

Einleitung und einführende Übung

Inhalt

1 Einleitung	2
1.1 GitHub	2
1.2 Forken eines Repositorys.....	2
1.3 Arbeiten im geforkten Repository	4
1.4 Einführung in Spring	6
1.5 Beans in Spring Boot.....	7
1.6 Erstellung einer Bean.....	8
1.7 Erweiterung um einen Service	9
1.8 Logging in Spring Boot.....	10
1.9 Testing in Spring Boot	12
2. Hausaufgaben zur Übung	13
Verweise	14

1 Einleitung

Alle Übungen zu der Vorlesung Web-basierte Anwendungssysteme finden Sie im GitHub Repository:
<https://github.com/alehmannFRA-UAS/WebAppSys>

1.1 GitHub

GitHub ist eine Codehosting-Plattform für die Versionskontrolle und Zusammenarbeit in Projekten. GitHub ist ein web-basierter Dienst, der Entwicklern hilft, ihren Code zu speichern und zu verwalten sowie Änderungen an ihrem Code zu verfolgen und zu kontrollieren. Um genau zu verstehen, was GitHub ist, müssen die zwei folgenden Prinzipien bekannt sein: [1]

- Versionskontrolle
- Git

Die **Versionskontrolle** unterstützt Entwickler bei der Verfolgung und Verwaltung von Änderungen am Code eines Softwareprojekts. Wenn ein Softwareprojekt wächst, wird die Versionskontrolle unerlässlich. Entwickler können mit der Versionskontrolle sicher durch Verzweigungen und Zusammenführungen arbeiten. Beim Verzweigen dupliziert ein Entwickler einen Teil des Quellcodes (das so genannte Repository). Der Entwickler kann dann sicher Änderungen an diesem Teil des Codes vornehmen, ohne den Rest des Projekts zu beeinträchtigen. Sobald der Teil des Codes des Entwicklers ordnungsgemäß funktioniert, kann er diesen Code wieder in den Hauptquellcode einbinden, um ihn publik zu machen. Alle diese Änderungen werden dann verfolgt und können bei Bedarf rückgängig gemacht werden. [1]

Git ist ein spezifisches Open-Source-Versionskontrollsystem, das 2005 von Linus Torvalds entwickelt wurde. Insbesondere ist Git ein verteiltes Versionskontrollsystem, was bedeutet, dass die gesamte Codebasis und Historie auf jedem Entwicklercomputer verfügbar ist, was ein einfaches Verzweigen und Zusammenführen ermöglicht. [1]

Weitere Informationen zu GitHub können Sie [hier](#) finden.

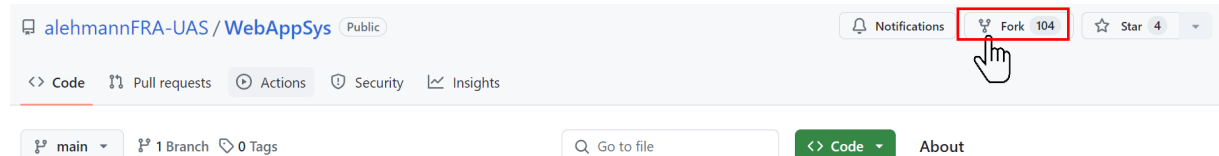
1.2 Forken eines Repositorys

Ein Fork ist ein neues Repository, das denselben Code und dieselben Sichtbarkeitseinstellungen verwendet wie das ursprüngliche Repository. Forks werden oft verwendet, um Ideen oder Änderungen zu überprüfen, bevor sie für das ursprüngliche Repository vorgeschlagen werden, z. B. in Open-Source-Projekten oder wenn ein Benutzer keinen Schreibzugriff auf das ursprüngliche Repository hat. Weitere Informationen finden Sie unter [Mit Forks arbeiten](#). [2]

Im Folgenden soll kurz dargestellt werden, wie Sie einen Fork des Repositorys der Übungen erstellen können. Hierzu benötigen Sie vorab einen GitHub Account, den Sie hier anlegen können:
<https://github.com/>

Hier die einzelnen Schritte, um einen Fork des Repositorys zu erstellen:

1. **Einloggen:** Stellen Sie sicher, dass Sie mit Ihrem GitHub-Konto angemeldet sind.
2. **Repository finden:** Öffnen Sie in einem neuen Browser Tab folgenden Link: <https://github.com/alehmannFRA-UAS/WebAppSys>
3. **Fork des Repositorys erstellen:** Auf der Repository-Seite klicken Sie oben rechts auf den "Fork"-Button. Dadurch wird eine Kopie des Repositorys in Ihrem GitHub-Konto erstellt.



