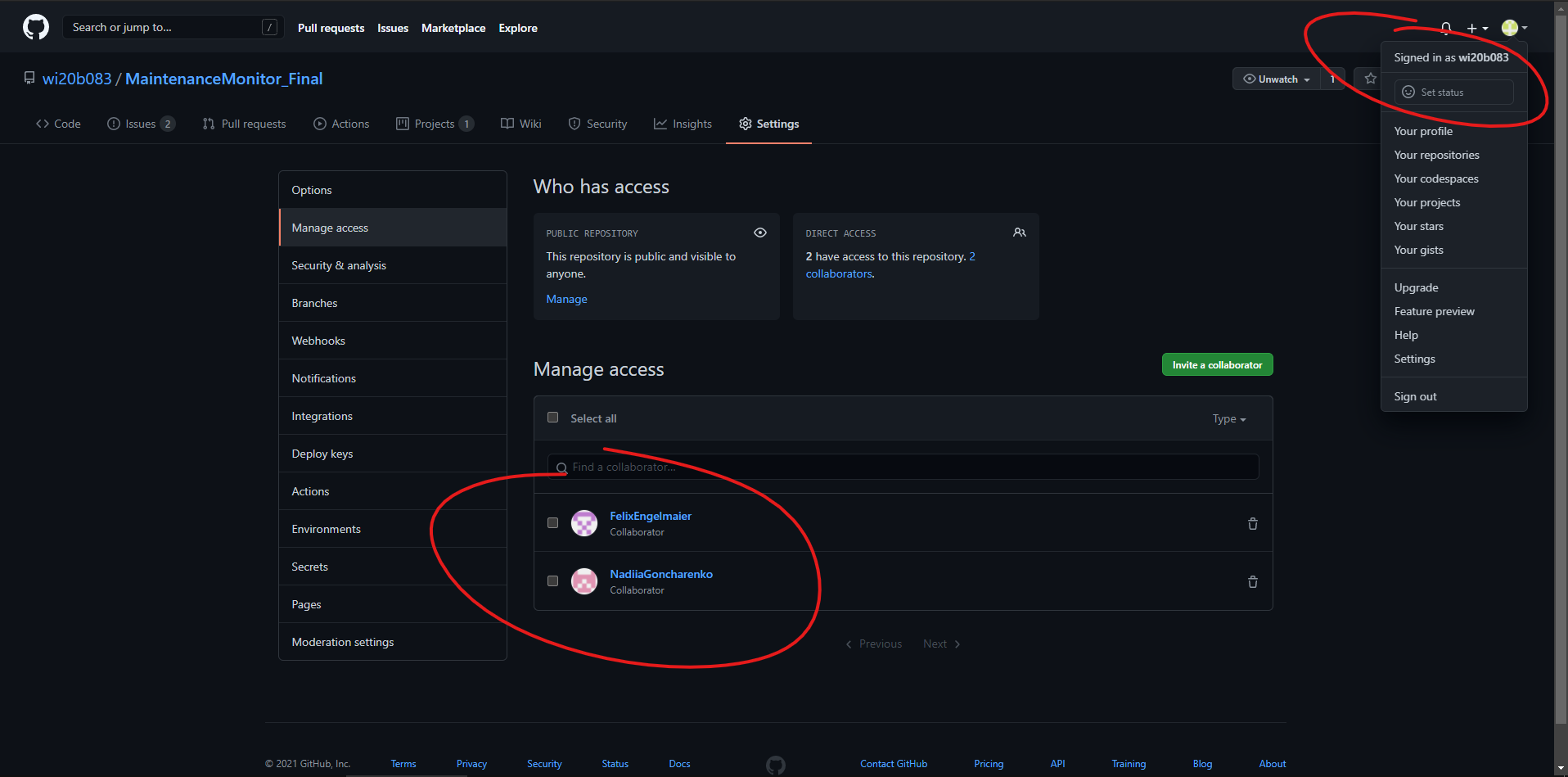
MaintenanceMonitor Dokumentation

Nadiia Goncharenko, Felix Engelmaier, Florian Huber

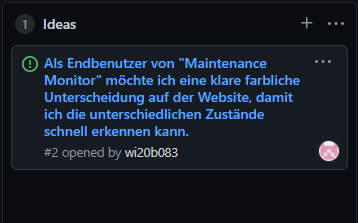
# Projekt in GitHub erstellt

Zunächst haben wir ein GitHub Projekt erstellt. Der Rest des Teams wurde als Contributor hinzugefügt.



# Board erstellt

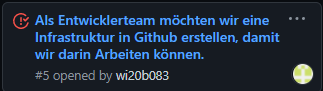
Hier haben wir das “Basic Kanban“ Board verwendet, wie es in der LV demonstriert worden ist. Als Lane haben wir „Ideas“ hinzugefügt und ein Demo-Issue hineingegeben, das zwar zu dem Projekt passend, allerdings nicht im Rahmen der Anforderungen ist.



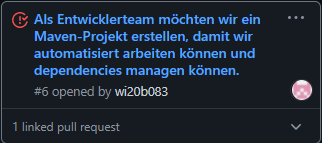
# User-Stories verfasst

Der erste Schritt im Projekt war die Verfassung der User-Stories. Hier aufgelistet (und gruppiert) sind die User-Stories in logischer Reihenfolge, nicht in der, in der sie verfasst wurden. Folgende User-Stories haben wir gefunden:

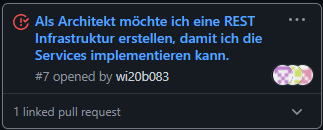
## GitHub-Infrastruktur



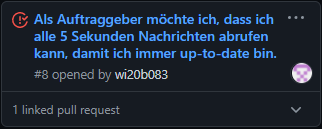
## Maven-Grundstruktur

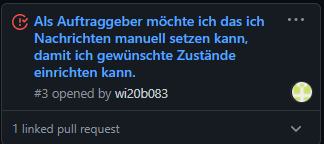


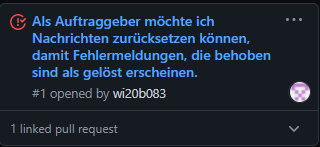
## REST-Service

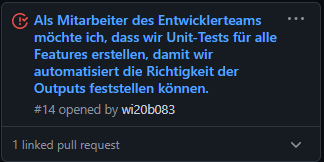


## Controller-Funktionen

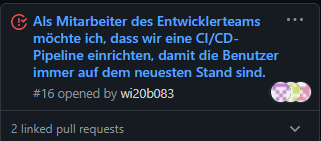




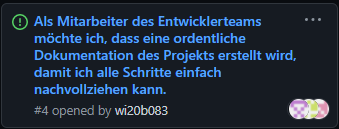




## CI/CD



## Dokumentation



# Reihenfolge festgelegt

Nachdem die User-Stories fertiggestellt waren, haben wir uns auf eine logische Reihenfolge (siehe Abschnitt 3) geeinigt. Dabei haben wir uns überlegt, welche Issues andere als Voraussetzung haben und so die logische Reihenfolge bestimmt. (zum Beispiel wäre es unlogisch einen Unit-Test vor dem Feature zu schreiben, oder Features zu programmieren ohne eine GitHub Infrastruktur zu haben).

# Sprint Scope festgelegt

Die Sprint-Scopes haben wir wie folgt festgelegt:

1. User-Stories verfassen
2. Basic-Structures implementieren (GitHub, Maven, REST)
3. Controller programmieren
4. CI/CD Pipeline bauen + Dokumentation abschließen

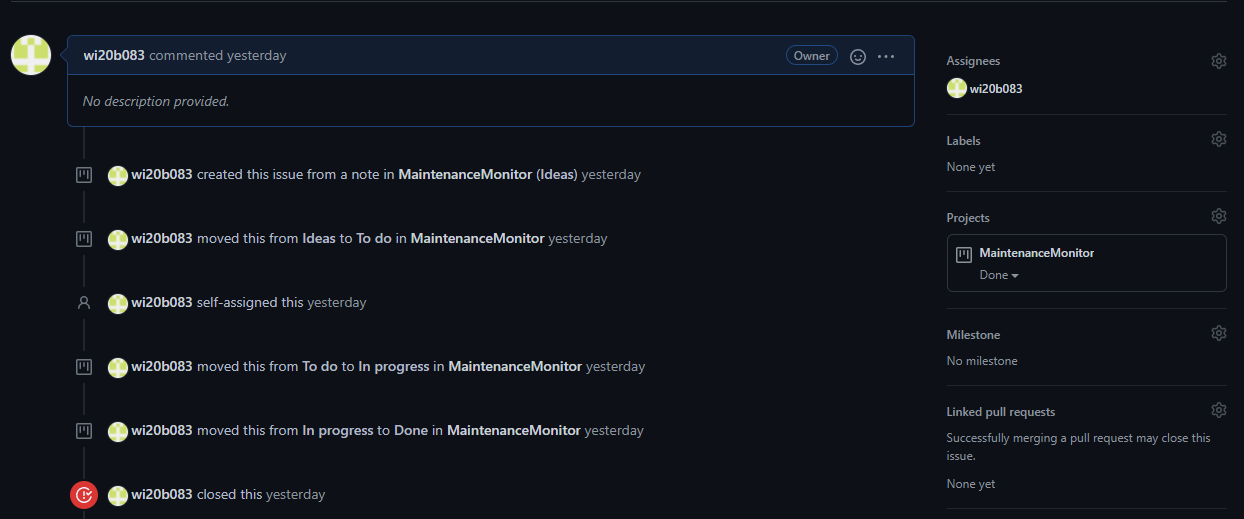
# Storys bearbeitet

Features haben wir jeweils als eigenen Branch im Format „feature/featureName“ mit GitFlow in Sourcetree bearbeitet.

Für die genauen Inhalte der User-Stories siehe Abschnitt 3.

