LB 1 : Lektion 1 / Sprint 1

Überblick

Sozialform	Form	Abgabe	Gewicht
😄 **Einzelarbeit**	🗣️ Fachgespräch	Woche 4	1/3 vom LB1
😄 **Einzelarbeit**	🖉 Schriftliche Reflexion	Woche 4	1/3 vom LB1
👥 **Gruppenarbeit**	👷 Arbeitsvorgang	Woche 4	1/3 vom LB1
...			
	🎖️ **LB1: Sprintnote**	Woche 4	**1/3 der Modulnote**

Teilnote: Fachgespräch

Sozialform	Form	Abgabe	Gewicht
😄 **Einzelarbeit**	🗣️ Fachgespräch	Woche 4	1/3 vom LB1

Am **Ende des Sprints** findet ein Sprint-Review, in Form eines **Fachgespräches** statt. Jedes Gruppenmitglied wird zur **Lektion 1** sowie zum **Sprint 1** und den dabei relevanten Themen **8 Minuten** befragt.

> ⚠️ **ACHTUNG!**
> Die Themen sind auf den jeweiligen Sprint-Beschreibungen ersichtlich.

Bewertung

Was	Wie	Punkte
Inhalt	Ist korrekt und entspricht der Fragen der Diskussion.	4
Wissen	Brauchte keine Hilfestellungen.	2
Fachbegriffe	Werden richtig und an stelle von Umschreibungen verwendet	2

Was	Wie	Punkte
Kommunikation	Die Antworten erfolgen deutlich, zügig und klar	2
...		
Total		**10**

Teilnote: Schriftliche Reflexion

Sozialform	Form	Abgabe	Gewicht
😄 **Einzelarbeit**	🖉 Schriftliche Reflexion	Woche 4	1/3 vom LB1

Am **Ende des Sprints** wird in **Einzelarbeit** eine **schriftliche Reflexion** über den Verlauf des Sprints verfasst.

Bewertung

Thema	Punkte
Alle Leitfragen sind beantwortet *(0.5 pro Frage)*	6
Die Antworten sind Tiefgründig und **kritisch** hinterfragt.	4
...	
TOTAL	**10**

> ⚠ **ACHTUNG!**
>
> - Die Reflexionen müssen auf **alle** Leitfragen eingehen.
> - Die Antworten sollen auch **kritisch** hinterfragen!
>   - "Wir waren alle super und die Kommunikation war toll", ist keine wirkliche Reflexion!
> - Die Antworten sollen auf das **wieso** eingehen und nicht einfach behaupten. **Ohne Begründung, kein Punkt für die Frage.**

Wo wird die Reflexion geschrieben?

- Für jede Reflexion pro Person soll ein `GitHub Issue` erstellt werden.

- Die Reflexion soll als `markdown`-Datei unter folgendem Pfad `/docs/reflections/ihr-nachname/reflexion-sprint1.md` geschrieben werden.
- Für jede Reflexion pro Person wird ein Branch erstellt.
- Es wird ein Pull-Request erstellt und mir `@codingluke` als Reviewer zugewiesen.
- Nach dem mergen, soll der Branch wieder gelöscht werden.

⚠ VORSICHT

- 💥 Die Commit-Message soll folgendermassen heissen.
  - `docs(ihr-nachname): Reflexion Sprint 1`

BEISPIELDATEI: `/docs/reflections/hodel/reflexion-sprint1.md`

```
# Hodel - Reflexion Sprint 1

## Welche verschiedenen Themen beinhaltete der Sprint?

lorem ipsum...

## Was wurde in diesem Sprint erreicht?

lorem ipsum...

...
```

Commit-Message: `docs(hodel): Reflexion Sprint 1`

WIESO SO KOMPLIZIERT?

1. Ihr wendet direkt Prinzipien von Git und GitHub an
2. Ich habe dich Möglichkeit im Pull-Request einzelne Zeilen zu kommentieren
3. Ihr verwendet mit conventional-commits direkt das Autorelease Feature

Leitfragen

Inhalt (Auch Lektion 1!)

- [ ] Welche verschiedenen Themen behandelten wir bis jetzt?
- [ ] Waren Themen bereits bekannt? Wenn ja, woher?
- [ ] Was wurde in diesem Sprint erreicht?
- [ ] Was wurde in diesem Sprint nicht erreicht?

Gruppendynamik

- [ ] Wie war die Zusammenarbeit im Team?

- [ ] Auf was achtet Ihr bei der Kommunikation, damit diese konstruktiv ist?
- [ ] Was ist **uns** besonders gut gelungen, worauf sind alle richtig stolz?

Lernzuwachs

- [ ] Was kann ich nun mehr als vorher?
- [ ] Dieses Thema vom Sprint ist besonders relevant? Begründen Sie.
- [ ] Was ist **mir** besonders gut gelungen, worauf bin ich richtig stolz.

Lerneffekt

- [ ] Was sollte beibehalten werden?
- [ ] Was sollte geändert werden?

Teilnote: Arbeitsvorgang

Am **Ende des Sprints** wird der **Arbeitsvorgang der Gruppe** analysiert und bewertet.

Sozialform	Form	Abgabe	Gewicht
Gruppenarbeit	Arbeitsvorgang	Woche 4	1/3 vom LB1

ACHTUNG KOLLEKTIVNOTE!

- Bitte kontrolliert euch selbst, dass alle sich an die Vorgaben halten.
- Diese Note ist bewusst nicht individuell, damit die Gruppe sich zusammenrauft.

Bewertung

Thema	Punkte
Der Projektberschierb ist aussagekräftig	2
Der Projektberschierb befindet sich am gewünschten Ort	1
Epics verweisen auf Issues die ihn umsetzen	1
Issues/Epics sind klar und verständlich beschrieben	2
Issues beinhalten eine Definition of Done	1
Issues beschreiben immer nur eine Aufgabe	1

Thema	Punkte
Alle Reflexionen befinden sich am gewünschten Ort und wurden als Pull-Request eingereicht	1
Alle Commit-Messages wurden wie gewünscht geschrieben	1
...	
TOTAL	**10**

LB 2 : Sprint 2

Überblick

Sozialform	Form	Abgabe	Gewicht
😄 **Einzelarbeit**	🗣️ Fachgespräch	Woche 8	1/3 vom LB2
😄 **Einzelarbeit**	🖍️ Schriftliche Reflexion	Woche 8	1/3 vom LB2
👥 **Gruppenarbeit**	👷 Arbeitsvorgang	Woche 8	1/3 vom LB2
...			
	🎖️ **LB2: Sprintnote**	Woche 8	**1/3 der Modulnote**

Teilnote: Fachgespräch

Sozialform	Form	Abgabe	Gewicht
😄 **Einzelarbeit**	🗣️ Fachgespräch	Woche 8	1/3 vom LB2

Am **Ende des Sprints** findet ein Sprint-Review, in Form eines **Fachgespräches** statt. Jedes Gruppenmitglied wird zum **Sprint 2** und den dabei relevanten Themen **8 Minuten** befragt.

> ⚠️ **ACHTUNG!**
>
> Die Themen sind auf den jeweiligen Sprint-Beschreibungen ersichtlich.

Bewertung

Was	Wie	Punkte
Inhalt	Ist korrekt und entspricht der Fragen der Diskussion.	4
Wissen	Brauchte keine Hilfestellungen.	2
Fachbegriffe	Werden richtig und an stelle von Umschreibungen verwendet	2

Was	Wie	Punkte
Kommunikation	Die Antworten erfolgen deutlich, zügig und klar	2
...		
Total		**10**

Teilnote: Schriftliche Reflexion

Sozialform	Form	Abgabe	Gewicht
😄 **Einzelarbeit**	🖉 Schriftliche Reflexion	Woche 8	1/3 vom LB2

Am **Ende des Sprints** wird in **Einzelarbeit** eine **schriftliche Reflexion** über den Verlauf des Sprint 2 verfasst.

Bewertung

Thema	Punkte
Alle Leitfragen sind beantwortet *(0.5 pro Frage)*	6
Die Antworten sind Tiefgründig und **kritisch** hinterfragt.	4
...	
TOTAL	**10**

> ⚠ **ACHTUNG!**
>
> - Die Reflexionen müssen auf **alle** Leitfragen eingehen.
> - Die Antworten sollen auch **kritisch** hinterfragen!
>   - "Wir waren alle super und die Kommunikation war toll", ist keine wirkliche Reflexion!
> - Die Antworten sollen auf das **wieso** eingehen und nicht einfach behaupten. **Ohne Begründung, kein Punkt für die Frage.**

Wo wird die Reflexion geschrieben?

- Für jede Reflexion pro Person soll ein `GitHub Issue` erstellt werden.

- Die Reflexion soll als `markdown`-Datei unter folgendem Pfad `/docs/reflections/ihr-nachname/reflexion-sprint2.md` geschrieben werden.
- Für jede Reflexion pro Person wird ein Branch erstellt.
- Es wird ein Pull-Request erstellt und mir `@codingluke` als Reviewer zugewiesen.
- Nach dem mergen, soll der Branch wieder gelöscht werden.

⚠ VORSICHT

- 💥 Die Commit-Message soll folgendermassen heissen.
  - `docs(ihr-nachname): Reflexion Sprint 2`

BEISPIELDATEI: `/docs/reflections/hodel/reflexion-sprint2.md`

```
# Hodel - Reflexion Sprint 2

## Welche verschiedenen Themen beinhaltete der Sprint?

lorem ipsum...

## Sind diese Themen zum Zeitpunkt im Projekt gut gewählt, wenn ja, wieso?

lorem ipsum...

...
```

Commit-Message: `docs(hodel): Reflexion Sprint 2`

WIESO SO KOMPLIZIERT?

1. Ihr wendet direkt Prinzipien von Git und GitHub an
2. Ich habe dich Möglichkeit im Pull-Request einzelne Zeilen zu kommentieren
3. Ihr verwendet mit conventional-commits direkt das Autorelease Feature

Leitfragen

NICHT GANZ GLEICH ZUM LB1!

Inhalt

- [ ] Welche verschiedenen Themen beinhaltete der Sprint?
- [ ] Sind diese Themen zum Zeitpunkt im Projekt gut gewählt, wenn ja, wieso?
- [ ] Was wurde in diesem Sprint erreicht?
- [ ] Was wurde in diesem Sprint nicht erreicht?

Gruppendynamik

- [ ] Wie war die Zusammenarbeit im Team?
- [ ] Auf was achtet Ihr bei der Kommunikation, damit diese Konstruktiv ist?
- [ ] Was ist **uns** besonders gut gelungen, worauf sind alle richtig stolz?

Lernzuwachs

- [ ] Was kann ich nun mehr als vorher?
- [ ] Dieses Thema vom Sprint ist besonders relevant? Begründen Sie.
- [ ] Was ist **mir** besonders gut gelungen, worauf bin ich richtig stolz.

Lerneffekt

- [ ] Was sollte beibehalten werden?
- [ ] Was sollte geändert werden?

Teilnote: Arbeitsvorgang

Am **Ende des Sprint 2** wird der **Arbeitsvorgang der Gruppe** analysiert und bewertet.

Sozialform	Form	Abgabe	Gewicht
Gruppenarbeit	Arbeitsvorgang	Woche 4	1/3 vom LB2

ACHTUNG KOLLEKTIVNOTE!

- Bitte kontrolliert euch selbst, dass alle sich an die Vorgaben halten.
- Diese Note ist bewusst nicht individuell, damit die Gruppe sich zusammenrauft.

Bewertung

Thema	Punkte
Issues sind **klar und verständlich** beschrieben	1
Issues beinhalten eine **Definion of Done**	1
Geschlossene Issues sind an einen **Pull Request geknüpft**	1
Geschlossene Issues erfüllen die **Definition of Done**	1

Thema	Punkte
Pull Requests besitzen eine **Berschreibung** (oder eine Referenz in der Beschreibung, auf den Issue den ihn beschreibt).	1
Pull Requests sind **konstruktiv** kommentiert (auch Code!)	2
Pull Requests beziehen sich **genau auf einen** Issue	1
Commits sind anhand **conventional-commits** geschrieben	1
Commits beschreiben den **Inhalt** kurz (und nicht die Datei die geändert wurde)	1
...	
TOTAL	**10**

LB 3 : Modul-Abschlussprüfung

Überblick

Sozialform	Form	Abgabe	Gewicht
😄 **Einzelarbeit**	📝 Schriftliche Prüfung	Woche 9	**1/3 der Modulnote**

Inhalt

- Alle Themen vom Unterricht
- Alle Informationen sind auf der Modulwebseite vorhanden

Art

- Multiple-Choice
- Schriftliche Antworten
- Grafische Antworten

GitHub Account erstellen

Das Modul baut auf GitHub auf. Daher ist es unerlässlich, dass jeder einen GitHub Account besitzt.

Aufgaben

- [x] GitHub - Signup öffnen
- [x] Einen Account mit der `ihr-name@bbzbl-it.ch` Email-Adresse erstellen.

> **HINWEIS**
>
> Ihr dürft auch euren persönlichen Account verwenden, diese muss der Lehrperson schriftlich kommuniziert werden!

SSH Konfigurieren

> **INFO**
>
> Das muss **nicht in der ersten Stunde** gemacht werden. Um Zeit zu sparen ist dieser Schritt erst nötig, wenn ein Repository lokal geklont wird.

Um lokal nach GitHub Änderungen "pushen" zu können (`git push`), muss lokal einen SSH key erstellt werden und dieser auf GitHub registiert werden.

Hier gibt es eine Anleitung von GitHub wie dies genau funktioniert.

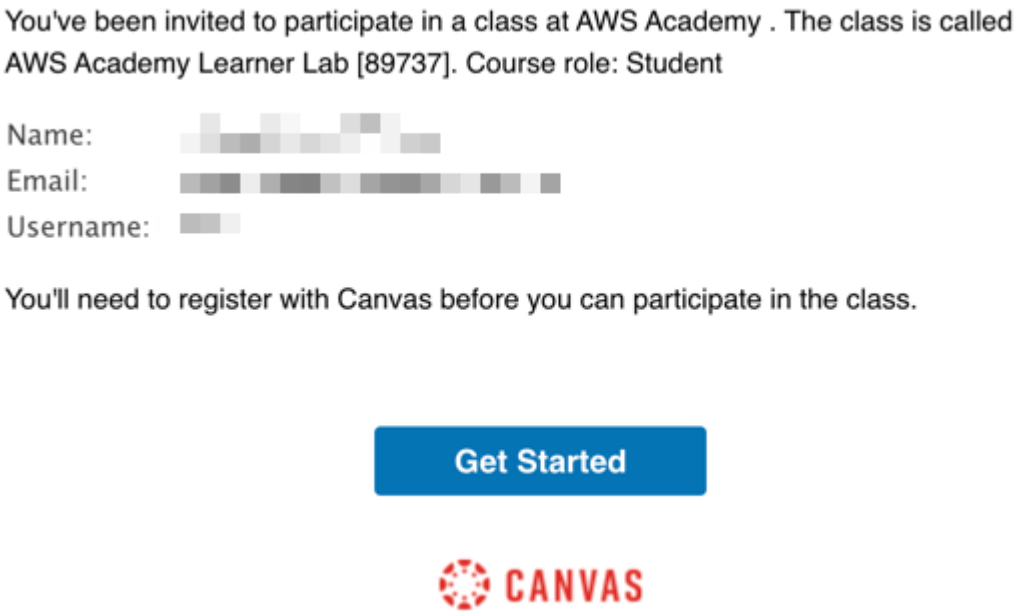
- GitHub SSH Dokumentation

AWS Academy Lerner Lab

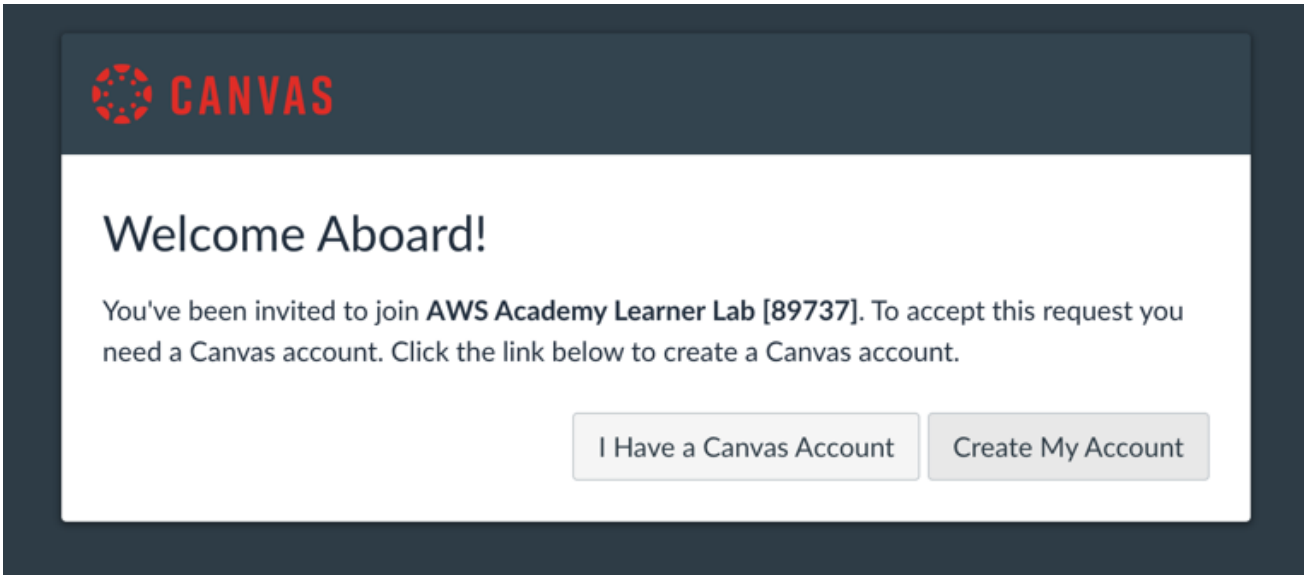
In dem Modul werden Web-Applikationen über eine Ci/CD-Pipeline direkt nach Amazon AWS ausgeliefert (deployed). Dafür haben alle eine Einladung per Email erhalten. Bitte folgt dieser Anleitung um das AWS-Lerner-Lab zu starten.

Aufgabe

1. Auf "Get Started" klicken

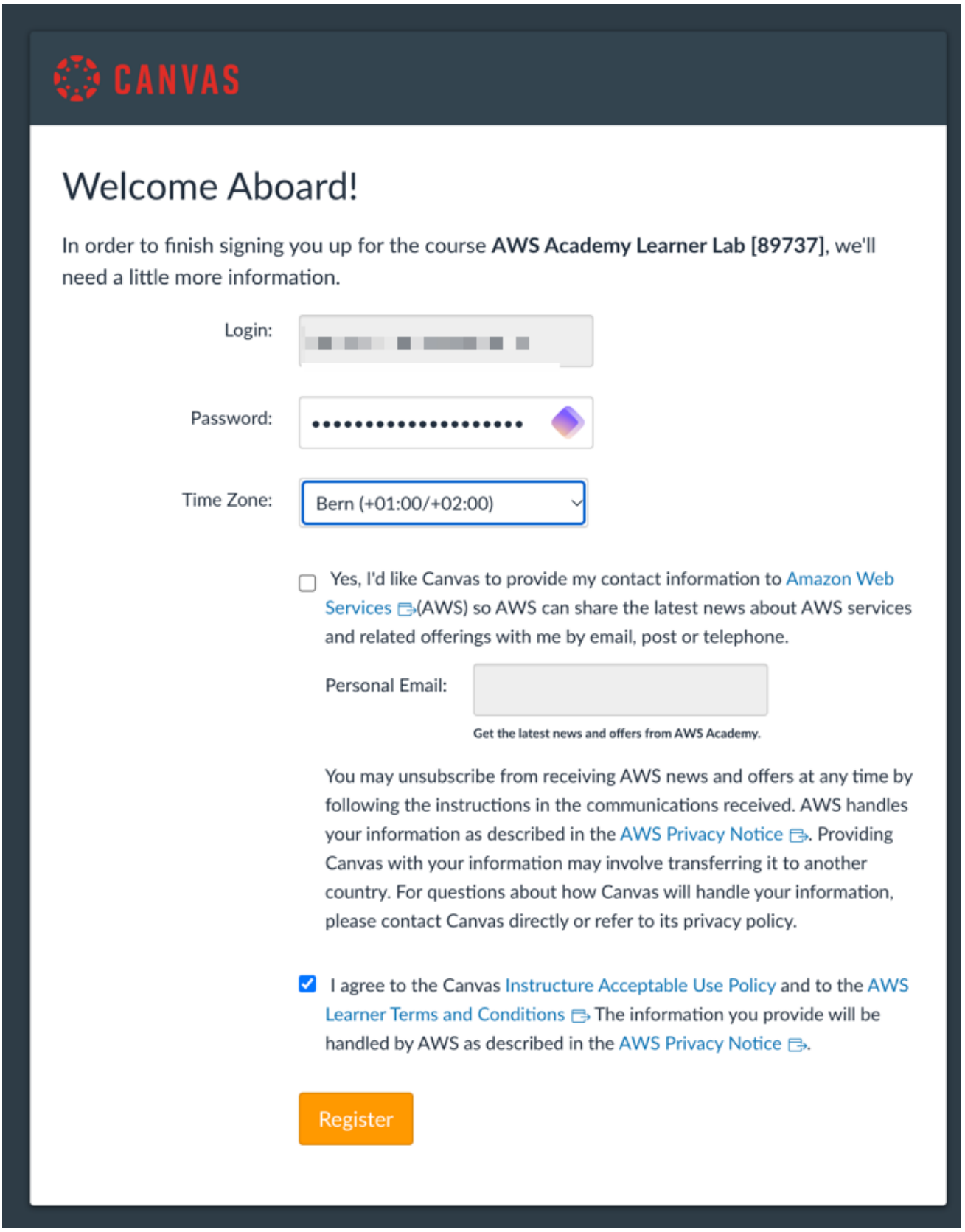


2. Auf "Create My Account" klicken



3. Formular ausfüllen

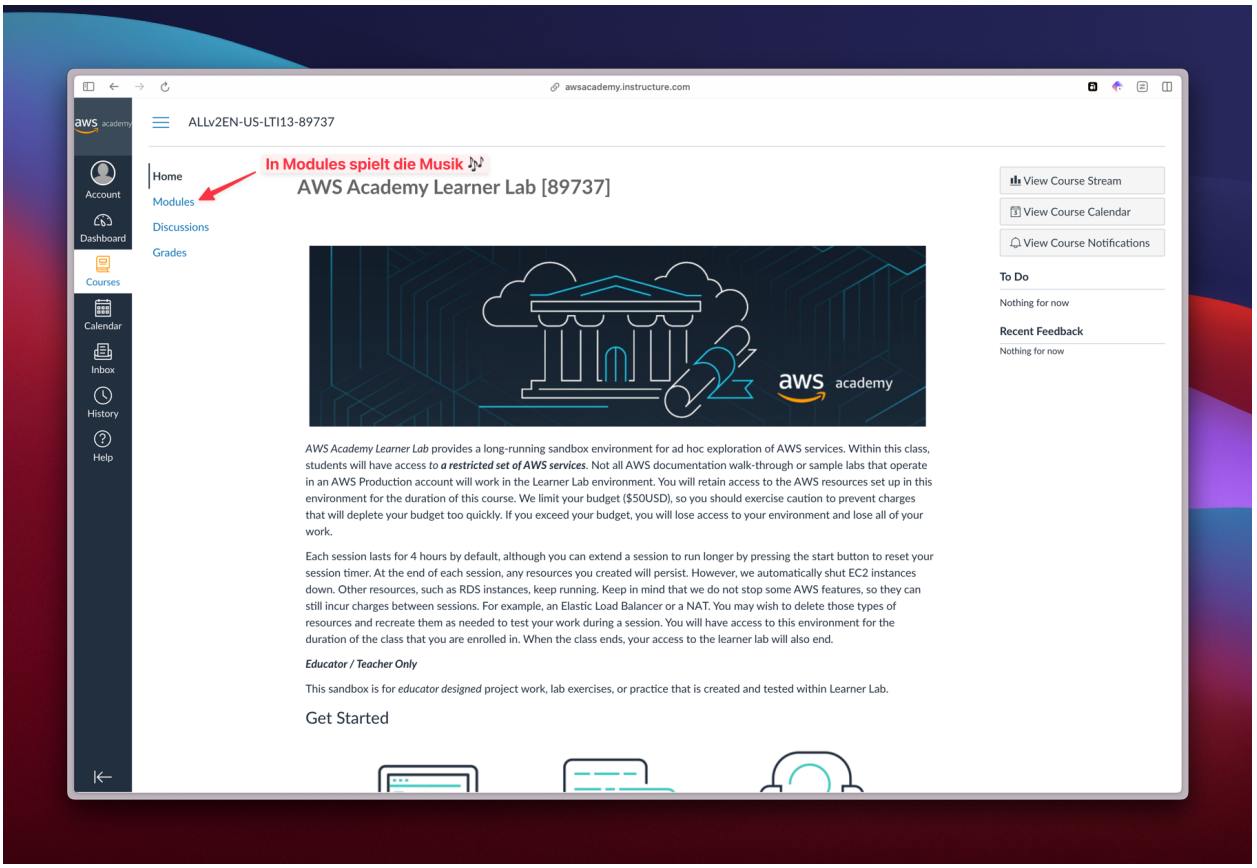
- Ein Passwort wählen
- Die **Zeitzone auf Bern** stellen
- AGB akzeptieren
- Mit **"Register"** abschicken



4. Academy - Home

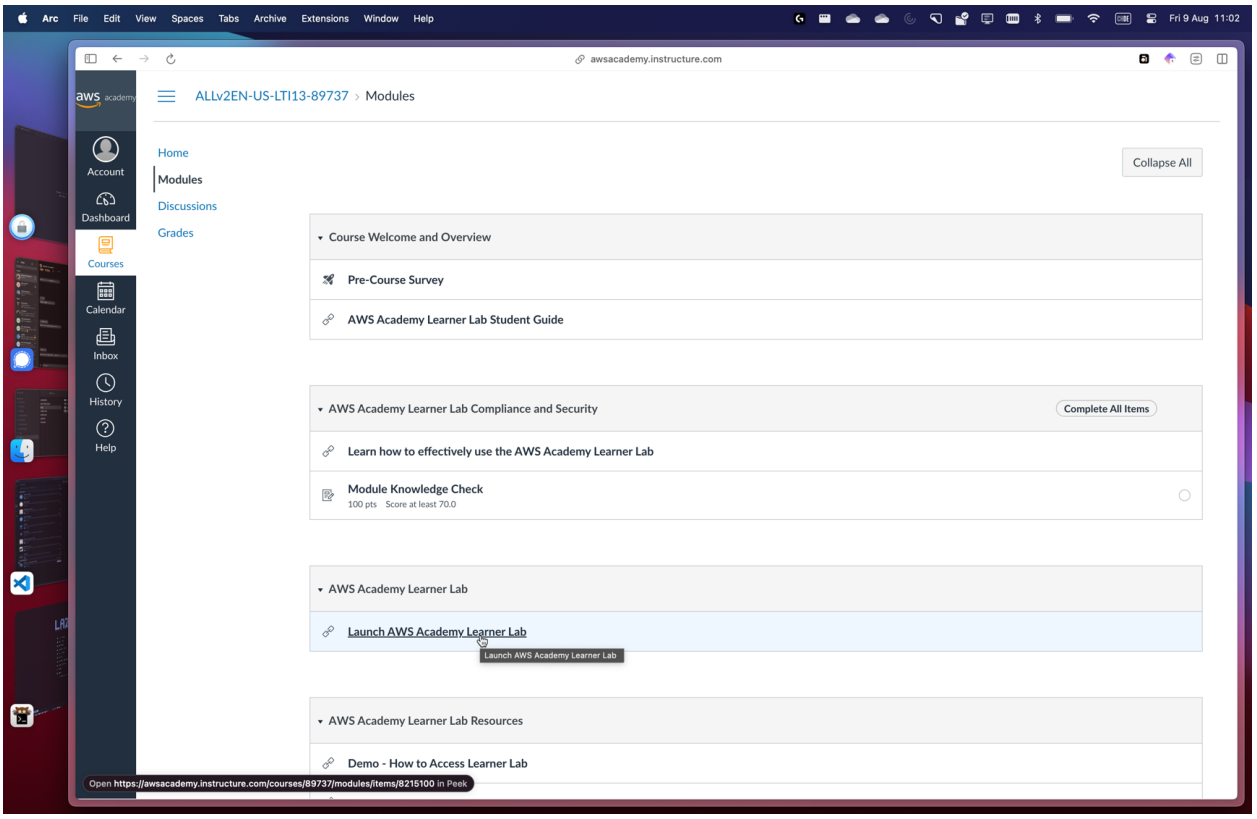
Nach dem Registrieren solltet Ihr auf folgende Übersichtsseite gelangen. Diese "Home"-Seite ist nicht relevant.