4. **Wählen** Sie unter „Owner“ mit dem Dropdownmenü Ihren Besitzernamen für das geforkte Repository aus.

Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks.](#)

Required fields are marked with an asterisk ().*

Owner * Repository name *

/


✔ WebAppSys is available.

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

☒ Copy the **main** branch only

Contribute back to alehmannFRA-UAS/WebAppSys by adding your own branch. [Learn more.](#)

 You are creating a fork in your personal account.

Create fork

Standardmäßig erhalten Forks den gleichen Namen wie die zugehörigen ursprünglichen Repositories. Um den Fork noch genauer zu unterscheiden, können Sie optional im Feld „Repositoryname“ den Namen anpassen.

5. **Klicken** Sie auf Create Fork.
6. **Warten** Sie ein paar Sekunden. GitHub erstellt nun eine Kopie des Repositorys. Dies kann einige Augenblicke dauern, je nach Größe des Repositorys.
7. **Arbeiten mit dem geforkten Repository:** Nachdem der Fork abgeschlossen ist, werden Sie auf die eigene Kopie des Repositorys weitergeleitet. Sie können nun Änderungen in diesem Repository vornehmen, neue Commits hinzufügen und sogar Pull Requests erstellen, um Änderungen an das ursprüngliche Repository zurückzusenden.

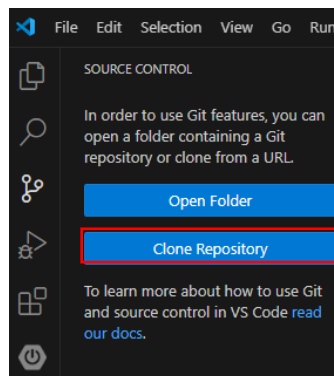
Denken Sie unbedingt daran, dass Sie möglicherweise die Lizenzen und die Nutzungsbedingungen des ursprünglichen Repositorys beachten müssen, insbesondere wenn Sie Änderungen veröffentlichen oder weiterverbreiten.

1.3 Arbeiten im geforkten Repository

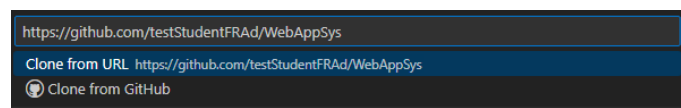
Im Folgenden soll Ihnen kurz dargelegt werden, wie Sie im geforkten Repository mit Visual Studio Code (VS Code) arbeiten können. Öffnen Sie zunächst VS Code. Falls ein vorhandenes Projekt geöffnet wird, so schließen Sie dieses bitte zuerst. Im nächsten Schritt soll das im vorigen Schritt geforkte Repository in VS Code lokal geladen werden. Rufen Sie dazu die Ansicht Source Control auf, indem Sie auf das Symbol Source Control in der Aktivitätsleiste an der Seite von VS Code klicken oder über den Befehl **View – Source Control** (Strg+Umschalt+G).



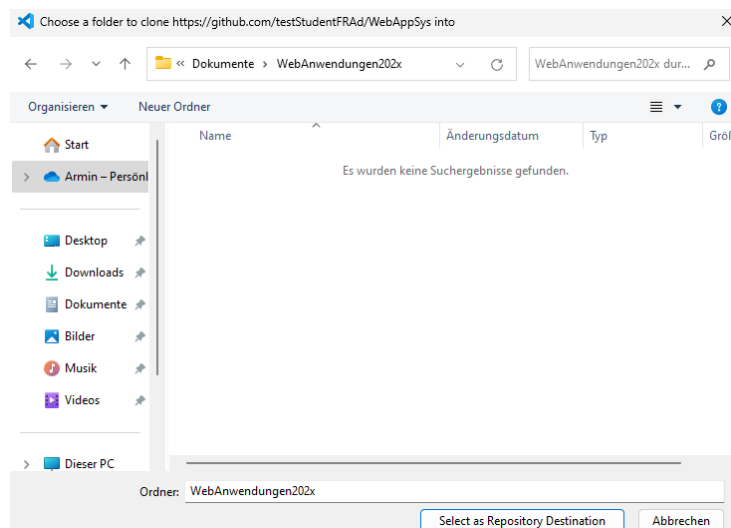
Klicken Sie nun auf Clone Repository.



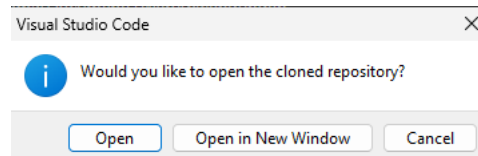
Geben Sie in der sich öffnenden Eingabemaske die URL zu Ihrem geforkten Repository an (<https://github.com/<Ihr GitHub Account Name>/<Name des Forks>>).



