

Login-API

Entwicklung einer RESTful API zur Registrierung und
dem Login von Benutzern mit OAuth2 als Container-Image



ZANCOR

Inhalt

- Vorstellung
- Unternehmen
- Projektbeginn
- Planung
- Entwurf
- Implementierung
- Qualitätsmanagement
- Übergabe
- Fazit



Vorstellung

- Daniel Schwarz
- 06.02.1985
- Berlin
- Bürokaufmann
- Angeln, Gaming, IT

Github: @Axlken

The screenshot shows the GitHub profile for user **Axlken**. The profile features a large circular profile picture of a red and orange eye. Below the picture, the name **Axlken** is displayed twice. The profile summary includes the following information:

- Edit profile** button
- 1 follower • 0 following
- Germany, Berlin

The **Highlights** section shows a **PRO** badge. The **Organizations** section lists one organization represented by a small icon. The **Contribution activity** section shows a heatmap of contributions over the last year, indicating activity primarily in February, March, April, and December. A message states "Axlken has no activity yet for this period." At the bottom, there is a link to "Show more activity". The footer of the page includes standard GitHub links like Terms, Privacy, Security, Status, Docs, Contact, Manage cookies, and a "Do not share my personal information" checkbox.

Praktikumsunternehmen

ZANCOR Consulting & Development UG
Rigaer Straße 49
10247 Berlin

www.zancor.de

- IT-Softwareprojekte
- Consulting
- Schulungen



ÜBER UNS

Herausforderungen & Bewältigung dieser ist unser Leben



UNSERE VISION

Wir begleiten die Unternehmen in ihre digitale Zukunft. Mit unserem exzellenten und umfassenden technologischen Know-how sind wir ein starker und kompetenter Partner für die Unternehmen. Wir kennen die technologischen Trends, entwickeln digitale und innovative Lösungen und setzen diese gemeinsam mit den Unternehmen um - individuell und bedarfsoorientiert.



UNSERE MISSION

Digitalisierung, Innovation und Technologien sind unsere DNA. Wir erfassen die Probleme unserer Kunden, geben strategische Orientierung und setzen IT-Projekte lösungsorientiert um kompetent um. Unser Know-how geben wir gern an die Unternehmen weiter und leisten mit Leidenschaft und persönlichem Antrieb einen entscheidenden Beitrag für die Sicherung der Wettbewerbsfähigkeit für die digitale Zukunft. Kurzum: wir als unterstützende Kraft auf Ihrem Weg zum Erfolg!



SICHERHEIT

Unsere Kunden können sich darauf verlassen, dass ihre digitalen Systeme und damit Ihre Daten geschützt und sicher sind. Datensicherheit genießt höchste Priorität.



ZUVERLÄSSIGKEIT

Unsere Systeme sind auf Langfristigkeit und Stabilität ausgerichtet und überfüllen die Erwartungen unserer Kunden.