- Ihr könnt direkt zur Seite **"Modules"** navigieren.



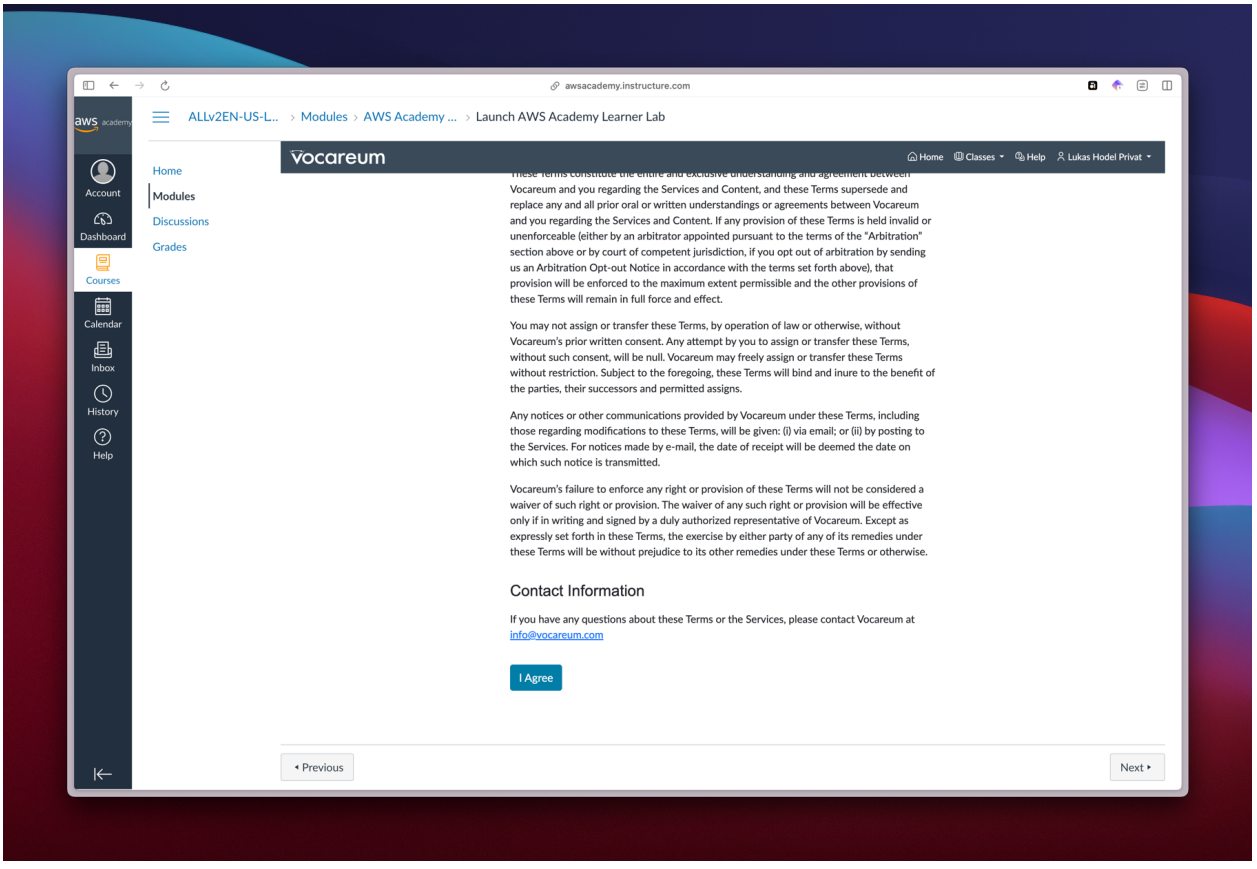
5. Academy - Modules

- Unter "Modules" interessiert nur **"Launch AWS Academy Learner Lab"**
- Öffnet **"Launch AWS Academy Learner Lab"**

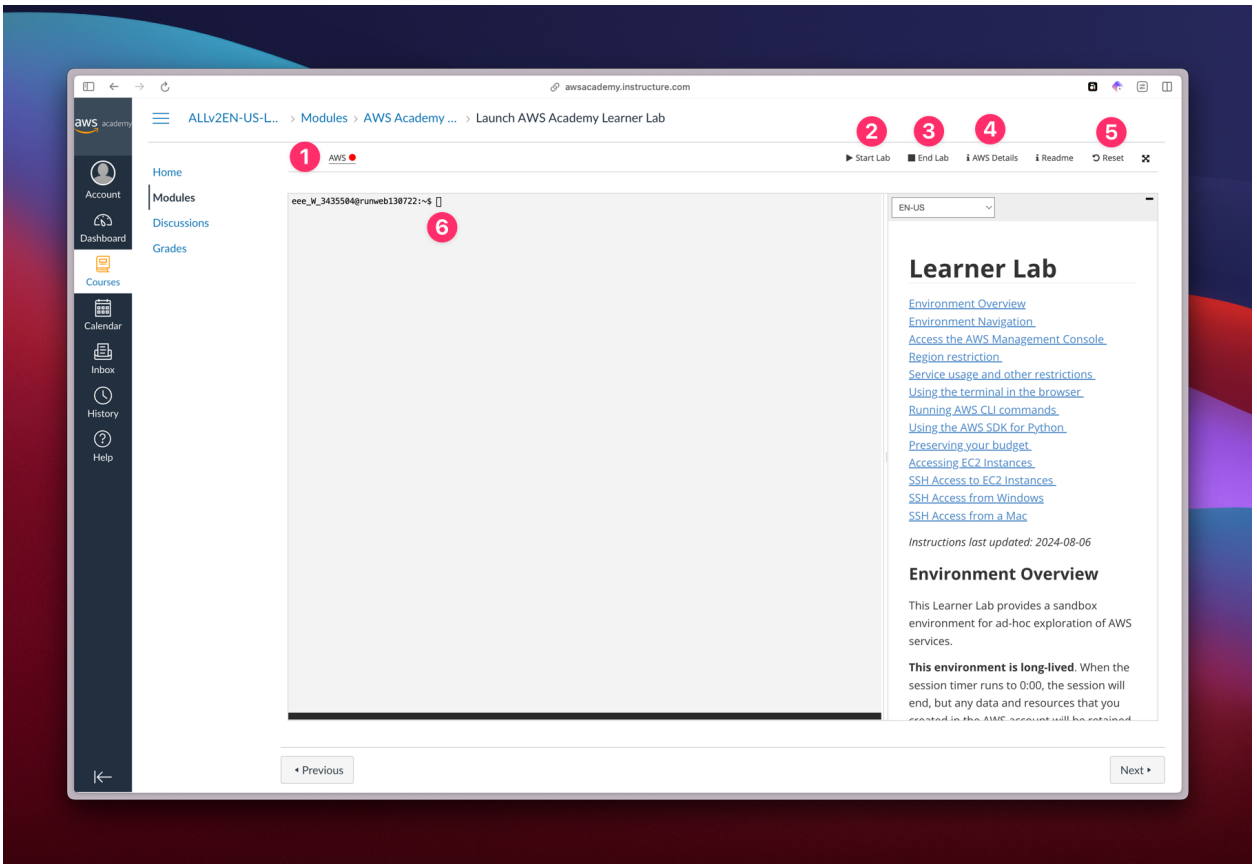


6. Launch Academy Learner Lab

- Als Erstes muss die AGB mit **"I Agree"** akzeptiert werden.



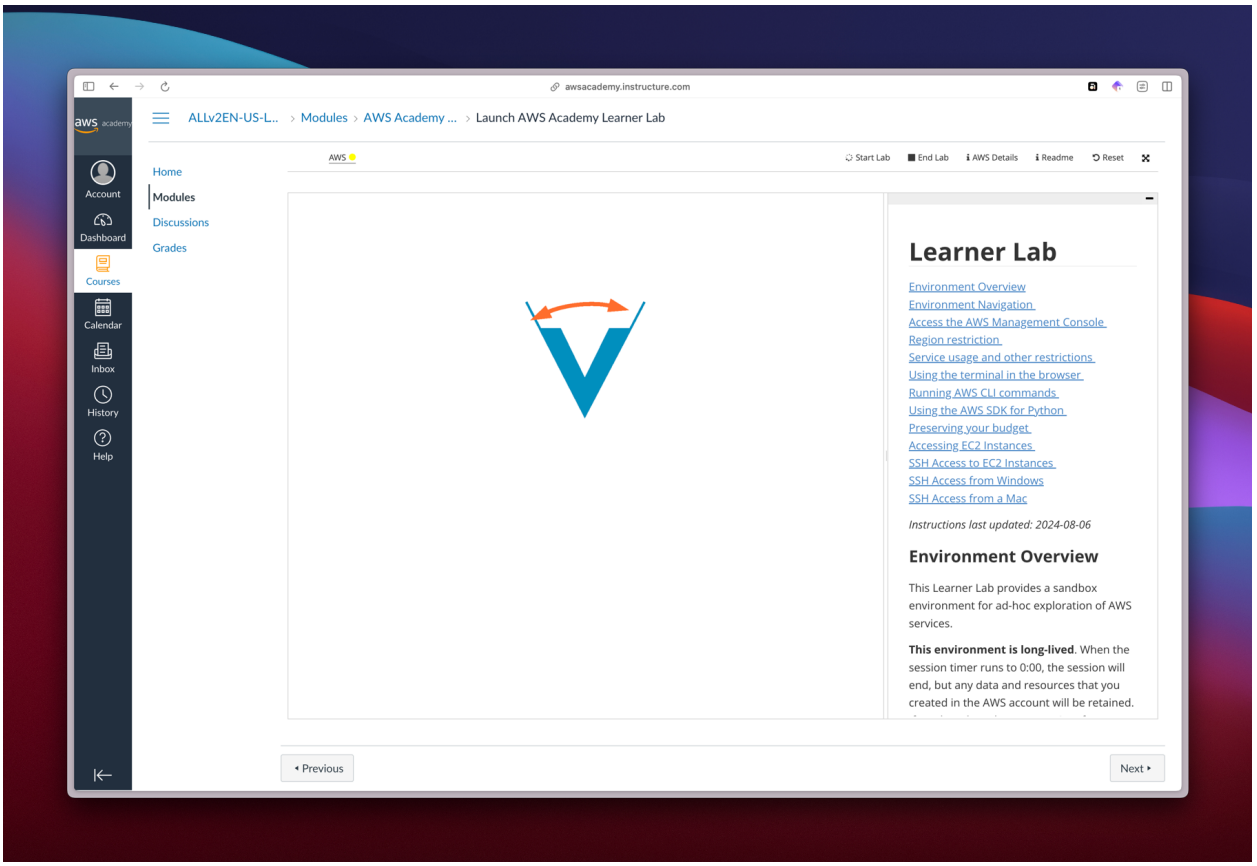
- Jetzt sollte folgende **Übersicht** zu sehen sein.



1. Link zur **AWS Management Console**
   - wenn **rot** ist die Umgebung gestoppt wenn **gelb** ist die Umgebung am starten
   - wenn **grün** ist die Umgebung gestartet
2. **Start Lab**: Startet die Umgebung
3. **End Lab**: Stoppt die Umgebung, (macht nix kaputt!)
4. **AWS Details**: Zeigt die Credentials und Account Informationen an
5. **Reset**: Setzt die Umgebung zurück (**löscht alles!**, braucht lange)
6. **Console**: AWS Academy Learner Lab console. (wird von uns nicht gross verwendet)

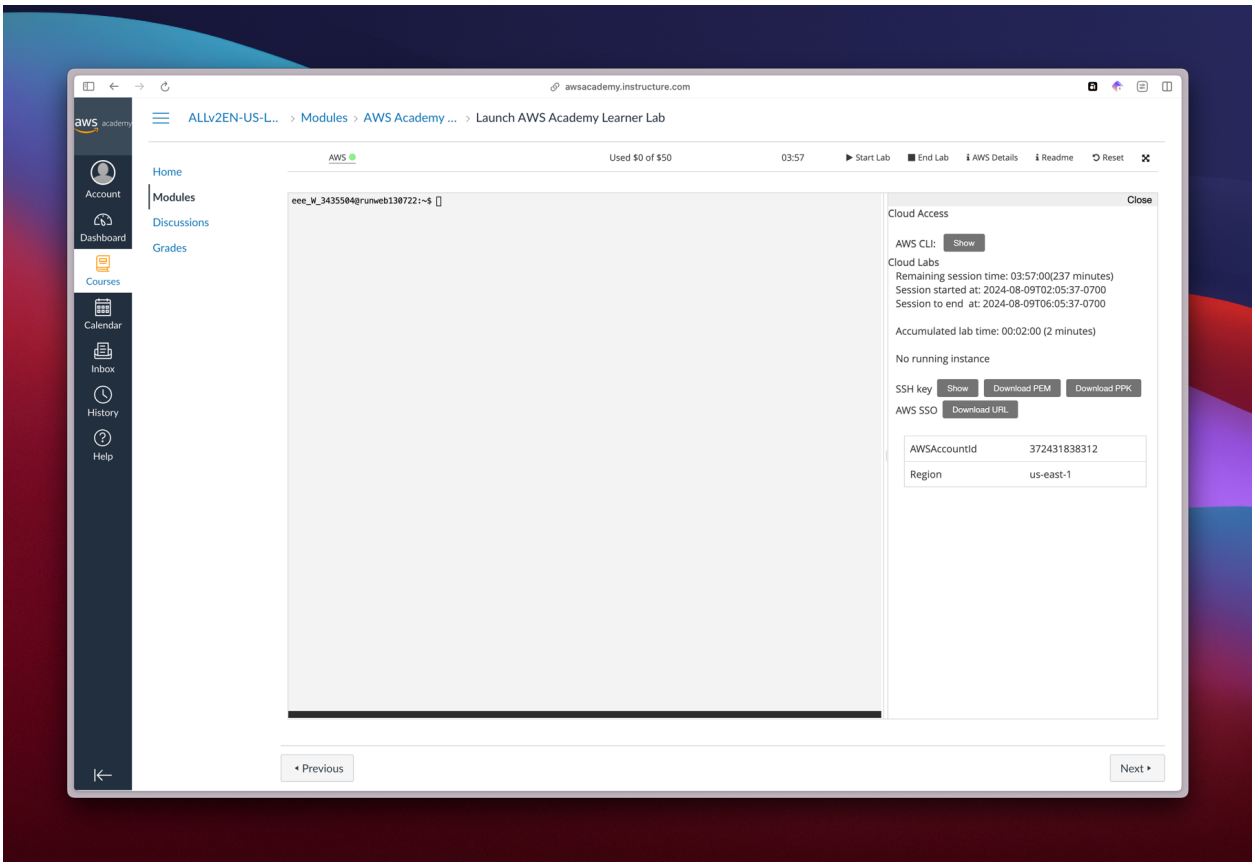
7. Start Lab

- Starte jetzt die Umgebung mit **Start Lab**
- Das "Lämbchen" wird nun gelb



8. AWS Details

- Ist das "Lämpchen" grün, ist die Umgebung gestartet.
- über "AWS Details" können nun alle Informationen und Credentials angezeigt werden.


awsacademy.instructure.com
ALLv2EN-US-L...
Modules
AWS Academy ...
Launch AWS Academy Learner Lab
Account
Dashboard
Courses
Calendar
Inbox
History
Help
Home
Modules
Discussions
Grades
AWS
Used $0 of $50
03:57
Start Lab
End Lab
AWS Details
Readme
Reset
eee_W_3435504@runweb130722:~$
Close
Cloud Access
AWS CLI:
Show
Cloud Labs
Remaining session time: 03:57:00(237 minutes)
Session started at: 2024-08-09T02:05:37-0700
Session to end at: 2024-08-09T06:05:37-0700
Accumulated lab time: 00:02:00 (2 minutes)
No running instance
SSH key
Show
Download PEM
Download PPK
AWS SSO
Download URL
AWSAccountId
372431838312
Region
us-east-1
Previous
Next

Web-Applikation Template Repository

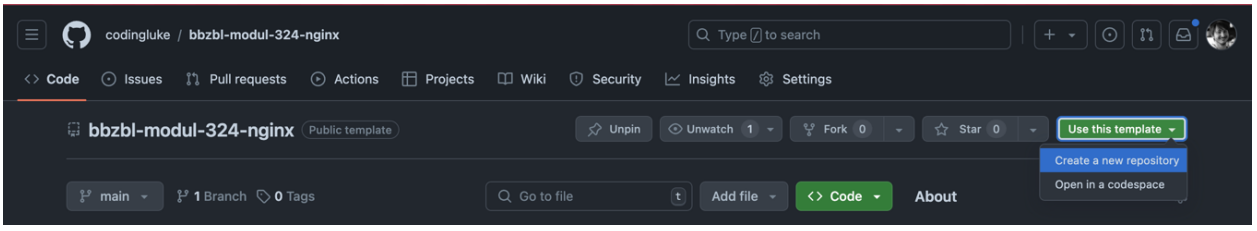
Das Modul verfolgt das Prinzip, ***make-it-work, then-make-it-yours***. Dies heisst es existiert ein funktionsfähiges, minimales Beispielprogramm. In diesem Fall ist es ein einfachen nginx Webserver, welcher über ein GitHub Template Repository von jedem Lernenden geklont und dann individuell angepasst wird.

> **ZIEL**
>
> - Das Ziel vom Modul ist es, dass jeder mindestens diese Beispiel App verstanden hat

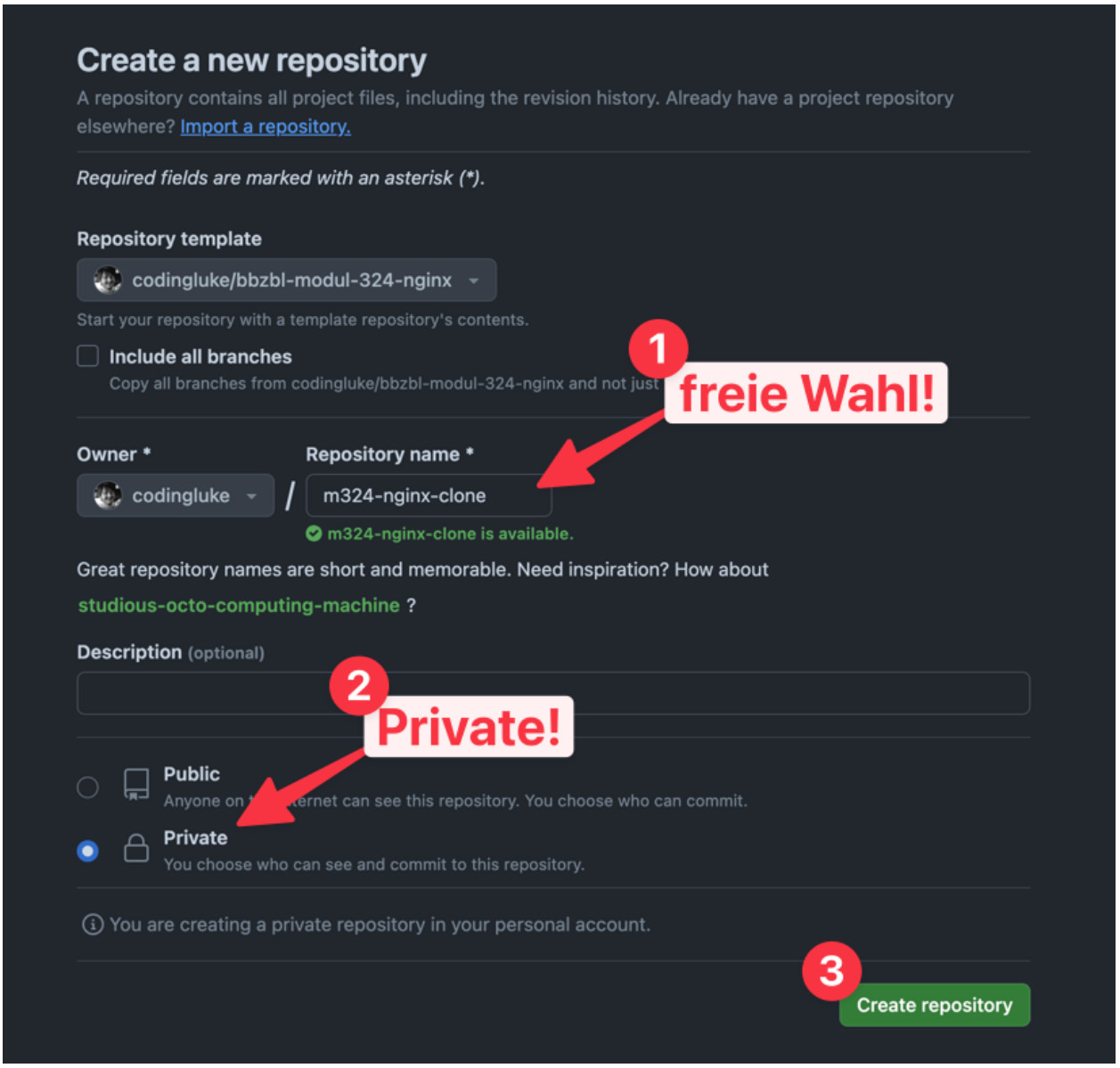
Aufgaben

Das Template `bbzbl-modul-324-template` klonen

- In den GitHub Account einloggen
- https://github.com/codingluke/bbzbl-modul-324-template aufrufen
- "Use this Template" -> "Create a new repository" wählen



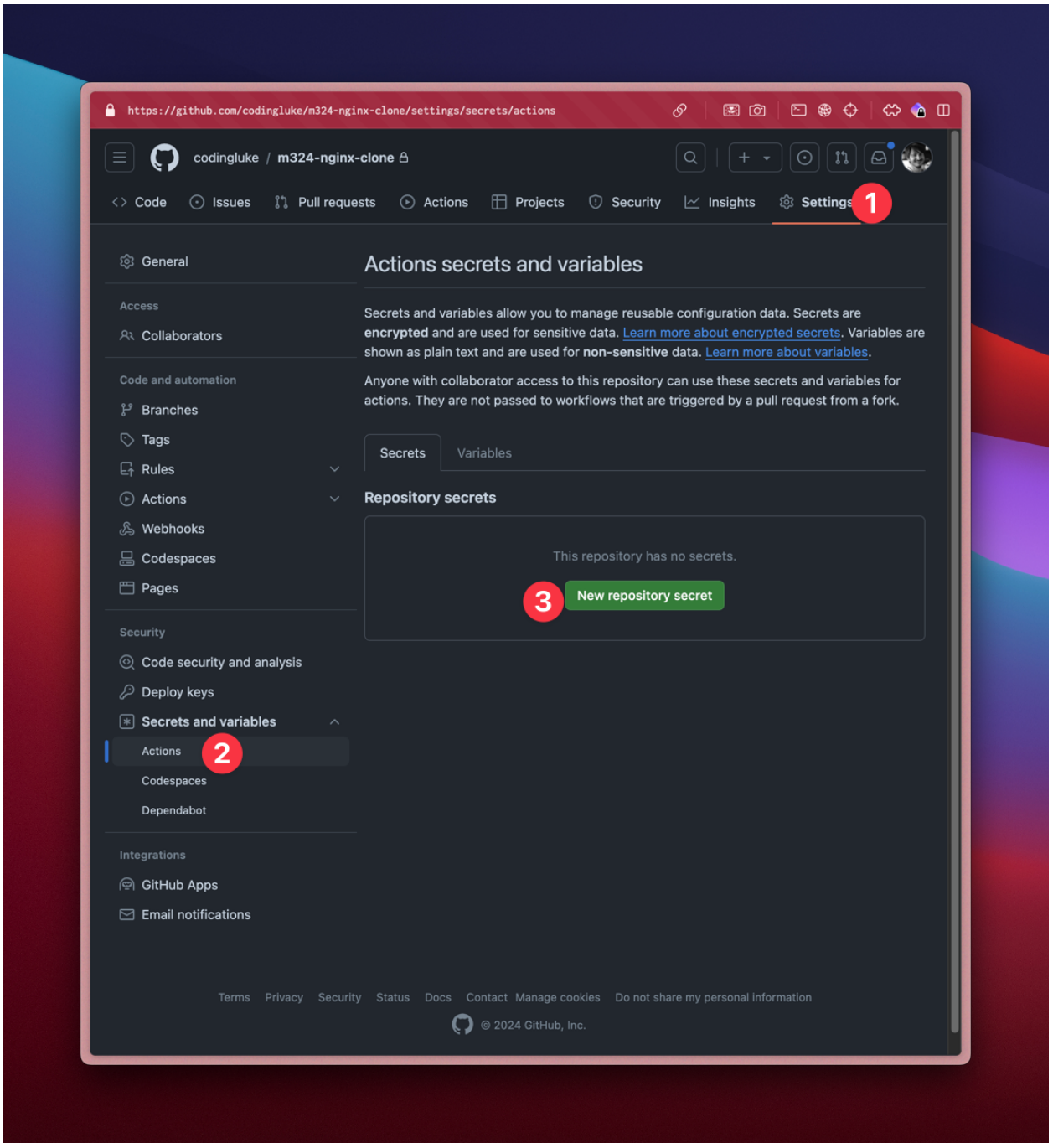
- Formular ausfüllen



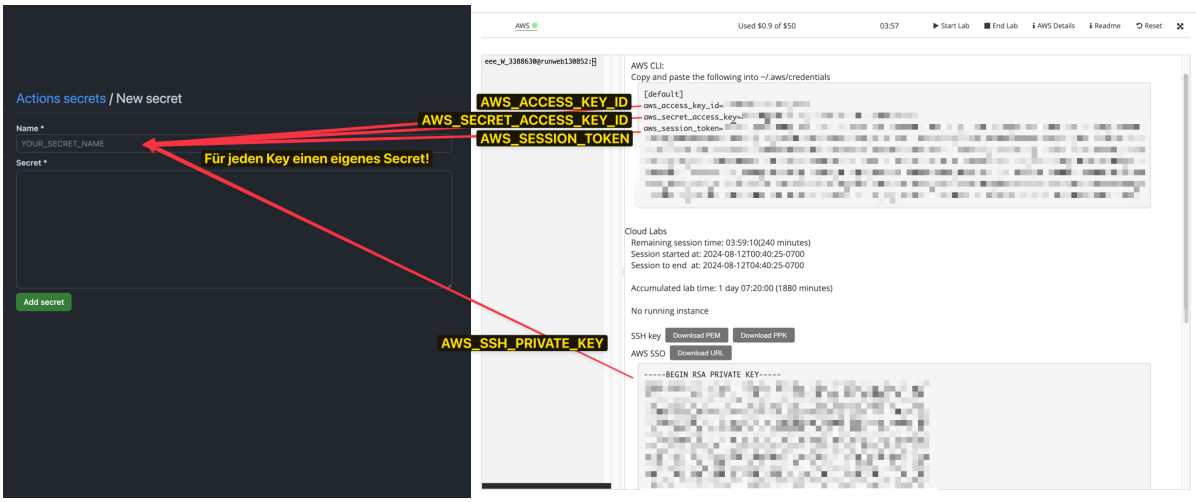
- Der Name dürft Ihr selber wählen.
- Bitte **"Private"** auswählen, später mir (codingluke) berechtigen
- Mit **"Create Repository"** bestätigen