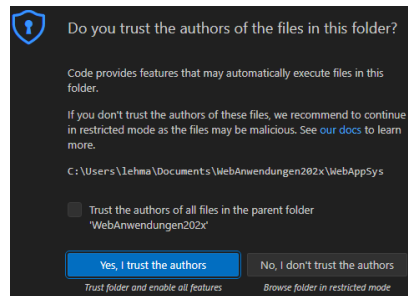
Bestätigen Sie durch Anklicken von Clone from URL die Eingabe und wählen Sie im nächsten Schritt den Speicherort lokal auf Ihrem Rechner für das geklonte Repository.



Öffnen Sie in der Folge das geklonte Repository.



Und bestätigen Sie, dass Sie dem Author der Dateien vertrauen.



Nun sollte das komplette geklonte Repository in VS Code angezeigt werden.

Eine weiterführende Anleitung zur Verwendung von Visual Studio Code und GitHub ist z.B. [hier](#) zu finden.

Bevor Sie mit der ersten Übung fortfahren können, muss noch eine sehr kurze Einführung zu Spring Boot gegeben werden.

1.4 Einführung in Spring

Spring IO Platform

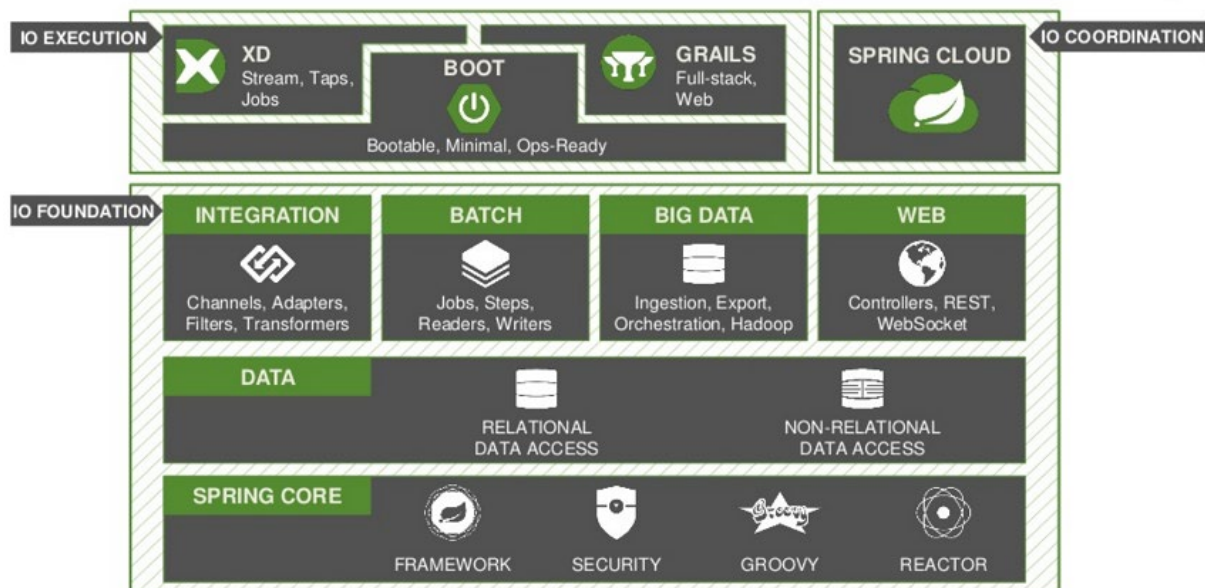


Abbildung 1: Spring Framework Übersicht

Das Spring-Framework wurde 2002 erstmals als Idee vorgestellt und ein Jahr später unter dem Namen Spring-Framework als quelloffenes Projekt veröffentlicht. Das Ziel – damals wie heute – ist, die Entwicklung mit Java zu vereinfachen und gute Programmierpraktiken zu fördern. Infrastruktur auf Anwendungsebene ist eines der Schlüsselemente von Spring. Springs Fokus liegt ganz klar auf Bereitstellung und Konfiguration nichtfunktionaler Anforderungen, so dass Entwickler von Anwendungen sich auf ihre eigentliche Aufgabe, Implementierung der Geschäftslogik, konzentrieren können. [3]