LEISTUNGSFÄHIGKEIT



ZUSAMMENARBEIT

Projektbeginn

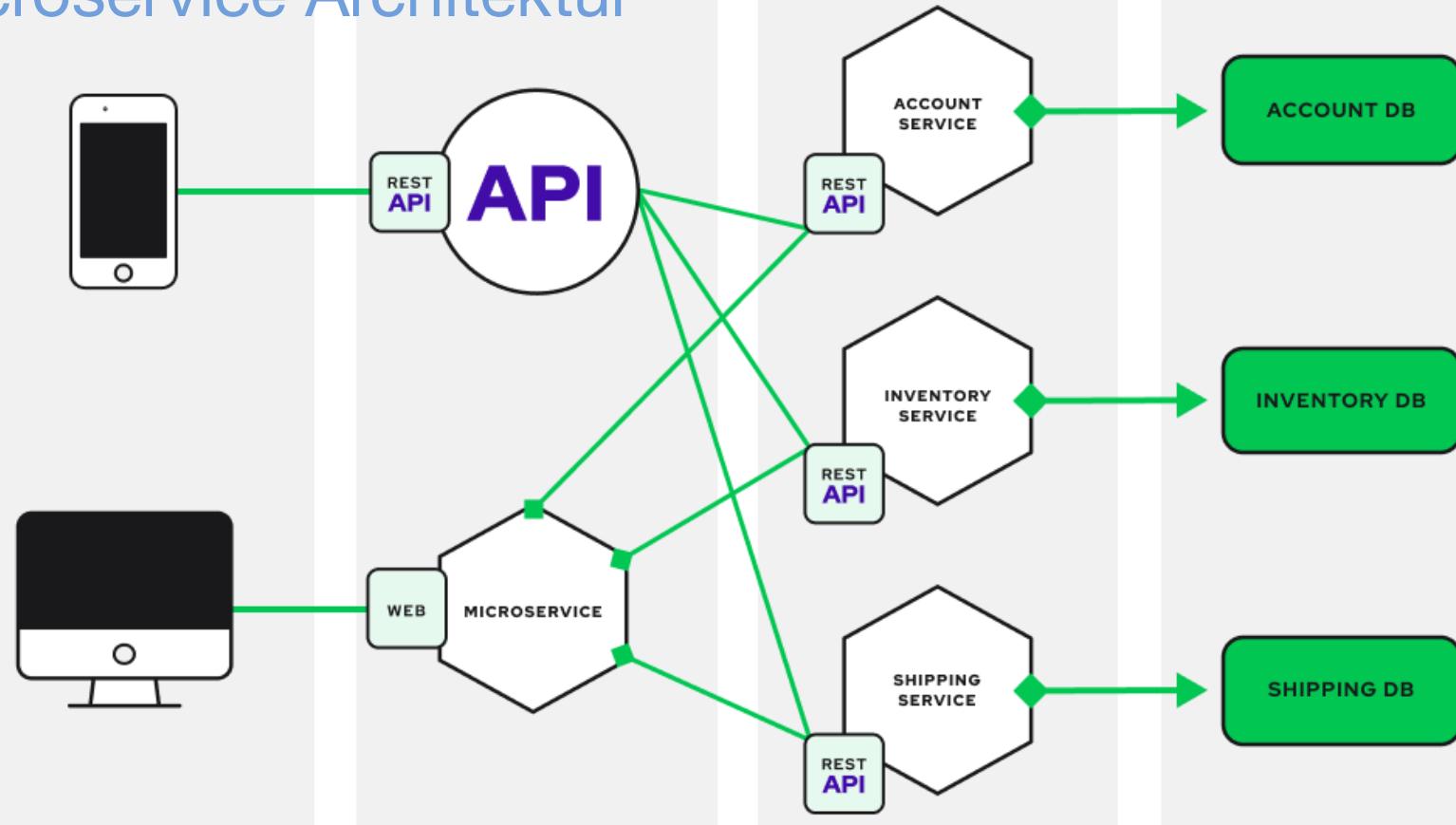
Eine immer wiederkehrende Aufgabe

Das Problem

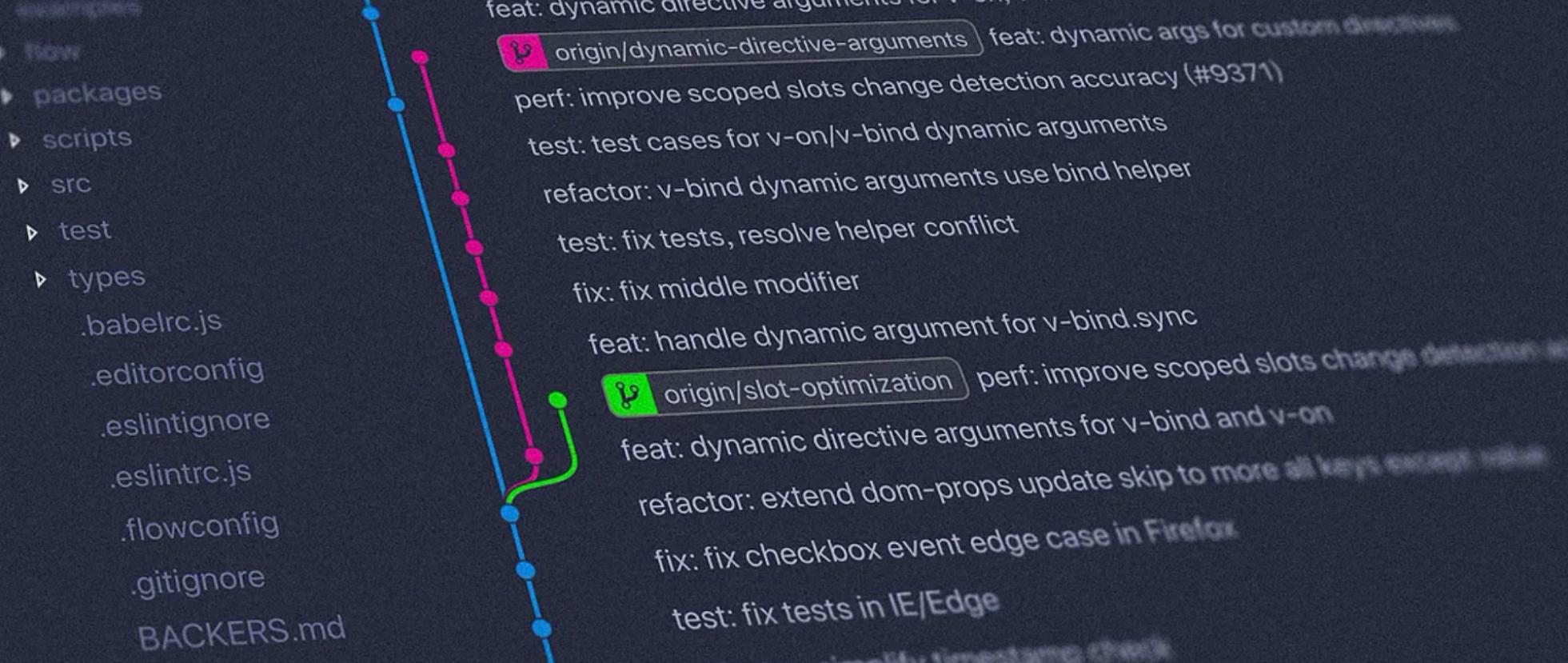
- Login Modul in fast jedem Projekt
- Monolithische Projekte
- keine Versionsverwaltung
- wechselnde Entwickler



Microservice Architektur



Versionsverwaltung



Ziele

- RESTful API
- Geltende Sicherheitsstandards
- Email-Verifizierung
- Daten in Datenbank Speichern
- Containerimage
- Codequalität
- Versionskontrolle
- auf Cloudinstanz bereitstellen

Planung

Das Wann und Womit

Planung

- 06.11. - 24.11.2024
- max. 80h
- Homeoffice
- keine Bereitstellungen

Phasen

Projektphasen	Zeit (in h)
Analyse/Planung	8
Entwurf	9
Implementierung	39
Abnahme	4
Dokumentation	13
Gesamt	73

Kosten

Stelle	Kosten
Entwickler	1499,42€
Geschäftsleiter	354,84€
Cloud-Server	2,51€
Gesamt	1856,77€

Armotisation

- 10 Projekte im Jahr 2022
