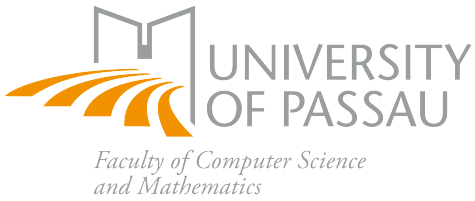

Implementierungsplan

TalentTauscher

Betreuer: Prof. Dr. Christian Bachmaier

Team 1

Phase	Verantwortlich
Pflichtenheft	Leon Föckersperger
Entwurf & Spezifikation	Jakob Edmaier
Implementierung & Validierung	David Sturm
Abschlusspräsentation	Leon Föckersperger

Version 1.0

29. Mai 2024

Inhaltsverzeichnis

1 Einleitung

Autor: Leon Föckersperger
Der Implementierungszeitraum dieses Projekts erstreckt sich vom 28.05.2024 bis zum 25.06.2024 und gliedert sich in drei Milestones. Jeder Milestone beinhaltet spezifische Arbeitspakete, die von der Benutzeroberfläche bis zur Datenverarbeitung reichen. Diese umfassen die Entwicklung von Facelets, Backing Beans, Service- und DAO-Methoden sowie DTOs und gegebenenfalls Validatoren und Converter. Die Einteilung in klar definierte Milestones ermöglicht es den Teammitgliedern, weitgehend unabhängig und effizient zu arbeiten. Schlüsselkomponenten wie der ExceptionHandler und der ConnectionPool werden zu Projektbeginn von erfahrenen Teammitgliedern entwickelt, um eine konsistente Basis für nachfolgende Entwicklungen zu schaffen und die Softwarearchitektur zu vereinheitlichen.

Milestone 1 (28.05.2024 bis 04.06.2024): Der Fokus liegt auf der Einrichtung der Infrastrukturkomponenten wie dem `ConnectionPool`, der Datenbankinitialisierung, dem Transaktionsmanagement, dem `LoggerProducer` und der `Config`-Klasse zum Einlesen von Konfigurationsdaten. Zusätzlich wird die `PasswordAuthentication` implementiert, die für die sichere Verschlüsselung von Passwörtern verantwortlich ist. Die Fehlerbehandlung wird durch den `AppExceptionHandler` gewährleistet. Grundlegende Systemfunktionen, einschließlich Systemstart durch den `StartAndStopListener`, Login-Prozess sowie die Hauptkomponenten für das Frontend (Template, `Footer` und `Header`), werden ebenfalls implementiert.

Milestone 2 (04.06.2024 bis 11.06.2024): Der Hauptfokus liegt auf der Implementierung der Anwendungskomponenten wie dem Benutzerprofil (`ProfileBean`), dem Anzeigenmanagement (`EditAdvertisementBean`) und der Startseite (`AllAdvertisementsBean`). Es wird ein Zugriffskontrollsystem über den `TrespassListener` implementiert. Die für viele Anwendungsbereiche zentralen paginierten und sortierbaren Tabellenkomponenten (`PaginatedDataModel`) werden fertiggestellt.

Milestone 3 (11.06.2024 bis 18.06.2024): In dieser letzten Phase konzentriert sich die Arbeit auf die Implementierung der verbleibenden Komponenten, insbesondere der Adminfunktionen (`UserAdministrationBean`) und der Bearbeitung von Footer-Seiten (Impressum, Datenschutz, Kontakt) sowie der Verwaltung von Benutzeranfragen und Anzeigen (`WriteRequestBean` und `OutgoingRequestsBean`).

Vorläufige Abgabe (25.06.2024): Das System wird in seiner vorläufig vollständigen Form zusammen mit dem Handbuch und dem Implementierungsbericht abgegeben, um eine umfassende Dokumentation und Nachvollziehbarkeit sicherzustellen.
Diese klar definierte Struktur und die detaillierte Planung der Milestones ermöglichen eine effiziente und systematische Entwicklung der Webapplikation. Trotz der Festlegung auf das Wasserfallmodell ermöglicht der agile Ansatz innerhalb des Teams Flexibilität, sodass Arbeitspakete bei Bedarf zwischen den Milestones verschoben werden können und Zuständigkeiten sowie Zeitpläne der Arbeitspakete flexibel gehandhabt werden.

2 Arbeitspakete

Autor: Jakob Edmaier

Im Folgenden sind alle Arbeitspakete nach Milestones gegliedert aufgelistet. Dabei sind jeweils die betroffenen Artefakte, Abhängigkeiten zwischen Arbeitspaketen, ein zeitlicher Rahmen und das verantwortliche Teammmitglied angegeben.

Da alle Klassen systemweit eindeutige Namen haben, verzichten wir zugunsten der Prägnanz auf die Angabe des jeweiligen Packages. Wenn in einem Arbeitspaket nicht die ganze Klasse implementiert werden muss, sind die zu implementierenden `public`-Methoden einzeln aufgeführt. Die Implementierung von Exceptions und DTO-Klassen ist trivial und wird deshalb in der Tabelle nicht erwähnt.

Die **Abhängig-von**-Relation ist als transitiv anzusehen, d.h. wenn C abhängig von B ist und B abhängig von A, dann ist auch C abhängig von A, ohne dass dies notwendigerweise explizit in der Tabelle steht.