## GitHub-Infrastruktur

### Verlauf der Story in GitHub



### Schritte

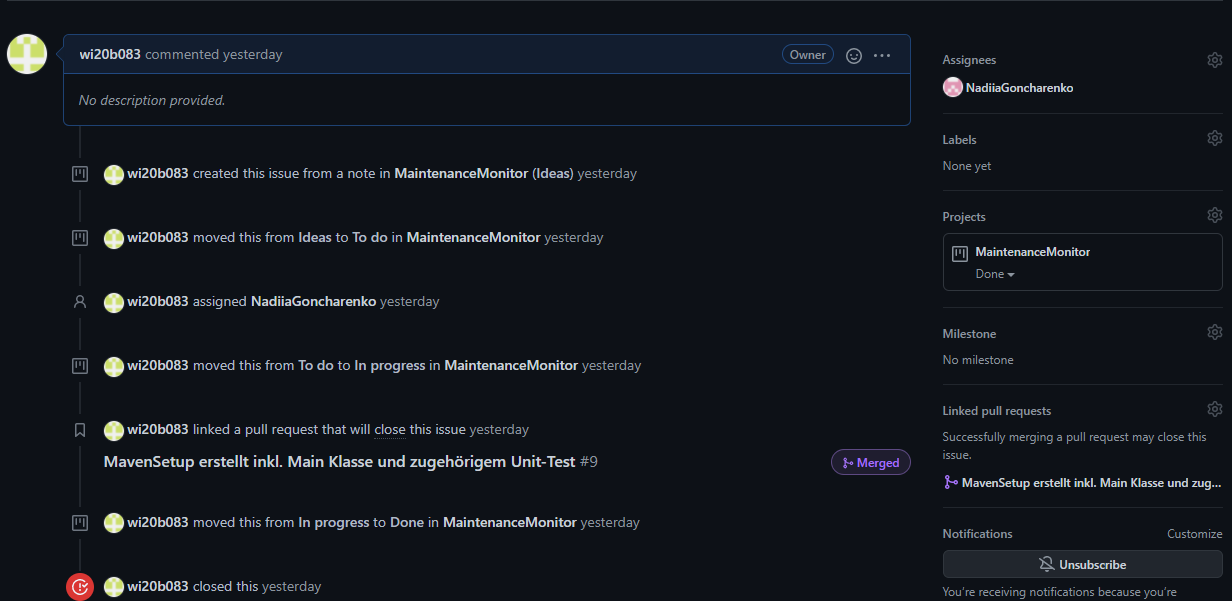
* Projekt in GitHub erstellen
* User-Stories verfassen
* Repo initialisieren
* Repo in SourceTree klonen

### Output

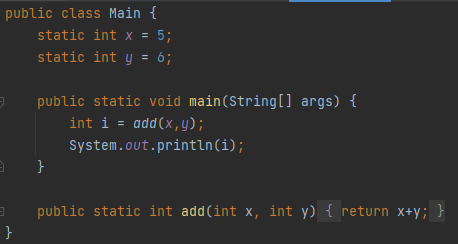
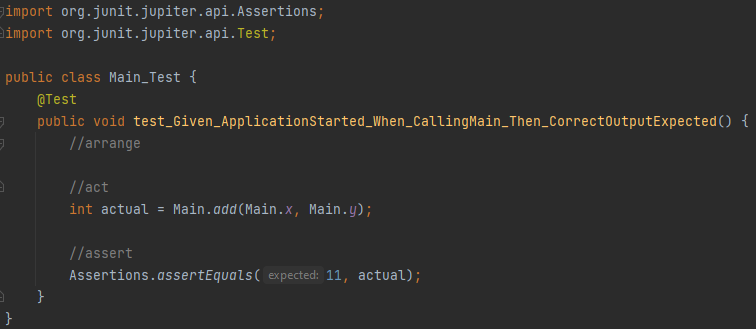
* Basic Infrastruktur in GitHub
* .gitignore File
* MIT-License File

## Maven

### Verlauf der Story in GitHub



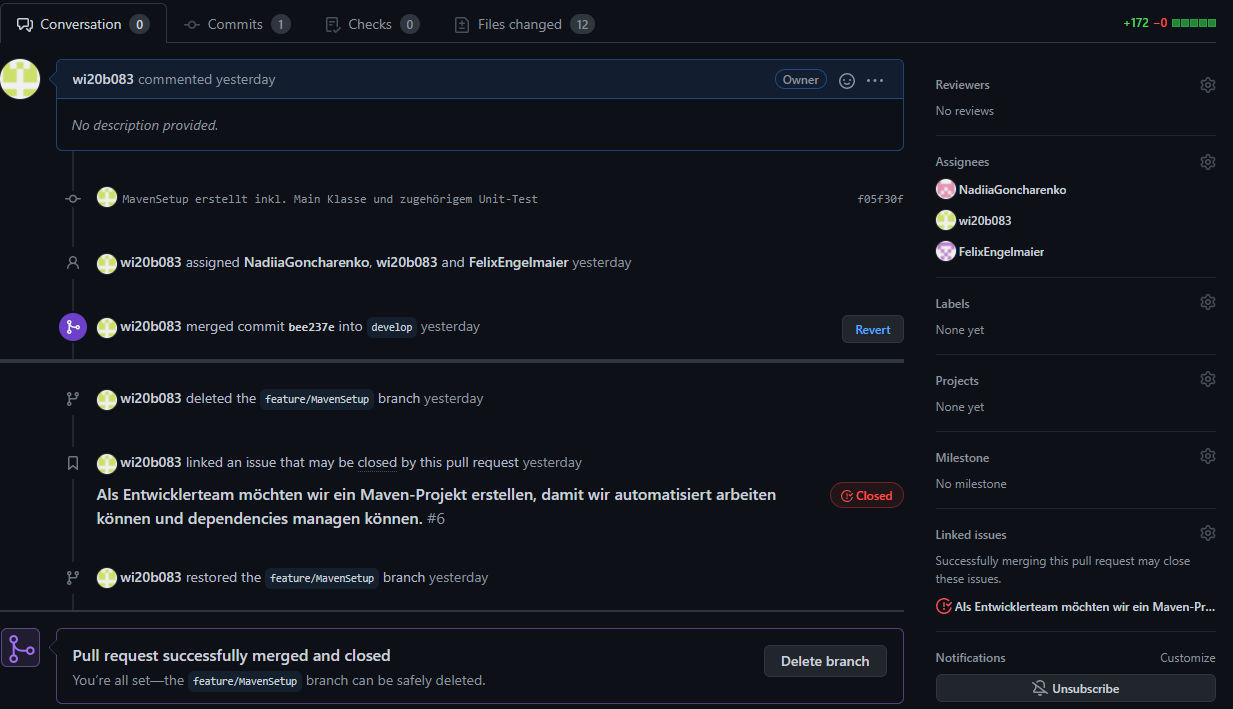
### Schritte

* Neues Feature starten 
* IntelliJ 🡪 neues Maven Projekt starten
* Main-Klasse erstellen 
* Main-Test erstellen 
* Den Maven-Lifecycle durchlaufen und pom File an Fehler anpassen (encoding, dependencies, plugins)

### Output

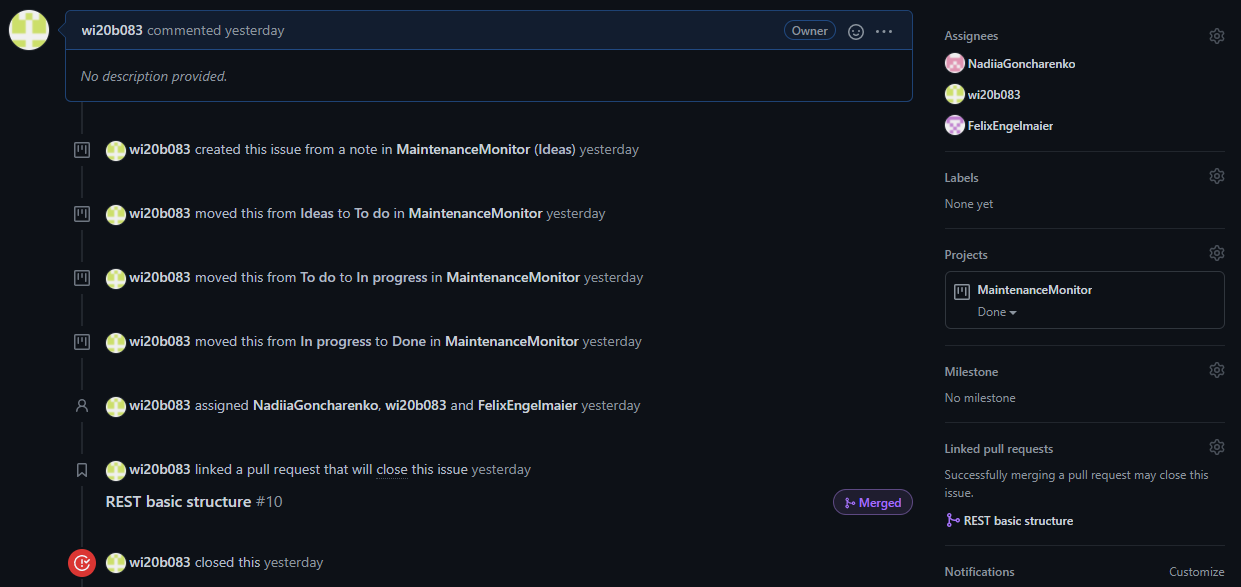
* 
* Basic Maven Projekt in IntelliJ angelegt
* Main-Klasse mit Unit-Test erstellt
* MavenSetup abgeschlossen