AWS Credentials in Repository Environment Variablen übertragen

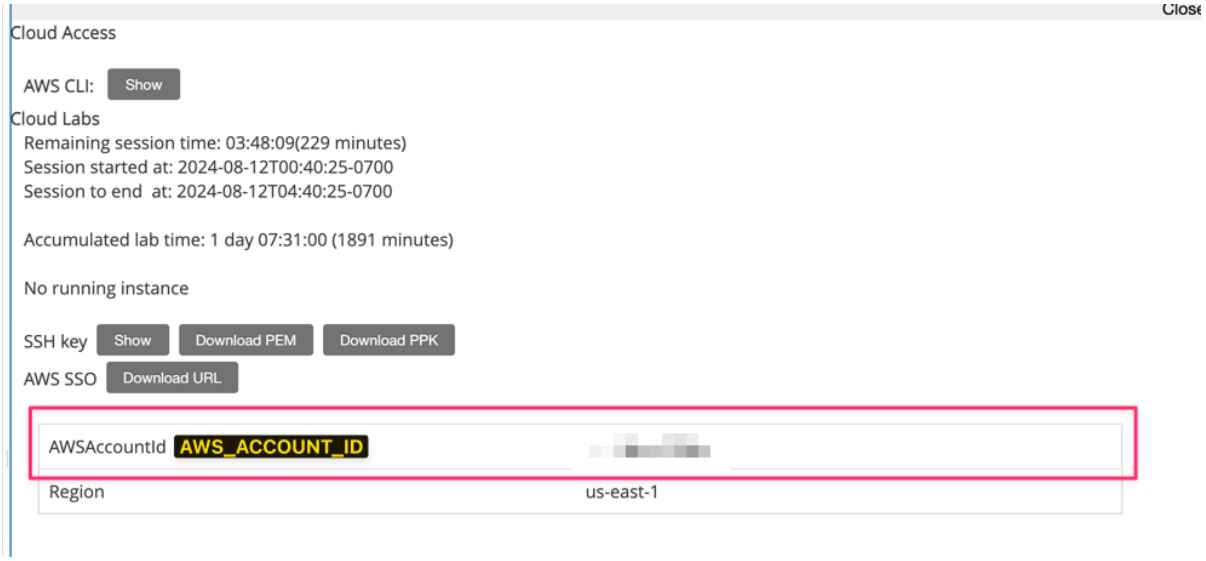
- Im neu erstellten Repository auf **"Settings" -> "Secrets and variables" -> "Actions"** navigieren
- Auf **"New Repository secret"** klicken



- Für alle Einträge unter **"AWS Details"** ein Secret erstellen, es sollten **5 Secrets** existieren
  - AWS_ACCESS_KEY_ID
  - AWS_SECRET_ACCESS_KEY
  - AWS_SESSION_TOKEN
  - AWS_SSH_PRIVATE_KEY

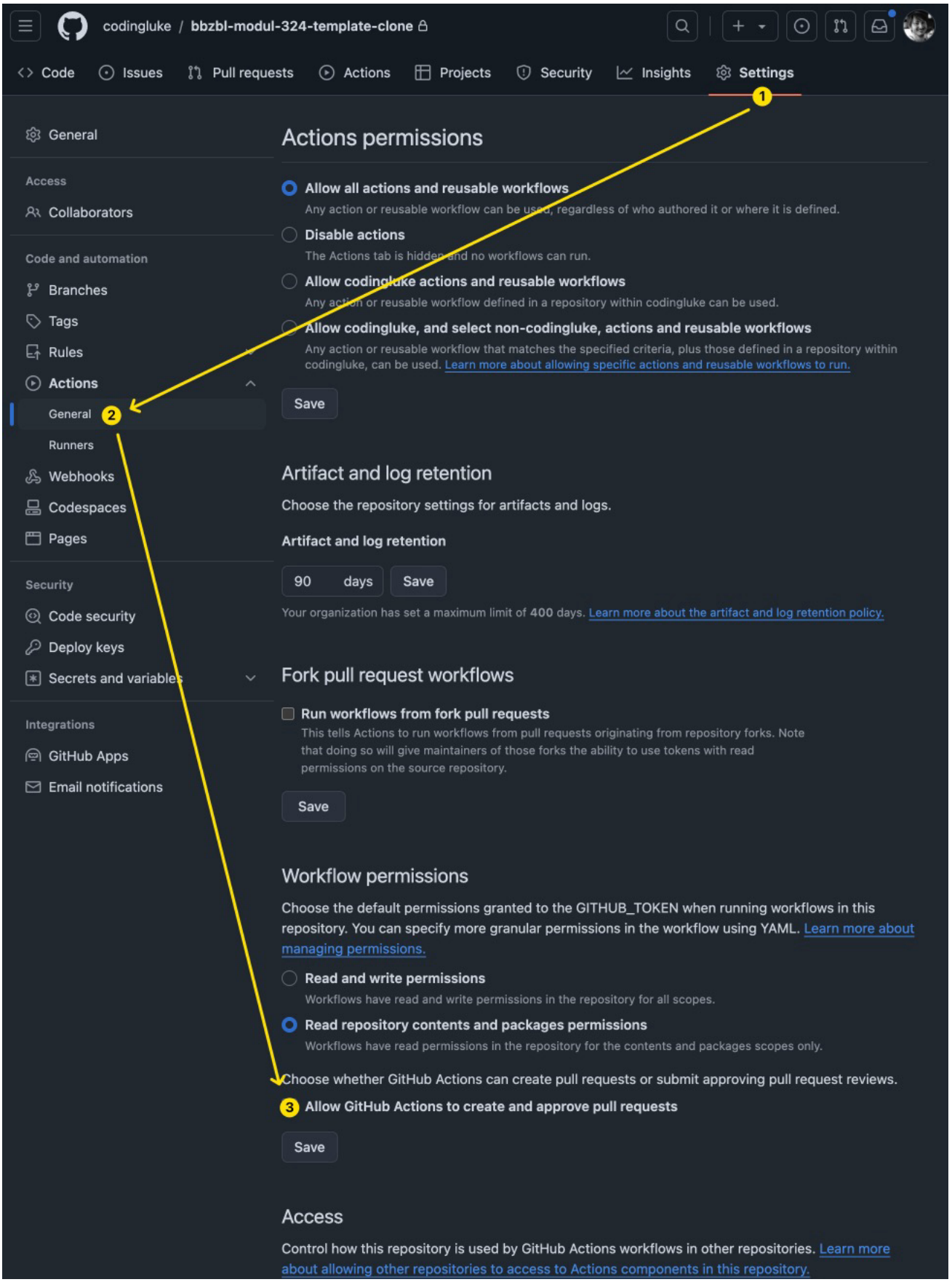


- AWS_ACCOUNT_ID



Berechtigungen setzen

1. Im neu erstellten Repository auf **"Settings" -> "Actions" -> "General"** navigieren
2. Ganz nach unten scrollen
3. **"Allow GitHub Actions to create and approve pull requests"** erlauben


codingluke / bbzbl-modul-324-template-clone
Code
Issues
Pull requests
Actions
Projects
Security
Insights
Settings
1
General
Access
Collaborators
Code and automation
Branches
Tags
Rules
Actions
General
2
Runners
Webhooks
Codespaces
Pages
Security
Code security
Deploy keys
Secrets and variables
Integrations
GitHub Apps
Email notifications
Actions permissions
Allow all actions and reusable workflows
Any action or reusable workflow can be used, regardless of who authored it or where it is defined.
Disable actions
The Actions tab is hidden and no workflows can run.
Allow codingluke actions and reusable workflows
Any action or reusable workflow defined in a repository within codingluke can be used.
Allow codingluke, and select non-codingluke, actions and reusable workflows
Any action or reusable workflow that matches the specified criteria, plus those defined in a repository within codingluke, can be used. Learn more about allowing specific actions and reusable workflows to run.
Save
Artifact and log retention
Choose the repository settings for artifacts and logs.
Artifact and log retention
90 days
Save
Your organization has set a maximum limit of 400 days. Learn more about the artifact and log retention policy.
Fork pull request workflows
Run workflows from fork pull requests
This tells Actions to run workflows from pull requests originating from repository forks. Note that doing so will give maintainers of those forks the ability to use tokens with read permissions on the source repository.
Save
Workflow permissions
Choose the default permissions granted to the GITHUB_TOKEN when running workflows in this repository. You can specify more granular permissions in the workflow using YAML. Learn more about managing permissions.
Read and write permissions
Workflows have read and write permissions in the repository for all scopes.
Read repository contents and packages permissions
Workflows have read permissions in the repository for the contents and packages scopes only.
Choose whether GitHub Actions can create pull requests or submit approving pull request reviews.
3
Allow GitHub Actions to create and approve pull requests
Save
Access
Control how this repository is used by GitHub Actions workflows in other repositories. Learn more about allowing other repositories to access to Actions components in this repository.

> **INFO**
>
> - Diese Berechtigung ist wichtig, dass automatisch Pull-Request erstellt werden können, was die Automatisierung von Releases ermöglicht.
> - Mehr dazu nächste Woche.

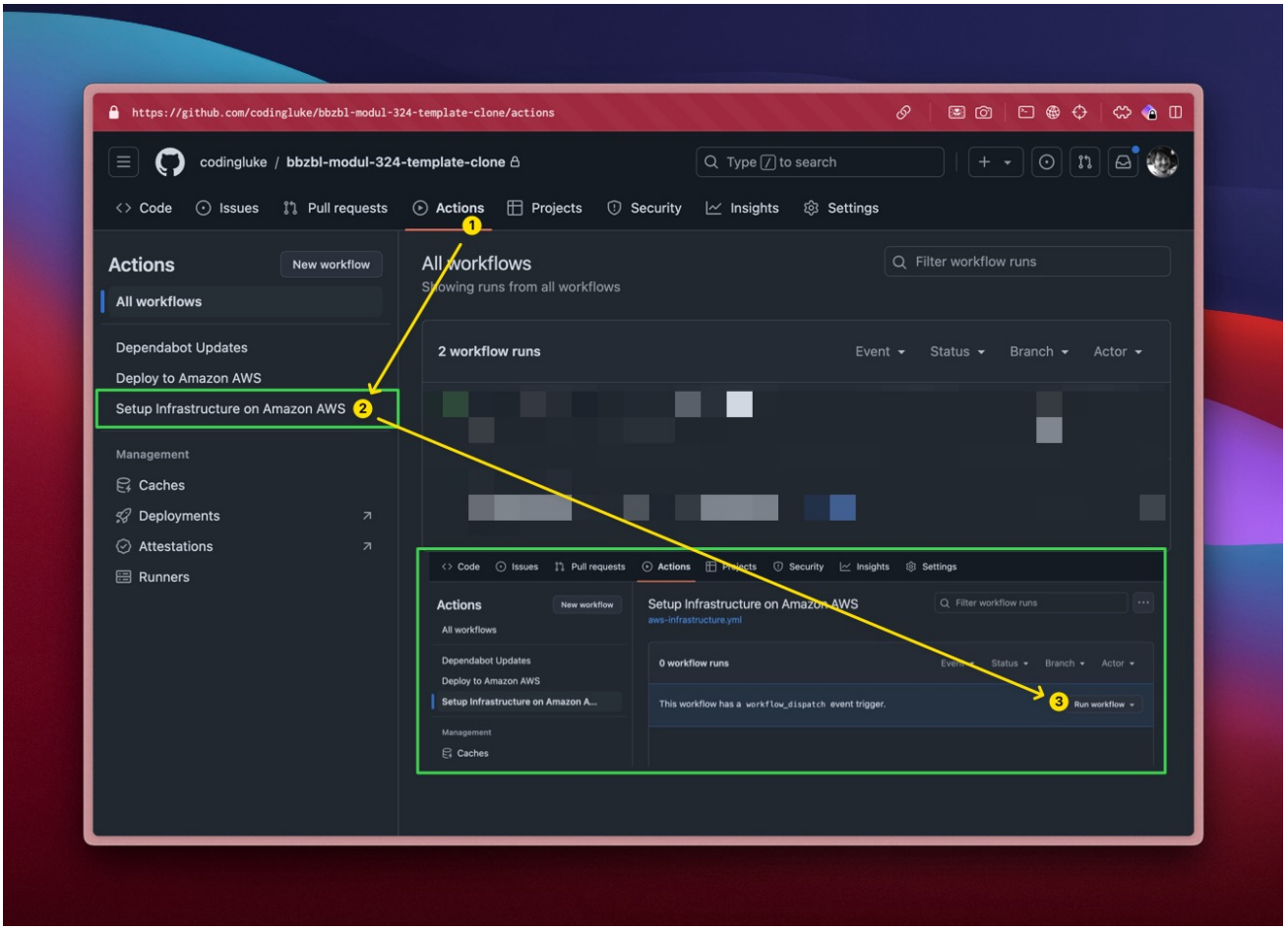
Amazon AWS Infrastruktur aufsetzten

Es existiert eine GitHub Action Setup Infrastructure on Amazon AWS

> **VORSICHT**
>
> - Diese kann manuell gestartet werden.
> - Da die Action sich mit Amazon verbindet, müssen **zuerst die Credentials übertragen** sein.

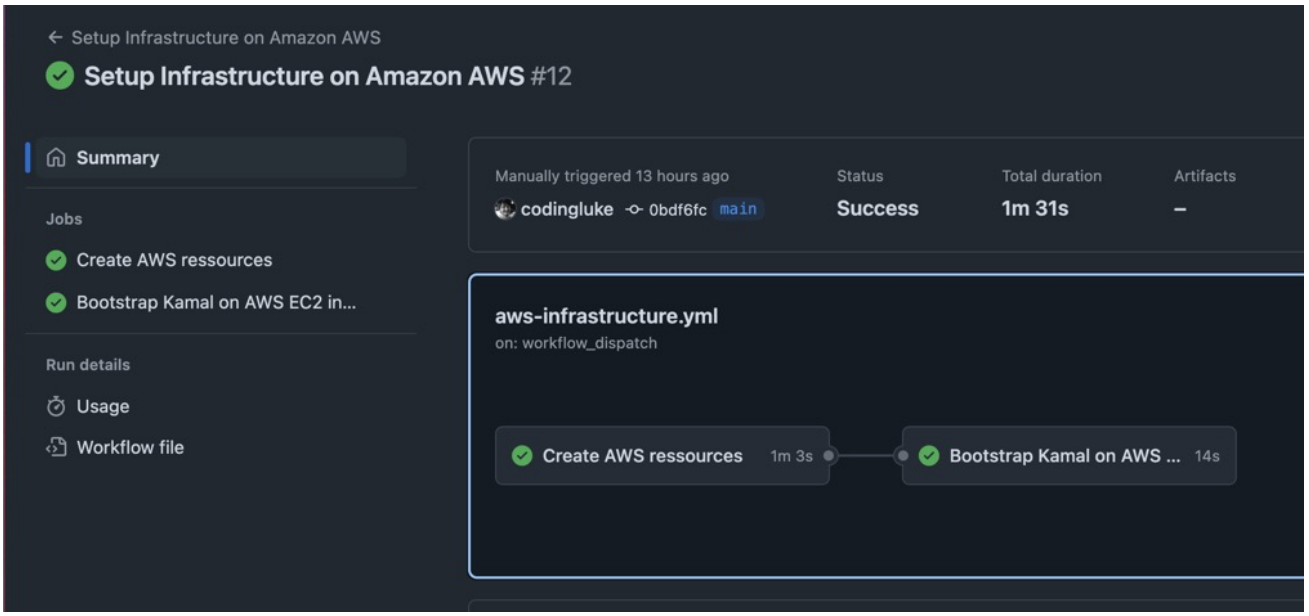
Danach die Action folgendermassen gestartet werden:

1. Nach **"Actions"** navigieren
2. **"Setup Infrastructure on Amazon AWS"** auswählen
3. **"Run workflow"** auswählen



Nun wird ein Workflow mit zwei Steps gestartet:

- der Erste "Create AWS Resourcen" erstellt alle AWS Resourcen, d.H. die Ubuntu VM und co.
- der Zweite "Bootstrap Kamal on AWS EC2 instance" installiert die nötigen tools in der Ubuntu VM

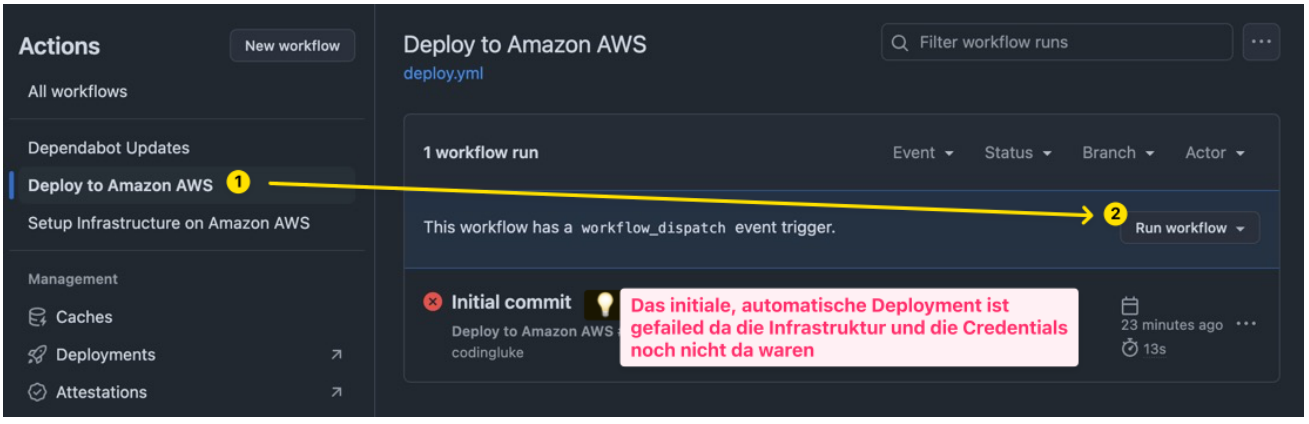


- Wenn alles grün ist (hoffentlich 😅) sollte die Umgebung parat sein, dass Docker-Images deployed werden können.

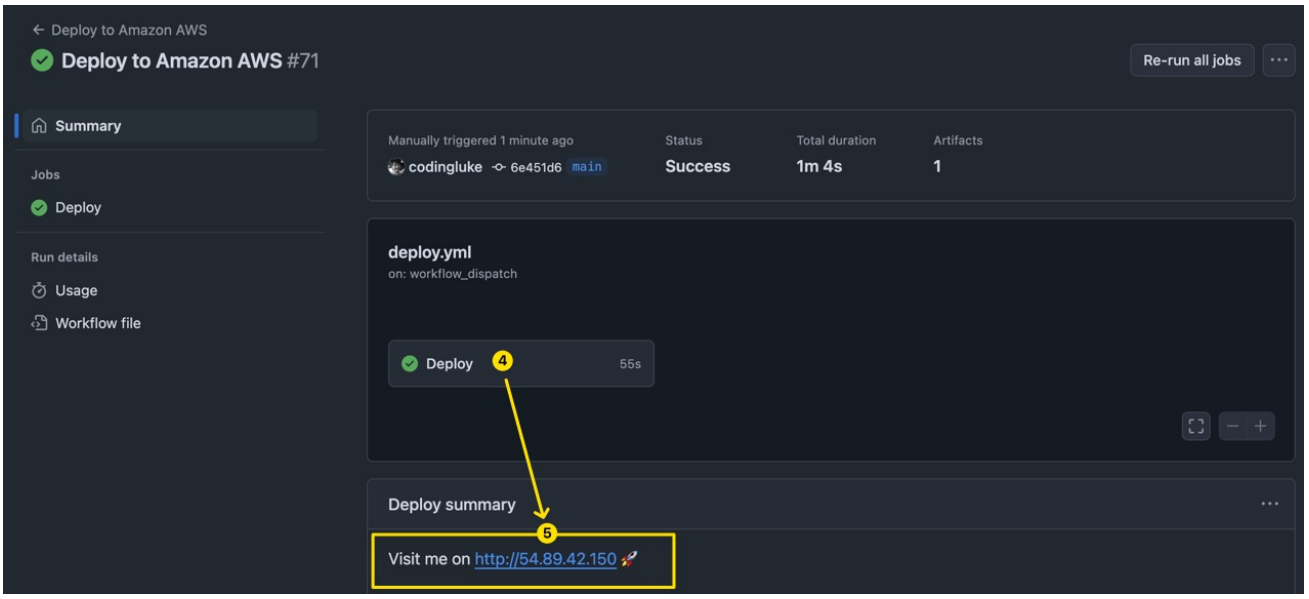
Nginx Image bauen und ausliefern (deployen)

Ist die AWS Umgebung aufgesetzt, kann die Action "Deploy to Amazon AWS" analog gestartet werden.

1. **"Deploy to Amazon AWS"** auswählen
2. **"Run workflow"** auswählen



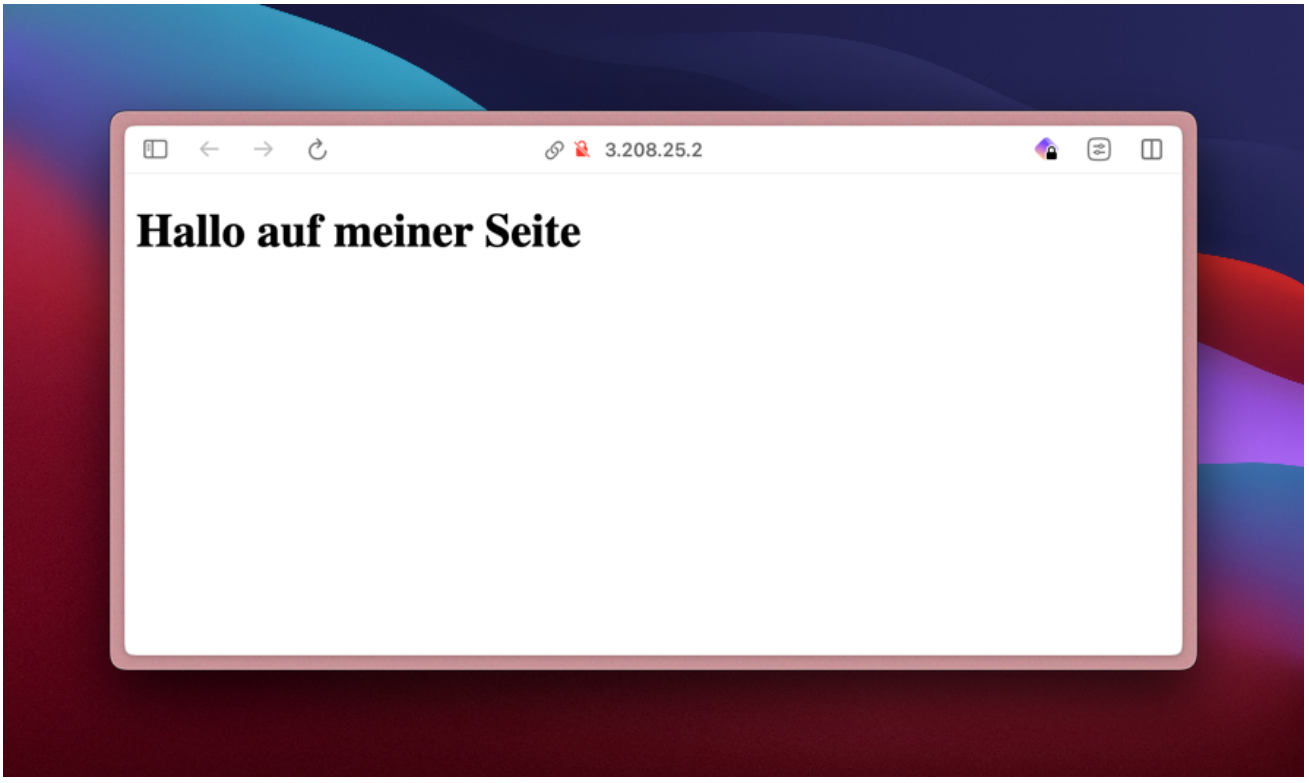
3. Warten bis das Deployment geklappt hat
4. Nun sollte einen Link "Visit me on http://public-ip 🚀" sichtbar sein



Webseite in Browser öffnen

Nun ist die Web-App auf AWS ausgeliefert und öffentlich verfügbar.