2.1 Milestone 1

ID	Arbeitspaket	Artefakte	Abhängig von	Beginn	Ende	Aufwand	Zuständig
101	Konfiguration	**AppConfig** **application.properties**		23.05.2024, 20:00	23.05.2024, 23:00	1h	Edmaier
102	Logging	**LoggerProducer** **logging.properties**		28.05.2024, 15:00	28.05.2024, 17:00	1h	Sturm
103	Systemstart	**StartAndStopListener** **StartAndStopBusiness** **StartAndStopPersistence**		20.05.2024, 9:00	20.05.2024, 11:00	1.5h	Föckersperger
104	Datenbankverbindung	**ConnectionPool**	101	20.05.2024, 11:00	20.05.2024, 15:30	3.5h	Föckersperger
105	Transaktions-management	**TransactionFactory** **PostgreSQLTransaction**	104	29.05.2024, 17:00	29.05.2024, 19:00	1h	Edmaier
106	Datenbanksetup	**PostgreSQLSetup** **StoreSetupFactory**	104	21.05.2024, 8:00	21.05.2024, 18:00	8h	Föckersperger
107	DAO	**PostgreSQLDAOFactory** **PostgreSQLDAO**	104	30.05.2024, 9:00	30.05.2024, 15:00	4h	Edmaier
108	Session Management	**UserSession** **UserService** .getAuthDataOfUser(UserAuthenticationDTO) **PostgreSQLUserDAO** .getAuthDataOfUser(UserAuthenticationDTO)	105, 106, 107	31.05.2024, 9:00	31.05.2024, 15:00	3h	Edmaier
109	Hashing	**PasswordAuthentication**		30.05.2024, 12:00	30.05.2024, 18:00	3h	Edmaier
110	JFUtils	**JFUtils**		03.06.2024, 09:00	03.06.2024, 12:00	2h	Edmaier

ID	Arbeitspaket	Artefakte	Abhängig von	Beginn	Ende	Aufwand	Zuständig
111	Template	**skeleton.xhtml** **header.xhtml** **HeaderBean** **footer.xhtml** **help.xhtml** **HelpBean** **SystemDataService** .get() **PostgreSQLSystemDataDAO** .getSystemData()		16.05.2024, 10:00	16.05.2024, 18:00	6h	Föckersperger
112	Emails	**EmailDispatcher**	101	20.05.2024, 17:00	28.05.2024, 19:00	3h	Sturm
113	Confirm-Dialog	**confirmButton.xhtml** **confirmDialog.xhtml**		23.05.2024, 10:00	30.05.2024, 23:00	5h	Edmaier
114	Fehlerbehandlung	**AppExceptionHandler** **AppExceptionHandlerFactory** **error.xhtml** **error404.xhtml**		30.05.2024, 14:00	31.05.2024, 20:00	6h	Sturm
115	Verifizierungstokens	**VerificationTokenGenerator**		30.05.2024, 14:00	30.05.2024, 16:00	1h	Sturm
116	Registrierung	**register.xhtml** **RegisterBean** **EmailFormatValidator** **EmailUniqueValidator** **NicknameFormatValidator** **NicknameUniqueValidator** **UserService** .create(UserDetailsDTO) .isNicknameUnique(UserAuthenticationDTO) .isEmailUnique(UserAuthenticationDTO) **PostgreSQLUserDAO** .createUser(UserDetailsDTO) .findUserByNickname(UserAuthenticationDTO) .findUserByEmail(UserAuthenticationDTO .putEmailVerificationToken(VerificationTokenDTO)	105, 106, 107, 109	31.05.2024, 10:00	02.06.2024, 15:00	8h	Sturm

Tabelle 3: Arbeitspakete Milestone 1

2.2 Milestone 2

Für alle Arbeitspakete in diesem Milestone bis auf DataTable ist ein funktionierender Datenbankzugriff notwendig. Daher werden die Pakete 101, 103, 104, 105, 106 und 107 als Abhängigkeiten vorausgesetzt.

ID	Arbeitspaket	Artefakte	Abhängig von	Beginn	Ende	Aufwand	Zuständig
201	DataTable	**PaginatedDataModel** **dynamicDataTable.xhtml** **sortFilterColumn.xhtml**		14.05.2024	15.05.2024	10h	Föckersperger
202	Email-Verifizierung	**emailVerification.xhtml** **EmailVerificationBean** **UserService** .verifyEmail(VerificationTokenDTO) **PostgreSQLUserDAO** .setEmailVerified(VerificationTokenDTO)	110, 112, 115, 116	05.06.2024, 17:00	06.06.2024, 12:00	4h	Edmaier
203	Login	**login.xhtml** **LoginBean** **UserService** .authenticate(LoginDTO) **PostgreSQLUserDAO** .findUser(LoginDTO)	108, 109, 116	03.06.2024, 14:00	03.06.2024, 22:00	5h	Sturm
204	Passwort zurücksetzen	**passwordResetInput.xhtml** **passwordResetRequest.xhtml** **register.xhtml** **PasswordResetInputBean** **PasswordResetRequestBean** **EmailBuilder** **UserService** .resetPassword(PasswordResetDTO) .putPasswordResetToken(VerificationTokenDTO) **PostgreSQLUserDAO** .resetPassword(PasswordResetDTO) .putPasswordResetToken(VerificationTokenDTO)	109, 110, 112, 115, 116	06.06.2024, 09:00	06.06.2024, 19:00	5h	Edmaier
205	Benutzerprofil	**profile.xhtml** **ProfileBean** **AvatarValidator** **UserService** .update(UserDetailsDTO) .getUserDetails(UserDetailsDTO) .delete(UserDTO) **PostgreSQLUserDAO** .updateUser(UserDetailsDTO) .getUserDetails(UserDetailsDTO) .deleteUser(UserDTO) .verifyAllUsers() **SystemDataService** .update(SystemDataDTO) **PostgreSQLSystemDataDAO** .updateSystemData(SystemDataDTO)	109, 113, 116, 203	06.06.2024, 10:00	09.06.2024, 22:00	8h	Sturm