### Linked pull requests

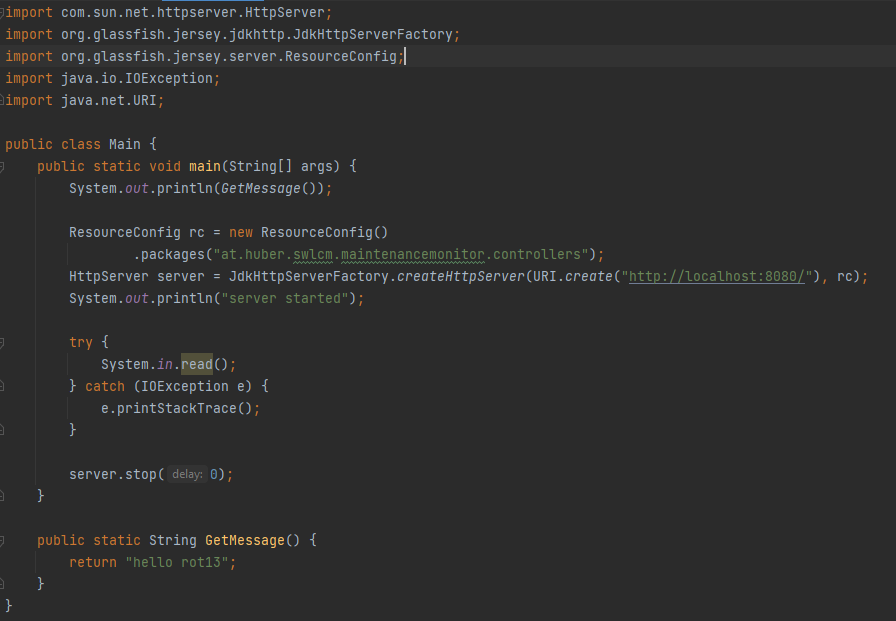


## REST

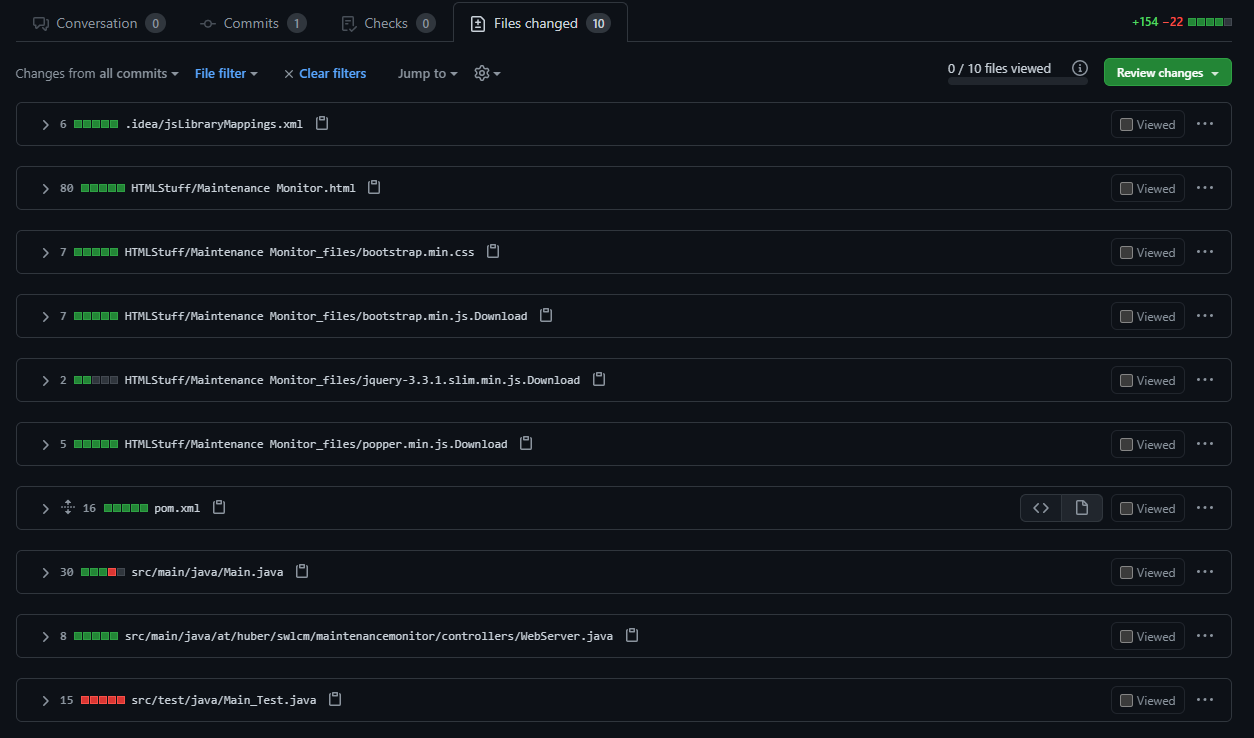
### Verlauf der Story in GitHub



### Schritte

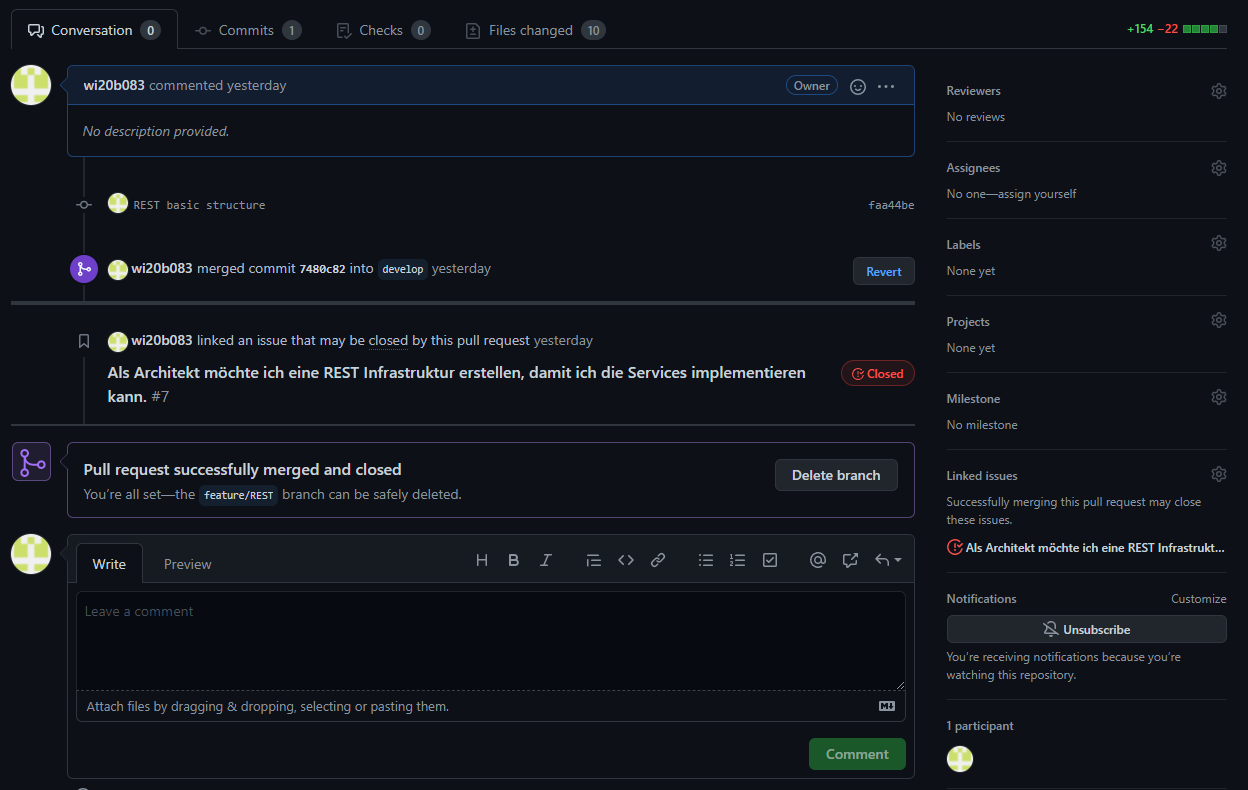
* Neues Feature Starten 
* Main-Klasse umschreiben (kopiert aus ROT13 Projekt, das in der LV gemacht wurde, danach Package-Name angepasst) 
* Main-Test entfernen
* (Hier war der Versuch das Frontend-HTML File hinzuzufügen (siehe Output))

### Output



* Server läuft, Besic REST-Struktur abgeschlossen

### Linked pull requests

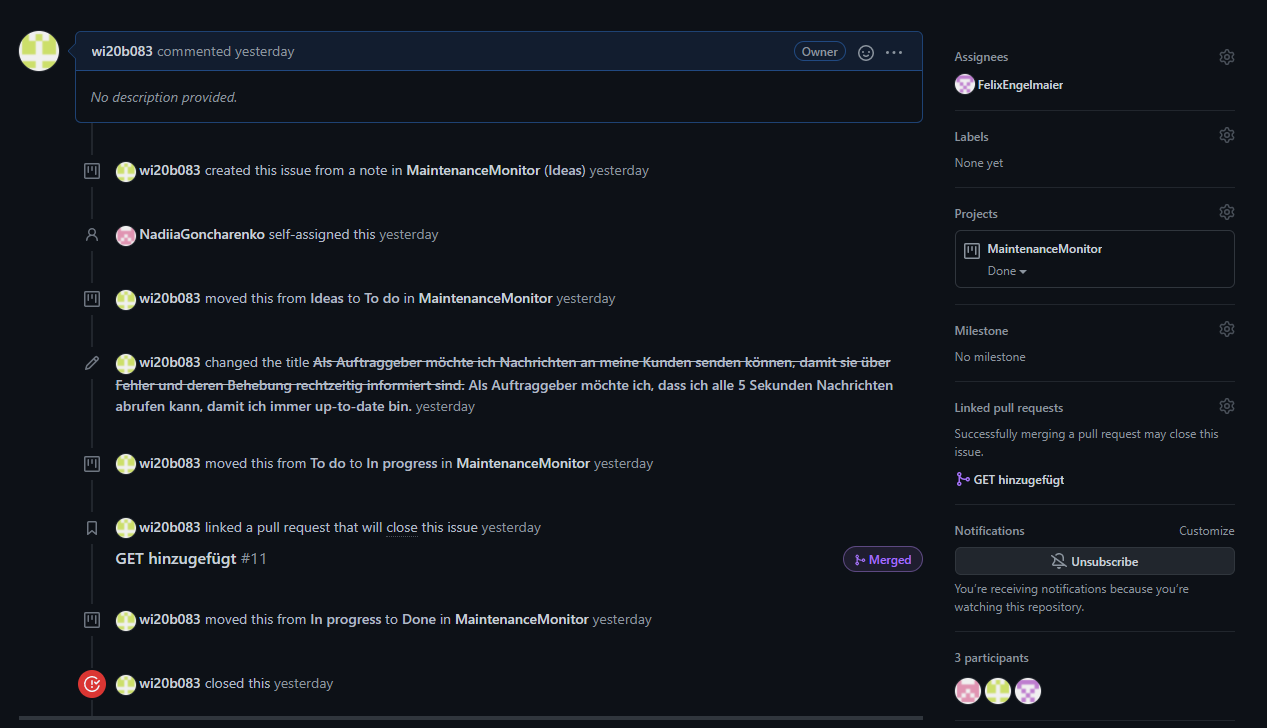


## Nachrichten Abrufen + Default message

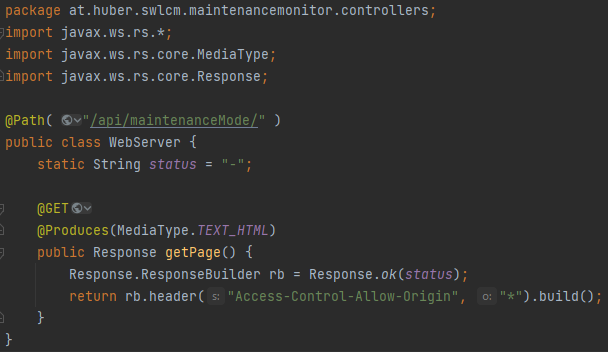
### Anmerkung

Hier war die Anstrengung groß, die Aufgabe zu verstehen und umzusetzen. Die implementierte Funktion ist mit dem Moodle-Frontend kompatibel. Außerdem wurden die Feature User-Stories (GET, POST und DELETE) aufgrund von Verständnisproblemen bearbeitet und neu Verfasst (die alten Fassungen wurden in Abschnitt 3 nicht aufgelistet)

