Modul 06

GitHub & Pull Requests

Lernziele

- GitHub nutzen: Repos, Issues und Projects
- Pull Requests erstellen und reviewen
- Branch Protection Rules einrichten
- GitHub Actions: CI/CD-Grundlagen am Beispiel AL-Go

GitHub-Überblick

GitHub ist tatsächlich mehr als nur Git-Repos:

Feature	Nutzen
Repositories	Code und History verwalten
Issues	Bugs und Aufgaben tracken
Projects	Kanban-Boards für Planung
Pull Requests	Code-Reviews und Merges
Actions	CI/CD-Pipelines
Releases	Versionierte Auslieferungen

Pull Requests (PRs)

Ein Pull Request ist ein **Vorschlag, Änderungen zu mergen**.

Workflow:

1. Feature-Branch erstellen und Commits machen
2. Branch pushen: `git push -u origin feature/customer-list`
3. PR auf GitHub erstellen
4. Team reviewed den Code
5. Nach Freigabe: PR mergen

> **Kern-Idee:** Kein Code kommt direkt in `main` - alles läuft über Pull Requests.

Einen PR erstellen

Auf GitHub:

1. "Compare & pull request" klicken (nach Push)
2. **Titel** - Kurze Zusammenfassung (wie Commit Message)
3. **Beschreibung** - Was wurde geändert und warum?
4. **Reviewer** zuweisen
5. **Labels** setzen (z.B. `bugfix`, `feature`)

Einen PR erstellen (Fortsetzung)

Via CLI (GitHub CLI):

```
gh pr create --title "Add customer list extension" \
  --body "Adds the customer list page extension" \
  --reviewer teammate
```

PR-Templates

Ein Template sorgt für einheitliche Beschreibungen.

`.github/pull_request_template.md`:

```
## Was wurde geändert?

## Warum?

## Wie getestet?

## Checkliste
- [ ] Code kompiliert fehlerfrei
- [ ] Manuelle Tests durchgeführt
- [ ] Commit Messages sind aussagekräftig
```

Draft Pull Requests

Für Arbeit, die noch nicht fertig ist:

- Signalisiert dem Team: "Noch nicht reviewen"
- Nützlich für frühes Feedback zur Richtung
- Kann jederzeit in einen regulären PR umgewandelt werden

```
gh pr create --draft --title "WIP: New report module"
```

Code Reviews

Als Reviewer:

- **Approve** - Code ist in Ordnung
- **Request Changes** - Änderungen nötig
- **Comment** - Frage oder Hinweis (blockiert nicht)

Gute Review-Praxis:

- Sachlich und konstruktiv kommentieren
- Konkretes Feedback statt "Das gefällt mir nicht"
- Vorschläge mit "Suggested Changes" direkt als Diff

Merge-Optionen für PRs

GitHub bietet drei Merge-Strategien:

Methode	Ergebnis	Wann nutzen
Merge Commit	Merge-Commit wird erstellt	Standard, History bleibt vollständig
Squash and Merge	Alle Commits zu einem	Feature hat viele kleine Commits
Rebase and Merge	Commits werden rebased	Lineare History gewünscht

> **Empfehlung für Teams:** "Squash and Merge" als Standard, damit `main` eine saubere History hat.

Branch Protection Rules

Schützt wichtige Branches vor direkten Pushes:

Typische Regeln für `main`:

- Require pull request before merging
- Require at least 1 approval
- Require status checks to pass
- Require branches to be up to date

Einrichten:

Settings → Branches → Add rule → Branch name: `main`

Ausblick: GitHub Actions

Automatisierte Workflows bei Git-Events:

```yaml
# .github/workflows/build.yml
name: Build
on:
  push:
    branches: [main]
  pull_request:
    branches: [main]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: echo "Build and test here"
```