

# LB 324

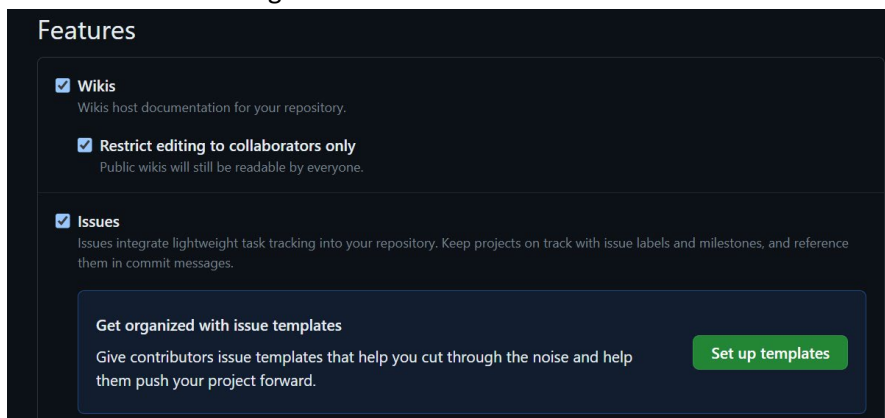
Olivier Kistler, I21b

# Aufgabe 1

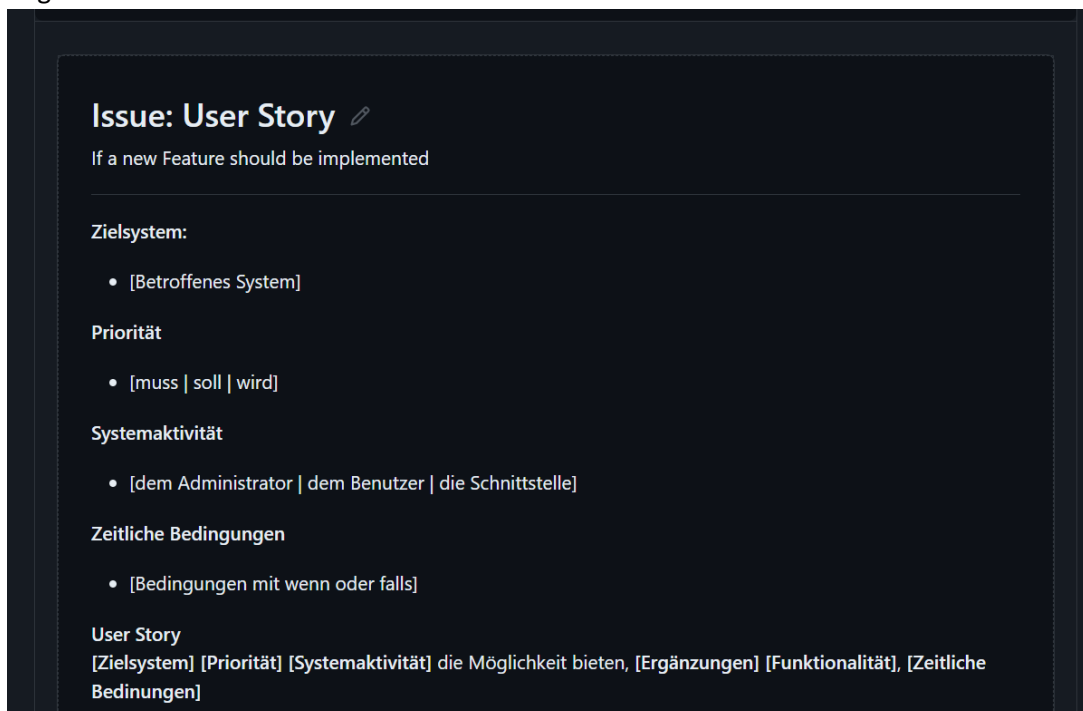
## Vorlage

In dieser Aufgabe habe ich eine Vorlage für User Storys erstellt. Diese Vorlage soll es dem Benutzer einfacher machen, ein neues Issue für eine Funktionalität zu erstellen. Um eine Vorlage für ein Issue zu erstellen, muss man wie folgt vorgehen:

- Zuerst navigiert man zu den Einstellungen des Repositorys und scrollt dann etwas herunter, bis man in der Features Section ankommt. Hier kann man dann einfach auf «set up Template» klicken und eine Vorlage erstellen.