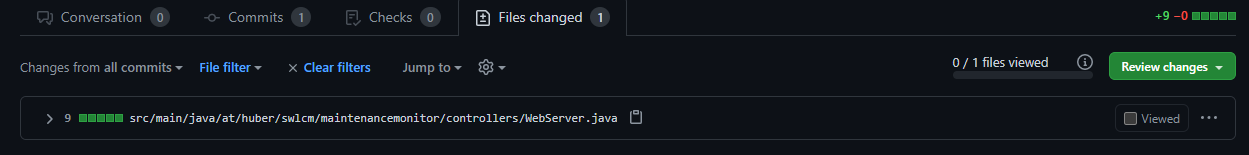
### Verlauf der Story in GitHub



### Schritte

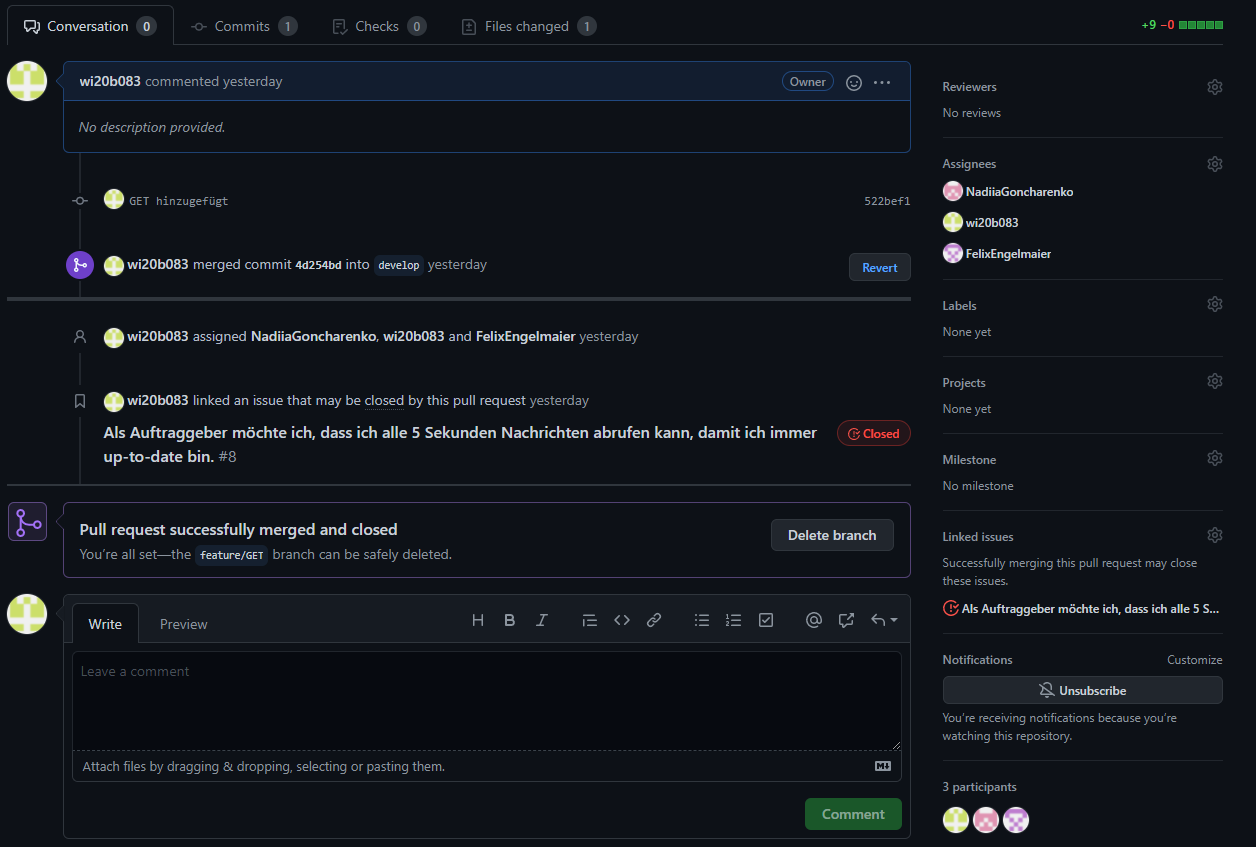
* Neues Feature starten 
* Klasse MaintenanceMonitorController erstellen (hier noch WebServer, da aus dem Forum kopiert und danach adaptiert) 

### Output



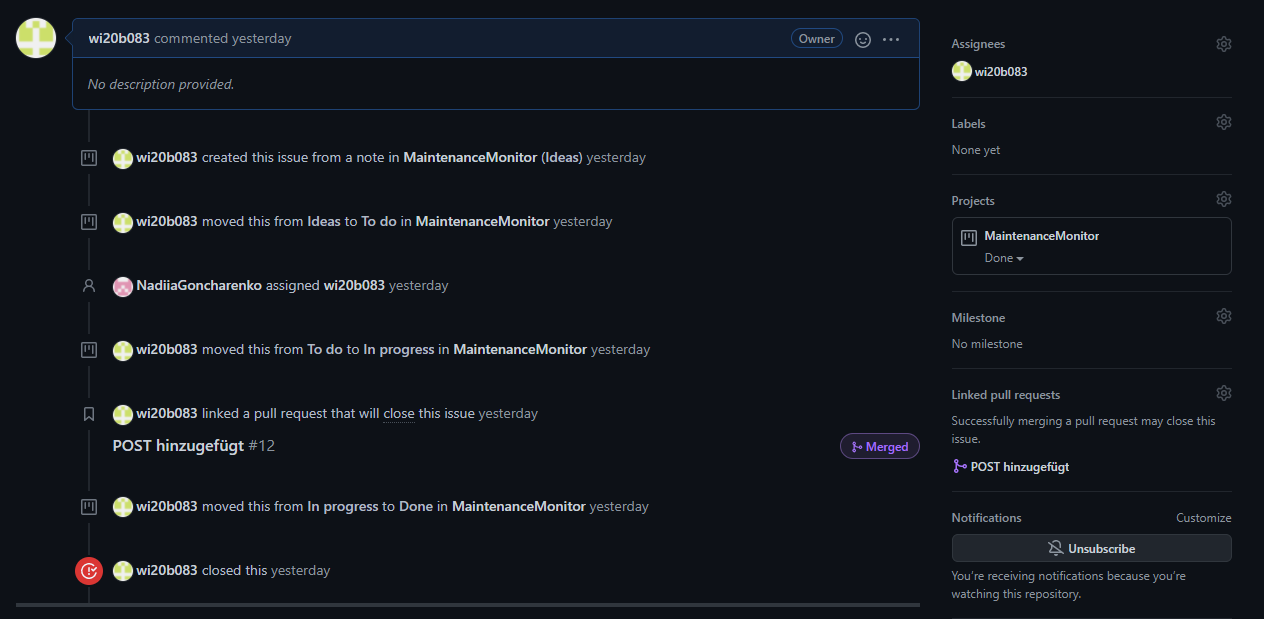
* Default Status “alles OK” hinzugefügt

### Linked pull requests

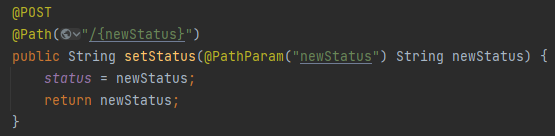


## Nachricht setzen

### Verlauf der Story in GitHub



### Schritte

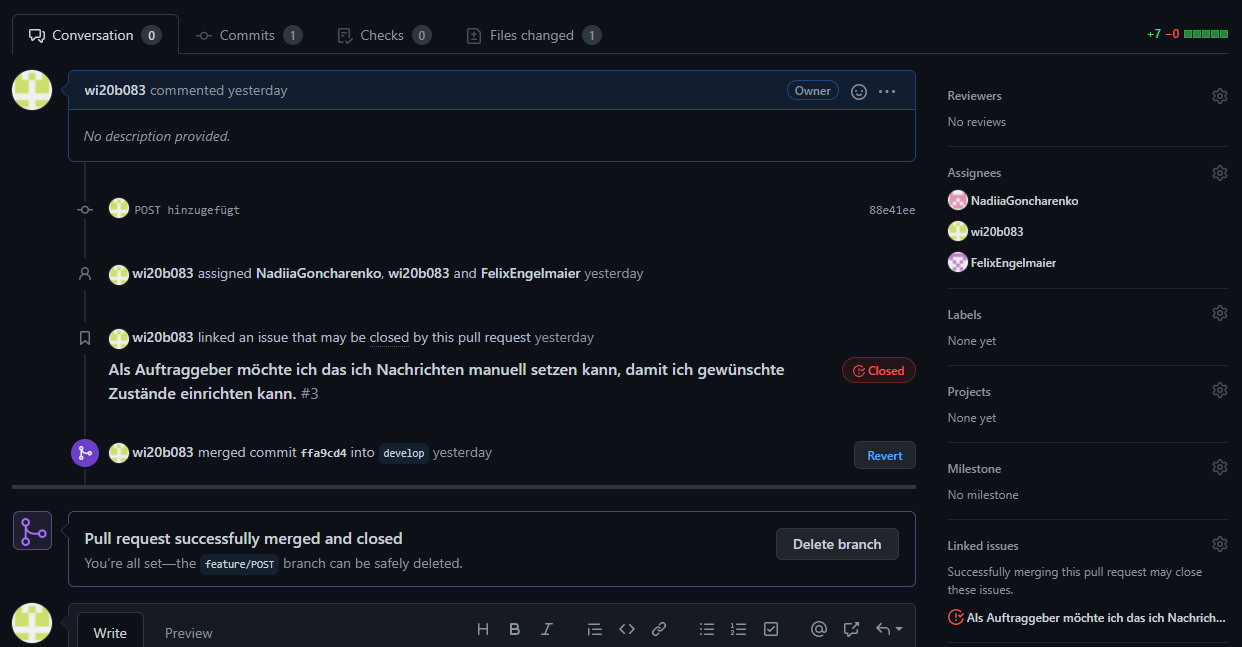
* Neues Feature starten 
* Eine set-Funktion mit der POST- Annotation hinzufügen (return-Wert wird beim Bearbeiten der Unit-Tests verändert)

### Output



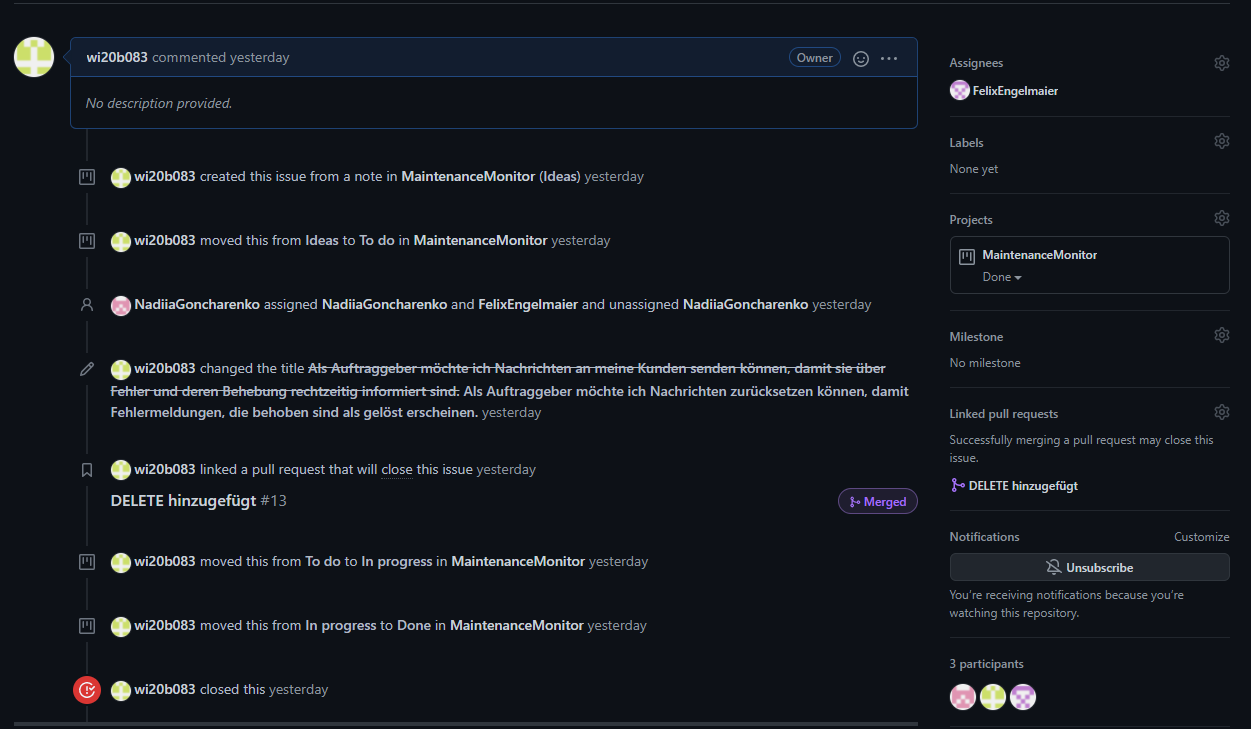
* POST-Funktion
* Manuelle Nachrichtensetzung möglich