Mittlerweile existiert eine Vielzahl an weiteren Projekten, die auf dem Spring Framework basieren. Eines dieser Projekte ist Spring Boot 2. Der Projektstart ist 2013 gewesen und das erste Release ist 2014 erschienen. Seit Januar 2019 kann das Release 2.1.2 eingesetzt werden. Spring Boot 2 vereinfacht die Entwicklungsarbeit noch weiter, indem erprobte und sinnvolle Standardkonfigurationen angeboten und verwendet sowie Stand-Alone-Anwendungen mit eingebettetem (Web-)Server erstellt werden. [5]

Spring Boot ist in diesem Kontext kein neues Framework, sondern eine Sicht auf die Spring-Plattform, die es ermöglicht, eigenständige und produktionsreife Anwendungen auf Basis des beschriebenen Spring-Frameworks zu bauen, die unter anderem folgende Eigenschaften haben: [4]

- eigenständige Anwendungen, die keine externen Laufzeitabhängigkeiten mehr haben
- eingebettete Container (zum Beispiel Tomcat, Jetty oder Undertow), so dass keine War-Dateien verteilt werden müssen
- automatische Konfiguration soweit möglich
- Bereitstellung von nicht funktionalen Eigenschaften, die zur Produktion in der Regel benötigt werden: Metriken, Health Checks und externe Konfiguration
- keinerlei Generierung von Code oder Konfiguration

Spring Boot basiert vollständig auf dem Spring-Framework. Es wurde mit dem Ziel erschaffen, die Entwicklung eigenständig lauffähiger Anwendungen mit dem Spring-Framework drastisch zu erleichtern. [4]

1.5 Beans in Spring Boot

Zuerst öffnen Sie bitte das Projekt „BeanExample“ mit VS Code. Nachdem Sie das Projekt geöffnet haben, analysieren Sie bitte die erstellte Projektstruktur.

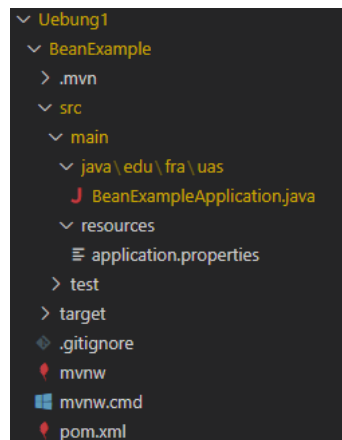


Abbildung 2: Projektstruktur

Öffnen Sie als nächstes die Datei `pom.xml`. Hier finden Sie wichtige Einträge wie z.B.:

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>3.3.1</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Dieser Eintrag bestimmt z.B. welche Version des Spring Boot Frameworks verwendet werden soll. Weitere Einträge wie die unten dargestellten definieren das Projekt genauer, unter anderem wird hier der Paketnamen (Package) für das Projekt festgelegt, hier z.B. `edu.fra.uas`

```
<groupId>edu.fra.uas</groupId>
<artifactId>BeanExample</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>BeanExample</name>
<description>Demo project for Spring Boot</description>
<properties>
  <java.version>17</java.version>
</properties>
```

Ein sehr wichtiger Eintrag wurde hier auch schon vorgenommen. Mittels dieses Eintrags wurde festgelegt, welche Java Version genutzt werden soll. Darüber hinaus wurden auch Abhängigkeiten wie die folgende festgelegt.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter</artifactId>
</dependency>
```

Hiermit wird z.B. definiert, dass es sich um eine Spring Boot-Anwendung handelt. Weiterhin wurde für das Projekt festgelegt, dass Maven zur Erstellung des Projekts genutzt werden soll.

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

Maven ist ein Build-Management-Tool. Maven geht dabei von einem Zyklus der Softwareerstellung aus, der häufig durchlaufen wird: Validierung, Kompilierung, Testen, Paketieren, Integrationstests, Verifizierung, Installation und Verteilung.

Die notwendigen Informationen zum Bau eines Maven-Projektes, die sich nicht aus Konventionen ableiten lassen, werden in einem ProjectObjectModel, dem POM, gespeichert. Das POM wird in der Regel in einer XML-Datei namens pom.xml abgespeichert. POMs können voneinander erben. Damit werden zentrale Definitionen und Konfigurationen ermöglicht, die an Teilprojekte weitergegeben werden können. Eine wichtige Aufgabe von Maven ist die Auflösung von Abhängigkeiten, die ein Softwareprojekt hat. Dies kann aus lokalen Quellen oder Quellen im Intranet oder Internet geschehen. Beide Quellen heißen Repositorys. Maven kann darüber hinaus zentrale Eigenschaften (Properties) eines Projektes verwalten, Quelltexte und andere Ressourcen während des Bauens filtern und vieles mehr. Maven hat eine modulare Architektur, die über Plugins erweitert werden kann. Mit dem Spring-Boot-Maven-Plugin steht Ihnen ein Plugin zur Verfügung, das Ihnen unter anderem beim Bau ausführbarer Artefakte hilft. [4]

1.6 Erstellung einer Bean

In Spring spielt der Begriff Bean eine spezielle Rolle. Eine Bean ist ein verwaltetes Java-Objekt innerhalb des Spring Containers. Die Bean wird erstellt, überwacht und gelöscht. Mit einer Annotation (Meta-Informationen) wird eine Bean im Programm ausgewiesen ([JavaBeans\(TM\) Specification 1.01 Final Release](#)).

Öffnen Sie nun die Klasse `BeanBeispielApplication` im Editor im Pfad `/src/main/java/edu/fra/uas`.

Diese Klasse ist mit der Annotation `@SpringBootApplication` ausgezeichnet und somit die Main-Klasse des Programms. Analysieren Sie folgende Methode in der Klasse.

