Leistungsbeurteilung 324

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Modulnummer | 324 | Verordnung | 2021 | Anbieter Version | 1.0 |
| Titel  Autor | DevOps  nicola.colic@bbbaden.ch | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Datum: | 22.9.25 |
|  |  |  |  |
| Name: | Cranganu-Cretu | Vorname: | Cezar |
| Klasse: | I23b | Experte: | Thut |
| Note: |  |

|  |  |
| --- | --- |
| Hilfsmittel | Folgende Hilfsmittel sind erlaubt:  Alle Unterlagen |
| Prüfungsdauer  Prüfungsinfrastruktur  Notenskala | 1 Woche Bearbeitungszeit, inkl. 5 Lektionen zu 45'  Folgende Infrastruktur steht zur Verfügung:  Eigenes Notebook, Beispiel-Applikation  Linear (Note = 1 + 5 \* ), auf halbe Noten gerundet.  1 ist die schlechteste, 6 die beste Note. |

|  |  |
| --- | --- |
| Objekt | Objekt aus Modulidentifikation |

|  |  |  |
| --- | --- | --- |
| HZ | Handlungsziel | Aufgabe |
| 1 | **Erfasst und verwaltet die Anforderungen und Umsetzungsschritte nachvollziehbar für die Entwicklung im Team.** | 1 |
| 2 | **Erfasst und verwaltet die Anforderungen und Umsetzungsschritte nachvollziehbar für die Entwicklung im Team.** | 2 |
| 3 | **Wendet die Sourcecode- und Artefaktverwaltung für eine anforderungsorientierte Entwicklung im Team an.** | 5 |
| 4 | **Nutzt einen automatisierten Integrationsprozess (Build, Qualitätssicherung/Test der Artefakte).** | 3 |
| 5 | **Nutzt einen automatisierten Deploymentprozess (Installation und Konfiguration der Laufzeitumgebung und Applikation).** | 4 |

Bewertung

|  |  |  |  |
| --- | --- | --- | --- |
| Aufgabe | Gewicht | Punkte | Bewertung |
| 0 | 1 | 3 | Die administrativen Vorgaben sind eingehalten |
| 1-1 | 1 | 3 | Die *labels* und Vorlage wurden gemäss der Vorgabe übernommen |
| 2-1 | 1 | 3 | Die *pre-commit*-Datei erhält die richtigen Befehle für *commit* |
| 2-2 | 1 | 3 | Die *pre-commit*-Datei erhält die richtigen Befehle für *push* |
| 2-3 | 1 | 3 | Die *readme.md* enthält die nötigen Instruktionen, um die *pre-commits* zu installieren. |
| 3-1 | 1 | 3 | Die *action* prüft jeden eingehenden *pull request* auf den richtigen Ast |
| 4-1 | 1 | 3 | Die Auslieferung funktioniert |
| 4-2 | 1 | 3 | Die Geheimnisse wurden korrekt auf die Auslieferungs-Platform übertragen. |
| 4-3 | 1 | 3 | Die *action* löst eine erneute Auslieferung auf dem richtigen Ast aus |
| 5-1 | 2 | 3 | Die Änderung wurde gemäss des *github flow* umgesetzt |

Hinweise

* Diese Prüfung ist eine Einzelarbeit.
* Es dürfen keine alten Prüfungen und/oder Daten von alten Prüfungen verwendet werden.
* Betrügereien, wie Spicken o.Ä., oder Betrugsversuche führen zum Ausschluss aus der Prüfung.
* Im Falle eines Betrugs gilt die Leistungsbeurteilung als absolviert, aber nicht bestanden.
* Alle Dateien müssen in einem Ordner abgelegt werden, dessen Name dem Format NameVornameLB-### entspricht. (z.B. MuellerHansLB-333).
  + Die gepackte Datei muss gleich wie der Ordner benannt werden.
  + Programmierprojekte müssen gleich wie der Ordner benannt werden.
* Folgende Formate sind bei den elektronischen Lösungen erlaubt:
  + Programmcode: Visual Studio-Projekte oder Dateien mit Ordnerhierarchie
  + Bilder: jpg, gif, png
  + Texte: Markdown md
  + Anderes: Microsoft Excel, Microsoft PowerPoint
* Die elektronischen Lösungen werden am Schluss der Prüfung auf Moodle hochgeladen.
* Elektronisch abgegebene Lösungen werden mit Vorteil gezippt.
* Sie sind verantwortlich dafür, dass Sie alle Dateien abgeben.
* Alle Daten dieser LB (Aufgabendateien und Lösungen) müssen sofort nach dem Abschluss der LB und dem Hochladen gelöscht werden.
* Vor Beginn der Prüfung abgegebene Daten:
  + Beispiel-Projekt
* Regelung zu verspätet eingereichter Leistungsbeurteilungen:
  + Pro angefangene 24h Verspätung wird eine halbe Note (0.5 Notenpunkte) abgezogen. Verspätet eingereichte Leistungsbeurteilungen werden korrigiert, bewertet und dann mit dem Abzug der Verspätung verrechnet.
  + Aufschiebende Wirkungen haben nicht planbare Absenzen wie Krankheit oder Unfall, sofern Sie vom Berufslernenden bei Eintritt des Ereignisses kommuniziert wurden. Die Lehrperson entscheidet über die Abgabetermine.