### Linked pull requests



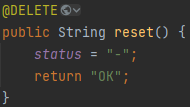
## Nachrichten- Reset

### Verlauf der Story in GitHub

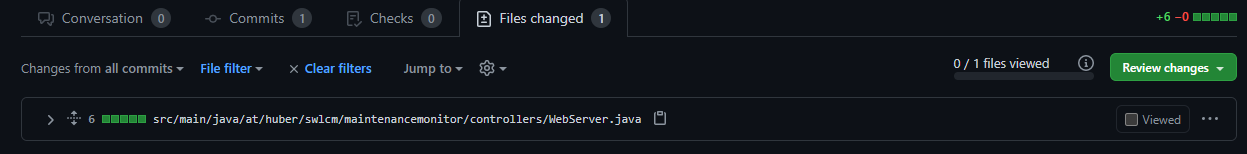


### Schritte

* Neues feature erstellen 
* Eine reset-Funktion mit der DELETE-Annotation hinzufügen (return-Wert wird beim Bearbeiten der Unit-Tests verändert)

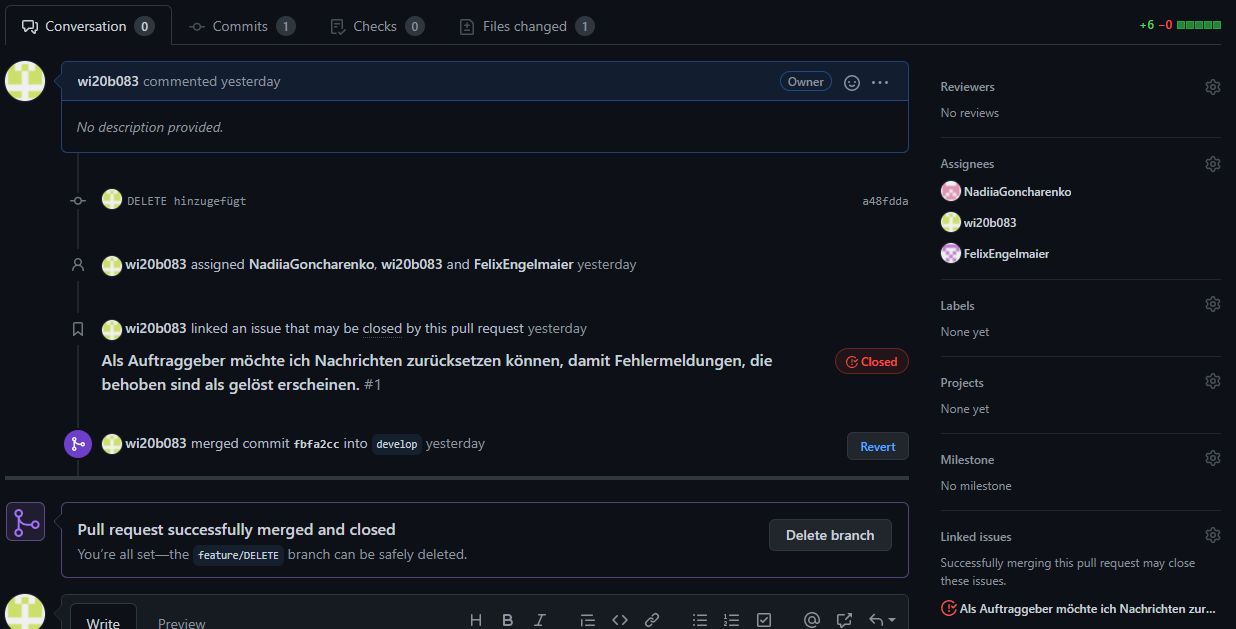


### Output



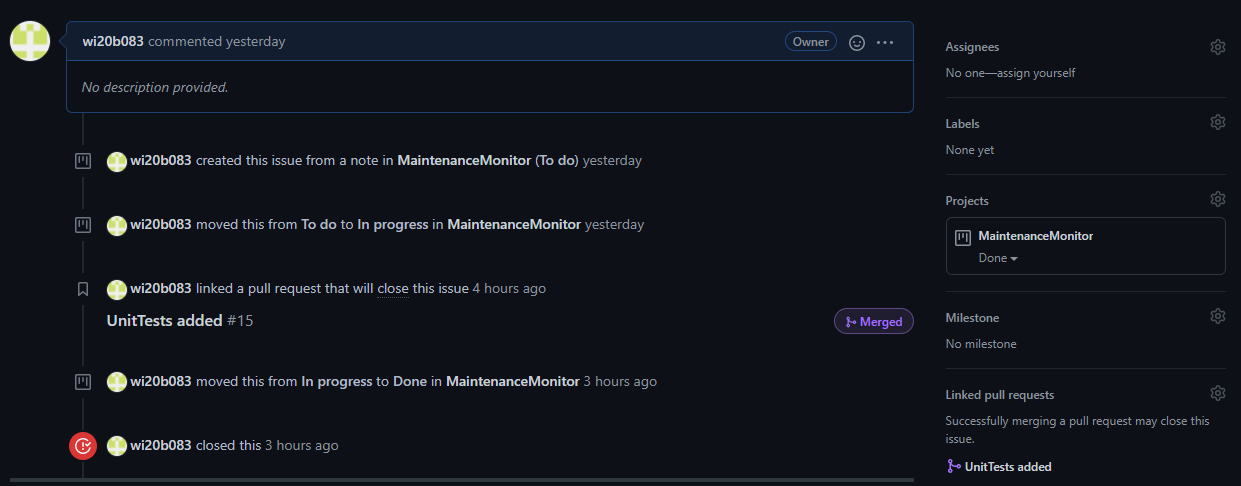
* DELETE-Funktion
* Status zurücksetzbar auf Default-message

### Linked pull requests



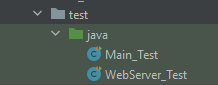
## Unit Tests

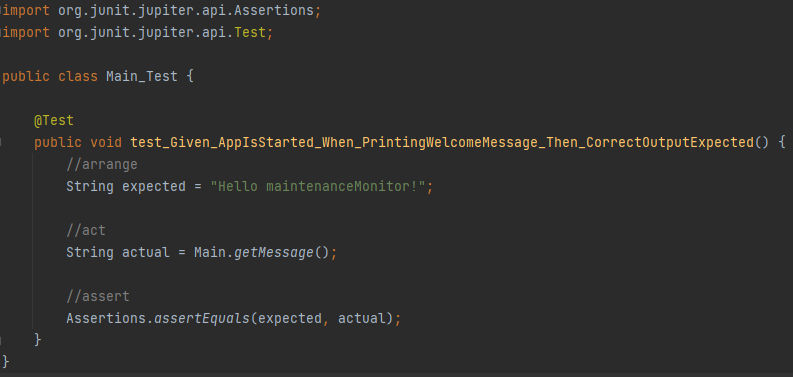
### Verlauf der Story in GitHub

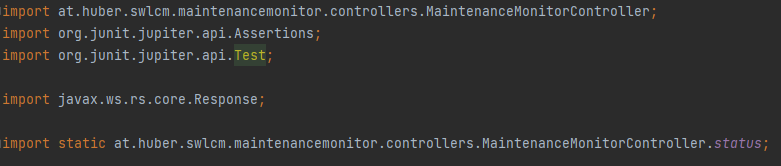


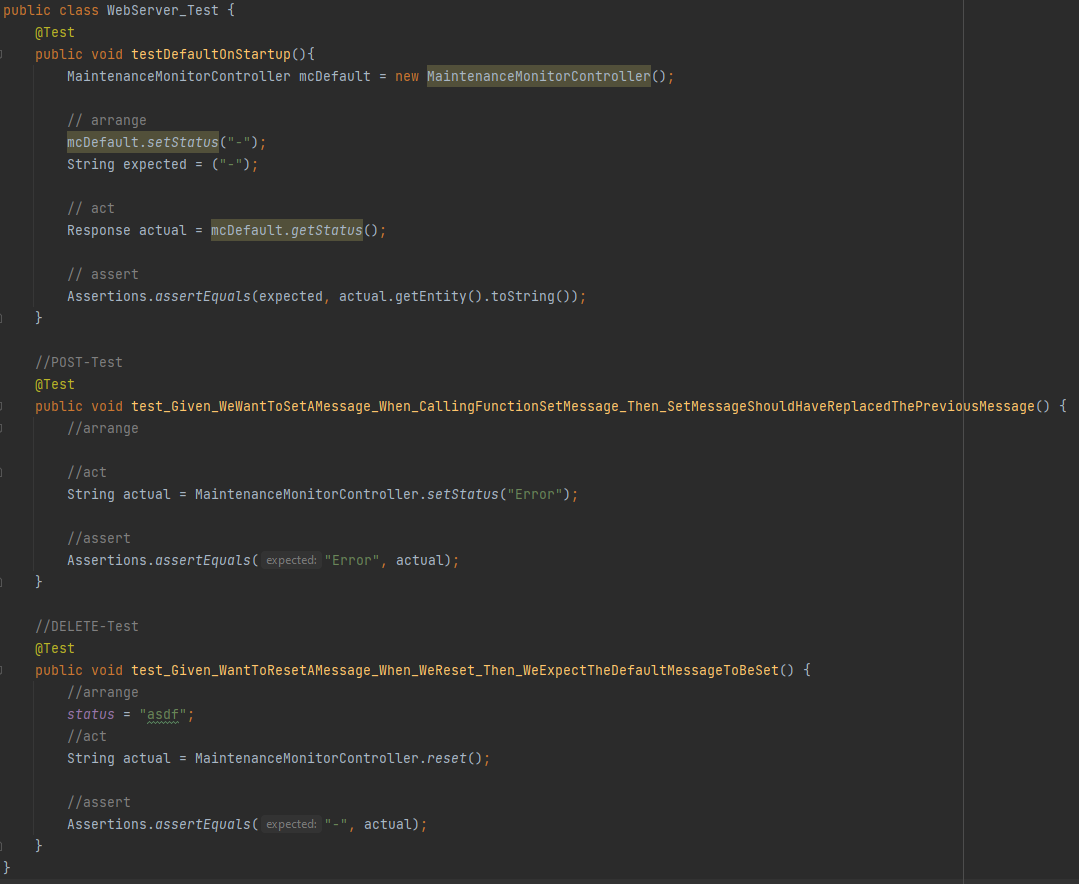
### Schritte

* Neues feature erstellen 
* (HTML Frontend aus Projekt gelöscht)
* Entsprechende Testklassen erstellen