```
@Bean
CommandLineRunner init() {
    CommandLineRunner action = new CommandLineRunner() {
        @Override
        public void run(String... args) throws Exception {
            System.out.println("Hello World");
        }
    };
    return action;
}
```

Wie zu sehen ist, ist die Methode `init()` mit `@Bean` annotiert. Dies bewirkt, dass der Spring Container nach dem Start der Anwendung die Bean unmittelbar ausführt. Die Bean ist dabei das Rückgabeobjekt der anonymen Klasse `CommandLineRunner`.

Zum Starten des Projektes genügt es, `BeanBeispiel` zu selektieren und dann mit dem Kontextmenü (rechte Maustaste drücken) *Run Java* auszuführen. *Bitte haben Sie etwas Geduld, bis die Ausführung startet*. Ihre erste Spring-Boot-Anwendung wird nun gestartet und ausgeführt. Was passiert, können wir im Terminal beobachten.

Aufgabe:

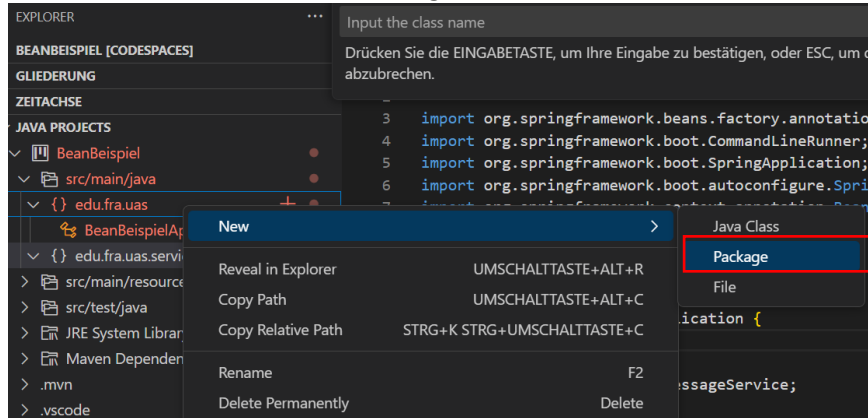
Löschen oder kommentieren Sie jetzt die Annotation `@Bean` und starten das Programm erneut. Vergleichen Sie die neue mit der alten Ausgabe. Gibt es Unterschiede? Welche Erklärung haben Sie dafür?

Diese und alle weiteren Übungen können Sie alle aus dem Git Repository klonen. Unter <https://github.com/alehmannFRA-UAS/WebAppSys> finden Sie alle Übungen bzw. Projekte zu diesem Kurs.

1.7 Erweiterung um einen Service

Erweitern Sie im nächsten Schritt das Programm durch Anlegen einer weiteren Klasse `MessageService`, wie im Folgenden dargestellt.

Legen Sie hierzu ein neues Package (Ordner) `service` im Projekt an. Das Package sollte den Pfad `/src/main/java/edu/fra/uas/service` ergeben. Unten im Explorer finden Sie den Reiter Java Projects. Hierüber sollten Sie das neue Package und die neue Klasse erstellen.



```
@Component
public class MessageService {

    private String message;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

}
```

Die oben dargestellte Klasse ist durch `@Component` annotiert und wird dadurch zu einer Bean-Klasse für den Spring Container. Damit wird das Objekt zur Bean und bekommt einen eindeutigen Namen zugeordnet, der identisch zum Klassennamen mit kleingeschriebenem erstem Buchstaben ist.

Die Main-Klasse wird nun wie folgt erweitert.