- Kosten: 1643,20€
- Zukünftige Kosten: 410,80€
- Ersparnis: 1232,40€/Jahr

Amortisationsdauer

18 Monate

Nutzwert API

Kriterium	Gewicht	FastAPI		Django		Flask	
		Punkte	Gewichtet	Punkte	Gewichtet	Punkte	Gewichtet
Performance	40%	10	4	6	2,4	8	3,2
Wartbarkeit	30%	9	2,7	5	1,5	6	1,8
Nützlichkeit	30%	10	3	10	3	9	2,7
Summe			9,7		6,9		7,8

Entwurf

Die Eckpfeiler

Datenmodell

Users(id, email_{NN}, hashed_pwd_{NN}, username_{NN}, is_activated_{NN}, created_at_{NN})

Endpunkte

- /register -> POST - Benutzerregistrierung - offen
- /login -> POST - Benutzerlogin - offen
- /user/change_email -> POST - Benutzer Email ändern - abgesichert
- /users -> GET - Erhalte eine Liste aller Nutzer - abgesichert
- /user/by_mail -> GET - Erhalte einen Benutzer mit EMail Angabe - abgesichert
- /user/change_pwd -> PUT - Benutzerpasswort erneuern - abgesichert
- /user -> DELETE - Lösche einen Benutzer - abgesichert