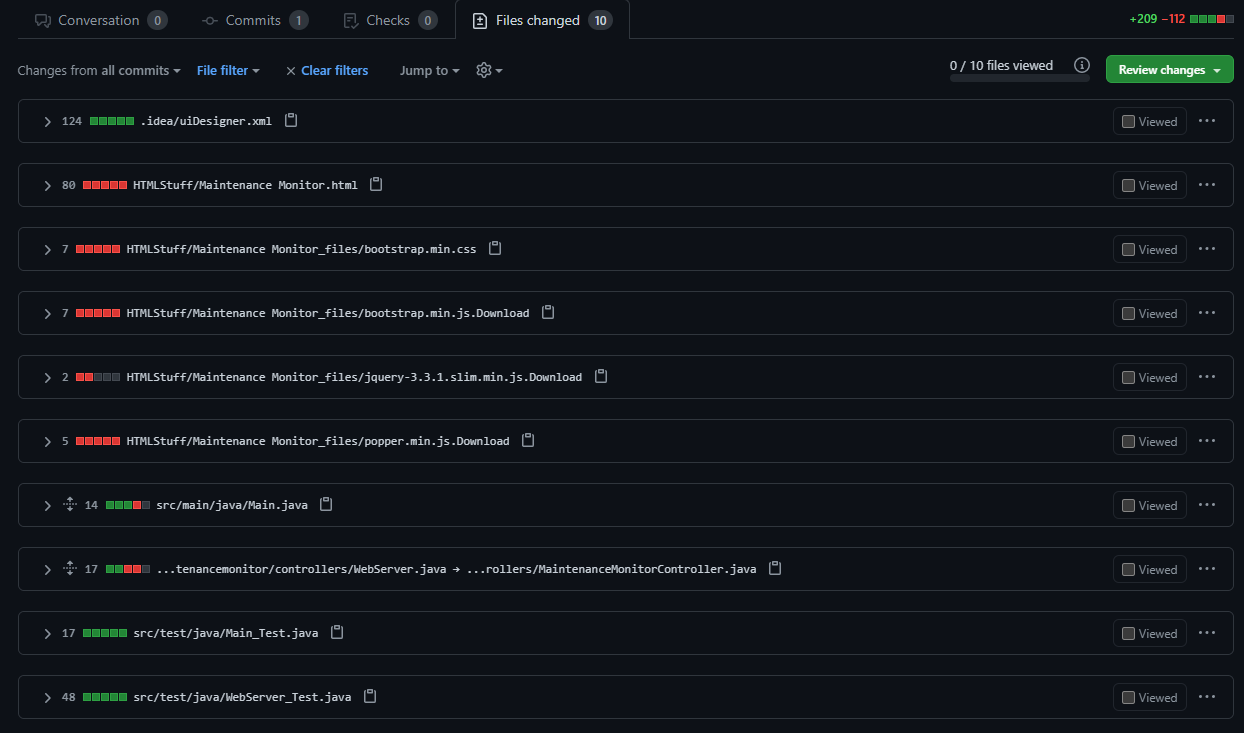


* Tests nach dem triple-a Schema schreiben



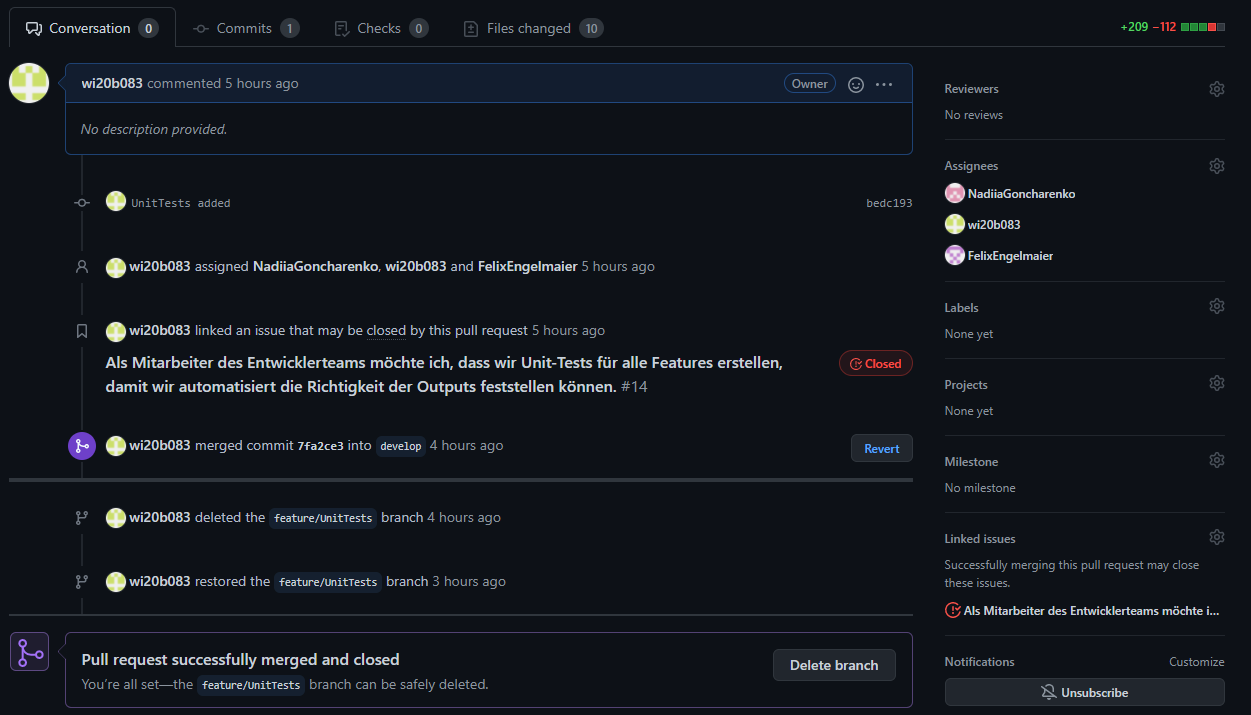


### Output



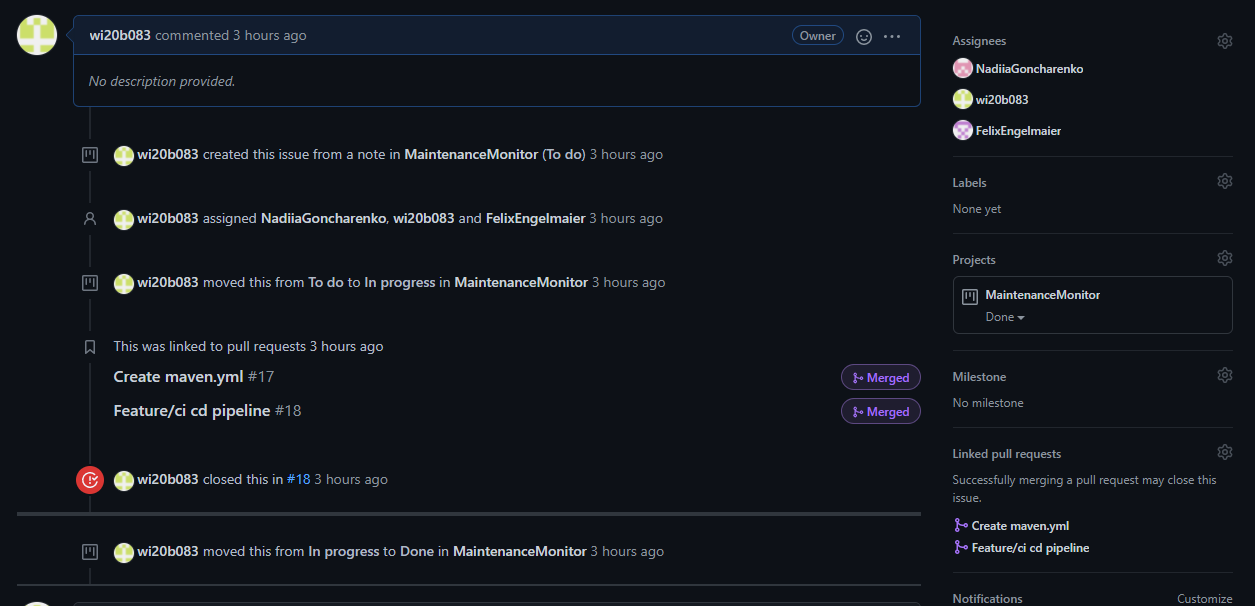
* Unit Tests für die Funktionen erstellt