```
@SpringBootApplication
public class BeanBeispielApplication {

    @Autowired
    private MessageService messageService;

    public static void main(String[] args) {
        SpringApplication.run(BeanBeispielApplication.class, args);
    }

    @Bean
    CommandLineRunner init() {
        CommandLineRunner action = new CommandLineRunner() {
            @Override
            public void run(String... args) throws Exception {
                messageService.setMessage("Hello World");
                System.out.println(messageService.getMessage());
                messageService.setMessage("--> HHHOHHH <--");
                System.out.println(messageService.getMessage());
            }
        };
        return action;
    }

}
```

Um eine Bean zu erhalten, annotieren wir die gewünschte Klasse mit `@Component` und injizieren diese Bean an der Stelle im Programm, an der wir sie verwenden möchten. Durch die Annotation `@Autowired` wird ein Objekt (genauer ein Bean-Objekt) zur Verfügung gestellt. Dies beinhaltet auch seine Instanziierung. In der `init()`-Methode rufen wir das `messageService`-Objekt auf und bekommen die eingegebenen Werte auf der Konsole ausgegeben.

Aufgabe:

Starten Sie das Programm und kontrollieren Sie die Ausgabe.

Löschen oder kommentieren Sie jetzt die Annotation `@Autowired` und starten das Programm erneut. Vergleichen Sie die neue mit der alten Ausgabe. Gibt es Unterschiede? Welche Erklärung haben Sie dafür?

1.8 Logging in Spring Boot

Sie haben bisher immer die Ausgaben der Programme in der Konsole erzeugt. Dies soll nun geändert werden, das Logging ist hierfür wesentlich besser geeignet.

Erzeugen Sie ein neues Package `controller` und legen Sie dort die Klasse `BeanController` an. Warum sollte Wert auf die Bezeichnung von Ordnern bzw. Packages gelegt werden? Mit der Bezeichnung `controller` wird aufgezeigt, dass die Klassen in dem jeweiligen Package spezielle Funktionalitäten aufweisen. Die hier verwendete Klasse dient als Schnittstelle nach außen und kontrolliert den Zugriff auf die Anwendung, daher die Bezeichnung. Im Package `service` hingegen befinden sich die Klassen für die entsprechende Geschäftslogik. Diese Klassen sind von außen nicht direkt zugreifbar, sondern nur über einen Controller.

Erstellen Sie die im Folgenden dargestellte Klasse `BeanController`. Hier wird die Methode `putMessage` definiert, die einen Parameter `message` besitzt und diesen Wert an die `MessageService`-Bean mit Hilfe der Methode `setMessage` übergibt. Wie zuvor wird auch hier die Annotation `@Autowired` zum Injizieren der Bean genutzt. (Hinweis: **Unbedingt den Rechtsschreibfehler `messgae` im Quelltext mit übernehmen!**)

```
@Component
public class BeanController {

    @Autowired
    private MessageService messageService;