Administrative Vorgaben

1. Laden Sie das Beispiel-Projekt von Moodle herunter, erstellen Sie auf github.com eine neue Ablage namens **NameVornameLB-324** und stellen Sie sicher, dass diese auf öffentlich gestellt ist. Alternativ dazu können Sie das [Projekt](https://github.com/Aequivinius/tagebbbuch)[[1]](#footnote-1) *forken*.

☝️ Die Abgabe der Prüfung erfolgt zweifach: Einerseits geben Sie die URL zu Ihrer Ablage über das Formular auf Moodle ab. Tun Sie dies am besten gleich jetzt. Andererseits laden Sie zur Sicherheit Ihr Projekt als Archiv auf Moodle hoch. Dies erledigen Sie, wenn Sie mit der Arbeit an dieser LB fertig sind, indem Sie Ihr Projekt von github.com herunterladen, das Archiv umbenennen, und auf Moodle wieder hochladen.

☝️ Wenn Sie eine nachfolgende Aufgabe danach fragt, etwas zu dokumentieren, tragen Sie dies in der vorbereiteten readme.md an der entsprechenden Stelle ein.

☝️ Sollten Sie ein bestehendes Projekt haben, welcher mit der nachfolgenden Beschreibung vergleichbar ist, dürfen Sie die Prüfung auch an diesem Projekt absolvieren. Wie Sie den Bewertungskriterien entnehmen können, wird die eigentliche Programmier-Arbeit nicht bewertet, sondern lediglich der Umgang mit den CI/CD-Werkzeugen. In diesem Fall halten Sie sich an alle nachfolgenden Aufträge, verwenden aber als Ausgangslage Ihren bestehenden Code, welchen Sie in eine neue, wie oben benannte Ablage auf github.com kopieren.

Fallbeschreibung

Ähnlich zur zu-bbbearbeiten-Applikation im Modul ist Gegenstand dieser LB eine Tagebbbuch-Applikation. Gegenwärtig ist diese noch in einer rudimentären Prototyp-Phase, doch die Entwicklung an der Applikation soll schnell vorangetrieben werden. Darum sind Sie eingestellt, um eine professionelle CI/CD-Umgebung einzurichten. Im Gegensatz zum Modul soll die Entwicklung an der Tagebbbuch-Applikation jedoch auf 3 Arten von Ästen stattfinden:

* Auf dem *main*-Ast werden nur Änderungen übernommen, wenn diese auch direkt ausgeliefert werden sollen. Die ausgelieferte Version der Applikation und der *main*-Ast stimmen also immer überein.
* Auf dem *dev*-Ast werden alle Änderungen übernommen, welche die Programmierer bei der Entwicklung von neuen Funktionen und Behebung von Fehlern für abgeschlossen betrachten. Hier darf nur getesteter Code übernommen werden.
* Auf den *feature*-Ästen findet die eigentliche Entwicklung statt, wobei für jede neue Programmier-Arbeit ein neuer Ast ausgehend vom *dev*-Ast erstellt wird.

Wie die zu-bbbearbeiten-Applikation lässt sich diese Applikation wie folgt starten:

|  |
| --- |
| pip install -r requirements.txt flask run |

☝️ Die Definition von $env etc. entfällt hier. Diese Applikation erwartet eine .env-Datei, welche eine Zeile im Format von PASSWORD="einSehrGeheimesPasswort" enthält. Diese müssen Sie erstellen.