### Linked pull requests



## CI/CD Pipeline

### Verlauf der Story in GitHub

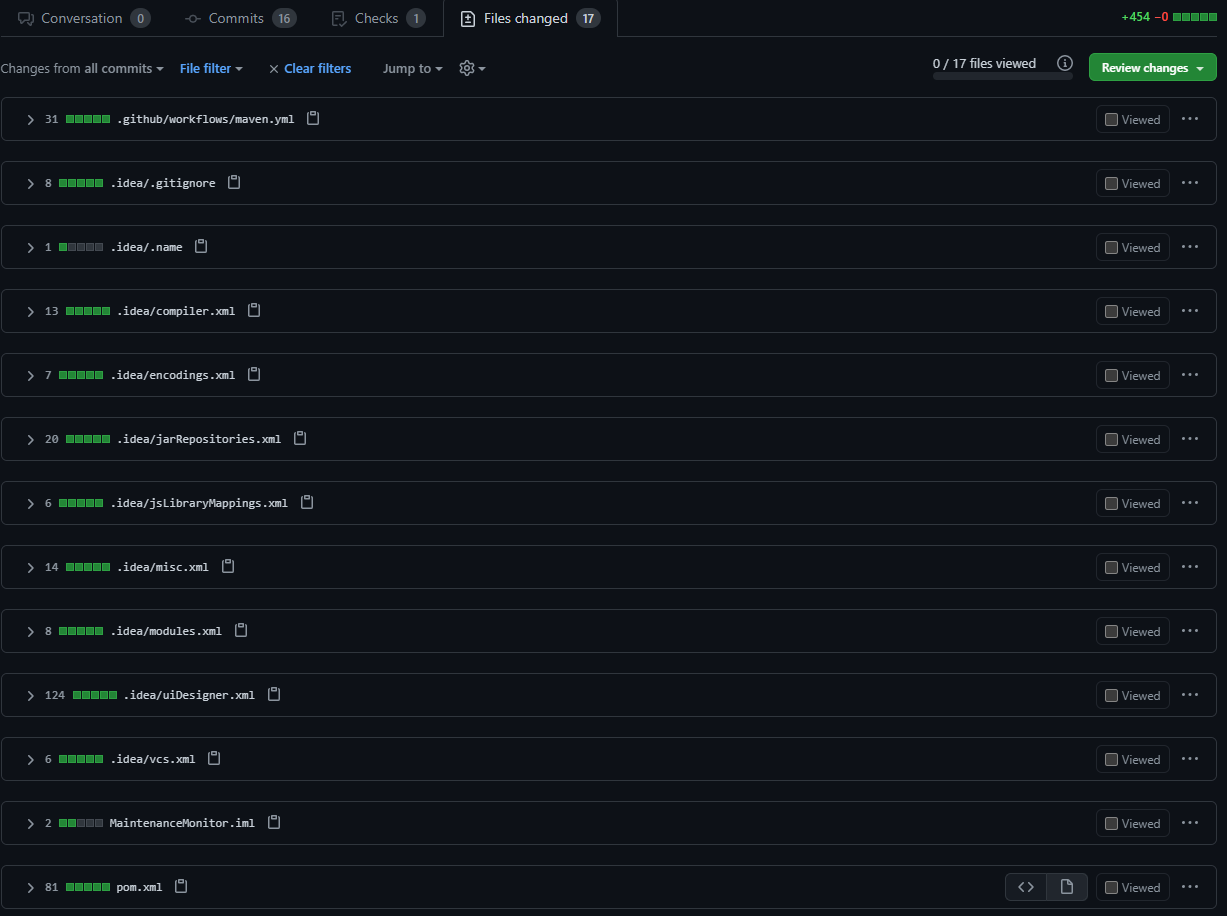


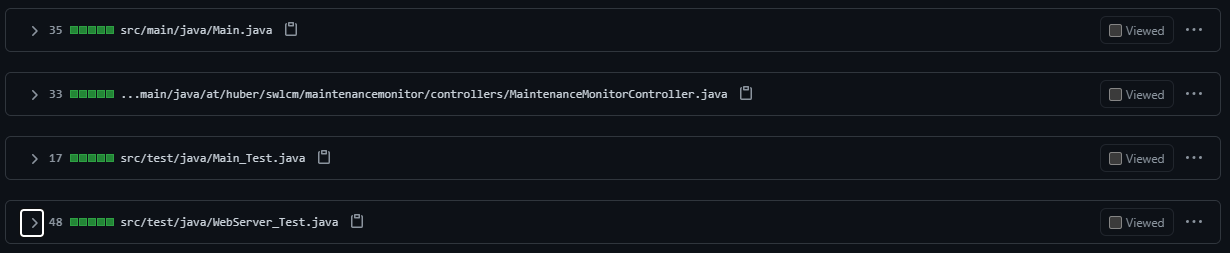
### Schritte

* Neues Feature erstellen 
* GitHub neue Action hinzufügen (Java with Maven)
* Maven.yml anpassen (kopiert von UptimeCalculator-Projekt), Java-Version angepasst

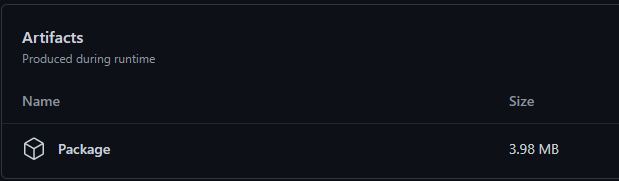
### Output



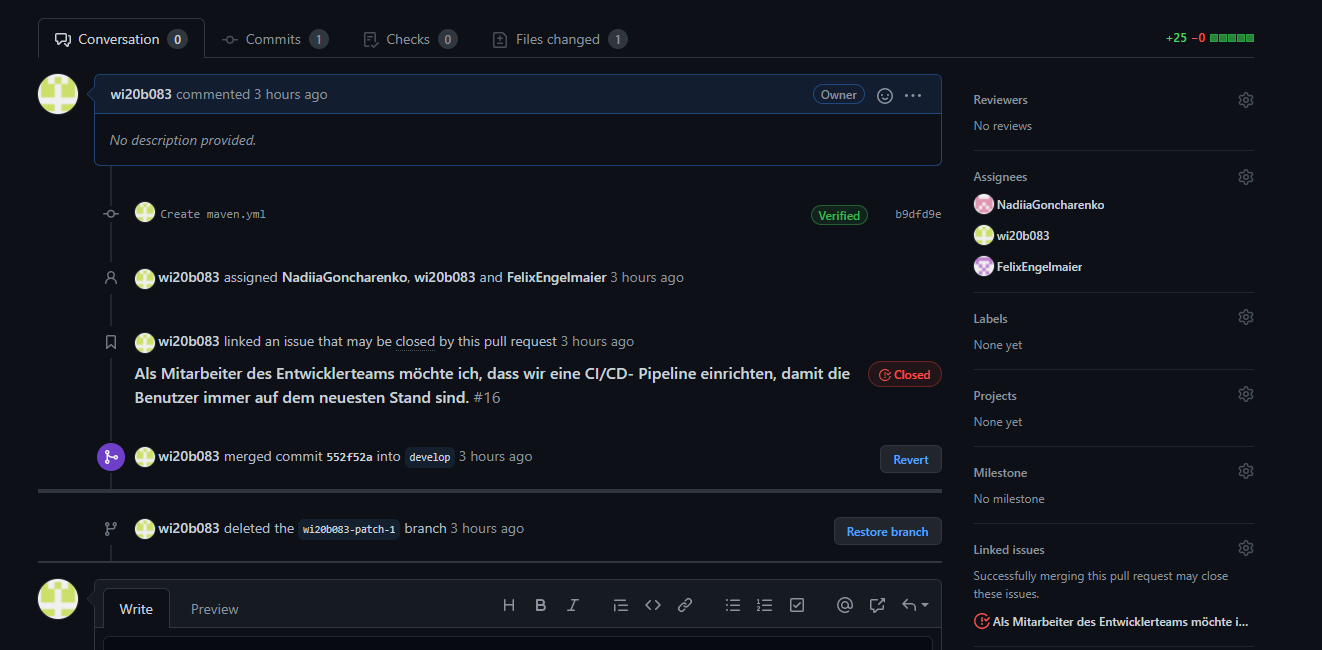


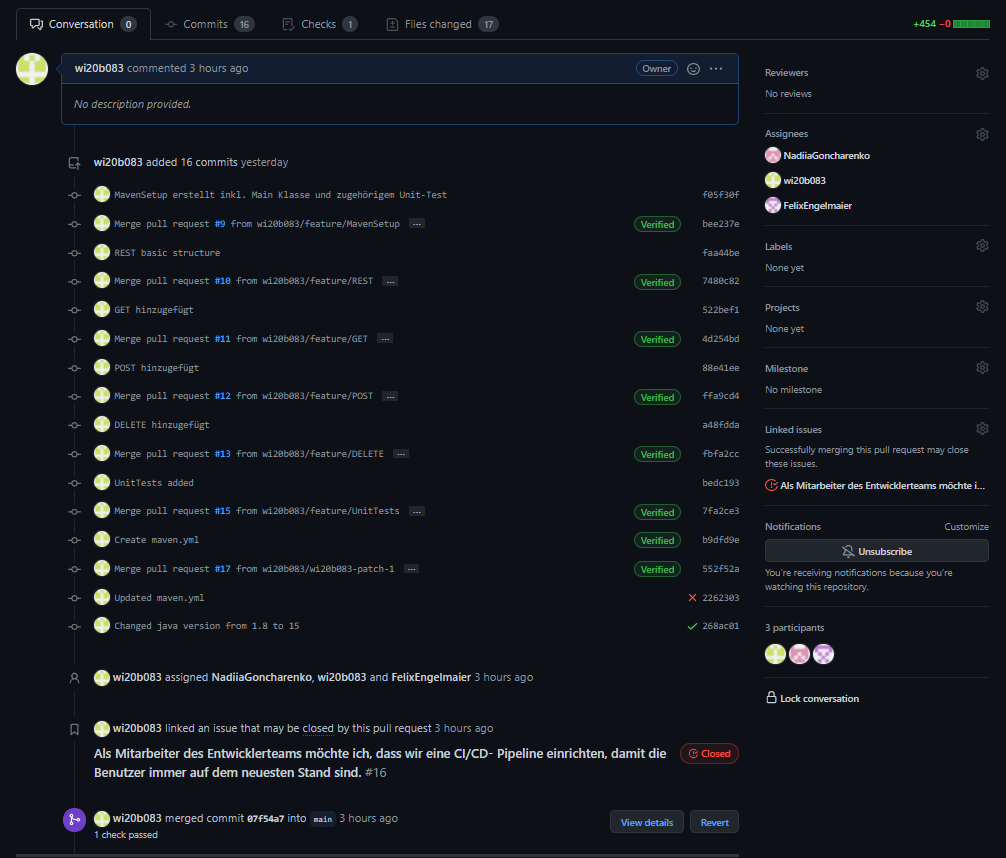


Anmerkung: aus Versehen nach main gemerged, daher die vielen Changes

* CI/CD Pipeline mit Action Artefakt

### Linked pull requests

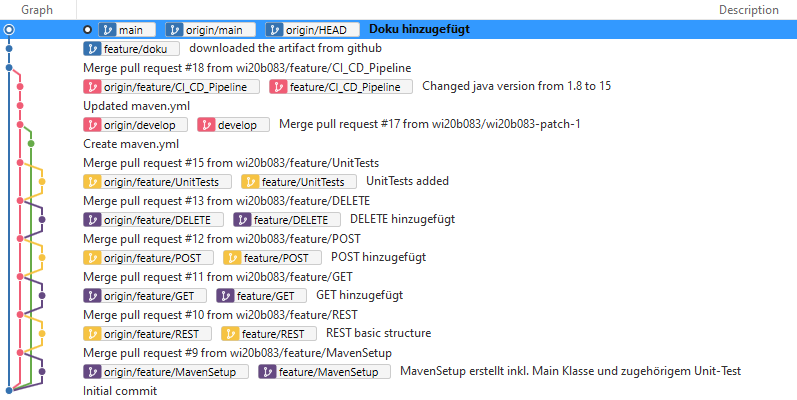




## Dokumentation

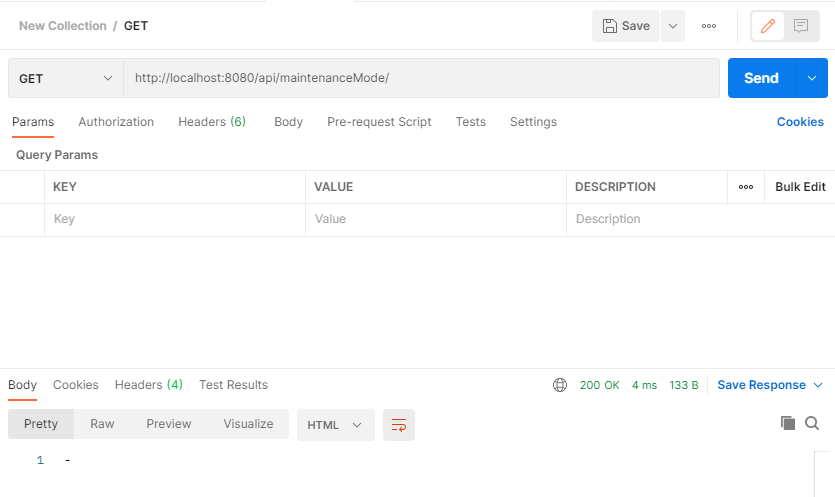
Siehe dieses Dokument.

