatic<UserRepository> mockedRepo;

    @BeforeAll
    static void mockRepository() {
        mockedRepo = mockStatic(UserRepository.class);
    }

    @AfterAll
    static void closeMock() {
        mockedRepo.close();
    }

    @Test
    void testGetUser() {
        User onlyId = new User();
        onlyId.setId(1);
        User fullData = new User();
        fullData.setId(1);
        fullData.setFirstName("Fabi");
        fullData.setLastName("Dorfner");

        mockedRepo.when(() -> UserRepository.get(eq(onlyId), any())).thenReturn(fullData);

        assertEquals(fullData, userService.get(onlyId));
    }

    @Test
    void testChangeUserName() {
        User newUser = new User();
        newUser.setId(1);
        newUser.setFirstName("Steffi");
        newUser.setLastName("Dorfner");

        User oldUser = new User();
        oldUser.setId(1);
        oldUser.setFirstName("Stefanie");
        oldUser.setLastName("Gürster");

        mockedRepo.when(() -> UserRepository.get(eq(oldUser), any())).thenReturn(oldUser);

        assertEquals(oldUser, userService.get(oldUser));

        mockedRepo.when(() -> UserRepository.get(eq(oldUser), any())).thenReturn(newUser);

        userService.change(newUser);

        assertEquals(newUser, userService.get(oldUser));
    }

    @Test
    void testChangeEmailVerified() {
        User user = new User();
        user.setId(1);
        user.setEmailAddress("stefanie@gmail.com");

        User changedEmail = new User();
        changedEmail.setId(1);
        changedEmail.setEmailAddress("st@gmail.com");

        Verification verification = new Verification();
        verification.setUserId(1);
        verification.setVerified(false);

        mockedRepo.when(() -> UserRepository.getVerification(eq(user), any())).thenReturn(verification);

        mockedRepo.when(() -> UserRepository.get(eq(user), any())).thenReturn(changedEmail);

        userService.change(changedEmail);

        assertFalse(userService.getVerification(user).isVerified());
    }
}
```

Listing 21: UserServiceTest.java

5.3.5 Arbeitspaket *NewReview*

Johannes Garstenauer

```
@ExtendWith(MockitoExtension.class)
public class ReviewServiceTest {

    // The required repository.
    private static MockedStatic<ReviewRepository> reviewRepoMocked;

    // The required service.
    private static ReviewService reviewService;

    // The required DTOs and their values
    private static Review review;
    private static FileDTO pdf;
    private static final Integer EXAMPLE_SUBMISSION_ID = 1;
    private static final Integer EXAMPLE_REVIEWER_ID = 10;
```

```java
private static final Integer EXAMPLE_PAPER_VERSION = 100;
private static final byte[] EXAMPLE_PDF = new byte[]{};
private static final String EXAMPLE_OLD_COMMENT = "Very good!";
private static final String EXAMPLE_NEW_COMMENT = "MY EYES ARE BLEEDING";

@BeforeAll
static void init() {
    // Mock repository and initialize service.
    reviewRepoMocked = mockStatic(ReviewRepository.class);
    reviewService = new ReviewService();

    // Mock get to return a review if the ids are correct.
    reviewRepoMocked.when(() -> ReviewRepository.get(eq(review),
            any(Transaction.class))).thenReturn(review);
}

@BeforeEach
void resetDTOS() {

    // Reset the dto's value's after each test.
    review = new Review();
    review.setPaperVersion(EXAMPLE_PAPER_VERSION);
    review.setSubmissionId(EXAMPLE_SUBMISSION_ID);
    review.setReviewerId(EXAMPLE_REVIEWER_ID);

    pdf = new FileDTO();
    pdf.setFile(EXAMPLE_PDF);
}

@AfterAll
static void close() {

    // Close the mocks
    reviewRepoMocked.close();
}

@Test
void testGet() {
    reviewService.add(review, pdf);

    Review gotten = reviewService.get(review);
    assertAll(
            () -> assertEquals(EXAMPLE_PAPER_VERSION, gotten.getPaperVersion()),
            () -> assertEquals(EXAMPLE_REVIEWER_ID, gotten.getReviewerId()),
            () -> assertEquals(EXAMPLE_SUBMISSION_ID, gotten.getSubmissionId())
    );
}

@Test
void testAdd() {
    reviewService.add(review, pdf);

    Review gotten = reviewService.get(review);
    assertAll(
            () -> assertEquals(EXAMPLE_PAPER_VERSION, gotten.getPaperVersion()),
            () -> assertEquals(EXAMPLE_REVIEWER_ID, gotten.getReviewerId()),
            () -> assertEquals(EXAMPLE_SUBMISSION_ID, gotten.getSubmissionId())
    );
}

@Test
void testGetFile() {
    reviewService.add(review, pdf);

    FileDTO gotten = reviewService.getFile(review);
    assertEquals(EXAMPLE_PDF, gotten.getFile());
}

@Test
```

```java
    void testChange() {

        // A review containing an old comment.
        review.setComment(EXAMPLE_OLD_COMMENT);
        reviewService.add(review, pdf);

        // Create a review with a new comment but the same ids.
        Review newReview = new Review();
        newReview.setPaperVersion(EXAMPLE_PAPER_VERSION);
        newReview.setSubmissionId(EXAMPLE_SUBMISSION_ID);
        newReview.setReviewerId(EXAMPLE_REVIEWER_ID);
        newReview.setComment(EXAMPLE_NEW_COMMENT);

        // Change the review to contain the new comment.
        reviewService.change(newReview);

        // Mock the get request to return the new review.
        reviewRepoMocked.when(() -> ReviewRepository
                .get(eq(review), any(Transaction.class))).thenReturn(newReview);

        // Request the changed review.
        Review result = reviewService.get(review);

        assertAll(
                () -> assertEquals(EXAMPLE_PAPER_VERSION, result.getPaperVersion()),
                () -> assertEquals(EXAMPLE_REVIEWER_ID, result.getReviewerId()),
                () -> assertEquals(EXAMPLE_SUBMISSION_ID, result.getSubmissionId()),
                () -> assertEquals(EXAMPLE_NEW_COMMENT, result.getComment())
        );
    }
}
```

Listing 22: ReviewServiceTest.java

✓