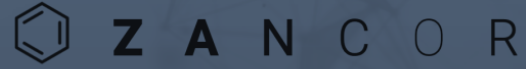
Login-API

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



Inhalt

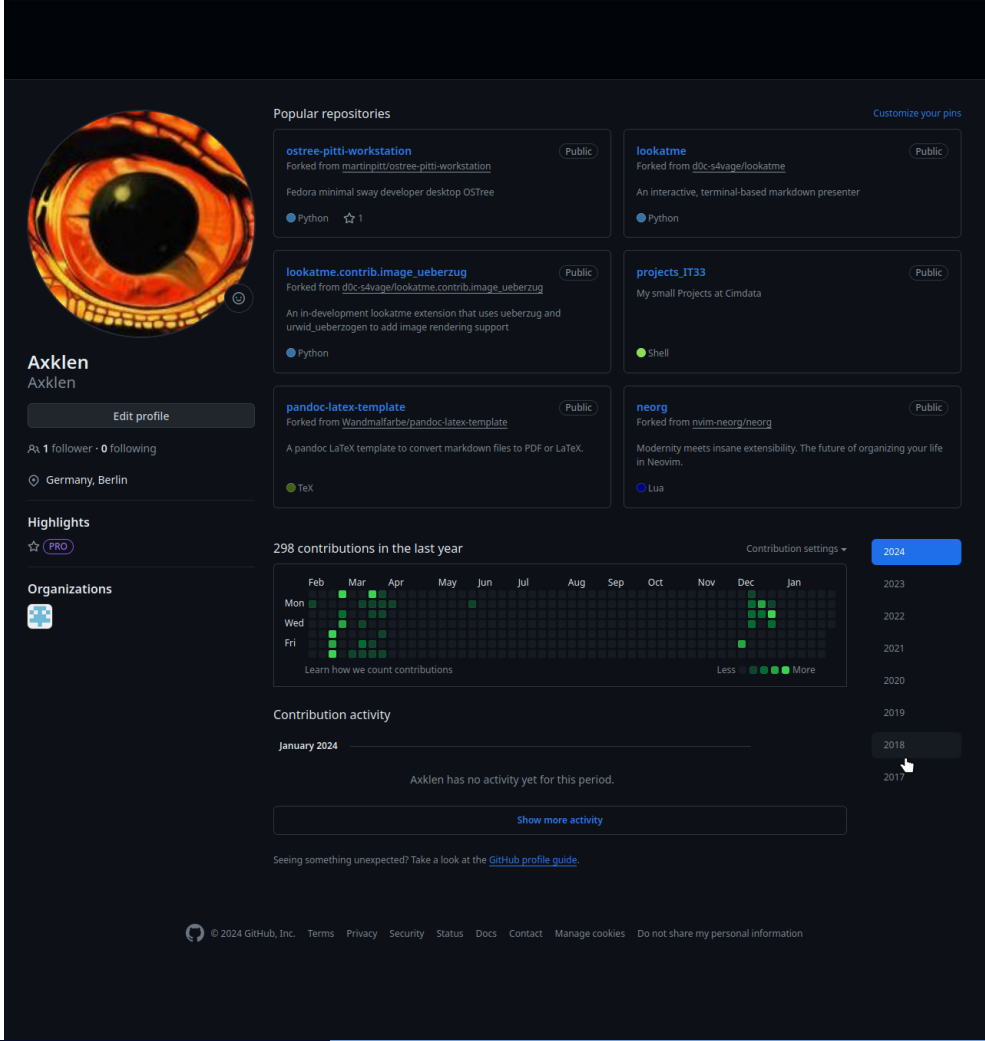
- Vorstellung
- Unternehmen
- Projektbeginn
- Planung
- Entwurf
- Durchfuehrung
- Qualitaetsmanagement
- Uebergabe
- Fazit