ID	Arbeitspaket	Artefakte	Abhängig von	Beginn	Ende	Aufwand	Zuständig
206	Anzeigen erstellen, bearbeiten, löschen	**editAdvertisement.xhtml** **AdvertisementImageValidator** **EditAdvertisementBean** **AdvertisementService** .create(AdvertisementDetailsDTO) .update(AdvertisementDetailsDTO) .delete(AdvertisementDetailsDTO) **PostgreSQLAdvertisementDAO** .createAdvertisement(AdvertisementDetailsDTO) .updateAdvertisement(AdvertisementDetailsDTO) .deleteAdvertisement(AdvertisementDetailsDTO)	203	07.06.2024, 09:00	07.06.2024, 19:00	6h	Edmaier
207	Eigene Anzeigen	**createdAdvertisements.xhtml** **CreatedAdvertisementsBean** AdvertisementService .getCreatedAdvertisementCount(UserDTO, PaginationDTO) .getCreatedAdvertisements(UserDTO, PaginationDTO) **PostgreSQLAdvertisementDAO** .getCreatedAdvertisementCount(UserDTO, PaginationDTO) .getCreatedAdvertisements(UserDTO, PaginationDTO)	201, 206	07.06.2024, 10:00	07.06.2024, 16:00	4h	Föckersperger
208	Startseite (alle Anzeigen)	**allAdvertisements.xhtml** **AllAdvertisementsBean** **AdvertisementService** .getCount(PaginationDTO) .getAdvertisements(PaginationDTO) **PostgreSQLAdvertisementDAO** .getCount(PaginationDTO) .getAdvertisements(PaginationDTO)	201, 206	23.05.2024, 8:00	23.05.2024, 15:00	5h	Föckersperger
209	Anzeigendetails	**advertisementDetails.xhtml** **AdvertisementDetailsBean** **AdvertisementService** .getAdvertisementDetails(AdvertisementDetailsDTO) **PostgreSQLAdvertisementDAO** .getAdvertisementDetails(AdvertisementDetailsDTO)	208	07.06.2024, 09:00	10.06.2024, 12:00	4h	Edmaier
210	TrespassListener	**TrespassListener**	108	04.06.2024, 10:00	04.06.2024, 20:00	5h	Sturm

Tabelle 6: Arbeitspakete Milestone 2

2.3 Milestone 3

Für alle Arbeitspakete in diesem Milestone ist ein funktionierender Datenbankzugriff notwendig. Daher werden die Pakete 101, 103, 104, 105, 106 und 107 als Abhängigkeiten vorausgesetzt.

ID	Arbeitspaket	Artefakte	Abhängig von	Beginn	Ende	Aufwand	Zuständig
301	Benutzerverwaltung	**userAdministration.xhtml** **UserAdministrationBean** **UserService** .getUsers(PaginationDTO) .getUserCount(PaginationDTO) **PostgreSQLUserDAO** .getUsers(PaginationDTO) .getCount(PaginationDTO)	201	10.06.2024, 14:00	10.06.2024, 22:00	5h	Sturm
302	Benutzer erstellen, bearbeiten, löschen	**editUser.xhtml** **EditUserBean**	109	11.06.2024, 10:00	13.06.2024, 18:00	6h	Sturm
303	Anfrage schreiben	**writeRequest.xhtml** **WriteRequestBean** **RequestService** .createRequest(RequestDTO) **AdvertisementService** .getAdvertisement(AdvertisementDTO) **EmailBuilder** .buildIncomingRequestEmail(ExtendedRequestDTO) **PostgreSQLRequestDAO** .createRequest(RequestDTO)	110, 112	11.06.2024, 09:00	11.06.2024, 13:00	3h	Föckersperger
304	Ausgehende Anfragen	**outgoingRequests.xhtml** **OutgoingRequestsBean** **StringLengthConverter** **RequestService** .getOutgoingRequestsForUser(UserDTO, PaginationDTO) .getOutgoingRequestsCount(UserDTO, PaginationDTO) **PostgreSQLRequestDAO** .getOutgoingRequests(UserDTO, PaginationDTO) .getOutgoingRequestsCount(UserDTO, PaginationDTO)	201	11.06.2024, 13:00	11.06.2024, 18:00	3h	Föckersperger