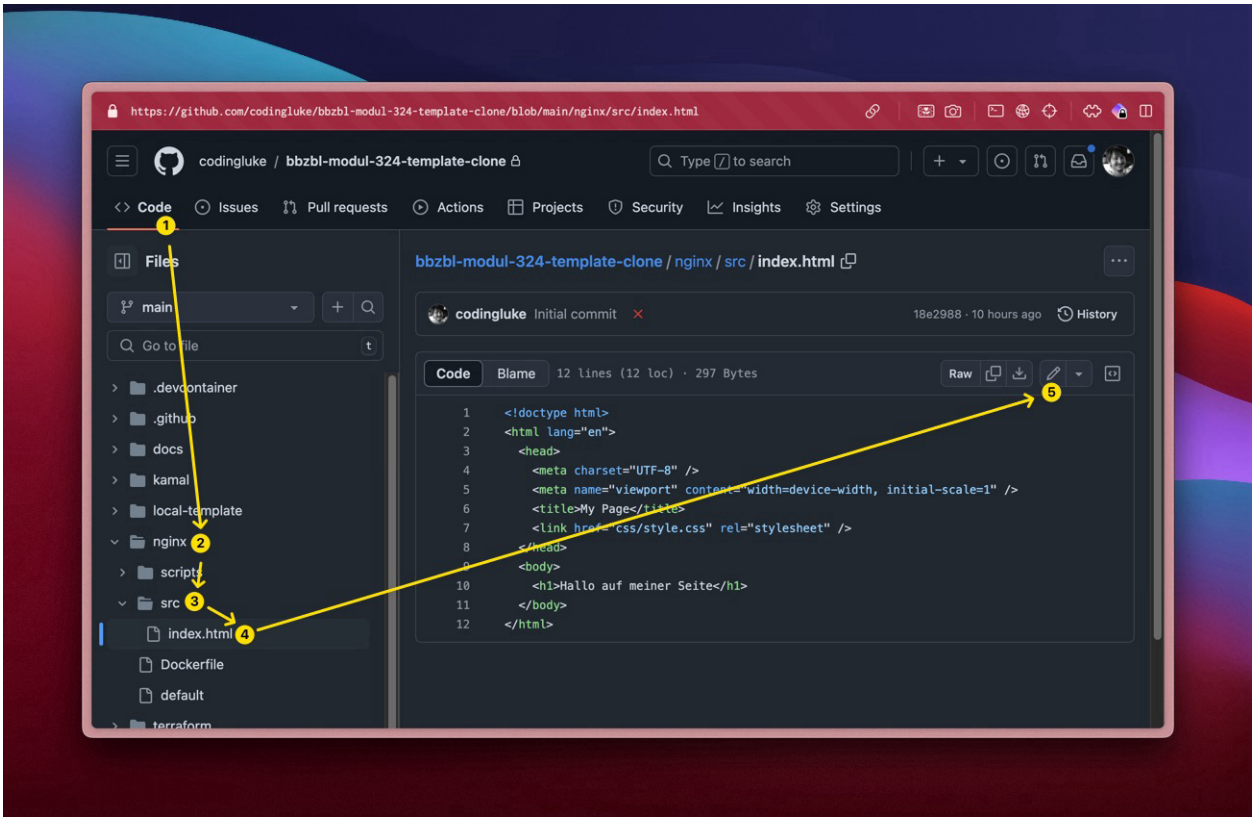
> **HTTP IST UNSICHER...**
>
> - Es handelt sich um **http** und nicht https!
> - Wenn euer Browser motzt, evt. Firefox verwenden.



HTML bearbeiten

Nun ist es möglich Änderungen an der App vorzunehmen, welche automatisch auf AWS deployed werden!

- Navigiert zu **"Code -> nginx -> src -> index.html"** und klickt auf **"Edit"**

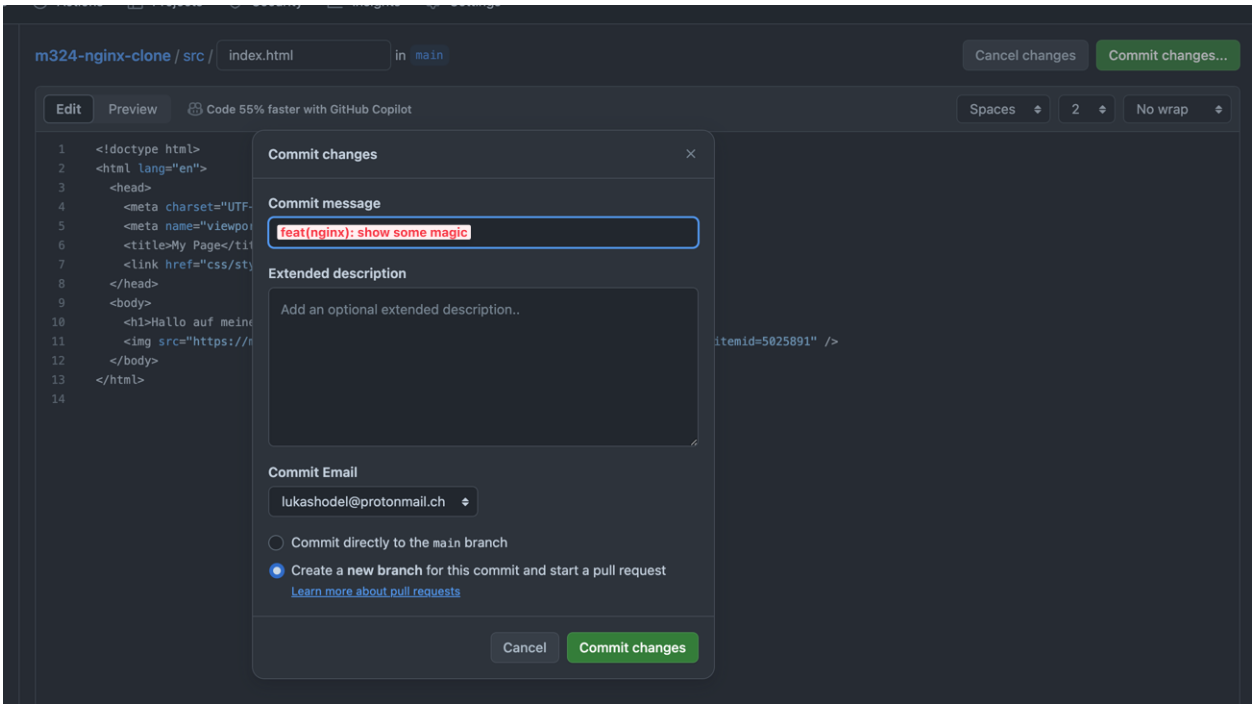


- Fügt folgenden Tag in den Body ein

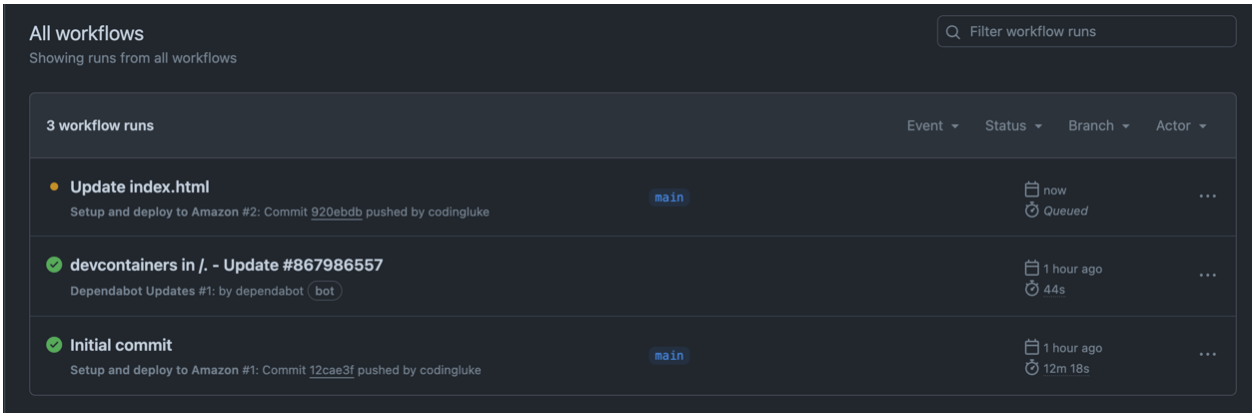
```
<img
  src="https://media1.tenor.com/images/b5c0aaca498dd84fa218239572165129/tenor.gif?itemid=5025891"
/>
```

- **"Commit changes"**.

> **VORSICHT**
>
> - 💥 Verwendet bitte die Commit-Message `feat(nginx): show some magic`!
> - 💥 Create a new branch for this commit and start a pull request



- In einem Pull-Request, könnte man nun den Code-Change kommentieren
- Nach dem mergen wird automatisch die GitHub Action **"Deploy to Amazon AWS"** gestartet.
- Warten bis die GitHub Action fertig ist



- 💥 **Nun die Webseite refreshen**

Teambildung

🎯 Jeder kennt sein Team und seine Rolle | 🧬 Gruppenarbeit | 🕐 10 Min

Ihr werdet ab der grossen Pause bis zum Ende vom Modul als Gruppe ein Projekt erarbeiten. Diese Aufgabe soll nun dazu dienen möglichst ausgewogene Gruppen zu finden, damit jede Gruppe gute Erfolgsbedingungen hat.

Eine Gruppe funktioniert am besten wenn möglichst viele Charaktertypen zusammenspielen. So muss nicht jeder der Super Programmierer sein, dafür gut koordinieren, Tickets beschreiben.

- 💥 **Nur zusammen ist man Stark!**
- 💥 **Gute Rollenverteilung ist die halbe Miete!**

📝 Auftrag

Auf der Wandtafel gibt es eine Tabelle mit folgenden Kategorien:

Koordinieren	Kontrollieren	Programmieren
-	-	-

- Überlegt in welcher Ihr euch am ehsten sieht.
- Anhand der Einschätzungen werden nun 3er Gruppen erstellt
- Am Besten ist es, wenn jede Gruppe je eine Person von jeder Kategorie beinhaltet.

> 💡 **TRAUT EUCH MIT NEUEN ZUSAMMENZUARBEITEN!**

Rollenbeschreibung

- Erstellt das GitHub Repository und ladet andere eine
- übernimmt Admin Aufgaben
- Kontrollieren -> **Stakeholder**:
  - Schaut sich die Arbeitspakete besonders gut
  - prüft die Kriterien
- Programmieren -> **Projektmitarbeiter**:
  - Ist besonders fleissig am Programmieren

GitHub Repo erstellen

🎯 Arbeitsgrundlage schaffen | 🧬 Gruppenarbeit | 🕐 10 Min

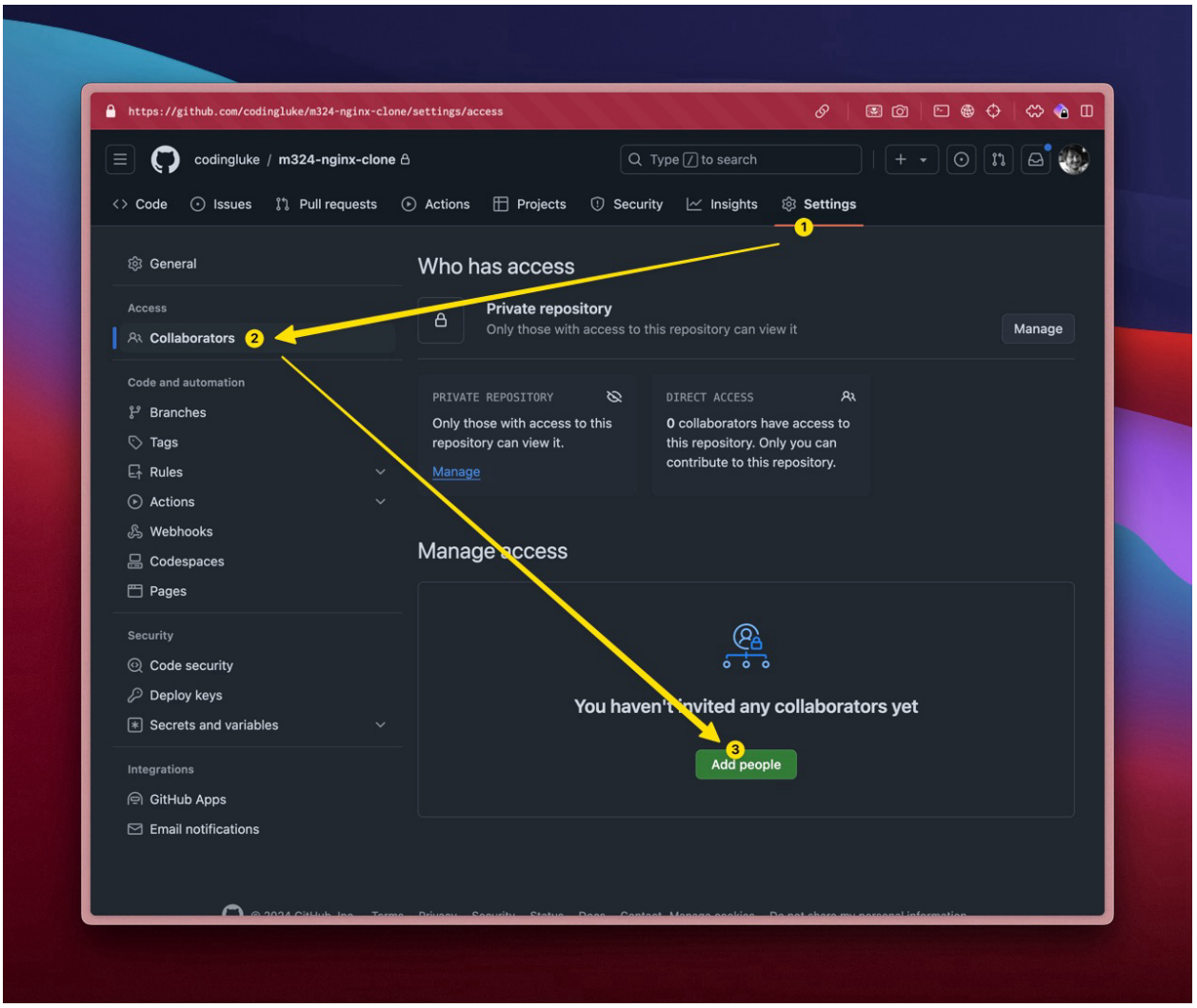
> ⚠️ **IM PROJEKT ARBEITET IHR ALLE IM GLEICHEN REPOSITORY!**

📝 Auftrag

1. Der **Teamleader erstellt** ein GitHub Repository genau gleich wie letze Woche.
   - Zur Aufgabe: Nginx Beispiel App - Das Template klonen
   - Die AWS Credentials könnt Ihr später noch einfügen

   > 💡 **TIPP**
   >
   > Der Teamleader kann auch **das bereits geklonte Repo einfach umbenennen**, sofern es noch nicht all zu fest angepasst wurde.

2. Der **Teamleader** gibt allen Mitgliedern die **Berechtigungen**
   - Bitte auch der Lehrperson `@codingluke`


https://github.com/codingluke/m324-nginx-clone/settings/access
codingluke / m324-nginx-clone
Type / to search
Code
Issues
Pull requests
Actions
Projects
Security
Insights
Settings
1
General
Access
Collaborators
2
Code and automation
Branches
Tags
Rules
Actions
Webhooks
Codespaces
Pages
Security
Code security
Deploy keys
Secrets and variables
Integrations
GitHub Apps
Email notifications
Who has access
Private repository
Only those with access to this repository can view it
Manage
PRIVATE REPOSITORY
Only those with access to this repository can view it.
Manage
DIRECT ACCESS
0 collaborators have access to this repository. Only you can contribute to this repository.
Manage access
You haven't invited any collaborators yet
3
Add people

GitHub Projekt erstellen

🎯 Arbeitsgrundlage schaffen | 🧬 Gruppenarbeit | 🕐 20 Min |

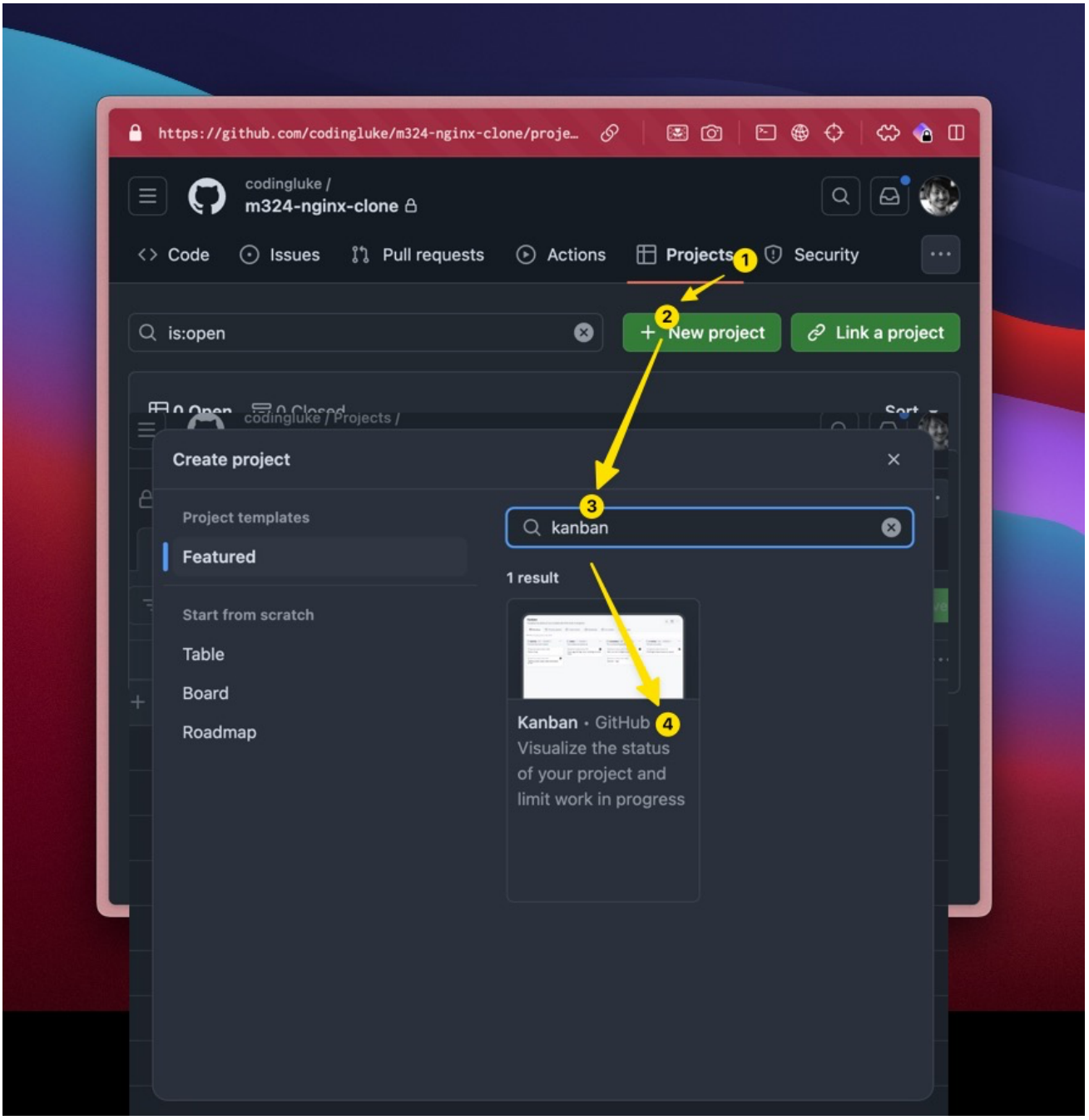
Das Projektmanagement fürs Projekt wird komplett in GitHub durch mehrere Issues und einem GitHub Projekt realisiert.

> **ⓘ HINWEIS**
>
> Ein GitHub Projekt ist nicht auf ein Repository gebunden. Das kann zu Verwirrung führen.
>
> - Es gibt auch einen offiziellen Thread dafür Projects Beta Support adding default repo for project

Auftrag

Erstellt ein GitHub Projekt im Projekt Repository

GitHub Projekt

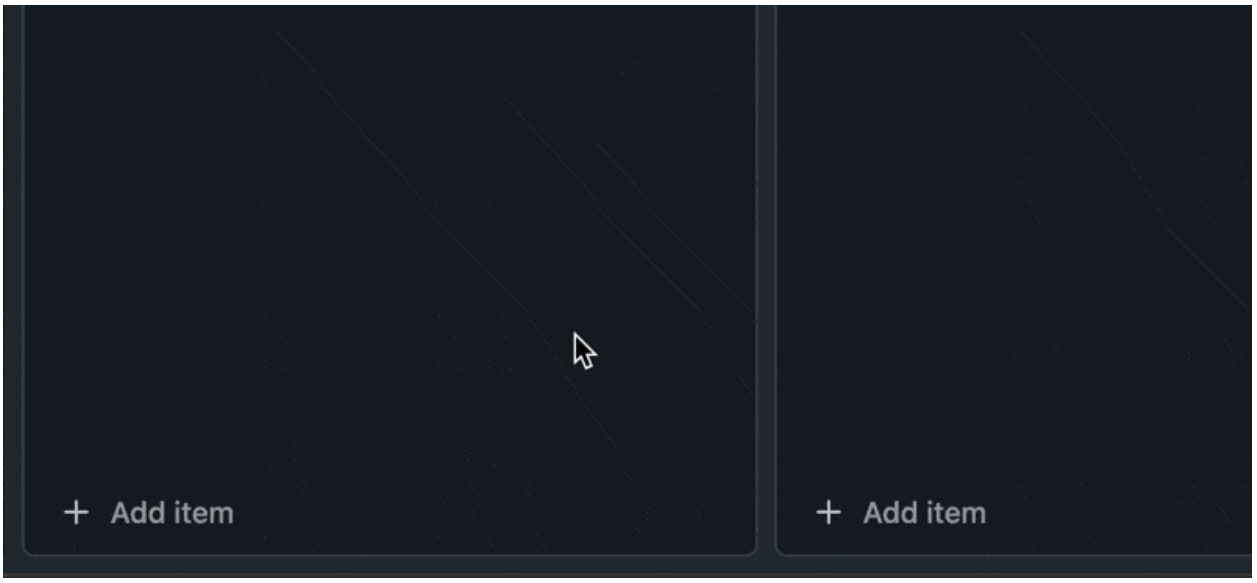


Öffnet die Project details und fügt als Beschreibung einen Link zum Repository rein.

- Dies ermöglicht das einfache Navigieren vom Projekt zum Repository

Erstellt erste 📜 GitHub Issues und spielt damit

- mit `#` kann das Repository ausgewählt werden, in welchem der Issue erstellt werden soll (siehe gif)
- Wird einfach nur ein "Item" erstellt, ist es ein "draft" im Projekt und noch nicht an einen Issue gebunden.

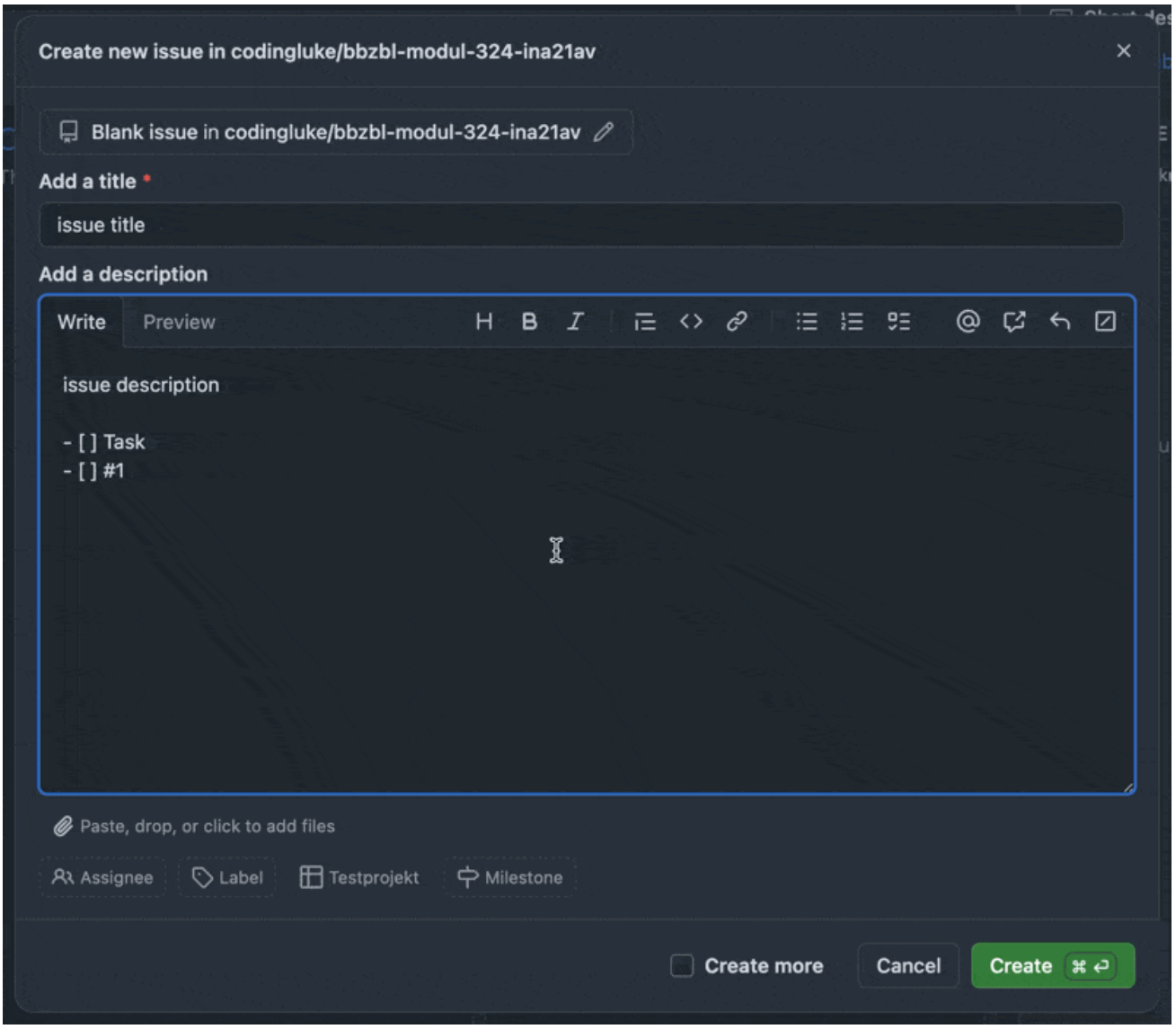


- Verknüpfen, Taggen
- Erstellt Tasks in Markdown (dieser kann man toggeln)