Qualitätssichernde Maßnahmen

- Unitests
- Linter
- Formatter
- Typechecking
- Dependency-Management
- automatisiert
- CI/CD
- sichere Verschlüsselung



Implementierung

Hier gehts zur Sache

Ablauf

- Github-Repositories
- CI / automatisiertes Testen
- Cloud-Instanz
- Container-Images Datenbank & API Service
- REST-API
 - Module Installieren
 - FastAPI
 - Endpunkte
 - OAuth2 Authentifizierung
 - ORM
 - Password-Hashing
 - Unitests



Continuous Integration

- pytest - Unitest
- deptry - Coverageauswertung
- mypy - Statischer-Typen-Checker
- tox - kompatibilitätstest
- ruff - Linter & Formatter
- git-hooks - automatisierung

The screenshot shows a GitHub Actions workflow for a repository named 'Axklen / login-api'. The workflow is titled 'fixed ENV error #23'. It consists of several jobs:

- quality**: This job contains sub-tasks for tox (3.8, 3.9, 3.10, 3.11), check-docs, and build-and-push-image. All sub-tasks are marked as green (Passed).
- Run details**: Includes sections for Usage and Workflow file.
- quality**: A summary of the workflow's execution, showing it succeeded 3 hours ago in 2m 51s. It lists the following steps:
 - Set up job
 - Checkout
 - Run actions/cache@v3
 - Set up the environment
 - Run checks**: This section is expanded, showing the log output for each step:
 - Run make check
 - Checking Poetry lock file consistency with 'pyproject.toml': Running poetry lock --check
 - All set!
 - Linting code: Running pre-commit
 - INFO Installing environment for https://github.com/pre-commit/pre-commit-hooks.
 - INFO Once installed this environment will be reused.
 - INFO This may take a few minutes...
 - INFO Installing environment for https://github.com/charliermarsh/ruff-pre-commit.
 - INFO Once installed this environment will be reused.
 - INFO This may take a few minutes...
 - INFO Installing environment for https://github.com/nsf/black.
 - INFO Once installed this environment will be reused.
 - INFO This may take a few minutes...
 - check for case conflicts.....Passed
 - check for merge conflicts.....Passed
 - check toml.....Passed
 - check yaml.....Passed
 - fix end of files.....Passed
 - trim trailing whitespace.....Passed
 - ruff.....Passed
 - black.....Passed
 - prettier.....Passed
 - Static type checking: Running mypy
 - Success: no issues found in 10 source files
 - Checking for obsolete dependencies: Running deptry
 - Scanning 10 files...
 - Success! No dependency issues found.
 - Post Set up the environment
 - Post Run actions/cache@v3
 - Post Check out

Container-Image

```
1  # syntax=docker/dockerfile:1
2  FROM python:3.11-slim-buster
3  ENV POETRY_VERSION=1.7.1 \
4  POETRY_VIRTUALENVS_CREATE=false \
5  POSTGRE_USER=auchGeheim \
6  POSTGRE_PASS=geheim \
7  POSTGRE_DB=postgres \
8  SECRET_KEY=sehrGeheimerKeyDerMitPythonSecretsErstelltWurde
9  ENV URL_DATABASE=postgresql://$POSTGRE_USER:$POSTGRE_PASS@localhost
10 :5432/$POSTGRE_DB
11 # Install poetry
12 RUN pip install "poetry==$POETRY_VERSION"
13 # Copy only requirements to cache them in docker layer
14 WORKDIR /code
15 COPY poetry.lock pyproject.toml /code/
16 # Project initialization:
17 RUN poetry install --no-interaction --no-ansi --no-root --no-dev
18 # Copy Python code to the Docker image
19 COPY login_api /code/login_api/
20 CMD [ "uvicorn", "login_api.main:app", "--reload", "--port", "8000", "
21 --host", "0.0.0.0"]
22
```