ID	Arbeitspaket	Artefakte	Abhängig von	Beginn	Ende	Aufwand	Zuständig
305	Eingehende Anfragen	**incomingRequests.xhtml** **IncomingRequestsBean** **RequestService** .getIncomingRequestsForUser(UserDTO, PaginationDTO) .getIncomingRequestsCount(UserDTO, PaginationDTO) **PostgreSQLRequestDAO** .getIncomingRequests(UserDTO, PaginationDTO) .getIncomingRequestsCount(UserDTO, PaginationDTO)	201	12.06.2024, 9:00	12.06.2024, 13:00	3h	Föckersperger
306	Anfragen annehmen	**writeResponse.xhtml** **WriteResponseBean** **RequestService** .getRequest(RequestDTO) .updateRequest(RequestDTO) **EmailBuilder** .buildIncomingResponseEmail(ExtendedRequestDTO) **PostgreSQLRequestDAO** .getRequest(RequestDTO) .updateRequest(RequestDTO) **PostgreSQLUserDAO** .transferTalentPoints(RequestDTO)	110, 112	13.06.2024, 09:00	14.06.2024, 18:00	4h	Edmaier
307	Inanspruchgenommene Anzeigen	**cleaimedAdvertisements.xhtml** **ClaimedAdvertisementsBean** **AdvertisementService** .getClaimedAdvertisementsForUser(UserDTO, PaginationDTO) .getClaimedAdvertisementsCount(UserDTO, PaginationDTO) **PostgreSQLAdvertisementDAO** .getClaimedAdvertisementsOfUser(UserDTO, PaginationDTO) .getClaimedAdvertisementsCount(UserDTO, PaginationDTO)	201	13.06.2024, 12:00	14.06.2024, 18:00	3h	Edmaier
308	Datenschutz	**privacyPolicy.xhtml** **PrivacyPolicyBean**		15.06.2024, 09:00	15.06.2024, 18:00	2h	Edmaier
309	Impressum	**imprint.xhtml** **ImprintBean**		15.06.2024, 09:00	15.06.2024, 18:00	2h	Edmaier
310	Kontakt	**contactUs.xhtml** **ContactBean**		12.06.2024, 13:00	12.06.2024, 16:00	2h	Föckersperger

Tabelle 9: Arbeitspakete Milestone 3

3 Abhängigkeiten

Autor: Leon Föckersperger

Um die verschiedenen Abhängigkeiten innerhalb unseres Projekts darzustellen, verwenden wir PERT-Diagramme. Diese Diagramme unterstützen uns dabei, die zeitlichen Abfolgen und Abhängigkeiten der einzelnen Meilensteine klar und übersichtlich zu visualisieren. PERT-Diagramme bieten eine grafische Darstellung der Projektaktivitäten und verdeutlichen die Beziehungen zwischen den verschiedenen Arbeitspaketen.

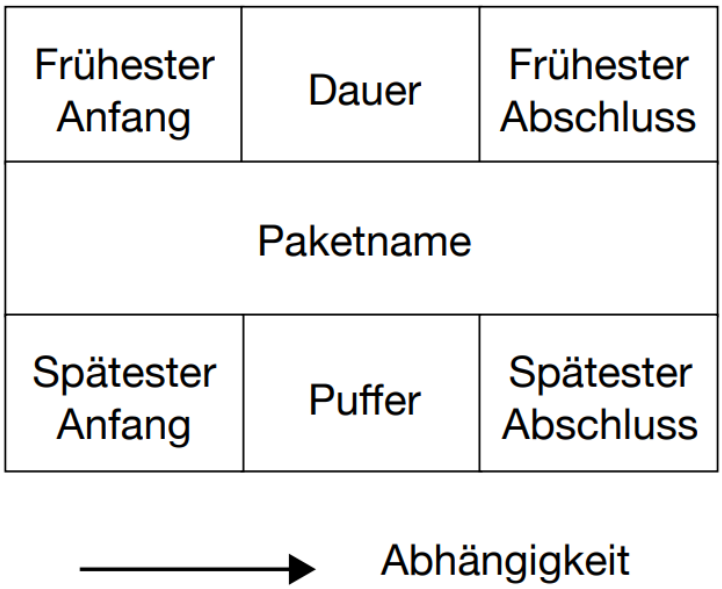

Abbildung 1: Legende der PERT-Diagramme

Die Legende in Abbildung 1 bietet eine Übersicht über die in den PERT-Diagrammen verwendeten Symbole und Elemente. Diese Legende ist essentiell, um die Diagramme korrekt zu interpretieren. Zusätzlich ist der kritische Pfad in den Diagrammen rot markiert.

In Abbildung 2 sehen wir die PERT-Diagramme, die die verschiedenen Meilensteine unseres Projekts darstellen. Jeder Knoten im Diagramm repräsentiert ein wichtiges Arbeitspaket, während die Pfeile die Abhängigkeiten und zeitlichen Beziehungen zwischen diesen Paketen anzeigen. Dieses Diagramm ist ein wichtiges Werkzeug zur Planung und Überwachung des Projektfortschritts. Es hilft uns, potenzielle Engpässe frühzeitig zu erkennen und entsprechend zu reagieren.