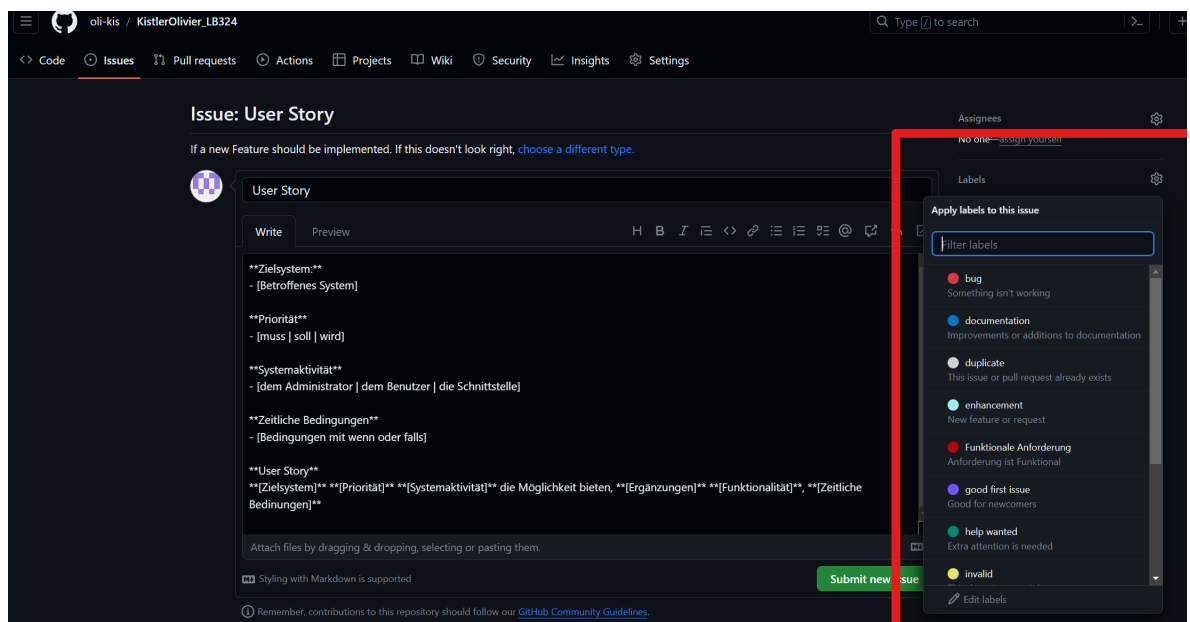
- Die Bearbeitung der Vorlage unterstützt die Markdownschreibweise. Meine Vorlage sieht wie folgt aus:



## Labels

Ausserdem musste ich drei neue Labels hinzufügen für Funktional-, Qualitäts-, und Randanforderungen. Um dies zu tun kann man folgende Schritte befolgen:

- Zuerst navigiert man in den Issue Tab des Repositorys
- Danach erstellt man ein neues Issue
- Beim Editieren des Issues, kann man rechts «Labels» auswählen
- Nun kann man die Option «Edit Labels» auswählen und einfach neue Labels für das Repo hinzufügen.



## Aufgabe 2

### Code testen bei «push»

Ich konnte dies so lösen, dass ich mit dem keyword «stages» die Phasen festlegen konnte, in der die Aktion ausgeführt werden soll.

```
- repo: local
  hooks:
    - id: run-tests
      name: Run Tests
      entry: python -m pytest
      language: system
      always_run: true
      stages: [push]
```

### Bei commit formatieren

Dies war einfach, da wir diese Aufgabe schon während dem Bearbeiten des Moduls lösen mussten. Ich musste auch keine «stages» angeben, da die Standardphase bereits «commit» ist. Ich verwende hier den Formater von «black» in der Version 23.10.1 und Python in der Version 3.11.

Ausserdem habe ich eine Anleitung für den Programmierer in der readme.md erstellt.