Situation

1. Allen Änderungen am Code geht eine auf github.com erstellte *issue* voraus, unabhängig davon, ob es sich um eine Fehlerbehebung oder neue Funktionalität handelt. Für letzteren Fall erstellen Sie aber eine Vorlage auf github.com, welche es dem Benutzer erleichtert, die zu programmierende Funktionalität als Anforderung festzuhalten. Die Anforderung soll dabei folgenden Aufbau haben:

|  |
| --- |
| *Zielsystem + Priorität + Systemaktivität + allf. Ergänzungen + Funktionalität + Bedingungen*, wobei:   * Zielsystem das zu betroffene (Teil-)System ist * Priorität durch *muss* bei hoher, *soll* bei mittlerer und *wird* bei niedriger Priorität angegeben wird * Systemaktivität beschreibt, ob das System selbständig handeln soll, einem Benutzer eine Funktion anbieten soll oder einer Schnittstelle bedarf. Wählen Sie die Formulierung *dem Administrator o. ä. die Möglichkeit bieten* für Benutzerfunktionen, und *fähig sein* für Schnittstellenanforderungen. * Zeitliche Bedingungen mit *wenn*, und logische Bedingungen mit *falls* beschrieben werden.   Beispiel: Reflect Media Player (Zielsystem) muss (= hohe Priorität) dem Benutzer die Möglichkeit bieten (= Benutzerinteraktion), J3D Szenengraphen aus einer wrml-Datei über das Netzwerk (Ergänzung) zu laden (Funktionalität), falls der Benutzer eingeschrieben ist. |

In der genauen Ausgestaltung der Vorlage sind Sie frei. Jede *issue* soll manuell etikettiert werden können mit ihrem Typ (Funktionale Anforderung, Qualitätsanforderung, Randanforderung).

✍️ Erstellen Sie die Vorlage und die nötigen Etiketten in Ihrer Ablage auf github.com.

1. Um sicherzustellen, dass nur getesteter Code eingebucht werden kann, benutzen Sie *pre-commit* wie in den Aufträgen des Moduls:

* Schreiben Sie eine entsprechende pre-commit-config.yaml, welche bei jedem *push* den Code testet.
* Erweitern Sie diese Datei, sodass Ihr Code bei jedem *commit* automatisch formatiert wird.
* Beschreiben Sie in Ihrer readme.md, wie diese zu benutzen ist; d.h. welche Befehle der Programmierer ausführen muss, damit die Automatisierungen ausgeführt werden.

☝️ Das ist anders als in den Aufträgen zum Modul, wo wir bei jedem *commit* getestet haben! Ggf. müssen Sie die Dokumentation von [pre-commit](https://pre-commit.com/#confining-hooks-to-run-at-certain-stages)[[2]](#footnote-2) konsultieren.

1. Erstellen Sie im Verzeichnis .github/workflows eine neue Datei im richtigen Format, welche bei jedem *pull request* auf den *dev*-Ast ebenfalls die Tests laufen lässt. Darüber hinaus sollen keine weiteren Aktionen ausgelöst werden.
2. Liefern Sie Ihre Applikation auf Azure aus.

* Fügen Sie die URL Ihrer laufenden Applikation Ihrer readme.md hinzu.
* Damit das Einschreiben richtig funktioniert, müssen Sie das geheime Passwort aus der .env-Datei korrekt in Azure übertragen. Dokumentieren Sie die nötigen Schritte hierzu in Ihrer readme.md
* Konfigurieren Sie Ihre Ablage so, dass bei jedem erfolgreichen *merge* in den *main*-Ast eine automatische Auslieferung auf Azure erfolgt.

🔑 Die .env-Datei finden Sie nur auf Moodle. Sie können sie aber auch einfach selbst erstellen im folgenden Format. Ändern Sie aber bei der Auslieferung das Passwort, sodass es gleich Ihrem github-Benutzernamen lautet.

PASSWORD="MaxMuster\_TheProGrammer"

1. Implementieren Sie eine neue, einfache Funktionalität gemäss des Ablaufs von *github flow*. Den dazugehörigen Testfall finden Sie nachfolgend. Wie Sie die Funktionalität tatsächlich implementieren, ist Ihnen überlassen.

def test\_add\_entry\_with\_happiness(client):

# Test adding an entry with happiness

response = client.post(

'/add\_entry', data={'content': 'Test Entry Content', 'happiness': '😃'})

# Check if the response is a redirect to the index page

assert response.status\_code == 302

assert response.headers['Location'] == '/'

# Check if the entry was added to the database with the correct happiness

entry = entries[0]

assert entry is not None

assert entry.content == 'Test Entry Content'

assert entry.happiness == '😃'

☝️ Löschen Sie den neu erstellten Ast *nicht*, auch wenn dies üblich wäre.

1. https://github.com/Aequivinius/tagebbbuch [↑](#footnote-ref-1)
2. https://pre-commit.com/#confining-hooks-to-run-at-certain-stages [↑](#footnote-ref-2)