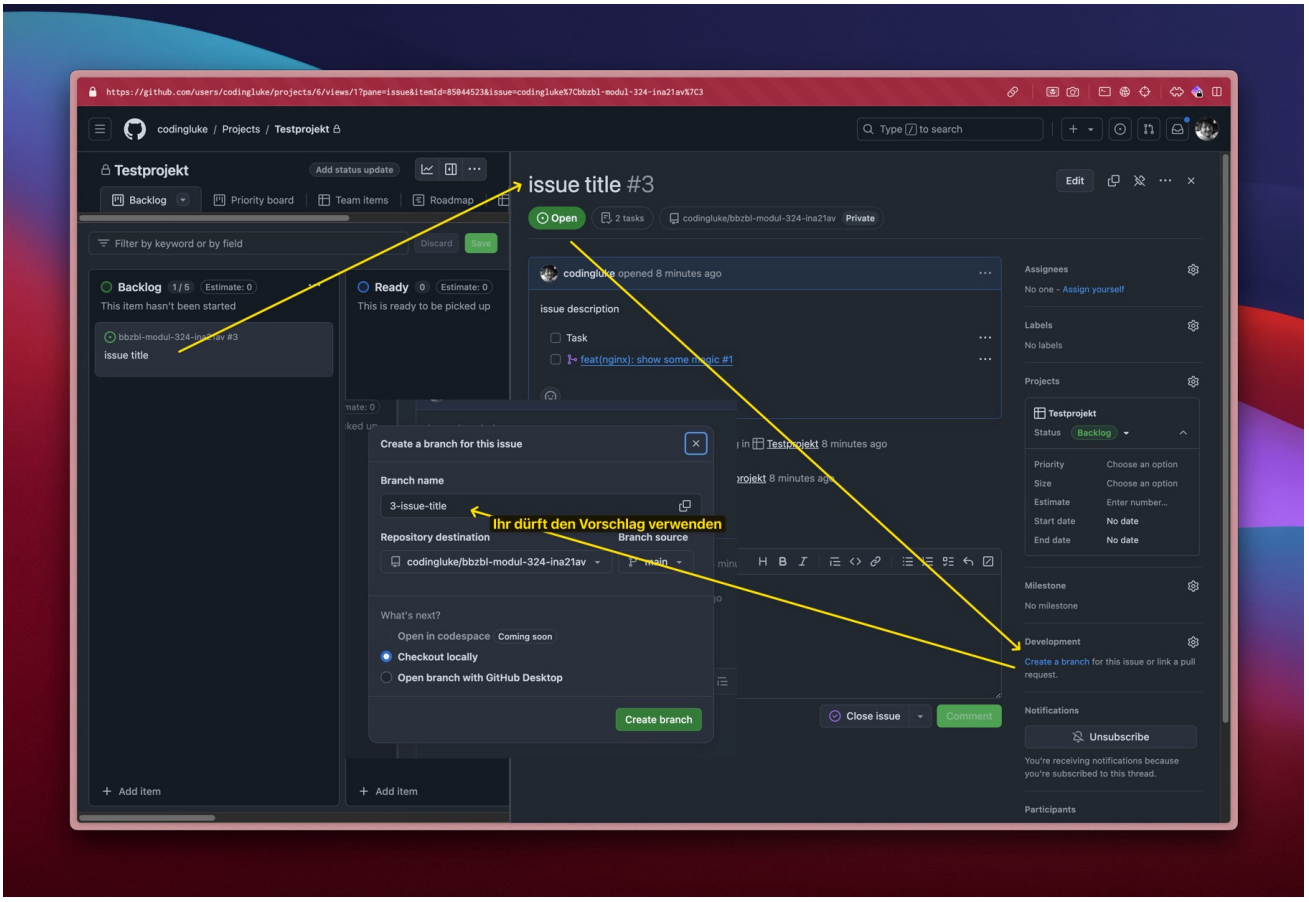
```
- [ ] Task 1
```

- Erstellt Eine Referenz zu einem Issue (hier #1) in Markdown

```
- [ ] #1
```


Create new issue in codingluke/bbzbl-modul-324-ina21av
Blank issue in codingluke/bbzbl-modul-324-ina21av
Add a title *
issue title
Add a description
Write
Preview
issue description
- [ ] Task
- [ ] #1
Paste, drop, or click to add files
Assignee
Label
Testprojekt
Milestone
Create more
Cancel
Create

Erstellt einen Branch direkt aus dem Issue

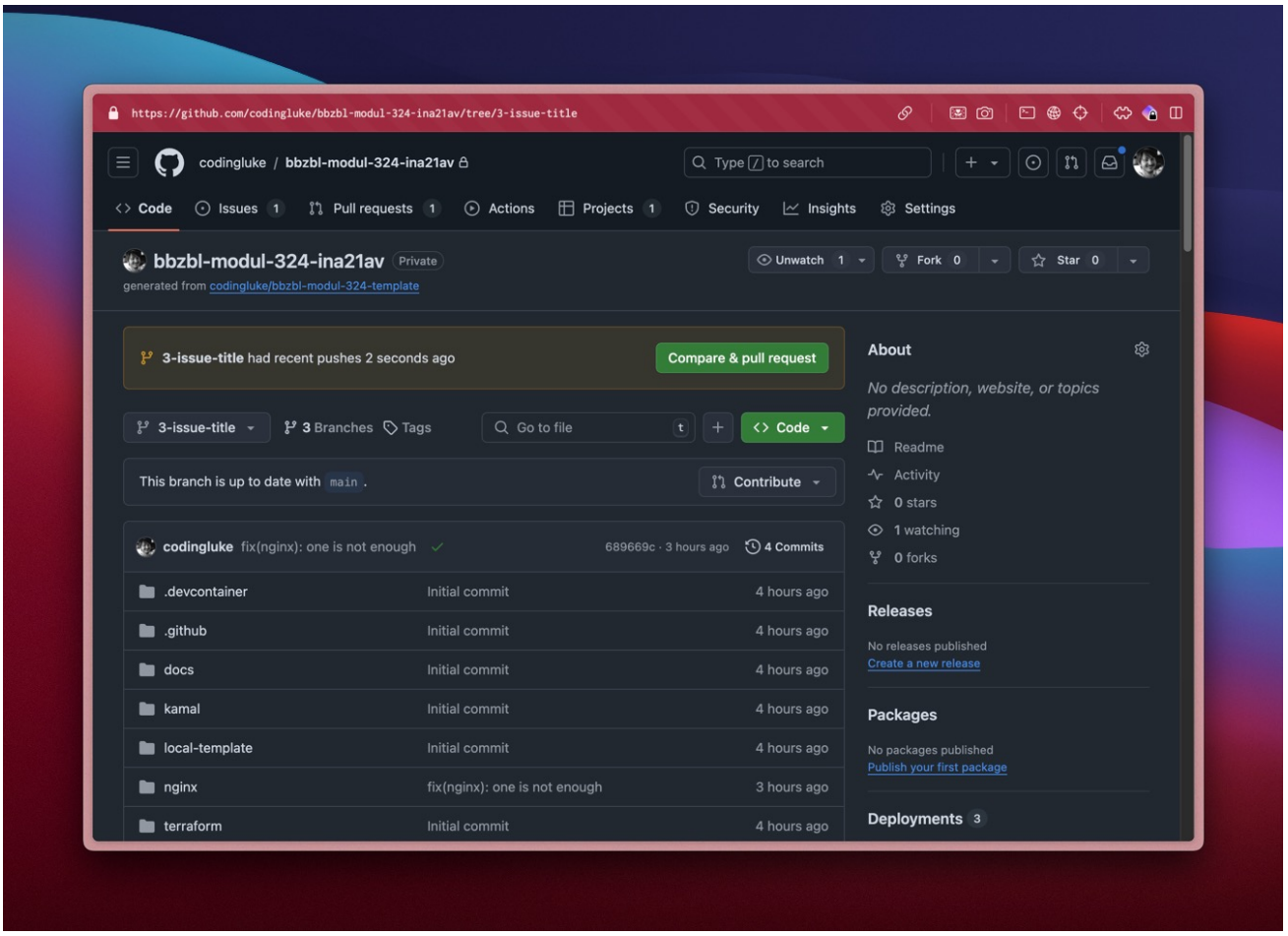


TIPP

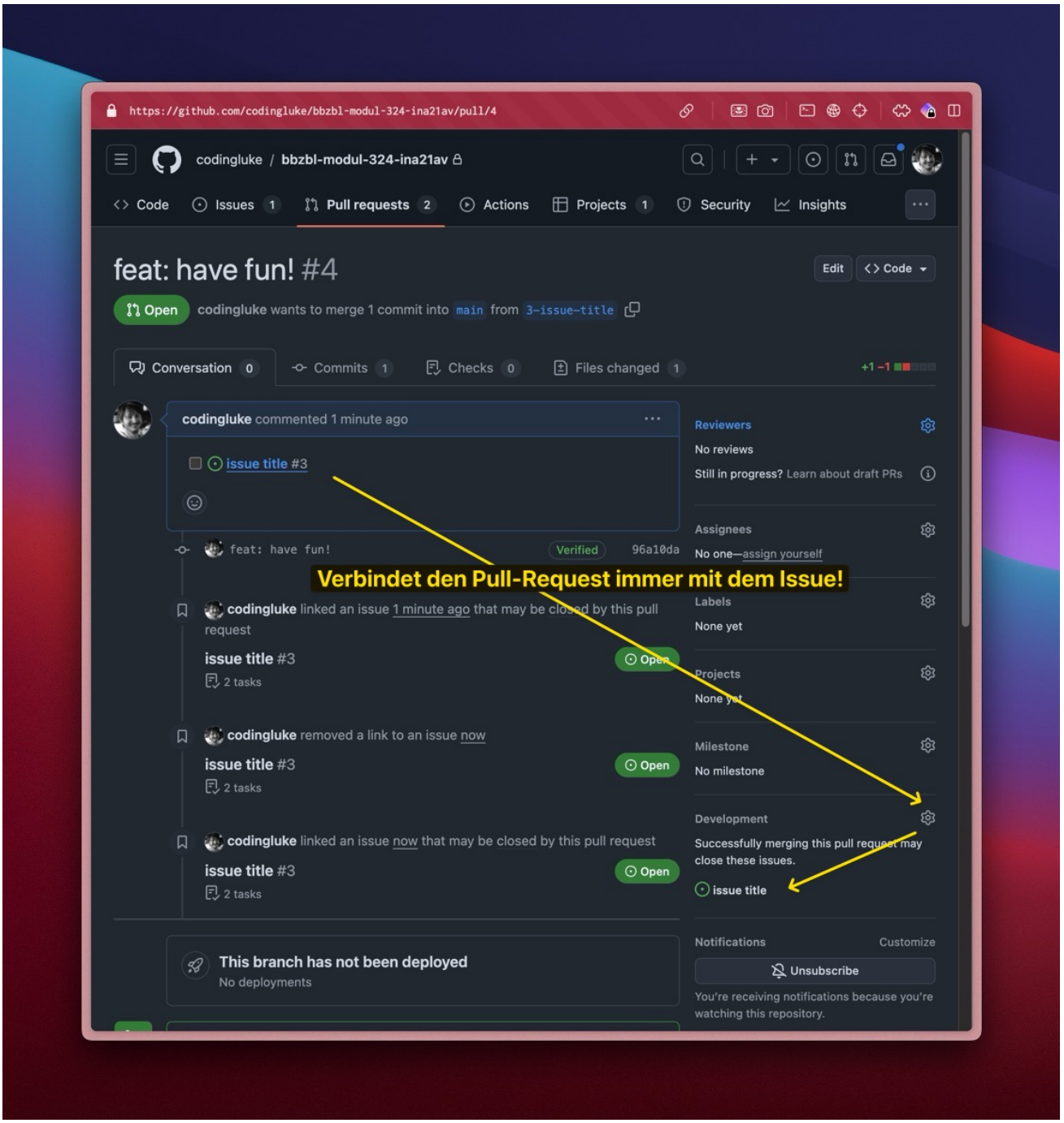
- Erstellt ihr die Branches auf diesem Weg, sind sie automatisch an den Issue geknüpft.
- Keine Angst es ist auf gleiche weise möglich einen Branch zu Verknüpfen
- Wichtig is

Erstellt einen Pull-Request vom Branch und linkt ihn zum Issue

- Öffnet den Branch im Repository
- Macht durch das GUI eine Änderung, am besten wieder die Datei `/nginx/src/index.html`
  - ❗ Dabei müsst ihr nicht nochmals einen Subbranch erstellen!



- Linkt den Issue zum Pull-Request!



NOTENRELEVANT

Es wird Bewertet, dass alle Issues welche geschlossen sind einen Pull-Request referenzieren.

- Ausnahmen: Issue welche keinen Code geneireren wie z.B. Mockub erstellen und co.
- Die Ausnahmen müssen aber als Definition of Done beweisen, wieso der Issue "done" ist.
  - Link zu eine externen Resource, wie z.B. ein Google Folder mit Bilder oder so.
  - Link zu erstellten Tickets resultiert von diesem
  - usw.

Projekt finden

🎯 Arbeitsgrundlage schaffen | 🧬 Gruppenarbeit | 🕐 30 Min

Ab jetzt arbeitet Ihr an eurem eigenen Projekt 🥳 Das Projekt könnt Ihr euch selbst aussuchen. Dabei sollt ihr jedoch ein paar Rahmenbedingungen beachten.

Rahmenbedingungen

- Es muss ein **Web-Projekt** sein
  - Desktop Apps lassen sich schlecht nach AWS deployen 😉
- Es sollte ein **neues Projekt** sein
  - Bestehenden Code anzupassen ist erfahrungsgemäss schwer für alle Mitarbeitenden die den Code noch nicht kennen.

> 💡 **PROJEKT IDEEN?**
>
> - 💡 Statische HTML-Seite, ToDo App oder was grösseres?
> - 💪 Seit mutig und kreativ!

- Es **muss nicht fertig werden**!
  - Es darf Ambitioniert sein, sollte aber klein anfangen, damit auch alle arbeiten können.
  - Das Projekt darf später gerne weiterverfolgt werden
- Es soll auf dem **Modul Template** aufbauen
  - Ich weis, einige von euch würden gerne von Scratch anfangen. Wir arbeiten hier aber im Team und **ich will verhindern, dass das Setup zu lange von einer Person abhängt**.

> 💡 **AWS MANAGEMENT KONSOLE BIS IM JANUAR OFFEN**
>
> Ambitionierte Teammitglieder dürfen gerne parallel dazu ihr eigenes Setup bauen und mir pitchen.
>
> - 🤓 Ich bin sicher ich kann viel von euch lernen.
> - 🚀 Gut möglich, dass ich sogar das Template erweitere.

📝 Auftrag

- Ein GitHub Issue, **Projekt finden**, erstellen.
- Issue sinngemäss beschreiben **mit Definition of Done**
- Ein **neuer Branch** erstellen
- Im Branch das Projekt im **root /README.md** beschreiben

- 💡 Ihr dürft auch zuerst in *Google Docs* die Beschreibung ausarbeiten.
- Am Ende sollte sie jedoch im README.md stehen.
  - **Bitte die bestehende Info nicht löschen.**
  - Den Titel *BBZBL Modul 324: Web-Applikation Template* zu einem Heading 2 ändern `##`
  - Einen eigenen Heading 1 Title, mit dem Projektnamen, einfügen `# Unser tolles Projekt`
  - Darunter die Beschreibung einfügen.

TIPP

- Wenn Sie das Repository noch nicht lokal geklont haben, können Sie direkt auf GitHub das `README.md` bearbeiten.

BEISPIELDATEI

/README.md

```
# Unser tolles Projekt

Eine einfach zu verstehende Beschreibung des Projekts. Es darf ruhig ein Pitch sein, wieso macht Ihr das, was ist die Inspiration und weshalb braucht die Welt dies.

## BBZBL Modul 324: Web-Applikation Template

alles hier bleibt gleich...
```

- Die Beschreibung *commiten*

VORSICHT

- 💥 **Zu verwendende Commit-Beschreibung:** `docs: Projektbeschreibung`
- 💡 Diese Commit-Beschreibung entspricht den conventional-commits, welche später noch genauer erläutert werden.

- Einen **Pull-Request** erstellen
- Der Pull-Request mergen, wenn er **von allen Approved** wurde.

Erste Epics und Issues definieren

🎯 Projekt grob erfasst haben | 🧬 Gruppenarbeit | 🕐 45 Min

Von Epics und Issues

Ein Issue sollte immer nur eine Aufgabe beschreiben. Am Anfang eines Projekts ist es jedoch illusorisch, schon viele, genaue Arbeitspakete erstellen zu können, da man das Projekt noch nicht genau kennt.

Hier kommen die Epics ins Spiel. Epics beschreiben nicht eine Aufgabe, sondern ein Bereich. z.B. könnte man sich in einem Web-Shop einen Epic "Warenkorb" vorstellen. Ein "Warenkorb" kann man nicht in einem Guss programmieren. Der Epic dient nun dazu zu beschreiben was man sich darin vorstellt. Dann kann man Issues erstellen die Teilbereiche abdecken wie z.B. Mockup erstellen, Produkt hinzufügen, Produkt löschen, Warenkorb anzeigen, Warenkorb bestellen, ...

> 💡 **BLEIBT AGIL**
>
> Es müssen noch nicht alle Epics und Issues erstellt werden. Es ist erlaubt und erwünscht im Projektverlauf neue Epics und Issues zu erstellen.

Issues und ihre Freunde

Wenn Arbeitspakete erstellt werden, sollten die so isoliert wie möglich sein, damit Sie unabhängig abgearbeitet werden können. Um dies zu ermöglichen ist es wichtig, dass die Issues nicht zu gross sind. Daher die Regel, dass ein Issue nur eine Aufgabe beinhalten soll.

Und trotzdem kommt es zwangsläufig dazu, dass Issues Abhängigkeiten zu anderen Issues haben. So macht z.B. "Produkt löschen" keinen Sinn, wenn "Produkt hinzufügen" nicht schon umgesetzt ist.

Diese Abhängigkeiten sollen in Form von Issue Referenzen (z.B. #1) in der Beschreibung abgebildet werden.

- Autolinked references and URLs

Ebenso sollen auch die Epics verlinkt sein!

NOTENRELEVANT

- Alle Issues sollten immer auf den zugehörigen "Epic" zeigen.
- Alle Issues sollen auf andere Issues verweisen, von welchem er abhängig ist.

Issue Beschreibung und die Definition of Done

Jeder Issue besitzt eine Beschreibung, diese **definiert die Aufgabe**, welche anhand vom Issue erstellt werden soll. Die Beschreibung richtet sich an den Programmierer.

INFO

- Die Beschreibung muss nicht einer Konvention wie "As a user a want to..." folgen, darf dies aber.
- Sie muss immer verständlich machen, was umgesetzt werden soll.

Um überprüfen zu können, ob ein Issue erledigt worden ist, benötigt es eine Kurzanleitung, wie dies verifiziert werden kann. **Hier sprechen wie von der "Definition of Done"**.

- **Die "Definition of Done" richtet sich an den Kunde!**

NOTENRELEVANT

- Alle Issues müssen eine Beschreibung beinhalten, der jeder, auch ohne Insider Wissen, entnehmen kann was zu tun ist.
- Alle Issues müssen eine "Definition of Done" beinhalten mit der der Kunde (also ich), die Umsetzung prüfen kann.

Sprint-Reflexionen

Die Reflexionen sollen auch als Pull-Request geschehen . Daher bietet es sich an für diese ein Epic mit Issues für jede Reflexion **pro Projektmitglied** zu erstellen.

TIPP

- Einen Epic "Sprint-Reflexionen" erstellen
- Für alle ein Issue für die Reflexion zum Sprint 1 und 2 erstellen

NOTENRELEVANT

- Achtet pingelig genau an die Anweisung unter Beurteilungen

Epics im Projekt

- **Es gibt keine Epics in GitHub Projects** 😳
- Erstellt ein Tag namens "epic".
- Erstellt Issues mit dem Tag "epic".
- *Ihr könnt auch* eine Swimmline "Epic" Erstellen, dies dient der Übersicht enorm.

> **HINWEIS**
>
> - 💡 Ab und zu muss man einfach kreativ sein und die Tools passend machen...

📝 Auftrag

Erstellt nun im GitHub Projekt neue Epics und Issues für euer Projekt.

Hier ein Beispiel von Themen:

- Administrative Issues wie z.B. Sprint-Reflexionen
- Technologiewahl treffen
  - Die Programmiersprache ist **frei wählbar**
- Infrastruktur aufbauen
- Hello World mit gewählter Technologie
- Erste Features beschreiben
- Mockup rsp. Wireframe erstellen

> **SCHAUT EUCH WOCHE 3 AN!**
>
> In der Woche 3 setzen wir zusammen auf, dass Ihr lokal entwickeln könnt. Das wäre doch ein paar Issues wert oder?

> **MOCKUPS (WIREFRAMES)**
>
> Am Anfang kann es gut möglich sein, dass noch nicht alle Code Schreiben können, da die Applikation noch nicht Lauffähig ist. Daher macht es Sinn auch **Issues für das "Mockup", also ein grafischer Preview** von Features zu erstellen.
>
> - 💡 Hat man bereits ein Mockup, kann man auch besser über Features sprechen und es kommt zu weniger Missverständnissen.
> - 💡 **Die Mockups könnt Ihr z.B. unter `/docs/mockups` einchecken.**
> - ❗ **Bitte keine Videos einchecken!**

Benötigte Tools installieren

Docker

Damit zügig am Projekt gearbeitet werden, wird das Template mit einem DevContainer ausgeliefert. Dieser benötigt docker. Auch ist es am einfachsten weitere Services, wie z.B. eine Datenbank, via Docker zu starten als manuell zu installieren.

- Docker Desktop
- Docker Compose Sollte mit Docker Desktop dabei sein.

> **KEIN DOCKER?**
> Grundsätzlich ist es möglich auch ohne docker lokal zu entwickeln. Ihr seit selber verantwortlich alle Tools zum entwickeln zu installieren und konfigurieren.

Git

- Git
  oder Github Desktop Macht das Clonen einfacher auf Windows

VS Code

Wir arbeiten im Modul mit VS Code. Sofern Ihr ein Java Projekt erstellt dürft Ihr auch IntelliJ verwenden. Ihr seit aber selber verantwortlich dass es funktioniert.

- VS Code
- Java Guys dürfen auch IntelliJ verwenden.
- ❗ Aber Achtung, devcontainer sind in VS Code besser unterstützt!

VS Code must have Plugins

Bitte Installiert all diese Plugins!

> **LEST DIE DOKUS!**
> - Die Plugins erklären sich selbst am besten ;)

Conventional Commits

Dieses Plugin "versteht" Conventional Commits und hilft dabei diese korrekt zu schreiben *(sofern man in VS Code die Git-Commits erstellt)*

EditorConfig for VS Code

Dieses Plugin garantiert, dass alle (auch Windows Benutzer) die richtigen Line Endings verwenden.

> **GEFAHR**
>
> Es gibt gewisse Bash-Scripte unter `/terraform/scripts` die nicht funktionieren mit Windows-Lineendigns!

Docker

Standard Plugin um laufende Docker containers in VS Code zu sehen. Geht auch ohne, VS Code wird euch sowieso nerven es zu installieren.

Dev Containers

Dieses Plugin ermöglicht es direkt in einem lokalen Container zu entwickeln.

> **KEIN BOCK AUF DEVCONTAINER?**
>
> Ihr dürft auch ohne DevContainer entwickeln. Müsst euch dann einfach absprechen, welche Tools (Programmierspachen und co.) in welcher Version verwendet werden sollen.

Remote - SSH

Wird vom Plugin "Dev Containers" benötigt.

SSH Key einrichten

Um ein eigenes Git-Repository von GitHub lokal zu klonen muss man sich authentifizieren. Grundsätzlich gibt es zwei Arten um auf ein Repository zuzugreifen, https und ssh. Da https bei jedem `git pull` und `git push` den Benutzer und das Passwort abfragt, empfehle ich jedem das Repository via `ssh` zu klonen. Damit kann man ohne user und passwort arbeiten. Man benötigt jedoch einen `ssh-key` den man auf GitHub registrieren muss.

Aufgabe

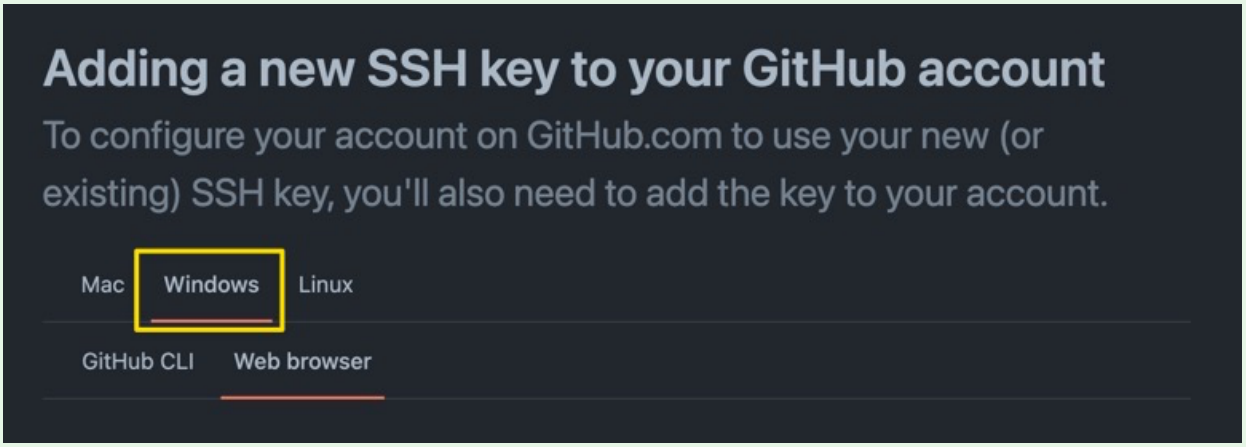
Bitte folgt den unten stehenden Anleitungen von GitHub um einen `ssh-key` zu erstellen und auf GitHub zu registrieren.

- Adding a new SSH key to your GitHub account

Alle die bereits einen Key haben können direkt hier vortfahren

- Adding a new SSH key to your account

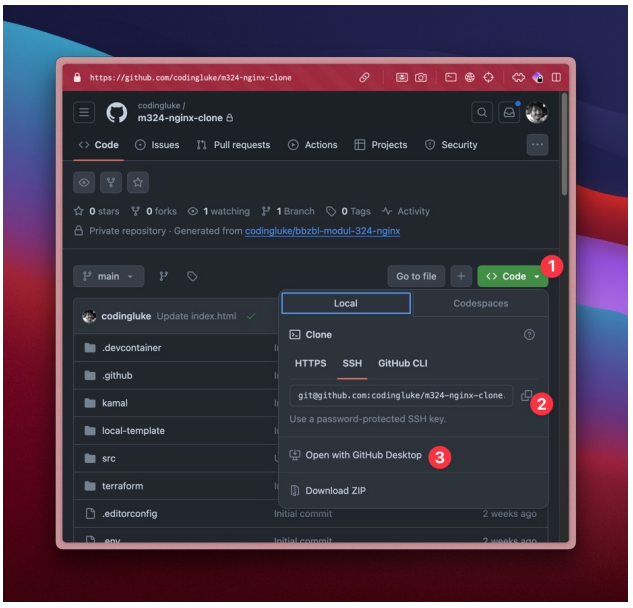
ES GIBT EINEN WINDOWS TAB AUF DEN WEBSEITEN ☺



Repository lokal klonen

Nachdem Ihr eine `ssh-key` generiert und in GitHub hinterlegt habt, sollte es möglich sein euer Repository lokal zu klonen.