### Aufgabe 3

Hier musste ich eine workflow-datei erstellen, welche bei jeder pull-request auf den dev-branch die Tests laufen lässt. Dies hat den Grund, dass auf dem dev-branch code gepusht wird, welcher kurz vor der Veröffentlichung steht. Deshalb ist es sehr wichtig, dass dieser Code einwandfrei funktioniert.

```
1  name: Test on Pull Request
2
3  on:
4    pull_request:
5      # Define which branches trigger the job
6      branches:
7        - dev
8
9  jobs:
10   test:
11     #Define system
12     runs-on: ubuntu-latest
13
14     steps:
15       # Check out of the Code
16       - name: Check Out Code
17         uses: actions/checkout@v2
18
19       # Set up the python environment with the correct version
20       - name: Set up Python
21         uses: actions/setup-python@v2
22         with:
23           python-version: 3.11
24
25       # Install the dependencies that are needed to run
26       - name: Install Dependencies
27         run: pip install -r requirements.txt
28
29       # Run the tests
30       - name: Run Tests
31         run: pytest
32
```

## Aufgabe 4

Ich hatte Probleme mit Azure, weshalb ich mich dazu entschieden habe, meine App zu dockerisieren und sie auf Render.com zu veröffentlichen. Damit kann ich dasselbe Ergebnis wie mit Azure erzielen.

Ich habe also damit angefangen ein Dockerfile zu erstellen:

```
Dockerfile > ...
1  # Select a Python image
2  FROM python:3.9-slim
3
4  # Set the working directory in the container
5  WORKDIR /app
6
7  # Copy the requirements file into the container
8  COPY requirements.txt .
9
10 # Install the requierement
11 RUN pip install --no-cache-dir -r requirements.txt
12
13 # Copy the rest
14 COPY . .
15
16 # Expose the port
17 EXPOSE 5000
18
19 # Define the command to run the application
20 CMD ["gunicorn", "app:app", "--bind", "0.0.0.0:5000"]
21
```

Ich konnte das Image dann mit diesem Befehl «docker push olikis/lb324-olivierkistler:latest» auf mein DockerHub Repo pushen.

Nun konnte ich das Image als Basis für die Veröffentlichung auf Render.com angeben.

You are deploying a web service for [docker.io/olikis/lb324-olivierkistler:latest](https://docker.io/olikis/lb324-olivierkistler:latest)

Name

A unique name for your web service.

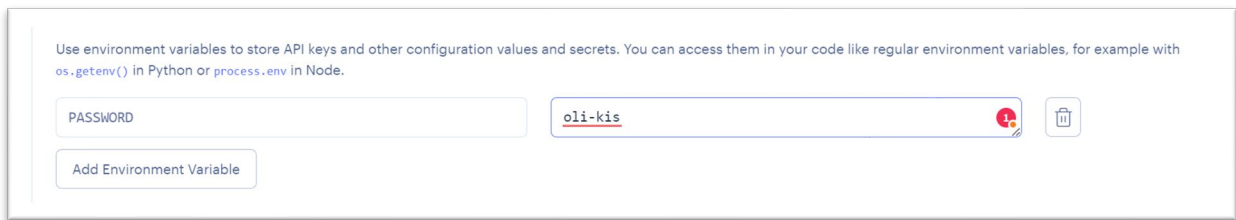
LB324-OlivierKistler

Region

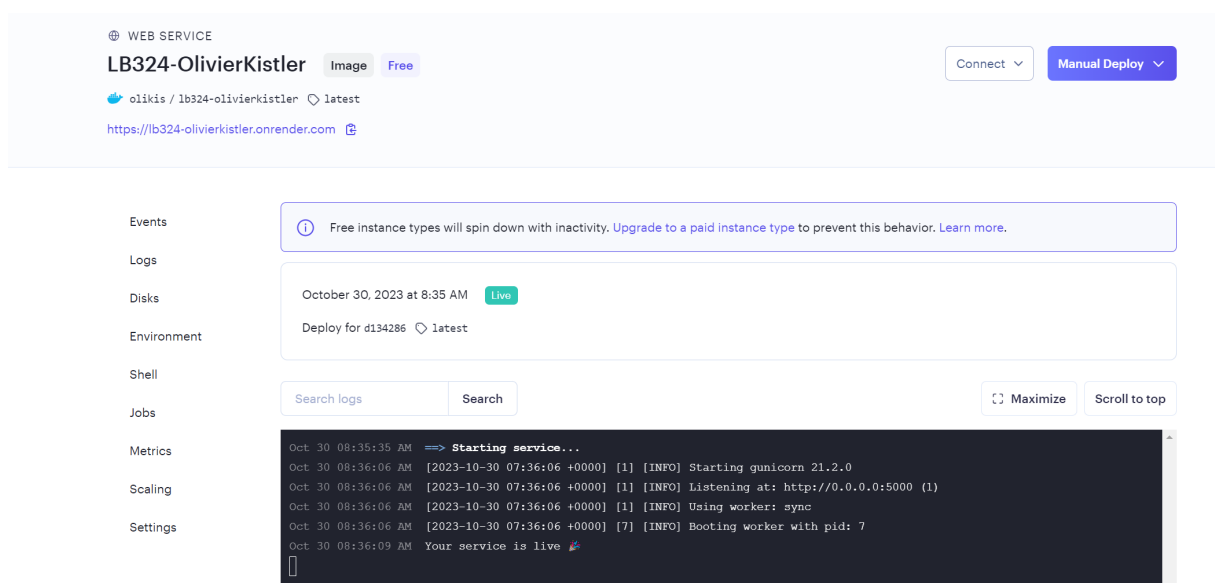
The region where your web service runs. Services must be in the same region to communicate privately and you currently have services running in Frankfurt.