Endpunkte

```
1   ...
2   @app.get("/users", tags=["user"])
3   async def get_all_users() → None:
4       pass
5   @app.get("/user/by_mail", tags=["user"])
6   async def get_user_by_mail() → None:
7       pass
8   @app.post("/register", tags=["user"])
9   async def register_user() → None:
10      pass
11  @app.post("/login", tags=["user"])
12  async def login_user() → None:
13      pass
14  @app.put("/user/change_email", tags=["user"])
15  async def update_email() → None:
16      pass
17  ...
18
```

Oauth2 - `auth_handler.py`

```
1  ...
2  oauth2_scheme = OAuth2PasswordBearer(tokenUrl="login")
3
4  SECRET_KEY = str.getenv("SECRET_KEY")
5  ALGORITHM = "HS256"
6  ACCESS_TOKEN_EXPIRE_MINUTES = 30
7
8  # data = {"email": user_email}
9  def create_access_token(data: dict, expires_delta: Union[timedelta, None] = None) → str:
10      to_encode_data = data.copy()
11      if expires_delta:
12          expires = datetime.utcnow() + expires_delta
13      else:
14          expires = datetime.now() + timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
15      to_encode_data.update({"exp": expires})
16      # Accesstoken erstellen
17      encoded_jwt = jwt.encode(to_encode_data, SECRET_KEY, algorithm=ALGORITHM)
18      return encoded_jwt
19  ...
20  ...
```

ORM - `models.py`

```
1  from datetime import datetime
2
3  from sqlalchemy import Boolean, DateTime, Integer, String
4  from sqlalchemy.orm import DeclarativeBase, mapped_column
5
6  class Base(DeclarativeBase):
7      pass
8
9  class User(Base):
10      __tablename__ = "users"
11
12      id = mapped_column(Integer, primary_key=True, index=True)
13      email = mapped_column(String, unique=True)
14      username = mapped_column(String, unique=True)
15      hashed_pwd = mapped_column(String)
16      is_activated = mapped_column(Boolean, default=False)
17      created_at = mapped_column(DateTime, default=datetime.now())
18
```

Unit-Tests - `test_main.py`

```
1  def test_change_email_no_auth():
2      response = client.put("/user/change_email", json={"new_email": "roland@peter.de"})
3      assert response.json() == {"detail": "Not authenticated"}
4      assert response.status_code == 401
5
6
7  # test Userendpoints
8  def test_get_all_users_with_auth():
9      response = client.get("/users", headers=access_token_header)
10     assert response.status_code == 200
11     response_data = response.json()
12     assert response_data[-1]["email"] == TEST_EMAIL
13
14
15 def test_get_user_by_mail_with_auth():
16     url = f"/user/by_mail?email={TEST_EMAIL}"
17     response = client.get(url, headers=access_token_header)
18     assert response.status_code == 200
19     response_data = response.json()
20     assert response_data["email"] == TEST_EMAIL
21     assert response_data["username"] == TEST_USERNAME
22
```

Projektsruktur

```
Documents/IHK-Projekt/login-api main • lt --level=4 --icons
.
├── docs
│   ├── index.md
│   └── modules.md
└── login_api
    ├── auth
    │   ├── __init__.py
    │   ├── auth_handler.py
    │   └── pwd_handler.py
    ├── db
    │   ├── __init__.py
    │   ├── crud.py
    │   ├── database.py
    │   ├── models.py
    │   └── schemas.py
    ├── __init__.py
    └── main.py
    └── tests
        ├── config.py
        └── test_main.py
! codecov.yaml
CONTRIBUTING.md
Dockerfile
LICENSE
Makefile
! mkdocs.yml
poetry.lock
poetry.toml
pyproject.toml
README.md
tox.ini
```