1. Unter Code Das grüne Dropdown `<> Code` öffnen
2. Die **SSH Url** kopieren
3. oder Sofern GitHub Desktop installiert wurde **Open with GitHub Desktop** wählen.



In Bash klonen

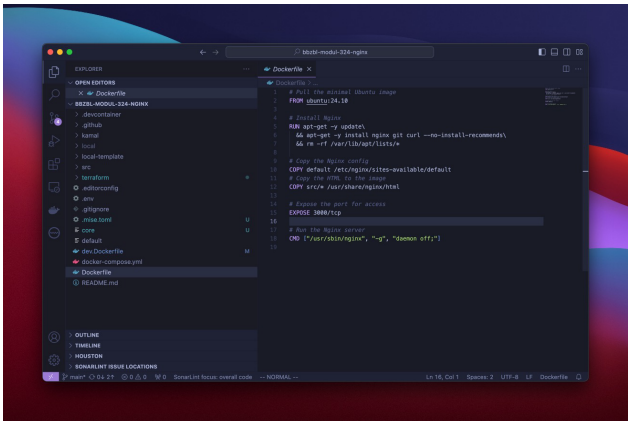
Danach könnt ihr lokal folgender `git` Befehl ausführen.

```
git clone {kopierte-ssh-url-zum-repo}
```

> **ICH WÜRDE ES SO MACHEN, DANN LERNT IHR GERADE DIE GIT BEFEHLE.**

In VS Code öffnen

1. `File -> Open folder`
2. Das frisch geklonte repository auswählen
3. Es sollte ungefähr wie rechts aussehen



Direkt in VS Code klonen

Alternativ könnt Ihr auch in VS Code das Repository klonen.

- How to Clone a Github Repository with VS Code (Example)

In Kürze:

- Die Kommando-Pallete öffnen mit `Ctrl+Shift+P` (Windows) oder `Command+Shift+P` (mac).
- Nach **"Git: clone"** suchen
- Bestätigen, die kopierte ssh-URL einfügen und dem Wizzard folgen.

Devcontainer einrichten

Der Devcontainer wird durch das Dockerfile unter `.devcontainer/Dockerfile` beschrieben. In dem `Dockerfile` können Tools installiert und konfiguriert werden, die fürs Projekt nötig sind. Dies erleichtert den Einstieg für neue Projektmitglieder und **garantiert dass alle die gleiche Umgebung haben**.

> ⓘ **KEIN MUST!**
>
> Theoretisch könnte auch jeder Projektmitarbeiter alle Tools `nodejs`, `java` und co. selbst installieren. Hat ja früher auch funktioniert ☺
>
> - ❗ **Es muss darauf geachtet werden, dass alle dieselben Versionen verwenden**.

> ⓘ **DEVCONTAINER VS PRODUKTIVES DOCKER-IMAGE**
>
> - Der Devcontainer (`.devcontainer/Dockerfile`) dient zum entwickeln. Darin wird gearbeitet.
>   - 💡 **Der Devcontainer wird lokal gestartet**.
> - Das produktive Image (`nginx/Dockerfile`, oder später `ihr-projekt/Dockerfile`) ist optimiert. Dieses sollte so klein wie möglich sein und nur das nötigste beinhalten. Meisten z.B. die gebaute Applikation (ohne node_moduls und co.) und nicht der dev Build.
>   - 💡 **Das produktive Image wird als Container auf AWS gestartet**.

Aufgaben

Mit `docker compose` den Devcontainer starten und prüfen

- Den Devcontainer starten

```
docker compose up devcontainer -d
```

- Mit `docker exec` eine shell im Container öffnen

```
docker exec -it devcontainer /bin/bash
```

- Im Container den `nginx` starten:

```
chmod +x nginx/scripts/start-nginx.sh # evt. nicht nötig, schadet aber nicht
sh nginx/scripts/start-nginx.sh
```

- Prüfen ob der Webserver läuft.

```
curl http://localhost:3000
```

- Mit `curl` kann man beliebige HTTP Request absetzen. `curl` wird daher oft fürs testen verwendet

PORT 3000 MUSS FREI SEIN

Wenn auf dem Port 3000 bereits ein Prozess läuft, muss dieser gekillt werden

Windows

```
$ netstat -ano | findstr :3000
>TCP     0.0.0.0:3000    0.0.0.0:0   LISTENING   2660
>TCP     [::]:3000       [::]:0      LISTENING   2660
$ taskkill /PID 2660 /F
```

Linux

```
kill $(lsof -t -i:3000)
```

Docker Container

Wenn der Port 3000 von einem anderen Docker Container belegt ist muss dieser gestoppt werden.

```
$ docker ps
> CONTAINER ID   IMAGE    COMMAND                  CREATED         STATUS
PORTS                                     NAMES
> 95a55d07e361   a-image   "/bin/sh -c 'while s…"   2 seconds ago   Up 2
seconds   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp  container-name
$ docker stop container-name
```

TIPP

Natürlich kann man auch via Browser den Webserver testen. Dafür muss `http://localhost:3000` im Browser geöffnet werden.

In VS Code Devcontainer starten und prüfen

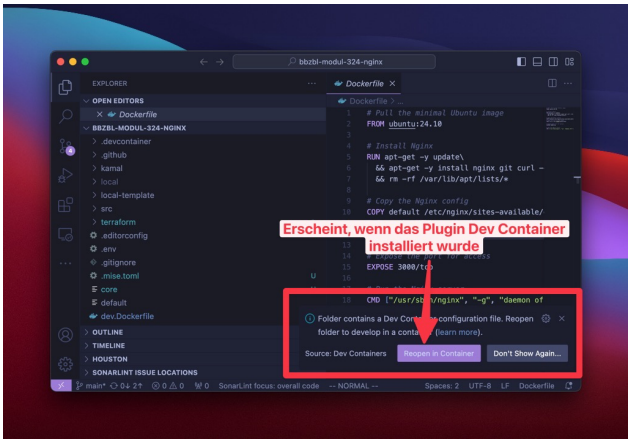
TIPP

- Bitte startet den Devcontainer zuerst mit `docker compose`.
- VS Code hat nicht so tolle Fehlermeldungen wenn was nicht klappt

VORSICHT

- Bitte zuerst alle benötigten VS Code Plugins installieren

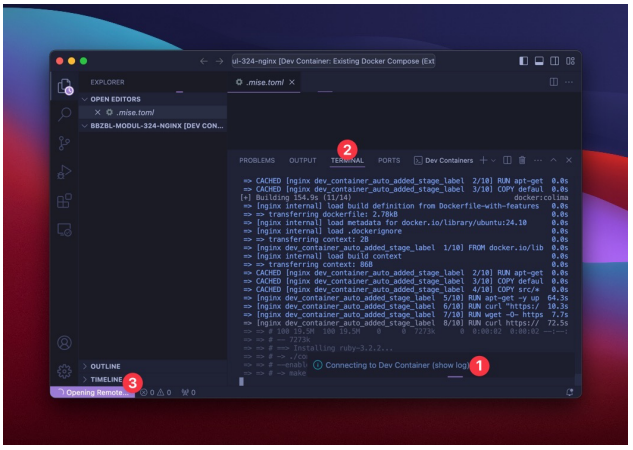
VS Code fragt automatisch nach, ob das Projekt im Container geöffnet werden soll sofern das Plugin "Dev Containers" installiert wurde.



- **"Reopen in Container"** klicken und warten
- Nun wird der Container gebaut und gestartet. **Das kann einige Minuten dauern!**

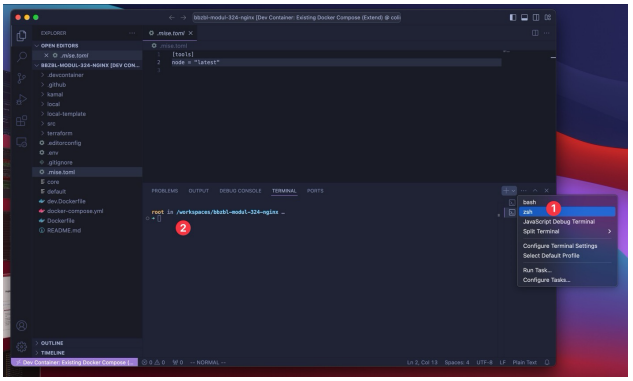
Unterdessen...

1. Wen auf "Connecting to Dev Container (Show Logs)" geklickt wird
2. erscheint folgender Log. Es zeigt wie das "Image" gebaut wird
3. Unten rechts ist ersichtlich ob VS Code in einem Container geöffnet wird/wurde.



Terminal öffnen und nginx starten

1. Mit `+` kann ein neues Terminal geöffnet werden, *(z.B. zsh oder bash)*
2. Nun existiert ein ubuntu Terminal im Container.
   (selbst unter Windows)
3. Die Dateien sind verfügbar unter `/workspace`



4. Nginx starten (wie ohne VS Code)

```
chmod -x nginx/scripts/start-nginx # evt. nicht nötig, schadet aber nicht
sh nginx/scripts/start-nginx
```

5. Prüfen ob der Webserver läuft.

```
curl http://localhost:3000
```

Produktives Dockerfile testen

1. Mit `docker compose` der productive service starten

```
docker compose up production -d
```

2. In der shell oder im Browser prüfen

```
curl http://localhost:3001
```

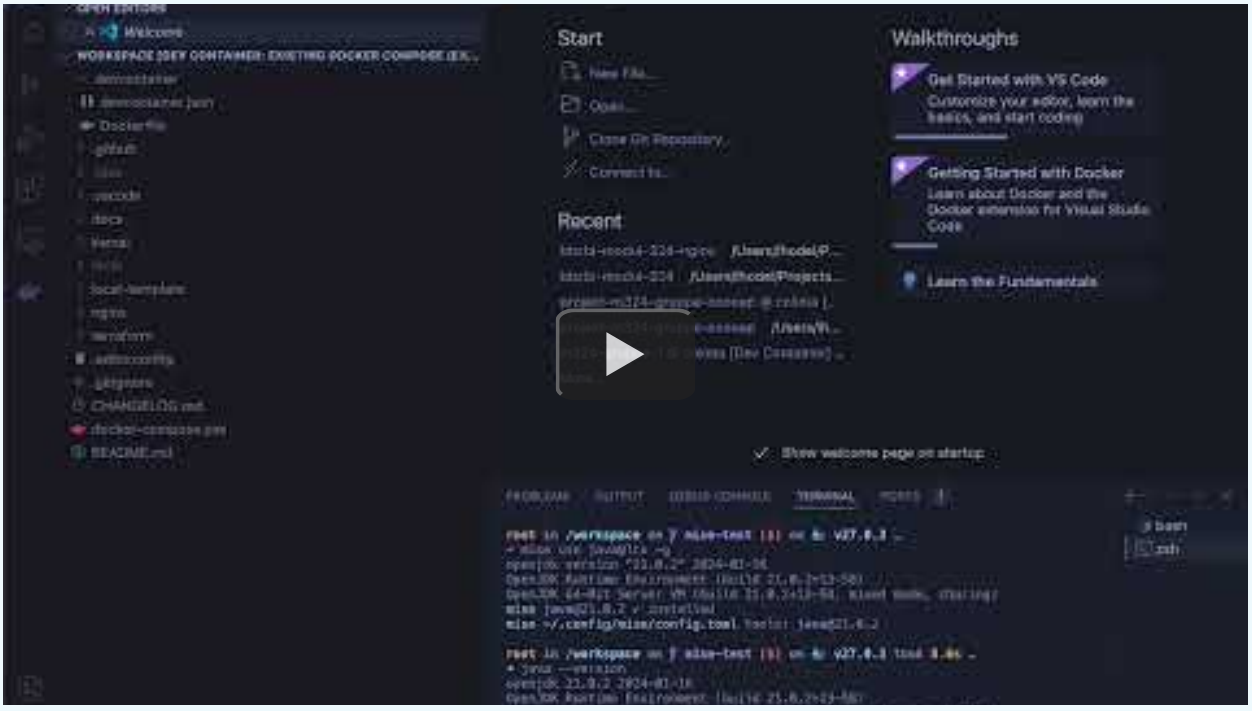
> **VS CODE DEVCONTAINER KANN "DOCKER IN DOCKER"**
>
> - Wenn Ihr ein Terminal im VS Code Devcontainer gestartet habt ist es möglich direkt darin mit `docker compose up production -d` zu starten.
> - Wenn Ihr den Devcontainer via `docker compose` selbst gestartet habt ist kein `docker` verfügbar
>   - kein Problem! Ihr könnt auch auf der Machine, in einer zweiten Shell den Befehl ausführen

Devcontainer anpassen

Der Devcontainer ist standardmässig nur mit Nginx ausgestattet. Es sind noch keine Programmiersprachen installiert. Dies wird für die meisten Projekte nicht reichen.

Es wurde jedoch das Tool `mise` installiert, mit welchem sehr einfach beliebige Sprachen installiert werden können. Mise ist ein universeller Packetmanager für Sprachen, aber auch andere Tools.

<details>
<summary>📺 Video der Aufgabe</summary>



⚠ VORSICHT

- Im Video verwende ich noch den Befehl `~/.local/bin/mise use`
- Neu könnt Ihr auch im Dockerfile `mise use` verwenden!
</details>

Aufgaben

Branch `mise-test` erstellen

Die folgende Aufgabe soll jeder für sich machen und ist nur bedingt Projektbezogen. Um Mergekonflikte zu vermeiden, sollen diese in einem eigenen Branch geschehen. Dieser Branch müsst ihr nicht pushen. Einen Branch erstellt ihr mit folgendem Befehl:

```
git checkout -b mise-test
```

Node installieren

Startet **ein Terminal im devcontainer** und führt folgende Befehle aus

- Zuerst schauen wir ob `node` vorhanden ist (es sollte nicht)

  ```
  node
  > bash: command not found: node
  ```

- Dann installieren wir node lts (long-term supported) global (-g)

  ```
  mise use node@lts -g
  ```

- Nun prüfen wir nochmals ob `node` vorhanden ist (es sollte sein)

  ```
  node
  Welcome to Node.js v22.10.0.
  Type ".help" for more information.
  >
  ```

Java installieren

Startet **ein Terminal im devcontainer** und führt folgende Befehle aus

- Zuerst schauen wir ob `java` vorhanden ist (es sollte nicht)

  ```
  java
  > bash: command not found: java
  ```

- Dann installieren wir java lts (long-term supported) global (-g)

  ```
  mise use java@lts -g
  ```

- Nun prüfen wir nochmals ob `java` vorhanden ist (es sollte sein)

  ```
  java --version
  openjdk 21.0.2 2024-01-16
  OpenJDK Runtime Environment (build 21.0.2+13-58)
  OpenJDK 64-Bit Server VM (build 21.0.2+13-58, mixed mode, sharing)
  ```

> **HINWEIS**
>
> So könnt ihr ziemlich jede noch so exotische Sprache installieren
>
> - wie rust (`mise use rust@stable -g`) oder zig: (`mise use zig@0.13.0 -g`)

Herausfinden was es für Versionen gibt

Mit dem Befehl `mise list-all [sprache] [prefix]` ist es möglich alle verfügbaren Versionen aufzulisten

Zeige alle node Versionen an

```
mise list-all node
0.1.14
0.1.15
...
22.10.0
23.0.0
23.1.0
```

Zeige alle Java "openjdk" Versionen an

```
mise list-all java openjdk
openjdk-9.0.0
openjdk-9.0.1
...
openjdk-23.0.0
openjdk-23.0.1
```

Für alle Programmiersprachen bereitstellen

Um zu garantieren, dass alle im Team die selbe Umgebung haben, müssen die Sprachen, nicht wie oben im Container, **sondern im** `.devcontainer/Dockerfile` installiert werden.

Dabei ist folgendes zu beachten:

- Öffne das Projekt im Devcontainer mit **"Reopen in Container"**
- Öffne die Datei `.devcontainer/Dockerfile`
- Suche folgenden Befehl

  ```
  RUN curl https://mise.run | sh \
  ```

  - Damit wird das cli tool "mise" installiert.

- Um nun damit Programmiersprachen direkt im Image zu installieren, muss dies **danach** passieren.
- Füge nun den folgenden **Befehl danach ins `Dockerfile`** ein. (Er sollte bereits als Kommentar existieren)

```
RUN  mise use node@lts -g
```

  - Damit wird die "long-term supported" Version von Nodejs installiert
- Wenn das `.devcontainer/Dockerfile` geändert wurde, muss der Devcontainer neu gebaut werden. **"Rebuild Container"**

> **TIPP**
>
> - Es macht durchaus Sinn zuerst im Devcontainer zu prüfen was möglich ist und erst danach den Code ins `Dockerfile` zu kopieren.
> - Auch macht es Sinn den devcontainer mit `docker compose build devcontainer` zuerst zu bauen, da VS Code nicht so schöne Fehlermeldungen generiert.

Am Projekt arbeiten

Wenn Ihr die folgenden Aufgaben ausgeführt habt und jetzt wisst, wie Ihr im Devcontainer arbeiten könnt arbeitet Ihr selbständig am Projekt weiter.

- Weiter Issues erstellen und definieren
- Erste Issues lösen

> **TIPP**
>
> Falls ihr bereits so weit seit, dass ihr eure eigene Applikation aufbauen könnt, dürft ihr in der Woche 5 weiter machen.

Am Projekt arbeiten

In diesem Sprint ging es vor allem darum, das Projekt zu finden und mit GitHub Issues zu beschreiben. Dies wird als LB1: Teilnote Arbeitsvorgang bewertet.

> **VORSICHT**
>
> Achten Sie darauf, dass alle Bewertungskriterien eingehalten wurden.

Fachgespräch

Bitte bereiten Sie sich auf das Fachgespräch vor. Die Reihenfolge ist am Beamer ersichtlich.

Ich bitte Sie **rechtzeitig vor dem Zimmer** fürs Fachgespräch zu warten, damit diese ohne grosse Zeitverzögerungen durchgeführt werden können.

- ℹ LB1: Teilnote Fachgespräch

ZWEI AUF EINEN STREICH

- Wenn Sie die schriftliche Reflexion schreiben, bereiten Sie sich direkt fürs Fachgespräch vor ;)

Reflexion Schreiben

Bitte schreiben sie die **schriftliche Reflexion bis zum Ende der 4. Lektion**.

- ℹ LB1: Teilnote schriftliche Reflexion

> **GEFAHR**
>
> - Es ist wichtig, dass Sie sich daran halten wo und wie die Reflexion abgegeben werden muss
>   - LB1: Wo wird die Reflexion geschrieben?
> - Bitte schreiben Sie die Reflexion alleine! Wenn ich zu starke Ähnlichkeit sehe, wird dies als Plagiat gehalten und kann die Note 1 geben!

Eigene Applikation aufsetzen

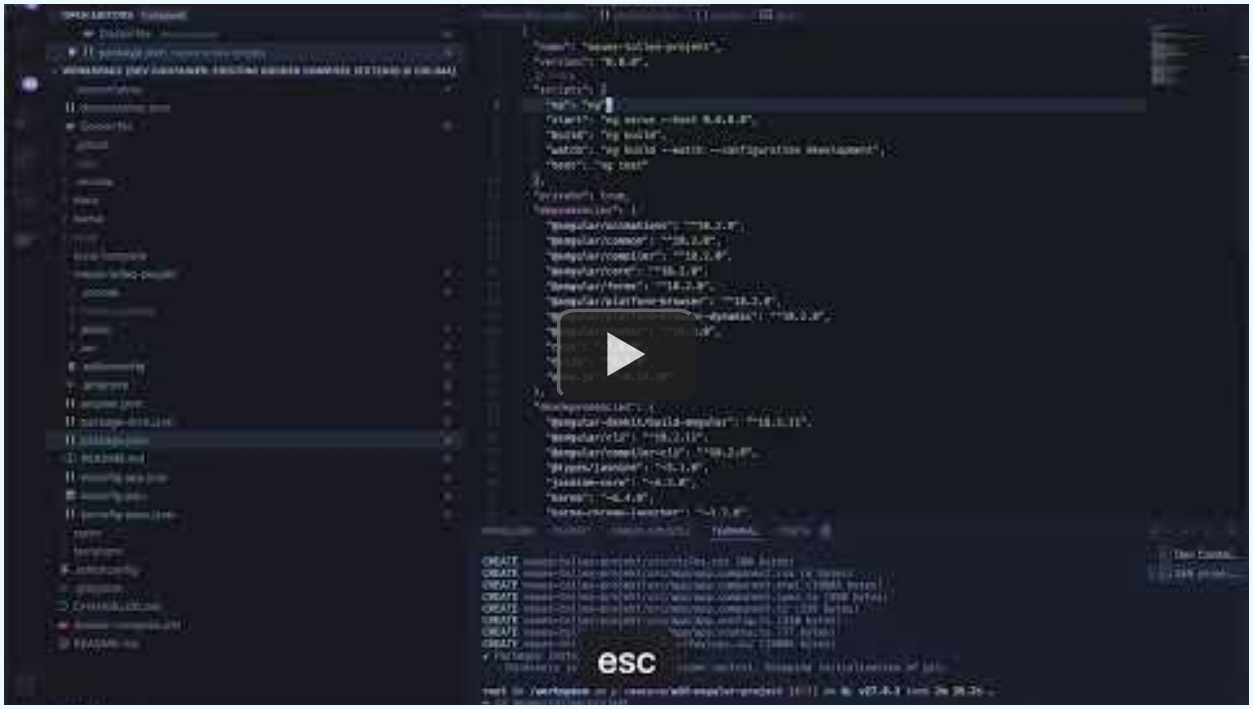
Das Fundament, damit Ihr alle zusammen arbeiten könnt, ist das Aufsetzen, oder "Bootstrappen", der eigenen Applikation. Unten sind für einige gängigen Technologien Beispiele vorhanden. Wählt einfach das aus, dass am besten zu eurem Projekt passt.

- 🎯 Eine `hello-world` App in der gewählten Technologie ist vorhanden
- 🎯 Alle Teammitglieder haben diese lokal gestartet
- 🎯 Die Grundlage ist gelegt, dass Tickets von allen Teammitglieder bearbeitet werden können.

SETZT ALS ERSTES DAS FRONTEND UM!

- 📣 **Full-Stack mit Backend und DB ist zu Zeitaufwändig für dieses Modul.**

📺 Video!



BEI DEN INSTALLIERVORGÄNGEN RUHIG VORSPULEN ;)

Angular

Da erfahrungsgemäss viele Angular verwenden habe besteht dafür ein Beispiel. Ihr dürft jedoch **jede beliebige Technologie** verwenden.

- Installationsanleitung von Angular

neuen Branch erstellen

```
git checkout -b feature/add-angular-project
```

@angular/cli im Devcontainer installieren

Angular braucht das Paket `@angular/cli`, damit der `ng` Befehl verfügbar ist. Um diese für alle zur Verfügung zu stellen, sollte sie direkt im `.devcontainer/Dockerfile` installiert werden.

- öffnet dazu `.devcontainer/Dockerfile`
- und erweitert den Node Install Befehl folgendermassen

```
RUN mise use node@lts -g \
  && mise x -- npm install @angular/cli -g
```

- nun sollte der Befehl `ng` von Angular global im Devcontainer verfügbar sein.

Ein Angular Projekt erstellen

- Öffne das Projekt im Devcontainer mit **"Reopen in Container"**
- Versichert euch, dass Ihr euch im Ordner `/workspace` befindet (`pwd`)
- Erstellt ein neues Angular Projekt mit folgendem Befehl und befolgt die Anleitung.

```
ng new <euer-projekt-name>
```

- wechselt in den neuen Projektordner mit `cd <euer-projekt-name>`

> ⚠ **WICHTG**
>
> - öffnet die Datei `package.json` und sucht nach `ng serve`
>   - Ersetzt den Befehl mit **`ng serve --host 0.0.0.0`**
>   - 💡 Dies ist nötig, sonst geht das Port-Forwarding zum Host nicht richtig. *(hat mich 1h gekostet)*

- Startet den Server mit `npm start`

- Nun erscheint eine Meldung von VS Code dass ein Prozess auf dem Port 4200 gestartet ist.
- Öffnet **http://localhost:4200** im Browser, 🎉

Commiten, pushen und einen PR erstellen

Wenn die Angular App erstellt wurde, sollte der Stand, ohne grosse Änderungen **commited werden**:

- Verwendet dafür eine **Conventional-Commit** Nachricht vom type `feat`
- Schaut dass ihr am eine der Nachricht mit (#issueId) den Issue verlinkt!
  - 💥 Natürlich ist `#iissueId` am Ende eine Nummer! Genau, die vom Issue wie z.B. `#1`

```
git add . # alle Dateien hinzufügen
git commit -m "feat: was auch immer (#issueId)" # Commiten und Issue verlinken
```

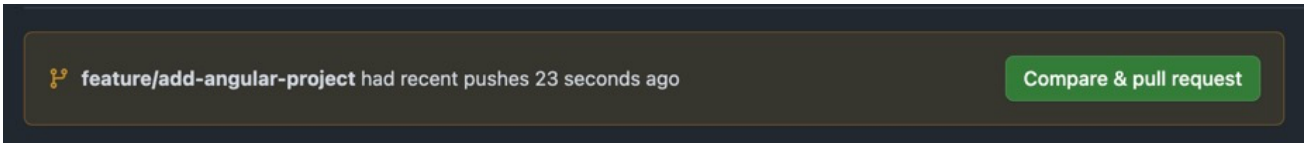
> **KLEINE COMMITS**
>
> - Es ist besser mehrere kleine commits zu machen.
> - Dies ist auch einfacher zum reviewen.
> - Es dürfen mehrere commits auf einen Issue zeigen

Nun könnt Ihr den **Branch nach GitHub pushen**:

```
git push origin feature/add-angular-project # Pushen
```

> **NATÜRLICH DÜRFT IHR AUCH EIN GUI VERWENDEN**

Auf GitHub müsst Ihr nun einen Pull-Request erstellen und diesen vom Team reviewen lassen.



Von allen Teammitglieder lokal testen lassen

Ist der Pull-Request in `main` gemerged, sollen alle Teammitglieder:

1. Den `main` Branch auschecken

```
git checkout main
```

2. Den `main` Branch von GitHub pullen

```
git pull origin main
```

3. Den Devcontainer in VS Code starten
4. Im Devcontainer in den Ordner der App wechseln `

```
cd euer-projekt-name
```

5. Die App starten `npm start` (oder ein alternativ Befehl der gewählten Technologie)
6. Beim VS Code Popup, das **Port-Forwarding freigeben**!

Eine eigene Technologie?

Ihr dürft das Projekt in jeder Beliebigen Technologie erstellen.