Frankfurt (EU Central)

In der Aufgabe war gefordert, dass man das Passwort im Azure Key Vault sicher aufbewahrt. Glücklicherweise hatte Render ein ähnliches System, um geheime Variablen aufzubewahren. Dadurch konnte ich mein Passwort als Environment-Variable angeben und ganz normal im Code darauf zugreifen.



Danach war die Website auch schon live und ich konnte darauf zugreifen.



Ebenfalls war gefragt, dass bei jedem erfolgreichen merge auf den main-ast, die App neu deployed wird. In Azure gibt es dazu eine Möglichkeit, mit Docker und Render allerdings nicht. Meine Lösung war es dann, ein neues workflow-file zu erstellen. Das Image wird bei jeder pull-request auf den main-branch auf das Dockerhub gepusht. Auf Github gibt es die Möglichkeit geheime Variablen zu speichern. Diese habe ich genutzt, um mein Docker Passwort sicher zu bewahren.

Docker würde die Funktion eigentlich auch unterstützen bei jedem neuen commit das Image neu zu erstellen. Diese Funktion ist leider aber kostenpflichtig.

### Automated Builds

Connect your account to GitHub or Bitbucket to:


- Automatically build and tag new images whenever your repository is updated
- Keep faulty images out of your repository with automated test runs
- Sync your Docker Hub readme with the one in your code repository

Available with Pro, Team and Business subscriptions. [Read more about automated builds](#).

[Upgrade](#)

### Build Activity

Overview of your build activity of the last 100 builds



Nun wird mein Image automatisch gepusht, wenn Änderungen in GitHub vorgenommen werden. Der nächste Schritt war es, Render zu automatisieren. Render stellt eine URL zur Verfügung, welche die App automatisch neu deployed, wenn sie aufgerufen wird. Ich konnte also nur noch ein Job hinzufügen, welcher die URL aufruft, nachdem das Docker-Image gepusht wurde. Daraus ergibt sich folgendes File und ein vollautomatischer Deployprozess.

```
1  name: Docker Image CI
2
3  on:
4    pull_request:
5      # Define what branch triggers the job
6      branches:
7        - oli-kis
8
9  jobs:
10   build-and-push:
11     # Define System
12     runs-on: ubuntu-latest
13     steps:
14       # Check out the Code
15       - name: Check out the code
16         uses: actions/checkout@v2
17
18       # Set up the docker environment
19       - name: Set up Docker Buildx
20         uses: docker/setup-buildx-action@v1
21
22       # Log into Docker
23       - name: Log in to Docker Hub
24         uses: docker/login-action@v1
25         with:
26           username: olikis
27           # Get the password, that I defined in the secrets tab in my repo
28           password: ${ secrets.DOCKERHUB_PASSWORD }}
29
30       # Build and push to docker
31       - name: Build and push Docker image
32         uses: docker/build-push-action@v2
33         with:
34           context: .
35           push: true
36           tags: olikis/lb324-olivierkistler:latest
37
38       # Open deploy url from render to trigger deploy
39       - name: Deploy to Render
40         run: curl -X POST 'https://api.render.com/deploy/srv-ckvlph237rbc7392dvsg?key=_uR-TjI5n6o'
```

Link zu meinem Docker Image: [Image](#)

Link zur Website (Könnte länger zum Laden brauchen, da Website nur bei Aufruf gestartet wird):  
[Website](#)