    public String putMessage(String message) {
        messageService.setMessage(" put messgae: " + message);
        return messageService.getMessage();
    }
}
```

In der Klasse `BeanBeispielApplication` wird nun die Bean `BeanController` verwendet. Nun soll aber nichtmehr `System.out.println` verwendet werden, sondern ein Logger.

Logging ist ein wichtiger Bestandteil des Spring-Frameworks selber. Spring Boot hat eine einfache Abstraktion zur Konfiguration gemeinsamer Logging-Parameter [4]. Logging zeichnet Zustände und Ereignisse in Programmausführungen auf, indem diese Informationen z.B. in Dateien gespeichert oder auf der Konsole ausgegeben werden. Log-Ausgaben können angepasst, konfiguriert und nach eigenen Vorzügen formatiert werden. Die eintretenden und zu protokollierenden Meldungen sind in verschiedenen `LogLevel` klassifiziert. Gängig und am meisten sinnvoll sind fünf Log Level (TRACE, DEBUG, INFO, WARN, ERROR), die in der Datei `application.properties` eingestellt werden können.

Die folgende Tabelle zeigt die Eigenschaften, die über den normalen Spring-Boot-Konfigurationsmechanismus konfiguriert werden können.

Name	Funktion
logging.config	Native Konfigurationsdatei des gewählten Frameworks
logging.file	Dateiname zum Loggen in Datei
logging.level.*	Level des Loggers, z.B. logging.level.org.springframework=WARN
logging.path	Pfad der erzeugten Logdatei
logging.pattern.console	Pattern, das zum Loggen in die Konsole genutzt wird (nur Logback)

Standardmäßig wird mit den Leveln `ERROR`, `WARN` und `INFO` in die Konsole geloggt. Wenn die Konsole, beziehungsweise das genutzte Terminal, ANSI-Escape-Sequenzen unterstützt, wird das Loglevel farblich markiert ausgegeben. `FATAL` und `ERROR` sind rot, `WARN` gelb und alle anderen Level grün. Wenn weder `logging.file` noch `logging.path` konfiguriert sind, loggt Spring Boot nur in die Konsole. Mit `logging.file` kann eine Datei angegeben werden, in die geloggt wird, mit `logging.path` ein Verzeichnis. Wird nicht zeitgleich eine Datei angegeben, so wird in die Datei `spring.log` geschrieben. Sowohl `logging.file` als auch `logging.path` können absolute Angaben sein oder relativ zum Arbeitsverzeichnis angegeben werden.

Durch die Einstellung `logging.level.edu.fra.uas=debug` wird für das Package `edu.fra.uas=debug` der Log Level auf `DEBUG` gesetzt.

Die einzelnen Log Level beschreiben Meldungen für

- **TRACE:** ausführlicheres Debugging inklusive detaillierter Information zum Ablauf
- **DEBUG:** Debugging inklusive Informationen zum Programmablauf
- **INFO:** Informationen zu auftretenden Ereignissen (wie Start und Ende eines Programmlaufes, Verbindung zur Datenbank hergestellt oder Dauer der Programmausführung)
- **WARN:** Situation, die nicht erwartet werden und zu Fehler führen könnte
- **ERROR:** Fehler (wie behandelte Ausnahme oder zu analysierende Probleme)

Passen Sie die Klasse `BeanBeispielApplication` wie dargestellt an.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@SpringBootApplication
public class BeanBeispielApplication {

    private static final Logger log = LoggerFactory.getLogger(BeanBeispielApplication.class);

    @Autowired
    private BeanController beanController;

    public static void main(String[] args) {
        SpringApplication.run(BeanBeispielApplication.class, args);
    }

    @Bean
    CommandLineRunner init() {
        CommandLineRunner action = new CommandLineRunner() {
            @Override
            public void run(String... args) throws Exception {
                log.debug(beanController.putMessage("Hallo Welt"));
                log.debug(beanController.putMessage("--> 000H000 <--"));
            }
        };
        return action;
    }
}
```

Starten Sie die Anwendung. Wo sind die Log-Ausgaben zu sehen?

Bisher werden noch keine selbstdefinierten Log-Ausgaben erzeugt. Die Ausgabe muss zuerst in der Datei `application.properties` im Ordner `ressources` eingeschaltet werden. Geben Sie dort folgendes ein.

```
logging.level.edu.fra.uas=debug
```

Starten Sie die Anwendung erneut.

Aufgabe:

Bauen Sie in der Klasse `MessageService` einen Logger ein und loggen die Vorgänge in den Methoden `getMessage` und `setMessage`. Verwenden Sie sinnvolle Logging-Texte.

Überlegen Sie welche Informationen noch interessant sein könnten, um diese zu protokollieren?

1.9 Testing in Spring Boot

Zum Abschluss der ersten Übung werden Sie noch ein kurzes Beispiel für das Testen der Software kennenlernen. Spring bietet hierzu bereits alle notwendigen Werkzeuge an. Erstellen Sie nun die Testklasse `ControllerTest` im Ordner `test/java/edu/fra/uas/beanbeispiel`.

```
import static org.assertj.core.api.Assertions.assertThat;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import edu.fra.uas.controller.BeanController;

@SpringBootTest
public class ControllerTest {

    @Autowired
    private BeanController beanController;

    @Test
    void testController() {
        assertThat(beanController.putMessage("Das ist ein Test"))
            .isEqualTo("Das ist ein Test");
    }
}
```

Durch die Annotationen `@SpringBootTest` und `@Test` wird festgelegt, das JUnit-Testläufe durchgeführt werden sollen. Der Testdurchlauf wird durch Klick auf die Testklasse im Reiter Java Projects und folgend durch rechte Maustaste und *Run Tests* gestartet.

In der Testklassen verwenden Sie die Java-Bibliothek AssertJ. Dies geschieht durch die Zeile

```
import static org.assertj.core.api.Assertions.assertThat;
```

Mit AssertJ werden Tests mit verketteten Aufrufen von Methoden lesbarer und leichter verständlich. Die Fehlermeldungen sind oft detaillierter und aussagekräftiger als die von JUnit verwendete Standardbibliothek Hamcrest.

Aufgabe:

Passen Sie den Test so an, dass dieser ohne Fehler durchläuft.