Codecovage

Axklen / 🔒 login-api / 🏷 main

Coverage Flags Commits Pulls Settings

Branch Context

main ▾

Coverage on branch

80.88%

165 of 204 lines covered

7 Days ▾ trend

+80.88%

Source: latest commit da323d8

Show Chart

Code tree File list 7 total files login-api

Search for files

Files	Tracked lines	Covered	Partial	↓ Missed	Coverage %
login_api/auth/auth_handler.py	40	24	2	14	<div style="width: 60.0%;"><div style="width: 60.0%;">60.00%</div></div>
login_api/main.py	73	59	7	7	<div style="width: 80.82%;"><div style="width: 80.82%;">80.82%</div></div>
login_api/db/database.py	13	9	0	4	<div style="width: 69.23%;"><div style="width: 69.23%;">69.23%</div></div>
login_api/db/crud.py	47	42	2	3	<div style="width: 89.36%;"><div style="width: 89.36%;">89.36%</div></div>
login_api/auth/pwd_handler.py	6	6	0	0	<div style="width: 100.00%;"><div style="width: 100.00%;">100.00%</div></div>
login_api/db/models.py	13	13	0	0	<div style="width: 100.00%;"><div style="width: 100.00%;">100.00%</div></div>
login_api/db/schemas.py	12	12	0	0	<div style="width: 100.00%;"><div style="width: 100.00%;">100.00%</div></div>

Testen & Bereitstellen

Die Zitter-Phase

Manualles Testen - `swagger-UI`

user

GET /users Get All Users

Parameters

No parameters

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8080/users' \
-H 'accept: application/json'
```

Request URL

```
http://localhost:8080/users
```

Server response

Code Details

Undocumented TypeError: NetworkError when attempting to fetch resource.

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

```
"string"
```

GET /user/by_mail Get User By Mail

POST /register Register User

POST /login Login User

Bereitstellung

- Containerimage via Github-Package-Registry
- PostgreSQL Container
- Login_API Container
- System-D Service
- Portweiterleitung Cloudfirewall
- Erreichbarkeit sicherstellen

Übergabe

Fast geschafft

Übergabe

- finales Meeting
- funtionalität vorgestellt
- schulung
- dokumentation
- repository übertragen
- Protokoll unterzeichnet

Übergabeprotokoll

für Softwareprojekt: *RESTful Login-API*

Datum: 24.11.2023

Verantwortlicher: Daniel Schwarz, Olbersstr. 49, 10589 Berlin

Kunde: Zancor Consulting & Development UG, Rigaer Straße 49, 10247 Berlin

Version der Software: 1.0.0

Übergebene Softwarekomponenten:

1. **Hauptanwendung:** RESTful Login-API als Container-Image
 - Version: 1.0.0
 - Hinweis: Die E-Mail-Verifizierung muss noch implementiert werden
2. **Datenbankkomponente:** PostgreSQL Container-Image über dockerhub.com
 - Hinweis: Beliebige andere Datenbank einfach Integrierbar

Lieferungsinhalt:

- Quellcode der Software auf Github
- Dokumentation der Software
- Installationsanleitung & Schulung
- Testprotokoll & Deploymentkonzept

Beschreibung der Übergabe:

Die Software „RESTful Login-API“ wird in der Version 1.0.0 an den Kunden, Zancor Consulting & Development UG, übergeben. Die Softwarekomponenten wurden gemäß den Anforderungen spezifiziert und entwickelt.

Installation und Inbetriebnahme:

Der Kunde ist für die Installation und Inbetriebnahme der Software verantwortlich. Die Installationsanleitung (auf Github) enthält detaillierte Anweisungen zur Installation.

Schulung und Support:

Eine Schulung des Kundenpersonals zur Verwendung der Software ist auf Anfrage verfügbar. Support-anfragen können über Email: schwarz.d@posteo.de gestellt werden. Eine Schulung des Kunden wurde bei der Übergabe durchgeführt.

Abnahme und Unterschrift:

Der Kunde, Zancor Consulting & Development UG, bestätigt den Erhalt der oben genannten Software und Dokumentation sowie die Kenntnisnahme der Installationsanleitung und Support-verfahren durch seine Unterschrift.

Fazit

Was habe ich gelernt

Das Negative

- Keine Email-Verifizierung
- Zeitplanung leicht überzogen
- Kein auto-update auf Server

Das Positive

- Im maximalen Zeitrahmen
- Alles funtional
- Viel gelernt
- Auftraggeber überaus zufrieden



Danksagung

Sind Sie noch Wach?