ZANCOR
Consulting & Development UG

Vorstellung

- Daniel Schwarz
- 06.02.1985
- Berlin
- Buerokaufmann
- Angeln, Gaming

Github: @Axklen



Praktikumsunternehmen

ZANCOR Consulting & Development UG
Rigaer Straße 49
10247 Berlin

www.zancor.de

- IT-Softwareprojekte
- Consulting
- Schulungen

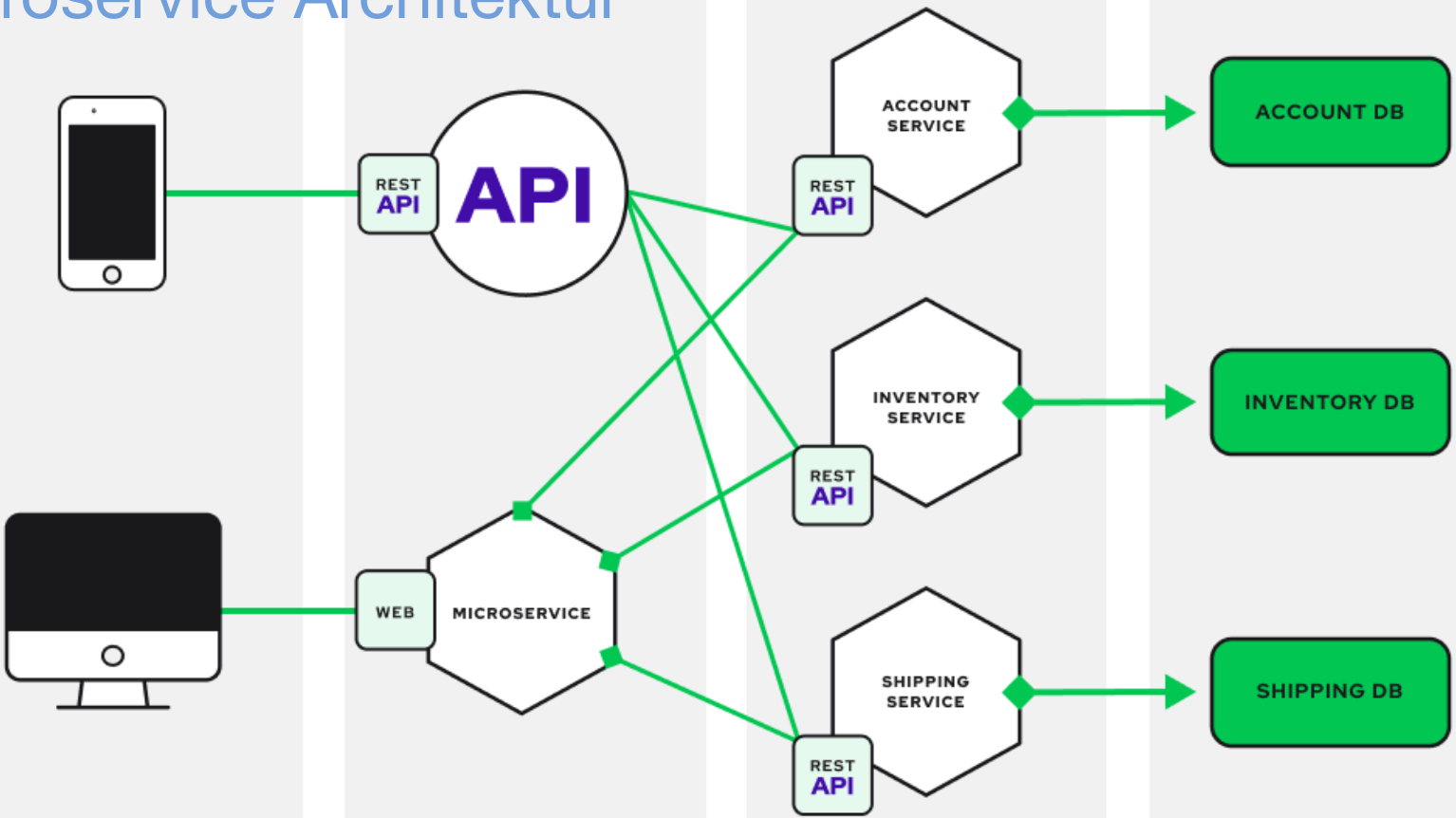
Projektbeginn

Eine immer wiederkehrende Aufgabe

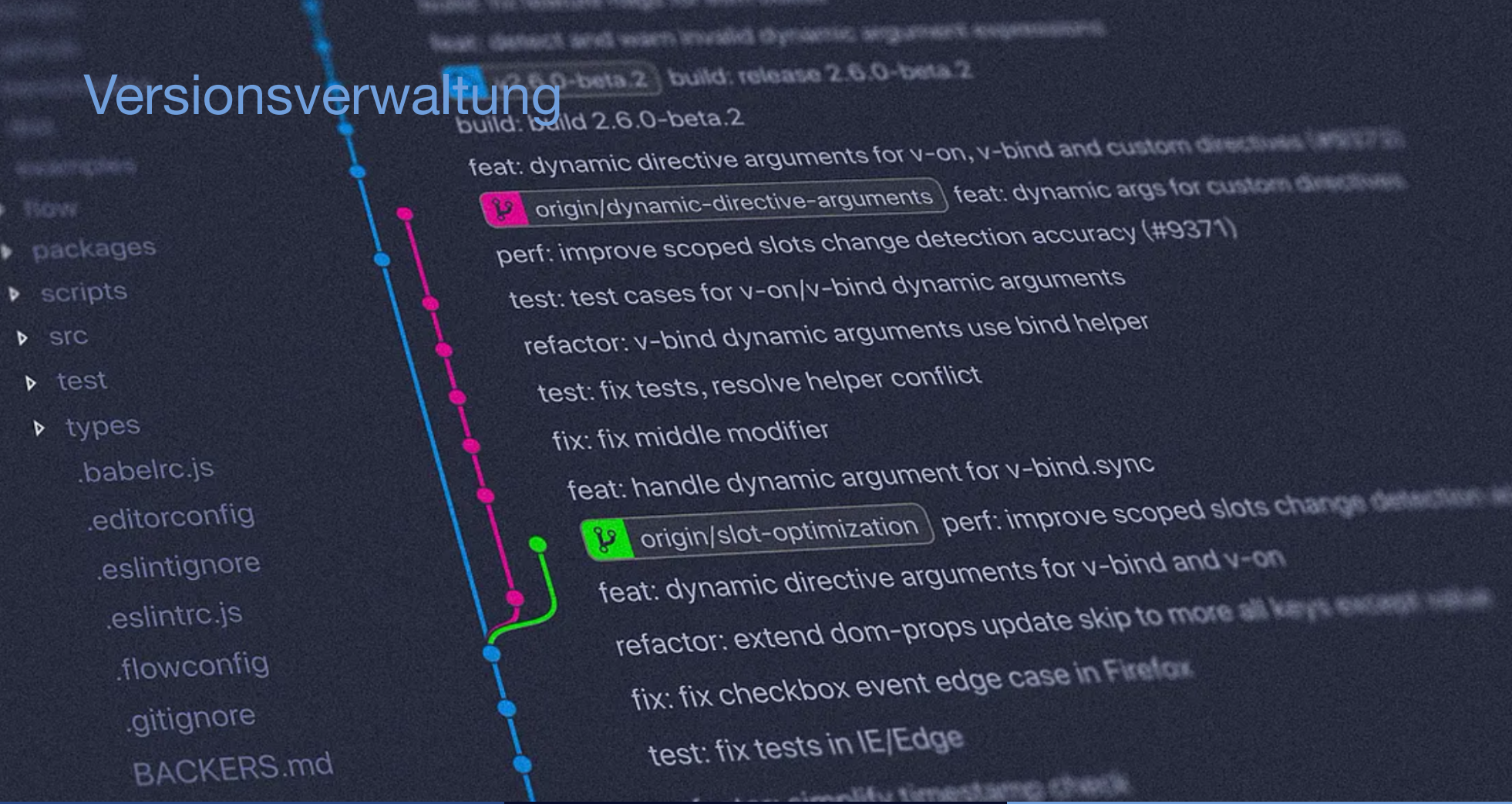
Das Problem

- Login Modul in fast jedem Projekt
- Monolitische 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€
Geschaeftsleiter	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

$$Users(\underline{id}, email_{NN}, hashed\_pwd_{NN}, username_{NN}, is\_activated_{NN}, created\_at_{NN})$$

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

- Unittests
- Linter
- Formatter
- Typechecking
- Dependency-Management
- automatisiert
- CI/CD
- sichere Verschluesselung