Erweitern Sie die Klasse `MessageService` um eine weitere Variable namens `counter` vom Datentyp `Integer`. Definieren Sie eine Methode `increment`, die den Wert der Variablen `counter` jeweils um 1 erhöht. Schreiben Sie dann in einer neuen Methode in der Klasse `ControllerTest` einen entsprechenden Test dazu.

2. Hausaufgaben zur Übung

Erstellen Sie ein neues Projekt zur Berechnung Ihres Notendurchschnitts anhand der hier gegebenen Vorgaben.

Überlegen Sie bevor Sie mit der Programmierung anfangen. Erstellen Sie sich z.B. Skizzen. Entwerfen Sie die Klassen, die Sie benötigen und bedenken Sie deren Attribute und Methoden. Erstellen Sie eine Projektstruktur, die unterschiedliche Zuständigkeiten erkennen lässt. Vielleicht helfen Ihnen zu Beginn User Stories, anhand derer Sie die zu realisierenden Methoden identifizieren können. Verwenden Sie Tests, um Ihre Anwendung zu realisieren und zu prüfen. **Erstellen Sie keine grafische Web-Schnittstelle!**

Welche Anforderungen soll die Anwendung erfüllen?

Eine relativ einfache Möglichkeit Anforderungen an ein System oder eine Anwendung zu beschreiben bieten User Stories [5]. In der Regel sind solche Beschreibungen nicht länger als zwei Sätze. Folgendes zeigt ein beispielhaftes Schema.

Als <Nutzer in Rolle> möchte ich <Wunsch/Ziel>, damit/weil/denn <Ergebnis/nutzen>.

Anhand des dargestellten Schemas lässt sich nun sehr gut die Anwendung etwas detaillierter beschreiben. Die folgende Tabelle gibt hierzu eine beispielhafte Übersicht für eine Chat-Anwendung.

Nr.	Als...	möchte ich...	damit/weil/denn ...
1	Registrierter Benutzer	mich anmelden,	ich Nachrichten austauschen kann.
2	Registrierter Benutzer	alle meine Konversationen sehen,	ich eine Konversation mit einem Kontakt auswählen kann.
3	Registrierter Benutzer	alle meine Konversationen sehen,	ich eine Konversation löschen kann.
4	Registrierter Benutzer	eine neue Konversation anlegen,	ich mit einem neuen Kontakt kommunizieren kann.
5	Registrierter Benutzer	eine existente Konversation auswählen,	ich mit dem ausgewählten Kontakt kommunizieren kann.
6	Registrierter Benutzer	meine Daten einsehen,	ich meine abgelegten Daten überprüfen kann.
7	Registrierter Benutzer	meine Nachrichten (ein- und ausgehend) zu einer Konversation sehen,	ich die gesendeten und empfangenen Nachrichten immer einsehen kann.
8	Registrierter Benutzer	meine Nachrichten innerhalb einer Konversation versenden,	ein Kontakt meine Nachrichten lesen kann.

Um die Anforderungen an eine Anwendung genauer zu definieren, können User Stories als erster Schritt dienen. Falls Sie mehr über User Stories lernen wollen, können Sie weitere Informationen in dem folgenden Buch finden: Wirdemann R., Mainusch J.: „[Scrum mit User Stories](#)“, 4. Überarbeitet und erweiterte Auflage, München, Hanser, 2022.

Vielleicht beantworten Sie für sich oder mit weiteren Studierenden die folgenden Fragen:

- Haben Sie alle User Stories verstanden?
- Wie würden die User Stories für Ihre Anwendung des Notendurchschnitts aussehen?
- Welche Ergänzungen würden Sie einfügen?
- Was fehlt Ihrer Meinung nach noch an Funktionalität?

Verweise

- [1] Kinsta Inc., „Was ist GitHub? Eine Einführung in GitHub für Einsteiger,“ 04 März 2020. [Online]. Available: <https://kinsta.com/de/wissensdatenbank/was-ist-github/>. [Zugriff am 19 August 2023].
- [2] GitHub Inc., „Forken eines Repositorys,“ 2023. [Online]. Available: <https://docs.github.com/de/get-started/quickstart/fork-a-repo>. [Zugriff am 19 August 2023].
- [3] M. Simons, Spring Boot 2 Moderne Softwareentwicklung mit Spring 5, dpunkt.verlag, 2018.
- [4] W. Golubski, Entwicklung verteilter Anwendungen – Mit Spring Boot & Co, Springer Vieweg, 2019.
- [5] R. Wirdemann, A. Ritscher und J. Mainusch, Scrum mit User Stories, München: Carl Hanser Verlag GmbH & Co. KG, 2022.