Die Verwendung von PERT-Diagrammen ermöglicht eine effektive Projektplanung und -steuerung, da sie eine klare und detaillierte Übersicht über die zeitlichen Abhängigkeiten und den kritischen Pfad bieten. Dies ist

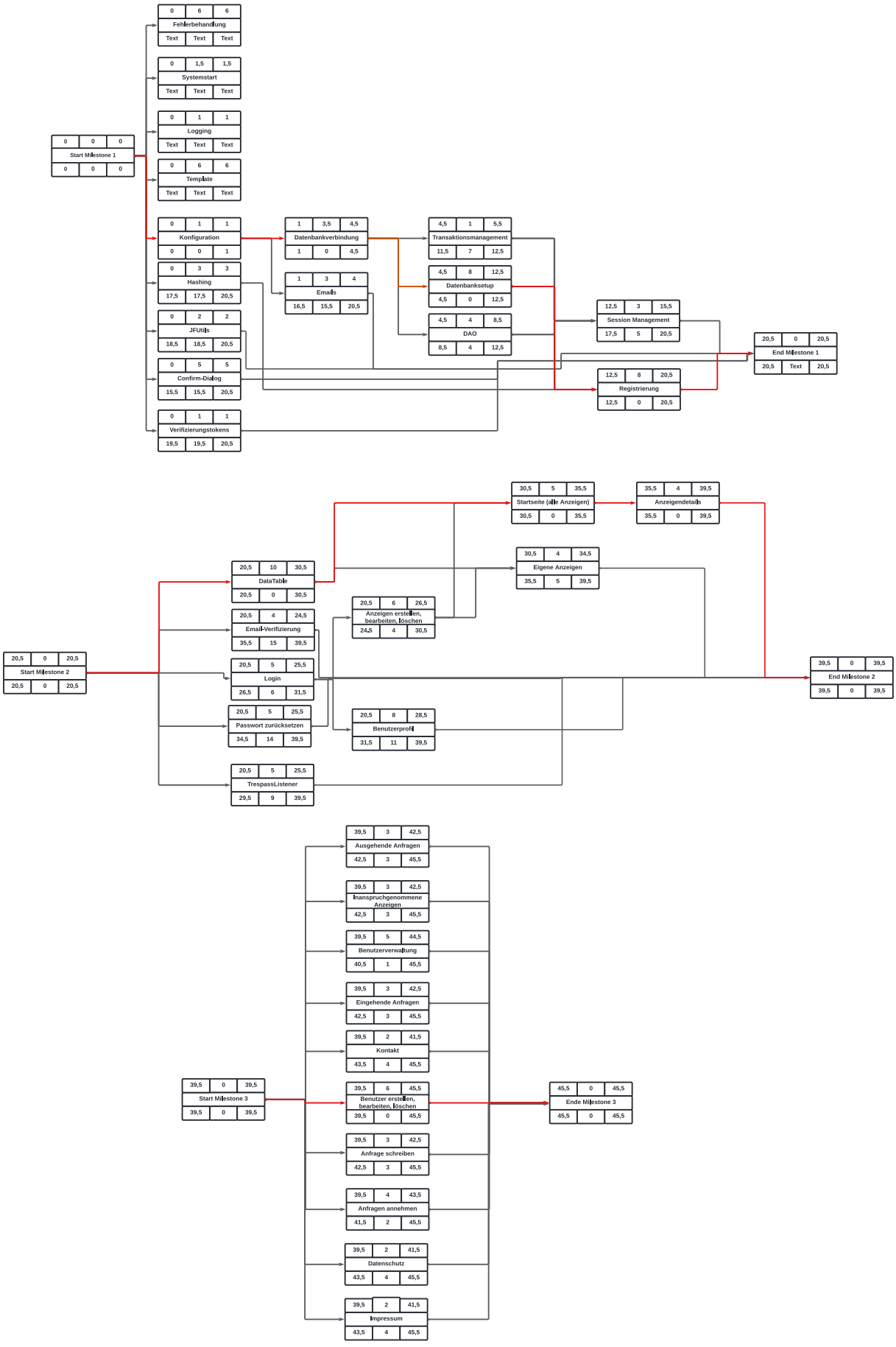

Abbildung 2: PERT-Diagramme der verschiedenen Meilensteine

besonders nützlich, um sicherzustellen, dass das Projekt innerhalb des geplanten Zeitrahmens abgeschlossen wird und alle wichtigen Meilensteine rechtzeitig erreicht werden.

4 Spezialgebiete

Autor: Leon Föckersperger

Während der bisherigen Projektphasen haben sich einige Teammitglieder als Experten für bestimmte Spezialgebiete herausgestellt oder wurden gezielt für neue Bereiche ausgewählt. Diese Aufteilung erleichtert die Verwaltung von Verantwortlichkeiten.