> **EINZIGE BEDINGUNG**
>
> - **Alle Teammitglieder müssen mit der Technologie vertraut sein!**

Eine kleine Anleitung was dabei zu beachten ist:

1. Die Programmiersprache sollte per `mise use` direkt im `Dockerfile` installiert werden
2. Sofern weitere global installierte Pakete benötigt werden, müssen die mit `mise x -- <install befehl>` im `Dockerfile` installiert werden.
   - Bei Angular ist das z.B. `mise x -- npm i @angular/cli -g`

> **NPX GEHT DIREKT!**
>
> - Nextjs installiert man z.B. durch `npx create-next-app@latest` dafür muss kein globales Paket installiert werden.

4. Das Projekt muss sich in einem eigenen Ordner befinden. z.B. `/coole-app`.
5. Das `Dockerfile` muss noch nicht angepasst werden. Dies machen wir später.
6. Commiten, pushen und einen PR erstellen
7. Von allen Teammitglieder lokal testen lassen

Produktives Dockerfile

Sofern Ihr noch keine Anpassungen an der CI/CD Pipeline vorgenommen habt, wird immer noch das `nginx` Image gebuildet und nach AWS deployed. Bis jetzt haben wir nur das `.devcontainer/Dockerfile` angepasst. Dieses dient dazu damit alle lokal eine einheitliche Entwicklungsumgebung haben.

Nun geht es darum ein Dockerfile für die eigene Applikation zu erstellen, damit diese nach AWS deployed werden kann.

Datei `docker-compose.yml` anpassen

In der Datei `docker-compose.yml` gibt es unter `service` den Eintrag `production`.

docker-compose.yml

```yaml
services:
  production:
    build:
      context: ./nginx/
      dockerfile: Dockerfile
    ports:
      - "3001:3000"
```

Wie zu sehen ist ist der `services.production.build.context: ./nginx/` gesetzt. Dies bedeutet, dass beim starten des Containers mit dem Befehl `docker compose up production -d` im Ordner `./nginx` nach einem `Dockerfile` gesucht wird. Dieses wird gebaut und gestartet. Dabei wir der Port 3000 nach 3001 umgeleitet, da der Devcontainer bereits den Port 3000 exposed.

Eigene App als context setzten

Um nun die eigene Applikation zu bauen und starten, muss hier den Kontext neu gesetzt werden.

docker-compose.yml

```yaml
services:
  production:
    build:
      context: ./neues-ssg-projekt/
      dockerfile: Dockerfile
    ports:
      - "3001:3000"
```

REGELMÄSSIG BUILDEN

- Während Ihr die folgenden Befehle ins `Dockerfile` schreibt, könnt ihr mit `docker compose build production` regelmässig prüfen ob alles klappt.
- Wenn ihr fertig seit und den Container starten möchtet, geht die mit `docker compose up production`. Dann sollte auf `http://localhost:3001` die eigene App erscheinen.

VORSICHT

Natürlich geht `docker compose build production` nun noch nicht, da zuerst das Dockerfile erstellt werden muss. Mit dieser Einstellung, können wir nun aber bequem das Dockerfile Schrittweise, während dem erstellen builden und testen.

Datei `.dockerignore` erstellen

Die Datei `.dockerignore` hat den gleichen Zweck wie die `.gitignore` Datei. Sie beinhaltet Pfade, die ignoriert werden sollen. Alle Pfade, die in der Datei `.dockerignore` eingetragen sind, werden von dem `COPY` Befehl bei einem `docker build` ignoriert. Sofern Sie ein Angular (oder anders Javascript) Projekt erstellen, wollen Sie bestimmt nicht, dass der Ordner `node_modules` kopiert wird. Dies kann vor allem auf Windows Systemen zu Probleme führen.

- Erstelle eine Datei `.dockerignore` im Projektordner
- Füge mindestens `node_modules` und `Dockerfile` ein

EIN BEISPIEL FÜR EIN ANGULAR PROJEKT

.dockerignore

```
node_modules
Dockerfile
.angular
.vscode
```

Dockerfile erstellen

Nun soll ein Dockerfile erstellt werden, welches die Applikation baut und startet. Dabei soll beachtet werden, dass der Server auf dem Port 3000 gestartet werden soll.

> **VORSICHT**
>
> - Diese Anleitung basiert auf einem Angular SSG Projekt mit dem Namen "neues-ssg-projekt", **dieser muss mit euren Namen ersetzt werden**.
> - Ein Angular app ohne SSG benötigt einen nginx, mehr dazu hier.

- Als erstes erstellt Ihr eine **neue Datei namens "Dockerfile" innerhalb vom Projektordner**.
  - Also nicht im Root vom Repository!
- Nun muss mit dem **Befehl** `FROM` definiert werden, von welchem Base-Image geerbt werden soll.
  - Es gibt für die meisten Programmiersprachen eines.
  - Für node: `FROM node:lts-slim`

> **MISE NICHT NÖTIG**
>
> - Da nun direkt ein Base image einer Sprache verwendet wird, muss diese nicht noch installiert werden.

- Der vom Projekt verwendete deployment Mechanismus **verlangt, dass das Image ein** `LABEL` **"service" besitzt**.
  - `LABEL service="neues-ssg-projekt"`
  - Diese LABEL ist später noch wichtig.
- Nun **definieren wir mit** `WORKDIR` **einen Arbeitsordner** innerhalb vom Image.
  - `WORKDIR /app`
- Als nächstes muss der **Projektcode mit** `COPY` **ins Image kopiert** werden.
  - Da sich das Dockerfile innerhalb vom Projektordner befindet, bedeutet folgender Befehl, dass alles vom aktuellen Ordner ins Image kopiert wird. Ausser Pfade im `.dockerignore`.
  - `COPY . .`
- Sind die Dateien verfügbar, müssen **mit dem** `RUN` **Befehlt die Pakete installiert und die App gebaut werden**. Für Angular (und andere node apps) geht es folgendermassen.
  - `RUN npm ci && npm run build`

<details>
<summary>npm ci error?</summary>

Wenn folgender Error erscheint, müssen Sie zuerst mit `npm install` im Devcontainer die Packages installieren und das `package-lock.json` commiten!

> npm error `npm ci` can only install packages when your package.json and package-lock.json or npm-shrinkwrap.json are in sync. Please update your lock file with `npm install` before continuing.

</details>

- Da die App auf dem Port 3000 laufen muss, müssen wir dies definieren. Unter node geht dies durch die Umgebungsvariable `PORT` die per `ENV` Befehl gesetzt werden kann.
  - `ENV PORT=3000`

- Zudem muss dem Image der Port 3000 auf freigegeben werden. Dies geschieht
  - `EXPOSE 3000/tcp` durch den `EXPOSE` Befehl.
- Wurde die App gebaut und der PORT 3000 ist parat, kann nun **mit dem `CMD` Befehl der Server auf PORT 3000 gestartet werden**
  - `CMD [ "node", "/app/dist/neues-ssg-projekt/server/server.mjs" ]`

Beispiel für eine Angular App mit SSG

Beispiel Angular Dockerile (unoptimized)

```
FROM node:lts-slim

LABEL service="neues-ssg-projekt"

WORKDIR /app

COPY . .

RUN npm ci && npm run build

ENV PORT=3000
EXPOSE 3000/tcp

CMD [ "node", "/app/dist/neues-ssg-projekt/server/server.mjs" ]
```

⚠ NOCH NICHT READY ZUM DEPLOYEN AUF AWS

- Die App wird zwar auf port 3000 geserved. Die angular app braucht aber noch eine route `/up` die ein HTTP code 200 zurück gibt!
- Auch sollte das Dockerfile wie folgt erläutert noch optimiert werden.

Dockerfile optimieren mit Multistage

Grundsätzlich ist das obere Dockerfile funktionsfähig. Da jedoch die Node App darin gebaut wurde und die Pakete installiert wurden, beinhaltet das Image unnötigen balast. Grundsätzlich sollte ein Dockerimage so klein wie möglich sein.

Um dies zu erreichen, gibt es **Multistage Dockerfiles**. Ein Multistage Dockerfile besitzt zwei Dockerfile Definitionen untereinander, wobei die erste der "builder" ist und nach dem build "weggeworfen" wird. Es wird nur das zweite Image verwendet.

Beispiel für eine Angular App mit SSG

Multistage Image

```
# Mit AS builder geben wir dem Image eine Namen um darauf zuzugreifen
FROM node:lts-slim AS builder

WORKDIR /app

COPY . .

RUN npm ci && npm run build

# Ab hier beginnt das Produktive Image!
FROM node:lts-slim

# Das LABEL muss hier gesetzt sein!
LABEL service="neues-ssg-projekt"

WORKDIR /app

# Hier kopieren wir nur den gebauten Code vom builder Image
COPY --from=builder /app/dist/neues-ssg-projekt/ .

ENV PORT=3000
EXPOSE 3000/tcp

CMD [ "node", "/app/server/server.mjs" ]
```

⚠ NOCH NICHT READY ZUM DEPLOYEN AUF AWS

- Nun ist das Dockerfile zwar optimiert. Die Route `/up` fehlt jedoch immer noch!

Beispiel für eine Angular SPA App, ohne SSG mit nginx geserved

Für alle die eine klassische SPA App entwickeln, welche nicht über einen node Server verfügen (SSG), müssen mit einem dedizierten Webserver geserved werden. In diesem Modul nehmen wir dafür einen Nginx.

Der Builder ist in diesem falle gleich. Das zweite Image ist jedoch einen Nginx. Auch wird dann die gebaute App zum nginx kopiert.

```
# Mit AS builder geben wir dem Image eine Namen um darauf zuzugreifen
FROM node:lts-slim AS builder

WORKDIR /app

COPY . .

RUN npm ci && npm run build

# Ab hier beginnt das Produktive Image!
FROM nginx

LABEL service="neues-spa-projekt"

# Hier kopieren wir nur den gebauten Code vom builder Image
COPY --from=builder /app/dist/neues-spa-projekt/browser /usr/share/nginx/html
# Hier kopieren wir die nginx Konfigurationsdatei (Ihr könnt diese vom nginx service kopieren!)
COPY default /etc/nginx/sites-available/default

EXPOSE 3000/tcp

# Eine standard nginx Konfigurationsdatei um eine SPA auf Port 3000 zu serven
CMD ["/usr/sbin/nginx", "-g", "daemon off;"]
```

Die dafür passende nginx *default* Konfigurationsdatei

default

```
server {
    listen 3000 default_server;
    listen [::]:3000 default_server;

    root /usr/share/nginx/html;
    index index.html index.htm;

    server_name _;
    location / {
        try_files $uri $uri/ = 404;
    }

    location /up {
        return 200;
    }
}
```

BEREITS READY ZUM DEPLOYEN AUF AWS

- Da der nginx mit der oberen "default" Konfigurationsdatei eine `location /up` definiert, kann dieses Image bereits deployed werden.

Ready to Deploy

Nun sollten wir soweit sein, dass mit `docker compose up produciton -d` jedes Teammitglied lokal die produktive App laufen lassen kann. Damit kann zwar nicht Entwickeln werden, da es immer neu gebaut werden muss, es hilft jedoch lokal testen zu können wie die produktive Applikation startet.

Deployment Konfigurationsdateien anpassen

Um das Image nach AWS deployen zu können braucht es nun noch folgende Anpassungen:

- Die Kamal config `./kamal/config/deploy.yml` muss als **service** den gleichen Namen besitzen wie das **LABEL** im Dockerfile.
  - Im Beispiel lautet dies `service: neues-ssg-projekt`

/kamal/config/deploy.yml

```
# INFO: muss gleich sein zum Label "service" des zu deployende Dockerfile
#       siehe: `../../nginx/Dockerfile`
- service: nginx
+ service: neues-ssg-projekt
...
```

- Der GitHub Action Workflow `./github/workflows/deploy.yml` muss das richtige Dockerfile builden und pushen
  - Sucht nach dem step **Build and push nginx Image**
  - ändert hier `context: ./nginx` nach `context: ./neues-ssg-projket`

.github/workflows/deploy.yml

```
...
      - name: Build and push nginx Image
        uses: docker/build-push-action@v6
        with:
-         context: ./nginx
+         context: ./neues-ssg-projekt
          push: true
          tags: ${{ steps.meta.outputs.tags }}
          cache-from: type=gha
          cache-to: type=gha,mode=max
...
```

Helthcheck Route `/up` erstellen

> **SPA APPS DIE MIT NGINX GESERVED WERDEN KÖNNEND DIESEN SCHRITT AUSLASSEN**

Auf AWS existiert ein reverse proxy, welcher die Route `/up` prüft. Diese ermöglicht ein sogenanntes zero-downtime deployment. Damit dies jedoch klappt muss die Route auch existieren.

Damit der node Server die Route `/up` kennt, muss die Datei `server.ts` erweitert werden:

- Öffnet die Datei `server.ts` im Projektordner
  - findet Ihr sie nicht, habt ihr ziemlich sicher nicht die SSG option gewählt.
- Sucht darin nach `server.set('views'`
- Nach diesem Befehl registrieren wir nun eine GET Route `/up` grün dargestellt.

/neues-ssg-projekt/server.ts

```
...
server.set('views', browserDistFolder);

// Health check route
server.get('/up', (req, res) => {
  res.status(200).send('OK');
});

// Example Express Rest API endpoints
// server.get('/api/**', (req, res) => { });
// Serve static files from /browser
server.get(
  '**',
  express.static(browserDistFolder, {
    maxAge: '1y',
    index: 'index.html',
  }),
);
...
```

> **VORSICHT**
>
> Die Route `/up` muss vor der Wildcard Route `**` gesetzt werden!

Auf AWS deployen

Wenn ihr nun diese Änderungen via Pull-Request in `main` merged, wird automatisch ein deployment nach AWS durchgeführt.

> **CREDENTIALS!**
>
> Achtet darauf, dass die Credentials immer wieder übertragen werden müssen!

Optional: Pull-Requests deployen

Momentan ist im `.github/workflows/deploy.yml` definiert, dass die Action nur ausgeführt wird, wenn in den `main` branch gepushed (merged) wird. Dies ist schade, da man ja nicht weis, ob das feature funktioniert.

Events, die 'Deploy to Amazon AWS' triggern

```yaml
on:
  workflow_dispatch:
  push:
    branches: ["main"]
```

Möchte man nun, dass die Action auch bei einem Pull-Request getriggert wird, kann dies folgendermassen eingestellt werden:

on.pull_request triggert nun auch die Aciton

```yaml
on:
  workflow_dispatch:
   pull_request:
  push:
    branches: ["main"]
```

> **DEPLOYMENT NACH AWS**
>
> Dies deployed ein feature branch nach AWS **und kann somit die App kaputt machen**, dies ist in diesem LAB O.K. in einem Produktiven System wäre dies fahrlässig!

Optional: Ein Test und Produktiv System aufbauen

Die GitHub Action `.github/workflows/deploy.yml` deployed automatisch nach AWS. Dafür verwendet sie die Credentials, welche Ihr eingegeben habt.

Ihre Gruppe besitzt jedoch vier AWS Umgebungen, für jedes Teammitglied eine. Ihr könntet somit **zusätzliche Credentials mit einem Prefix "TST_"** für eine Testumgebung anlegen.

- **TST**_AWS_ACCESS_KEY_ID, **TST**_AWS_SECRET_ACCESS_KEY, **TST**_AWS_SESSION_TOKEN, **TST**_AWS_SSH_PRIVATE_KEY, **TST**_AWS_ACCOUNT_ID

Danach die Datei `.github/workflows/deploy.yml` nach `.github/workflows/deploy-tst.yml` **kopieren** und in der Kopie folgende Änderungen vornehmen:

- Den Namen nach "**Deploy to Amazon AWS Test**" ändern
- Die Secrets **Umbenennen** damit sie mit **TST_** anfangen
- `on.pull_request` hinzufügen damit beim PR deployed wird
- `on.push.branches: ["main"]` entfernen damit nicht doppelt deployed wird

Nur noch beim pull_request triggern

```
on:
  workflow_dispatch:
+ pull_request:
- push:
-   branches: ["main"]
```

Test Infrastruktur aufsetzen

Das Test Deployment wird fehlschlagen, da das Testsystem auf Amazon noch nicht erstellt wurde. Dies kann gemacht werden, indem auch der Workflow `.github/workflows/aws-infrstructure.yml` nach `.github/workflows/aws-infrstructure-tst.yml` kopiert wird.

Auch diesem müssen die Credentials ersetzt werden. Dann kann man Ihn manuell triggern. Sobald dies geschehen ist, sollte das Test Deployment ebenfalls gehen. 😄

Und tataa, hat man fast ein Test und ein Produktiv System! 🎉

> **TIPP**
>
> - Hier wird nun ersichtlich wie mächtig Automatisierung sein kann! Ein Testsystem *for-free*, indem man nur zwei Dateien kopiert und leicht anpasst!

Automatisiert Testen

Ein Punkt im DevOps Modell ist die *Qualitätssicherung*. Qualitätssicherung soll sicher stellen, dass das Produkt auch den Erwartungen entspricht. Dies kann funktional sein: "Tut das Programm was es soll", aber auch qualitativ: "tut es dies robust und fehlerfrei?".

Die funktionale Erwartung soll durch das Einbinden des Kunden und schnellem Feedback der umgesetzten Features gesichert werden. Dies kann schlecht automatisiert geschehen. Die Fehlerfreiheit jedoch ist zu einem gewissen Grad automatisiert testbar.

Arten von automatisierten Tests sind *Unit Tests* und *Integration Tests*

Unit-Test

Unit Tests beziehen sich auf einzelne Methoden. Mit Ihnen soll gewährleistet werden, dass eine Methode korrekte Resultate liefert. Dafür wird **die Methode mit verschiedensten Argumenten aufgerufen und geprüft ob das Resultat stimmt**.

Gängige Unit-Test Frameworks sind:

- Java: JUnit
- Node:
  - Jasmine (out-of-the-box von Angular verwendet, trotzdem deprecated)
    - Karma (out-of-the-box von Angular verwendet, trotzdem deprecated)
    - Karma führt die Unit-Tests im Browser aus, ist aber kein Integration Testing framework.
  - Jest (wird neu von Angular empfohlen)
    - Web-test-Runner (wird neu von Angular empfohlen)
    - Web-test-Runner führt die Unit-Tests im Browser aus, ist aber kein Integration Testing framework.
  - MochaJS

Integration-Test

Integration Tests testen das Zusammenspiel von Methoden. Dafür wird oft ein "headless" Browser (playwright, puppeteer) verwendet, mit welchem die **Benutzerinteraktion nachempfunden** wird. So kann zum Beispiel getestet werden, ob nach dem Einloggen die Applikation direkt zu einer bestimmten Seite weitergeleitet wird.

Gängige Integration Testing Frameworks sind:

- Selenium
- Gauge / Taiko

- Cypress
- Cucumber (die Erfinder...)

> **TIPP**
>
> Integration Tests sind meiner Meinung nach Unit-Tests in vielen Bereichen überlegen:
>
> - Sie testen Benutzerinteraktion und sind daher nicht so theoretisch.
> - Sie sind Grobmaschiger, was ein Refactoring nicht behindert.
>   - Viele Unit-Tests können das Refactoring hindern, da viele Test angefasst werden müssen, was Arbeit bedeutet ;)

> **SPRACHUNABHÄNGIGE INTEGRATION-TESTS**
>
> Da Integration-Tests einen user in einem Browser simulieren, müssen Integration-Tests nicht in der Sprache geschrieben werden, in der auch die Applikation programmiert wurde.

Aber Achtung!

> Program testing can be used to show the presence of bugs, but never to show their absence!
>
> -- Edsger Dijkstra

Das bedeutet soviel wie: man kann gar nicht immer Wissen ob ein Programm Fehlerfrei ist, wenn man dem Fehler noch nicht begegnet ist. Dies ist kein Grund nicht zu testen, sondern eine Mahnung sich bewusst zu sein, dass auch 100% getesteter Code, fehlerhaft sein kann.

Wozu dienen Tests wenn keine Garantie?

- Beim Schreiben von Tests befasst man sich tiefer mit der implementierten Methode. Man Reflektiert somit auch das eigene Schaffen.
- Beim Refactoring kann sichergestellt werden, dass die Methode sich noch gleich verhaltet!
- Test dienen als Beschreibung und sind somit direkt **lebende Dokumentation** auf Englisch "living documentaion".
  - Dazu dienen vor allem Integration Tests, geschrieben in Gherkin wie Cucumber es definiert hat und viele sich davon inspirieren liessen. (optional)

Testen mit Angular

Angular besitzt den command `ng test`. Sofern nichts anderes definiert, wird automatisch "jasmin" mit "karma" verwendet und verweise auf **die offizielle Anleitung**.

- Versucht nun nach der Anleitung einige Tests für euer Projekt zu erstellen.

- In diesem Modul werden **nur Unit Tests** geschrieben. Ich lade Sie jedoch dazu ein sich mit Integration Tests auseinander zu setzen.

> **JASIME/KARMA SIND VERALTET**
>
> Angular selbst will in Zukunft auf Jest mit Web-Test-Runner wechseln. Dies, da Karma und Jasmin doch langsam in die Jahre gekommen sind. Wer also lust hat, darf das Projekt so anpassen, dass Jest und Web-Test-Runner verwendet wird.
>
> Ich habe dazu zwei Anleitungen gefunden:
>
> - https://alfredo-perez.dev/setup-jest-in-angular-18
> - https://danywalls.com/testing-in-angular-replace-karma-to-web-test-runner

> **ANGULAR.IO -> ANGULAR.DEV**
>
> Mir ist aufgefallen, dass die alte Domain angular.io nur bis Version 17 abdeckt.
>
> - **Neuere Angular Versionen verwenden angular.dev**!

Karma Konfiguration

Um Karma Browser Tests im Dockerfile laufen zu lassen, braucht es die Option einen sogenannten "Headless Browser" zu verwenden. Am gängigsten ist der ChromeHeadless. Damit dieser von Karma verwendet wird, muss folgende Konfiguration erstellt werden.

Neuer Branch erstellen

Wie immer, wenn man ein neues Feature entwickelt sollte dies in einem neuen Branch geschehen.

`git checkout -b feat/karma-config`

Karma config generieren

Als erstes muss die Karma configuration generiert werden, damit diese angepasst werden kann. Ohne diesen Schritt wird automatisch immer die Standardkonfiguration verwendet. Nach dem folgenden Befehl sollte die Datei `./karma.conf.js` zur Verfügung stehen.

```
ng generate config karma
```

ChromeHeadlessNoSandbox launcher hinzufügen

Als nächstes muss ein neuer `customLaunchers` erstellt werden. Damit wird Karma gesagt, wie genau der Browser gestartet werden soll. Es gibt bereits den "ChromeHeadless", welcher kein GUI Browser öffnet, sondern ein Browser als CLI Befehl verwendet.

Da wir jedoch im Dockerfile arbeiten, und der ChromeHeadless root access braucht, müssen wir noch den Parameter `--no-sandbox` hinzufügen. Wieso jetzt genau ist nicht so wichtig, kopiert einfach den grünen Teil in der Datei `karma.conf.js` direkt nach dem Key `browsers`. Dieser launcher heisst nun `ChromeHeadlessNoSandbox`.

karma.conf.js

```
//...
    browsers: ['Chrome'],
    customLaunchers: {
      ChromeHeadlessNoSandbox: {
        base: 'ChromeHeadless',
        flags: ['--no-sandbox']
      }
    },
    restartOnFileChange: true
//...
```

Script `test:ci` im package.json definieren

Als letztes muss in der Datei `package.json`, unter `scripts` ein neues script namens `test:ci` hinzugefügt werden. Dieses startet die tests mit dem oben erstellten "Laucher" `ChromeHeadlessNoSandbox`. Auch wird mitgeteilt, dass kein Ladebalken ausgegeben werden soll und auch nicht auf Dateiänderungen gehört werden soll.

Der Befehl `npm run test:ci` wird später verwendet um die Tests in der CI Pipeline laufen zu lassen.

package.json

```
  "scripts": {
    "//": "..."
    "test": "ng test",
    "test:ci": "ng test --no-watch --no-progress --browsers=ChromeHeadlessNoSandbox",
    "//": "..."
  },
```

Testen in der GitHub Action

Besitzt die Applikation Unit-Test, macht es nur Sinn, dass diese auch automatisiert ausgeführt werden. Es gibt nun **zwei Orte** an denen die Tests ausgeführt werden können.

Behilflich ist uns nun das script `npm run test:ci` welches in der vorherigen Anleitung erstellt wurde.

Dedizierter Test-Job im `deploy.yml`

Die gängige Art ist es, einen dedizierten Test-Job direkt in das bestehende `deploy.yml` einzufügen. Dafür muss in der GitHub Action node und chrome installiert werden.

> **HINWEIS**
>
> Das untere Beispiel sollte für eine Angular Applikation direkt funktionieren. Sie müssen Einzig den Pfad `ihr-projekt/` jeweils anpassen.

Damit der Deploy-Job auf den Test-Job wartet, muss dieser den test job als Abhängigkeit besitzen. Dies geschieht mit dem key `needs: test`

.github/workflows/deploy.yml

```yaml
name: Deploy to Amazon AWS
# ...
jobs:
  test:
    name: Test
    runs-on: ubuntu-latest
    steps:
      - name: Install headless chrome
        uses: browser-actions/setup-chrome@v1
        with:
          chrome-version: 120
          install-chromedriver: true
      - uses: actions/checkout@v4
      - name: Install node
        uses: actions/setup-node@v4
        with:
          node-version: 22 # lts
          cache: "npm"
          cache-dependency-path: ihr-projekt/package-lock.json
      - name: Test
        working-directory: ihr-projekt
        run: npm ci && npm run test:ci
  deploy:
    needs: test # wartet auf den Test-Job
    # ...
```

Dies alles hat zur Folge, dass die Action nicht mehr so schnell ist.

Es gibt aber auch Vorteile:

1. Der Test-Job wird vor dem Deploy-Job ausgeführt. Dadurch muss nicht nach AWS eingelogged werden und kein Dockerfile gebaut werden, sofern der Test fehlschlägt.
2. Es ist klar ersichtlich, ob der Build wegen einem Test fehlgeschlagen ist oder doch einem anderen Problem.
3. Die Testresultate können als Artefakte an den Job Angeheftet werden. (optional)
   - https://www.npmjs.com/package/karma-jasmine-html-reporter
   - https://github.com/actions/upload-artifact

> **HINWEIS**
>
> - 🚀 GitHub Actions Cache ist super schnell, cached immer, wenn möglich!

Direkt im Dockerfile *(Altervative)*

Sofern Sie bereits die Applikation mit einem "Multistage Dockerfile" builden, könnten die Tests auch direkt vor dem Build, mit dem Befehl `npm run test:ci` ausgeführt werden.

> **HINWEIS**
>
> - Chromium nur im `builder` installieren. Das produktive Image braucht Google Chrome nicht.