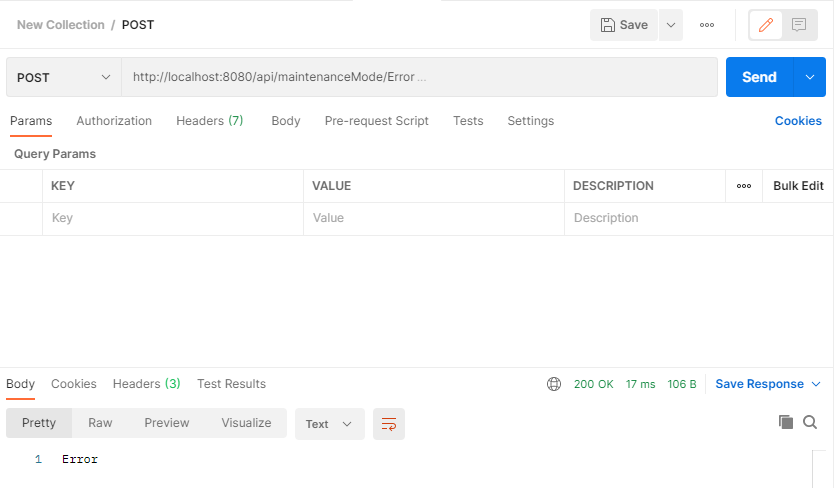
# GitFlow

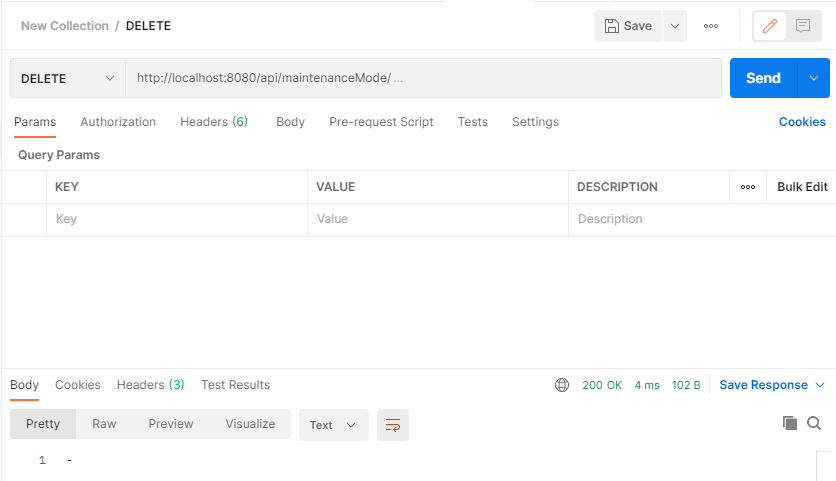


Anmerkung: „downloaded the artifact from github” wurde im falschen Feature gemacht.

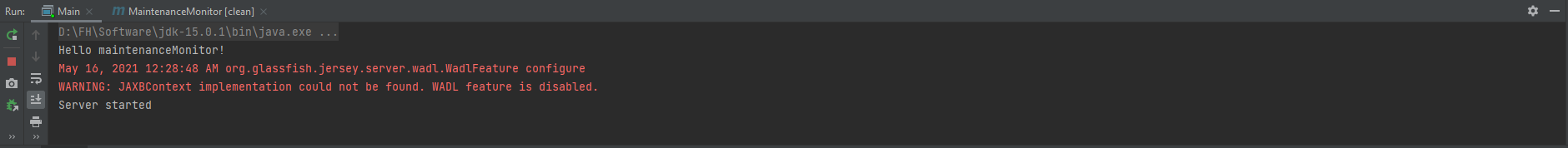
# Moodle Frontend und Postman

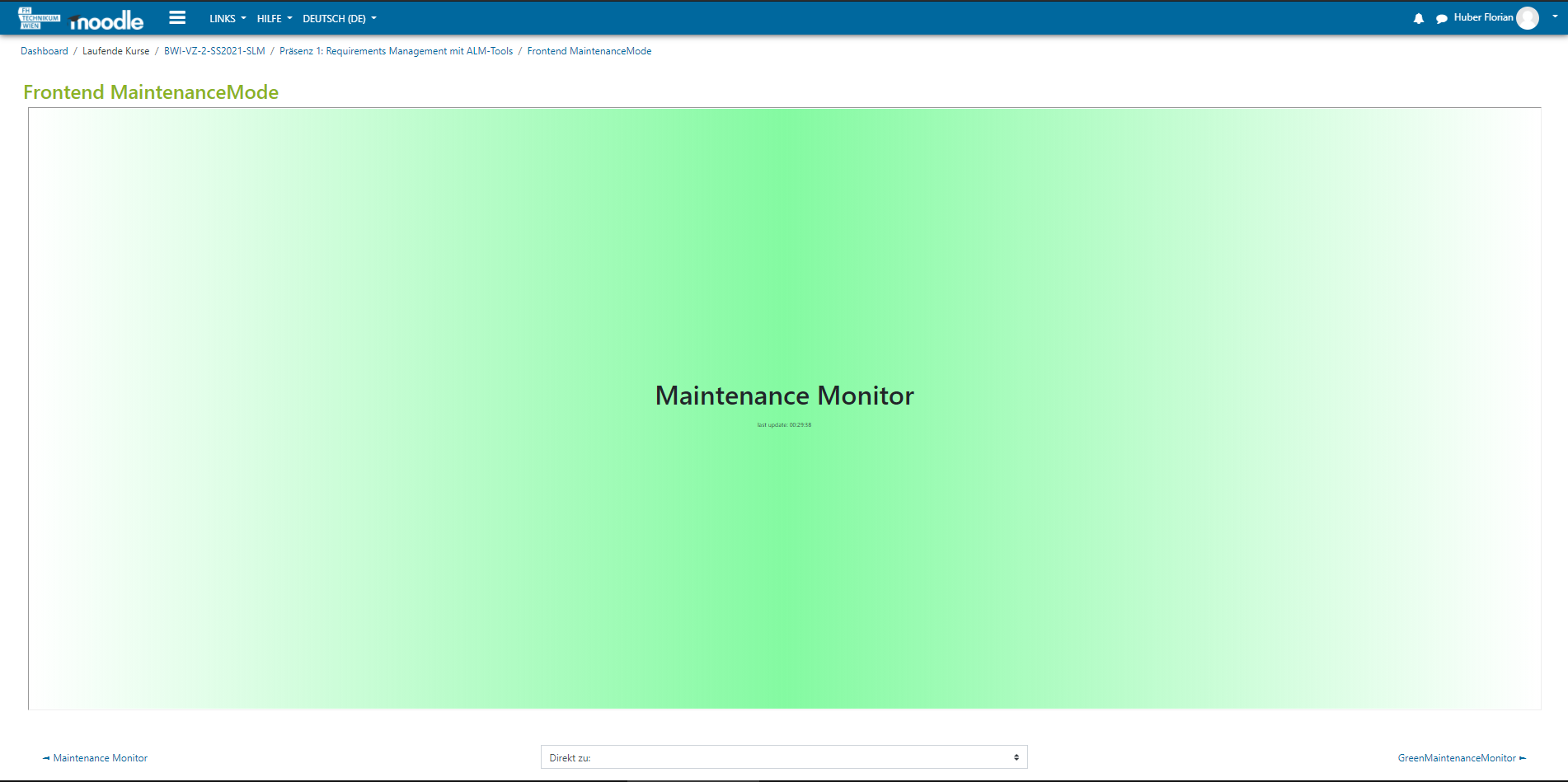
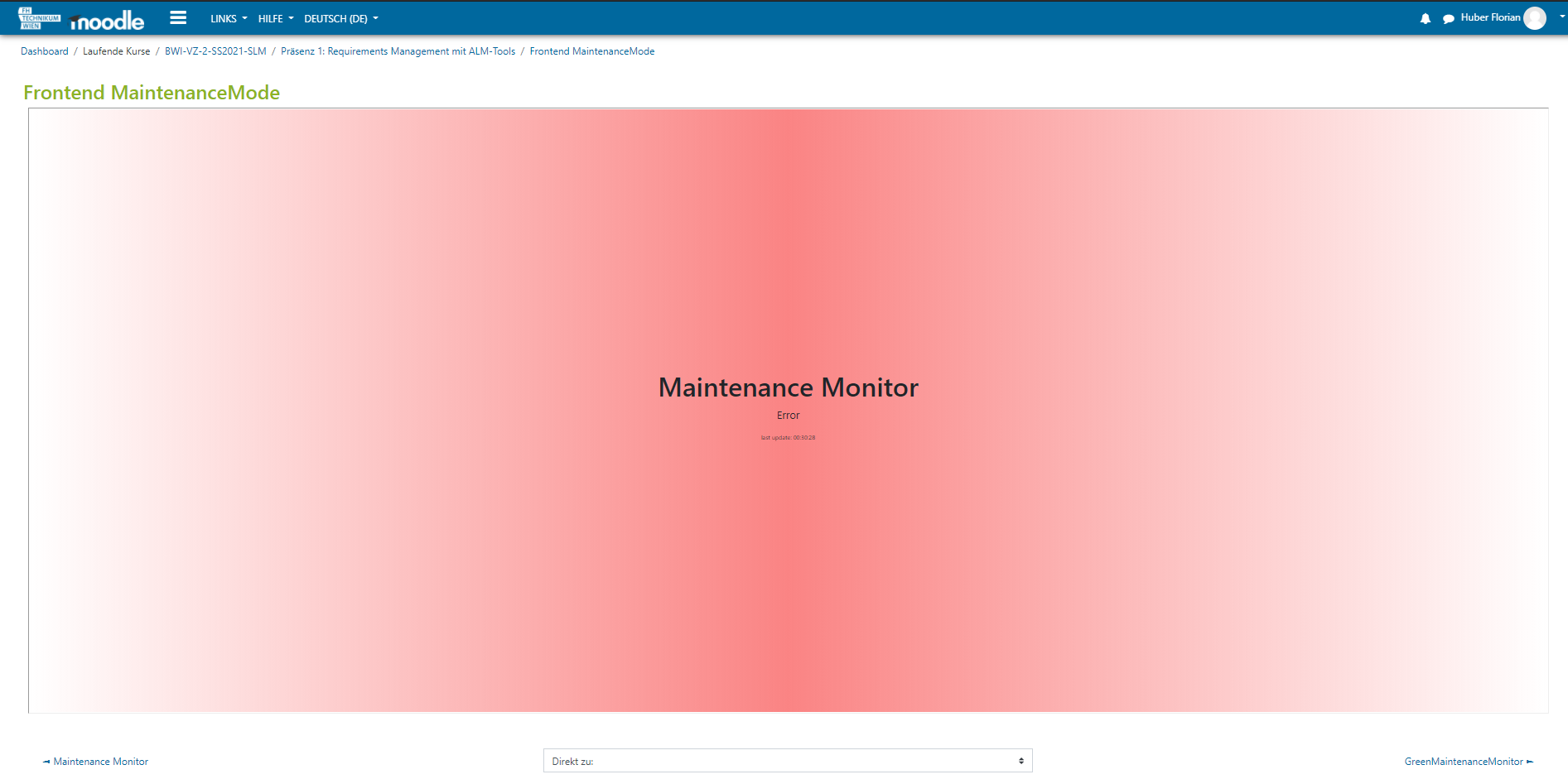
* In Postman die Requests vorbereitet





* Server gestartet



* Postman-GET gesendet
* Postman-POST gesendet
* Postman-Delete gesendet