Die Spezialgebiete und die zugehörigen Experten sind in der folgenden Tabelle aufgelistet:

Spezialgebiet	**Experte**
CSS	Jakob Edmaier
CDI	Leon Föckersperger
Versionsverwaltung	Jakob Edmaier
Logging	David Sturm
Konfigurationsdateien	Leon Föckersperger
Connection Pool	Leon Föckersperger
E-Mail-Versand	David Sturm
LaTeX	Jakob Edmaier
Kryptographie	Jakob Edmaier
Validatoren und Konverter	Jakob Edmaier
Pagination	Jakob Edmaier
Fehlerbehandlung	Jakob Edmaier
Selenium	David Sturm
Continuous Integration / Continuous Deployment	Leon Föckersperger
Sicherheit (web.xml)	Leon Föckersperger
URL-Parameterverarbeitung	David Sturm
SQL	Leon Föckersperger
Softwaretests (JUnit, Mockito)	David Sturm
Internationalisierung	Jakob Edmaier
Composite Components	Leon Föckersperger
Jakarta Faces (Facelets)	David Sturm
Jakarta Faces (Beans)	David Sturm
Rollenprüfung (Trespasslistener)	David Sturm
SQL und Datenbank-Setup	Leon Föckersperger

5 Tests

Autor: David Sturm

Im folgenden Kapitel werden auzugsweise einige Whitebox-Tests für die einzelnen Meilensteine aufgeführt. Diese sind hierbei nur Beispiele der für die Implementierung genutzten Tests und bilden somit nicht die vollständige Liste der verwendeten Tests ab.
Da das Programm für die Sicherheit der Benutzer nur gehashte Passwörter speichert, wird auf das folgende, vorher berechnete Passwort zurückgegriffen:
Passwort = “passWord1”
Salt = “123456”
Hash-Verfahren = “SHA512”
Gehashtes und Encodedes Passwort = “NThBMTE0OTg1NDU0MjgzNjA1MUI2RTk1OUZBOUZFN0U0MEJBMjY1QjIwQTI5OE

Q0NBQ0VFRUNBQzc4NEM3QzBFMDdFQzdDMjgzMzhBQUFCOTA2RDlFMDA3RjlDOU-

U1ODFBODEwMkRDNzJENTI2QTI4NEZCMjAxQUNGMkExNzA=”

5.1 Milestone 1

T101 - Registrieren

GEGEBEN Ein *UserDetailsDTO* user mit den Attributen
nick=“mmustermann”, email=“mustermann@mail.com”, firstname=“max”, lastname=“mustermann”,
street=“musterstraße”, houseNumber=“1”, city=“musterstadt”, postalCode=“12345”.

UND Ein *PasswordInputDTO* passwordInput mit den Attributen password = “passWord1” und repetition = “passWord1”.

DANN Läuft der folgende Test erfolgreich durch:

```
RegisterBean registerBean = new RegisterBean();
registerBean.setUser(user);
registerBean.setPasswordInput(passwordInput);
registerBean.submit();

LoginDTO login = new LoginDTO();
login.setNickname(user.getNickname() );
login.setPassword(passwordInput.getPassword());
Optional optional = UserService.authenticate(login);

assertTrue(optional.isPresent());
assertEquals(user.getNickname() ,optional.get().getNickname());
assertEquals(user.getLastname() ,optional.get().getLastname());
```

T102 - E-Mail Verifikation

GEGEBEN Ein Benutzer mit den Attributen id=“1”, nick=“mmustermann”, email=“mustermann@mail.com”, firstname=“max”, lastname=“mustermann”, street=“musterstraße”, houseNumber=“1”, city=“musterstadt”, postalCode=“12345” sowie dem gehashten Passwort ist in der Datenbank gespeichert.

UND Kein Benutzer mit der E-Mail “beispielmail@email.de” ist in der Datenbank vorhanden.

DANN Läuft der folgende Test erfolgreich durch:

```
EmailFormatValidator formatValidator = new EmailFormatValidator();
EmailUniqueValidator uniqueValidator = new EmailUniqueValidator();

String correctMail = "beispielmail@email.de";
String falseMail = "false@input@mail.de";
String doubledMail = "mustermann@mail.com"

FacesContext fctx = null;
UIComponent component = null;

assertDoesNotThrow(ValidatorException.class, ()-> {
    EmailFormatValidator.validate(fctx, component, correctMail);
    EmailUniqueValidator.validate(fctx, component, correctMail);
    });

assertThrows(ValidatorException.class,
    ()-> EmailFormatValidator.validate(fctx, component, falseMail));

assertThrows(ValidatorException.class,
    ()-> EmailUniqueValidator.validate(fctx, component, doubledMail));
```

T103 - Session Management

GEGEBEN Ein Benutzer mit den Attributen id=“1”, nick=“mmustermann”, email=“mustermann@mail.com”, firstname=“max”, lastname=“mustermann”, street=“musterstraße”, houseNumber=“1”, city=“musterstadt”, postalCode=“12345” sowie dem gehashten Passwort ist in der Datenbank gespeichert.

UND Die UserSession userSession.

DANN Läuft der folgende Test erfolgreich durch:

```
LoginBean loginBean = new LoginBean();
LoginDTO login = new LoginDTO();
login.setNickname("mmustermann");
login.setPassword("passWord1");
loginBean.setLogin(login);
loginBean.submit();

assertTrue(userSession.isUserLoggedIn());
assertEquals(1, userSession.getCurrentUserId());
```

T104 - Database Connection

GEGEBEN Die Instanz connectionPool der Klasse *ConnectionPool*.

DANN Läuft der folgende Test erfolgreich durch:

```
Connection connection = connectionPool.getConnection();
assertNotNull(connection);
assertTrue(connection.isValid());
```

T105 - Token Generation

GEGEBEN Die Instanz tokenGenerator der Klasse *VerificationTokenGenerator*.