Damit diese funktionieren, muss zuerst ein HeadlessChrome im Dockerfile installiert werden. Das untere Script sollte bei jedem (auch Apple Silicon) funktionieren.

Dockerfile

```
FROM node:lts-slim AS builder

# ...

# Install Chromium
# (Use Chromium from "Playwright" instead of Puppeteer, for getting ARM64 build,
which is not provided by Puppeteer)
RUN mkdir -p /tmp/cli-chromium && \
  cd /tmp/cli-chromium && \
  npm i playwright@latest && \
  PLAYWRIGHT_BROWSERS_PATH=/usr/local/bin/pw-browsers npx playwright install --with-
deps chromium && \
  ln -s $(find /usr/local/bin/pw-browsers -name "chrome" -executable | head -n 1)
/usr/local/bin/chrome && \
  rm -rf /tmp/cli-chromium
ENV CHROME_BIN="/usr/local/bin/chrome"

# ...

RUN npm ci && npm run build
RUN npm ci && npm run test:ci && npm run build

# ...

# Ab hier beginnt das produktive Image!
FROM node:lts-slim

# ...
```

Der Vorteil an dieser Art ist, dass die App nur einmal gebaut werden muss.

Es gibt jedoch auch Nachteile!

1. Dadurch, dass der Test im Dockerfile gemacht wird, ist es nicht direkt klar, dass der Build fehlgeschlagen ist, weil ein Test fehlgeschlagen ist. Es könnte auch der Build an sich

fehlschlagen.

2. Das Installieren vom Chrome im Dockerfile ist komplex und dauert lange.
3. GitHub Actions erlauben es sogenannte Artefakte zu speichern. Dies könnten z.B. die Testresultate sein. Wird der Tests im Dockerfile ausgeführt können diese nicht (oder nur sehr schwer) als Artefakte gespeichert werden.

TIPP

- Daher ist es gängiger, dafür einen eigenen "Job" in der GitHub Action einzubauen.

Am Projekt arbeiten

In diesem Sprint ging es darum am Projekt aktiv zu arbeiten, Dockerfiles zu erstellen, Tests zu schreiben und die CI/CD Pipeline nach AWS aufzusetzen. Dies wird als LB2: Teilnote Arbeitsvorgang bewertet.

> **VORSICHT**
>
> Achten Sie darauf, dass alle Bewertungskriterien eingehalten wurden.
>
> - **Die sind anders als bei der LB1!**

Fachgespräch

Bitte bereiten Sie sich auf das Fachgespräch vor. Die Reihenfolge ist am Beamer ersichtlich.

Ich bitte Sie **rechtzeitig vor dem Zimmer** fürs Fachgespräch zu warten, damit diese ohne grosse Zeitverzögerungen durchgeführt werden können.

- ℹ LB2: Teilnote Fachgespräch

ZWEI AUF EINEN STREICH

- Wenn Sie die schriftliche Reflexion schreiben, bereiten Sie sich direkt fürs Fachgespräch vor ;)

Reflexion Schreiben

Bitte schreiben sie die **schriftliche Reflexion bis zum Ende der 4. Lektion**.

- ℹ LB2: Teilnote schriftliche Reflexion

> **GEFAHR**
>
> - Es ist wichtig, dass Sie sich daran halten wo und wie die Reflexion abgegeben werden muss
>   - LB2: Wo wird die Reflexion geschrieben?
> - Bitte schreiben Sie die Reflexion alleine! Wenn ich zu starke Ähnlichkeit sehe, wird dies als Plagiat gehalten und kann die Note 1 geben!

DevOps

DevOps Best Practices

Präsentation

Im Browser öffnen | PDF Speichern

Agiles Projektmanagement

Handlungsziel

1. Erfasst und verwaltet die Anforderungen und **Umsetzungsschritte nachvollziehbar** für die Entwicklung im Team.
   i. Kennt den Nutzen bezüglich **kontinuierlicher toolunterstützter Entwicklung** und Wartung (z.B. MVP, Kundenfeedback, Kosten/Nutzen, Qualität, -Risikoreduktion).
   ii. Kennt **Vorgehensweisen zur Verwaltung von Anforderungen** (z.B. Storys, Issues, Akzeptanzkriterien etc.).
   iii. Kennt Vorgehensweisen zur nachvollziehbaren Entwicklung im Team (z.B. **Verknüpfung Commit mit Story, Pull Request/Peer-Review**).

Projektdefinition

> Vorhaben, das im Wesentlichen durch die Einmaligkeit aber auch Konstante der Bedingungen in ihrer Gesamtheit gekennzeichnet ist, wie z.B. Zielvorgabe, zeitliche, finanzielle, personelle und andere Begrenzungen; Abgrenzung gegenüber anderen Vorhaben; projektspezifische Organisation.
>
> -- DIN 69901

Agiles Manifest

> Wir erschliessen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen. Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:
>
> - **Individuen und Interaktionen** mehr als Prozesse und Werkzeuge.
> - **Funktionierende Software** mehr als umfassende Dokumentation.
> - **Zusammenarbeit mit dem Kunden** mehr als Vertragsverhandlung.
> - **Reagieren auf Veränderung** mehr als das Befolgen eines Plans.
>
> Das heisst, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein.
>
> -- agilemanifesto

Das Vorgehensmodell Kanban

Im agilen Projektmanagement haben sich die Vorgehensmodelle *SCRUM* und *Kanban* durchgesetzt. Beiden Modelle verwenden eine **iterative Vorgehensweise**. Anders als in klassischen Vorgehensmodelle wie dem Wasserfall-Model werden die **Arbeitspakete fortwährend erstellt und angepasst**.

Wo *SCRUM* einer klar definiertem Struktur mit vordefinierten Zeremonien folgt (Daily, Refinement, Review, Retrospektive) ist *Kanban* einfacher/flexibler gehalten.

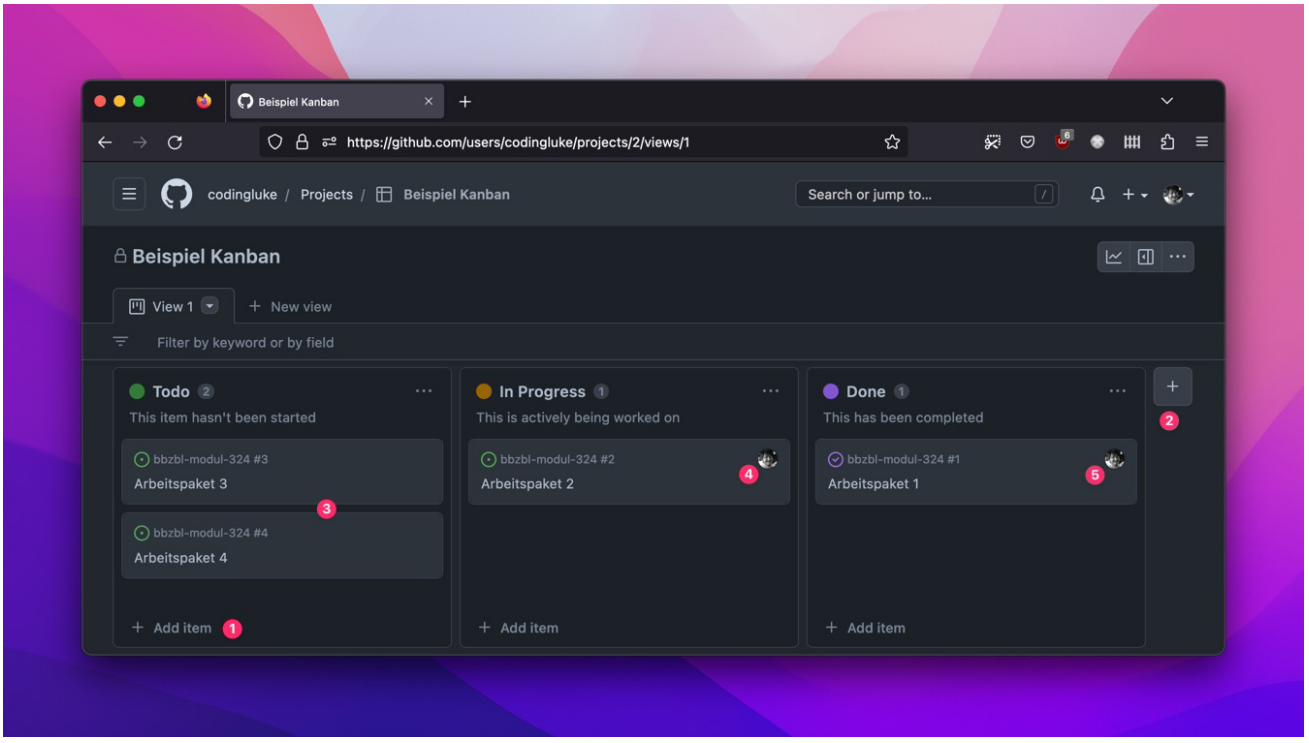
In *Kanban* werden die Arbeitspakete in verschiedene **Status** eingeteilt. Standardmässig heissen diese:

- **To do:** Arbeitspakete, welche bereits definiert wurden, jedoch noch nicht bearbeitet werden
- **In Process:** Arbeitspakete, welche momentan gerade bearbeitet werden
  - immer nur 1 Arbeitspaket pro Person
- **Done:** Arbeitspakete welche erfolgreich umgesetzt wurden
- **Waiting:** Für Arbeitspakete, welche "In Process" sind, jedoch auf externe Einflüsse gewartet wird darf ein optionaler Status *Waiting* eingeführt werden. (z.B. Warten auf eine Unterschrift, Infrastruktur usw.)

Da das Modul 324 nicht Projektmanagement im Fokus hat, werden wir für das Projekt Kanban verwenden.

> **PROTIP**
>
> *Kanban* eignet sich auch für kleine Privatprojekte, oder sogar zum Planen vom eigenen Tagesablauf.

Einfaches GitHub Kanban Board



1. Mit Add item können direkt neue Arbeitspakete (*Issues*) erstellt werden.
2. Mit dem rechten + können neue Status hinzugefügt werden (z.B. *Waiting*).

3. Die *Arbeitspakete 3* und *4* wurden noch nicht begonnen.
4. Das *Arbeitspaket 2* ist begonnen und es wurde eine Person zugeteilt.
5. Das *Arbeitspaket 1* wurde bereits fertiggestellt.

Arbeitspakete (GitHub Issues)

Die Arbeitspakete beschreiben die Aufgaben, welche erledigt werden müssen, um das Projekt erfolgreich umzusetzen.

Sie werden je nach Software und Vorgehensmodell auch Issues, Stories, Cards, Tickets usw. genannt und:

- müssen eine **klare "Definition of Done"** (Akzeptanzkriterien) besitzen.
- dürfen **nur eine Aufgabe** beschreiben.
  - Ein "*und*" im Titel ist ein Hinweis darauf, dass ein Arbeitspaket aufgesplittet werden sollte.
- müssen **auf den Code verweisen** (Branch, Pull-Request), welcher die Beschreibung implementiert.
  - Natürlich geht dies erst wenn das Paket "In Progress" ist und es sich durch Code lösen lässt.
- müssen vorhandene **Abhängigkeiten** aufzeigen.
  - Es gibt auch Arbeitspakete die isoliert, also unabhängig implementiert werden können.

Definition of Done (Akzeptanzkriterien)

Die Akzeptanzkriterien geben vor, **wie geprüft werden kann, ob das Arbeitspaket korrekt umgesetzt wurde**. Grundsätzlich kann zwischen zwei Arbeitspaket-Typen unterschieden werden, *formelle Tätigkeiten* und *programmierbare Funktionalitäten*.

1. Administrativen Tätigkeiten

Definieren der erwarteten Artefakte** (z.B. GUI Skizzen, Dokumente usw.) und wo diese gefunden werden können:

- Beschaffung von Infrastruktur
- Beschaffung von Lizenzen
- Meetings mit Kunden
- Ausarbeiten der Arbeitspakete
- ...

2. Programmierbare Funktionalitäten

Definieren der erwarteten Funktionalität, **wo sie gefunden und getestet werden kann**:

- Jegliche Software Funktionalität die umgesetzt werden soll
- *DevOps* Automatisierungen

- *Infrastruktur as Code*
- Alles was in Code resultiert
- ...

WICHTIG

Ein Arbeitspaket ist erst dann **"Done"** wenn die **Akzeptanzkriterien** auf dem **Testsystem** erfolgreich durchgeführt wurden.

SCHNELLES KUNDENFEEDBACK

Wenn das Projekt im *DevOps* Model mit einer CI/CD Pipeline umgesetzt wird, können die abgeschlossenen Arbeitspakete direkt im Testsystem überprüft werden, da neue Funktionen *(Features)* automatisiert online gestellt *(deployed)* werden.

- Optimalerweise kann dies direkt vom Kunde übernommen werden.

AUTOMATISIERTE INTEGRATIONS-TESTS

Automatisierter **Integrations Test** durch ein Testing Framework wie Puperteer, Playwright, Selenium **spart Zeit und garantiert das Funktionieren auf Zeit**

- Geht nur für Web-Applikationen
- Vergleichbar mit einem Suchmaschinen-Bot

Eine klare Aufgabe pro Arbeitspaket

Jede komplexe Aufgabe besteht aus vielen einfacheren Unteraufgaben. Dabei gibt es häufig Unteraufgaben, die schneller und solche die weniger schnell gelöst werden können.

Wenn nun eine komplexe Aufgabe in einem Arbeitspaket beschreiben wird, werden diese meistens sowieso im Arbeitspaket nacheinander gelöst. Dies führt dazu, dass Teilbereiche der Aufgabe bereits fertiggestellt sind, jedoch noch nicht abgeschlossen werden können, da sie von der komplexeren Aufgabe blockiert werden.

Vorteil von kleinen Arbeitspakete

- Sie sind **schneller gelöst**
- Es gibt ein **Erfolgsgefühl** ein Arbeitspaket abschiessen zu können
- Die **Motivation steigt** das Arbeitspaket überhaupt anzufangen
- Es **verhindert Knotenrisiko** ("Eigentlich sind wir ja fertig, wenn nur XY funktionieren würde")
- Garantiert schnelles **Feedback vom Kunden**
  - erhöht die **Kundenzufriedenheit**
  - vermeidet, **löst Missverständnisse** frühzeitig

- Macht die **Gedanken frei** um sich auf etwas zu konzentrieren
- Wenn bereits beim Erstellen der Arbeitspakete überlegt wird, ob man dieses noch mehr unterteilen kann, setzt man sich schon früh mit der Funktionalität auseinander. Dies führt dazu, dass **eventuelle Probleme schon früher erkennbar werden**.

> **IMPORTANT**
>
> - Natürlich darf man im agilen Projektmanagement ein Arbeitspaket auch noch im Nachhinein splitten! 😁

Verweist auf den Code

Handelt es sich bei einem Arbeitspaket um eine **programmierbare Funktionalität**, muss das Arbeitspaket auf den **Code verweisen**, welches es umsetzt.

Dies wird in diesem Modul durch die Versionskontrolle *Git* in Kombination mit der Plattform *GitHub* ermöglicht.

Natürlich muss die Verweisung erst angefügt werden, **sobald das Arbeitspaket umgesetzt wird** und in den Status "In Progress" gesetzt wurde.

Verweist auf Abhängigkeiten

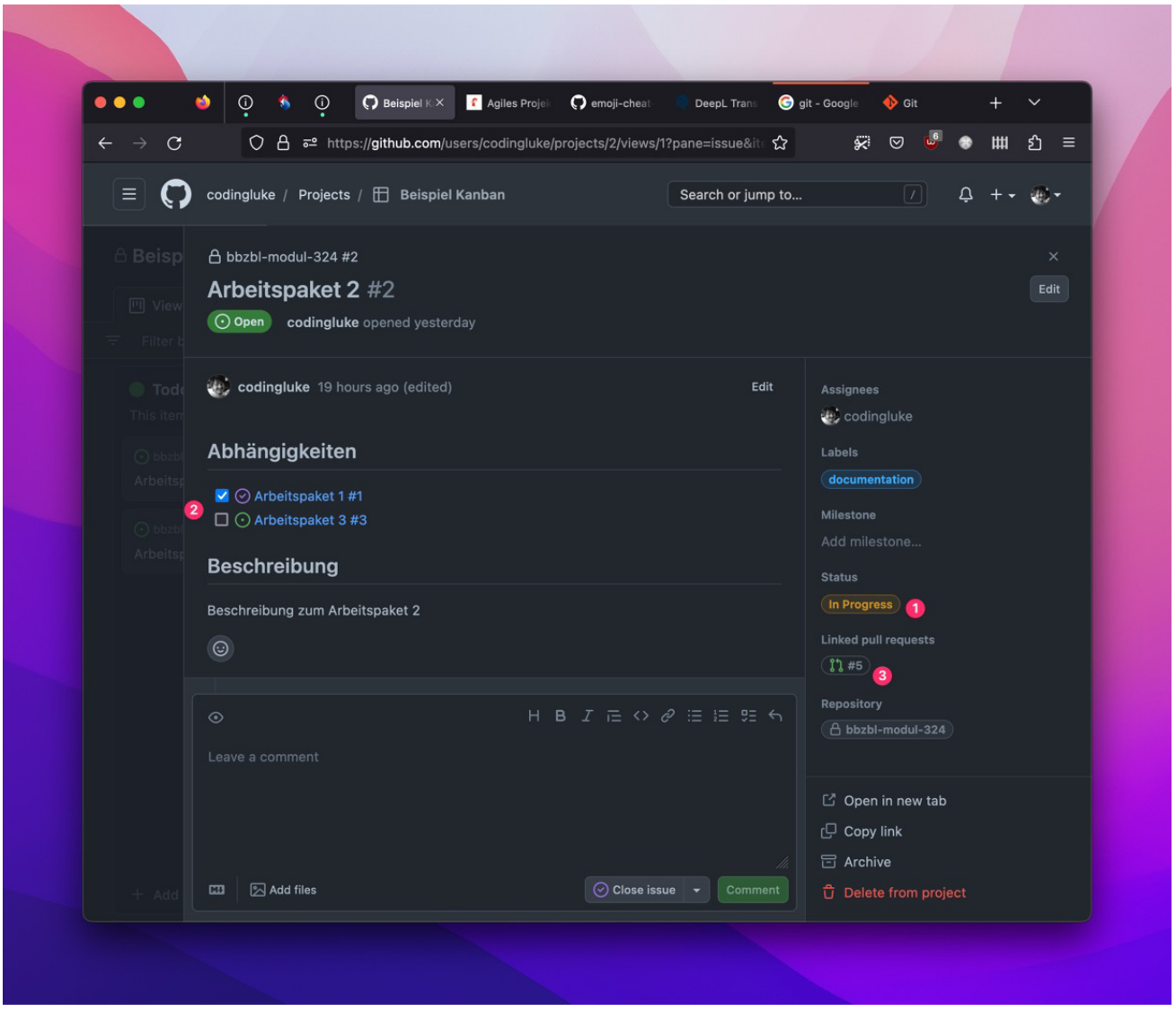
Ist ein Arbeitspaket abhängig von anderen Arbeitspaketen, muss diese Abhängigkeit klar ersichtlich sein. Dafür wird am Anfang der Beschreibung eine Liste erstellt, welche **auf Arbeitspakete verweist, welche davor gemacht werden müssen**.

In GitHub Markdown sieht eine Liste folgendermassen aus:

```
# Abhängigkeiten

- [x] #1
- [ ] #2

# Beschreibung

Die genaue Beschreibung...
```

- `#1` ist die **Nummer des GitHub Issues** und wird automatisch von GitHub aufgelöst

GitHub Issues: Verweisung auf Code und Abhängigkeiten in



1. Das *Arbeitspaket 2* ist **"In Progress"**
2. Es hat **Abhängigkeiten**, wobei eine noch aussteht
3. Es zeigt auf den **"*Pull Request*", also den Code** welche zur Implementation benötigt wird.

Kosten / Nutzen der agilen Vorgehensweise

Bei der agilen Vorgehensweise mit dem DevOps Model, entstehen traditionellerweise **am Anfang höhere Kosten**, da alle Entwicklungsumgebungen, das Testsystem, die CI/CD Pipelines usw. aufgebaut werden müssen. Während des Projektes sinken die Kosten für gewöhnlich.

Vorteile der Agilen Vorgehensweise

- Missverständnisse werden früh gefunden durch **schnelles Kundenfeedback**
- Es existiert von Anfang an ein **Funktionsfähiges System**
- Für alle **klarer Entwicklungsstand** *(Keine Katze im Sack, works-on-my-machine wird ausgehebelt)*
- Das **Risiko wird minimiert**, da der Kunde frühzeitig einschreiten kann, wenn was nicht stimmt.
- Teure **Nachbesserungsarbeiten, werden vermieden**
- Die **administrativen Tätigkeiten werden gesenkt**, da weniger über das Einhalten von Verträgen (Pflichtenheft) verhandelt werden muss
- Eine Funktionalität wird meistens erst klar, wenn man sie umsetzt. Die "hands-on" Mentalität führt dazu, dass schnell umgesetzt wird und somit schneller klar ist, **was der Kunde genau will** und wie es **am besten umsetzbar** ist.

Kundenfeedback

Tools

- **GitHub Projects** ☜ Im Unterricht verwendet
- Trello
- Jira
  - wird von vielen Grossfirmen verwendet in Kombiation mit BitBucket und Confluence
- Basecamp

Versionskontrolle

Handlunsziele

3. Wendet die Sourcecode- und Artefaktverwaltung für eine anforderungsorientierte Entwicklung im Team an.
   i. Kennt Praktiken, um Sourcecode featurebasiert zu verwalten (z.B. **GIT-Workflow**, **semantic Versioning** etc.).
   ii. Kennt Möglichkeiten, um benötigte und erstellte Artefakte/Binaries zu verwalten (z.B. Artefakt-Repository/Registry, Dependency-Management wie npm/maven etc.).

Wofür ist es gut?

Probleme beim Programmieren

Git

Tutorial

Pull Request

Machen Erläutern Grösse

Code Review

Vorteile Kommunikationsstiel

Git Flow

Tutorial

Feature Branch Release Branch Semantic versionin

Semantic Versioning

Formatierung und Linting

Handlungsziele

2. Konfiguriert und nutzt die lokale Entwicklungsumgebung integriert mit den automatisierenden Tools.
    i. Kennt Möglichkeiten, um automatisierende Tools in der Entwicklungsumgebung zu nutzen (z.B. **Linting**, Build, Ausführen, Testen, Versionierung, Abhängigkeiten/Pakete).

Was ist Formatierung?

Formatting bezieht sich auf die Darstellung des Codes, indem Leerzeichen, Einzüge und Zeilenumbrüche verwendet werden.

Das Ziel ist es, den Code lesbar und konsistent zu gestalten.

Wieso ist Formatierung wichtig?

- Lesbarkeit
- Wartbarkeit
- Konsistenz

Automatische Formatierung

VS Code

Tastenkürzel

- Windows: `Shift` + `Alt` + `F`
- Mac: `Shift` + `Option` + `F`
- Linux: `Ctrl` + `Shift` + `I`

Eclipse

Tastenkürzel

- Windows / Linux: `Ctrl` + `Shift` + `F`
- Mac: `⌘` + `⇧` + `F`

> **AUTOFORMAT BEIM SPEICHERN**
>
> `Window -> Preferences -> Java -> Editor -> SaveActions` das Häkchen **Source Code Formatierungen** setzen. Nun sollte der Source Code automatisch formatiert werden, sobald die Datei gespeichert wird.

IntelliJ

Tastenkürzel

- Windows / Linux: `Ctrl` + `Alt` + `L`
- Mac: `⌥` + `⌘` + `L`

> **AUTOFORMAT BEIM SPEICHERN `Preferences -> Tools -> Actions On Save` DIE**
>
> Häkchen **Reformat Code** und **Optimize imports** setzen. Jetzt sollte der Source Code automatisch formatiert werden, sobald die Datei gespeichert wird.

Tools

- Google Java Format
- Prettier (JS/TS)

Was ist Linting?

Linting sowie Formatierung dienen beide dazu, die Code-Qualität zu verbessern. Während das Formatting auf die ästhetische Darstellung abzielt, konzentriert sich das Linting auf die **funktionale Korrektheit**.

Linting identifiziert:

- potenzielle Fehler
- stilistische Probleme
- unsichere Praktiken

Das Ziel von Linting ist:

- Konformität und Standards
- einheitliche Qualität
- **Sicherheit**

Tools

- SonarLint (Ganz viele Sprachen)

- eslint (JavaScript/Typescript)

Externe Links

- Java formatting and linting

Paketmanager

Handlungsziele

3. Wendet die Source-Code- und Artefaktverwaltung für eine anforderungsorientierte Entwicklung im Team an.
   i. Kennt Praktiken, um Sourcecode featurebasiert zu verwalten (z.B. Git-Workflow, Semantic Versioning etc.).
   ii. Kennt Möglichkeiten, um benötigte und erstellte Artefakte/Binaries zu verwalten (z.B. **Artefakt-Repository/Registry**, **Dependency-Management** wie **npm/maven/gradle** etc.).

Was ist ein Paketmanager?

Wofür braucht man diese?

Typen

Gradle

Maven

NPM / Yarn

Bundler

Composer (PHP)

:superhero: asdf / mise en place

Testautomation

Handlungsziele

2. Konfiguriert und nutzt die lokale Entwicklungsumgebung integriert mit den automatisierenden Tools.
    i. Kennt Möglichkeiten, um automatisierende Tools in der Entwicklungsumgebung zu nutzen (z.B. Linting, Build, Ausführen, **Testen**, Versionierung, Abhängigkeiten/Pakete).

Wofür ist es gut?

Arten

Unit Test

- JUnit (Java)
- Jest (Javascript)

Integration Test

- Selenium
- Playwright (Browser automation)
- Puppeteer (Browser automation)
- Taiko (Browser automation)
- Guauge (Browser automatin und BDD)

Tutorial

CI / CD

Handlungsziele

4. Nutzt einen automatisierten Integrationsprozess (Build, Qualitätssicherung/Test der Artefakte).
    i. Kennt Möglichkeiten, um die Applikation und Komponenten automatisiert zu bauen, auszuführen und zu testen (z.B. Build-Management wie `gradle`/`npm`, Container, Pipelines etc.).
5. Kennt Möglichkeiten, um die Applikation automatisiert zu deployen und konfigurieren (z.B. *Infrastructure as Code*, Scripting, Virtualisierung, Konfiguration mit YAML, Vaults, Profile, SSH-Key etc.).

Continious Integration

Automatisierungstools

- GitHub Actions
- Jenkins
    - How to Install and Run Jenkins With Docker Compose

Image bauen

- Dockerfile
- `docker build`

In Registry pushen

- `docker push`
- `npm publish`

Continious Delivery

Docker Compose

- Bauen des *images* durch Github Action oder Jenkins
- Ändere das `docker-compose.yaml` um das neuste Image zu verwenden
- SSH in prod system `git pull` restart compose
    - Docker Compose Continuous Deployment setup