DANN Läuft der folgende Test erfolgreich durch:

```
String token1 = tokenGenerator.generateToken();
String token2 = tokenGenerator.generateToken();
String token3 = tokenGenerator.generateToken();

assertFalse(token1.equals(token2));
assertFalse(token1.equals(token3));
assertFalse(token2.equals(token3));
```

T106 - Race-Condition

GEGEBEN Die zwei wie folgt befüllten *UserDetailsDTO*'s:
user1 mit den Attributen nick="mmustermann", email="mustermann@mail.com", firstname="max", lastname="mustermann", street="musterstraße", houseNumber="1", city="musterstadt", postalCode="12345" and an *AuthTokenDTO* containing the values salte="12345678", passwordHash="987654321" and algorithm="guessing"
sowie *user2* mit den Werten
nick="mmustermann", email="mustermann@othermail.com", firstname="marie", lastname="mustermann", street="musterstraße", houseNumber="1", city="musterstadt", postalCode="12345" and an *AuthTokenDTO* containing the values salte="5316761", passwordHash="732723747" and algorithm="guessing".

DANN Läuft der folgende Test erfolgreich durch:

```
Transaction transaction1 = TransactionFactory.createTransaction();
Transaction transaction2 = TransactionFactory.createTransaction();

DAOFactory daoFactory1 = transaction1.getDAOFactory();
DAOFactory daoFactory2 = transaction2.getDAOFactory();

UserDAO userDAO1 = daoFactory1.createAdvertisementDAO();
UserDAO userDAO2 = daoFactory2.createAdvertisementDAO()
userDAO1.createUser(user1);

assertThrows(StoreValidationException.class,
    ()-> userDAO2.createUser(user2););
```

5.2 Milestone 2

T201 - File Validation bei Upload

GEGEBEN Ein Mock des interfaces *Part* part.

DANN Läuft der folgende Test erfolgreich durch:

```
MockitoAnnotations.openMocks(this);
Mockito.when(part.getContentType()).thenReturn("image/gif");
AvatarValidator validator = new AvatarValidator();
FacesContext fctx = null;
UIComponent component = null;

assertThrows(ValidatorException.class,
    ()-> validator.validate(fctx, component, part));
```

T202 - Erfolgreiches Einloggen

GEGEBEN Ein Benutzer mit den Attributen id="1", nick="mmustermann", email="mustermann@mail.com", firstname="max", lastname="mustermann", street="musterstraße", houseNumber="1", city="musterstadt", postalCode="12345" sowie dem gehashten Passwort ist in der Datenbank gespeichert.

DANN Läuft der folgende Test erfolgreich durch:

```
LoginDTO login = new LoginDTO();
login.setNickname("mmustermann");
login.setPassword("passWord1");
Optional optional = UserService.authenticate(login);

assertTrue(optional.isPresent());
```

T203 - Fehlgeschlagenes Einloggen

GEGEBEN Ein Benutzer mit den Attributen id="1", nick="mmustermann", email="mustermann@mail.com", firstname="max", lastname="mustermann", street="musterstraße", houseNumber="1", city="musterstadt", postalCode="12345" sowie dem gehashten Passwort ist in der Datenbank gespeichert.

DANN Läuft der folgende Test erfolgreich durch:

```
LoginDTO login = new LoginDTO();
login.setNickname("mmustermann");
login.setPassword("1droWssap");
Optional optional = UserService.authenticate(login);

assertFalse(optional.isPresent());
```

T204 - Passwort Zurücksetzen

GEGEBEN Ein Benutzer mit den Attributen id="1", nick="mmustermann", email="mustermann@mail.com", firstname="max", lastname="mustermann", street="musterstraße", houseNumber="1", city="musterstadt", postalCode="12345", secret_for_password_reset="123456789101112" sowie dem gehashten Passwort ist in der Datenbank gespeichert.

DANN Läuft der folgende Test erfolgreich durch:

```
passwordResetRequestBean resetBean = new passwordResetRequestBean();

PasswordResetDTO resetDTO = new PasswordResetDTO();
resetDTO.setVerificationToken("123456789101112");

PasswordInputDTO inputDTO = new PasswordInputDTO();
inputDTO.setPassword("newPassword");
inputDTO.setRepetition("newPassword");

resetBean.setPasswordInput(inputDTO);
resetBean.setPasswordReset(resetDTO);
resetBean.submit();

LoginDTO login = new LoginDTO();
login.setNickname("mmustermann");
login.setPassword("newPassword");
Optional optional = UserService.authenticate(login);

assertTrue(optional.isPresent());
```

T205 - Anzeige Erstellen, Bearbeiten und Löschen

GEGEBEN Ein Benutzer mit den Attributen id="1", nick="mmustermann", email="mustermann@mail.com", firstname="max", lastname="mustermann", street="musterstraße", houseNumber="1", city="musterstadt", postalCode="12345", secret_for_password_reset="123456789101112" sowie dem gehashten Passwort ist in der Datenbank gespeichert.

UND In dem Table ad ist die id="3537" nicht vorhanden.

DANN Läuft die folgende Testfolge erfolgreich durch:

Anzeige Erstellen:

```
EditAdvertisementBean editAdBean = new EditAdvertisementBean();
AdvertisementDetailsDTO advertisement = new AdvertisementDetailsDTO();

advertisement.setId(3537L);
advertisement.setUserId(1L);
advertisement.setTitle("Test Anzeige");
advertisement.setCost(20);
advertisement.setStartDate(LocalDate.now());
advertisement.setCountry("Deutschland");
advertisement.setCity("Musterstadt");
advertisement.setPostalCode("12345");

editAdBean.submit();

advertisement = new AdvertisementDetailsDTO();
advertisement.setId(3537L);

advertisement =
    AdvertisementService.getAdvertisementDetails(advertisement);

assertNotNull(advertisement);
assertEquals("Test Anzeige", advertisement.getTitle());
```

Anzeige laden:

```
EditAdvertisementBean editAdBean = new EditAdvertisementBean();

AdvertisementDetailsDTO advertisement = new AdvertisementDetailsDTO();
advertisement.setId(3537L);

editAdBean.loadData();

assertNotNull(editAdBean.getAd());
assertEquals("TestAd",editAdBean.getAd().getTitle());
```

Anzeige bearbeiten:

```
EditAdvertisementBean editAdBean = new EditAdvertisementBean();

AdvertisementDetailsDTO advertisement = new AdvertisementDetailsDTO();
advertisement.setId(3537L);

editAdBean.loadData();

advertisement = editAdBean.getAd();
advertisement.setTitle("Neuer Titel");
editAdBean.setAd();
editAdBean.submit();

advertisement = new AdvertisementDetailsDTO();
advertisement.setId(3537L);

advertisement =
    AdvertisementService.getAdvertisementDetails(advertisement);

assertNotNull(advertisement);
assertEquals("Test Anzeige", advertisement.getTitle());
```

Anzeige löschen:

```
EditAdvertisementBean editAdBean = new EditAdvertisementBean();

AdvertisementDetailsDTO advertisement = new AdvertisementDetailsDTO();
advertisement.setId(3537L);

editAdBean.loadData();
editAdBean.deleteAdvertisement();

advertisement = new AdvertisementDetailsDTO();
advertisement.setId(3537L);

assertThrows(EntityNotFoundException.class,
    ()-> AdvertisementService.getAdvertisementDetails(advertisement));
```

5.3 Milestone 3

T301 - Anfrage schreiben

GEGEBEN Ein Benutzer mit den Attributen id=“1”, nick=“mmustermann”, email=“mustermann@mail.com”, firstname=“max”, lastname=“mustermann”, street=“musterstraße”, houseNumber=“1”, city=“musterstadt”, postalCode=“12345”, talent_points=“50” sowie dem gehashten Passwort ist in der Datenbank gespeichert.

UND Ein weiterer Benutzer mit den Attributen id=“2”, nick=“amusterfrau”, email=“musterfrau@mail.com”, firstname=“anna”, lastname=“musterfrau”, street=“musterweg”, houseNumber=“5”, city=“musterstadt”, postalCode=“12345”, talent_points=“50” sowie dem gehashten Passwort ist ebenfalls in der Datenbank gespeichert.

UND Eine Anzeige mit den Attributen id=“3537”, creator_id=“1”, title=”Test Anzeige”, cost_in_talent_points=“20”, date_of_publication=“27.05.2024”, country=“Deutschland”, city=“Musterstadt”, postal_code=“12345” ist ebenfalls im System gespeichert.

DANN Läuft die folgende Testfolge erfolgreich durch:
Anfrage schreiben:

```
WriteRequestBean witeRequestBean = new WriteRequestBean();
RequestDTO request = new RequestDTO();
UserDTO adCreater = new UserDTO();
AdvertisementDTO ad = new AdvertisementDTO();

adCreater.setId(1L);
ad.setId(3537L);
request.setAdvertisementId(3537L);
request.setSenderId(2L);
request.setId(98765L);
request.setRequest("Test Text");

witeRequestBean.setRequest(request);
witeRequestBean.submit();

request = new RequestDTO();
request.setId(98765L);

request = RequestService.getRequest(request);
assertEquals("Test Text", request.getRequest());
```

Anfrage laden:

```
WriteResponseBean writeResponseBean = new WriteResponseBean();
RequestDTO request = new RequestDTO();

request.setId(98765L);
writeResponseBean.setRequest(request);

writeResponseBean.loadData();
request = writeResponseBean.getRequest();

assertEquals("Test Text", request.getRequest());
```

Anfrage annehmen:

```
WriteResponseBean writeResponseBean = new WriteResponseBean();
RequestDTO request = new RequestDTO();

request.setId(98765L);
writeResponseBean.setRequest(request);

writeResponseBean.loadData();
request = writeResponseBean.getRequest();
request.setResponse("Antworttext");
writeResponseBean.accept();

request = new RequestDTO();
request.setId(98765L);
UserDTO requestCreator = new UserDTO();
requestCreater.setId(2L);
UserDTO adCreator = new UserDTO();
adCreator.setId(1L);

request = RequestService.getRequest(request);
requestCreator = userService.getUserDetails(requestCreator);
adCreator = userService.getUserDetails(adCreator);

assertEquals("Antworttext", request.getResponse());
assertTrue(request.getAccepted());
assertEquals(70, adCreator.getTalentPoints());
assertEquals(30, requestCreator.getTalentPoints());
```