Implementierung

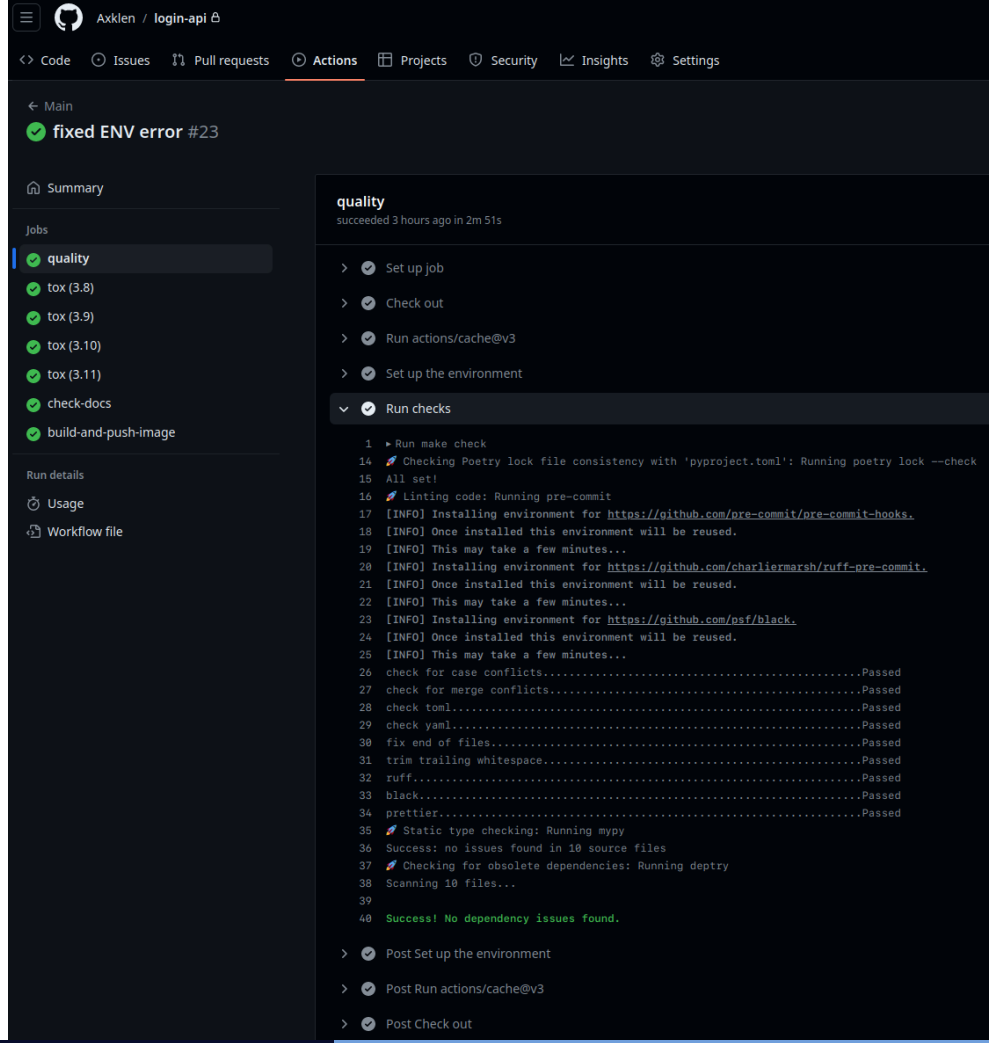
Hier gehts zur Sache

Ablauf

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

Continuous Integration

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



Container-Image

```
# syntax=docker/dockerfile:1
FROM python:3.11-slim-buster
ENV POETRY_VERSION=1.7.1 \
POETRY_VIRTUALENVS_CREATE=false \
POSTGRE_USER=auchGeheim \
POSTGRE_PASS=geheim \
POSTGRE_DB=postgres \
SECRET_KEY=sehrGeheimerKeyDerMitPythonSecretsErstelltWurde
ENV URL_DATABASE=postgresql://$POSTGRE_USER:$POSTGRE_PASS@localhost
:5432/$POSTGRE_DB
# Install poetry
RUN pip install "poetry==$POETRY_VERSION"
# Copy only requirements to cache them in docker layer
WORKDIR /code
COPY poetry.lock pyproject.toml /code/
# Project initialization:
RUN poetry install --no-interaction --no-ansi --no-root --no-dev
# Copy Python code to the Docker image
COPY login_api /code/login_api/
CMD [ "uvicorn", "login_api.main:app", "--reload", "--port", "8000", "
--host", "0.0.0.0"]

```

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`

```python
...
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="login")

SECRET_KEY = str(getenv("SECRET_KEY"))
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30

# data = {"email: user_email"}
def create_access_token(data: dict, expires_delta: Union[timedelta,None] = None) -> str:
    to_encode_data = data.copy()
    if expires_delta:
        expires = datetime.utcnow() + expires_delta
    else:
        expires = datetime.now() + timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    to_encode_data.update({"exp": expires})
    # Accesstoken erstellen
    encoded_jwt = jwt.encode(to_encode_data, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt
...

```

ORM - `models.py`

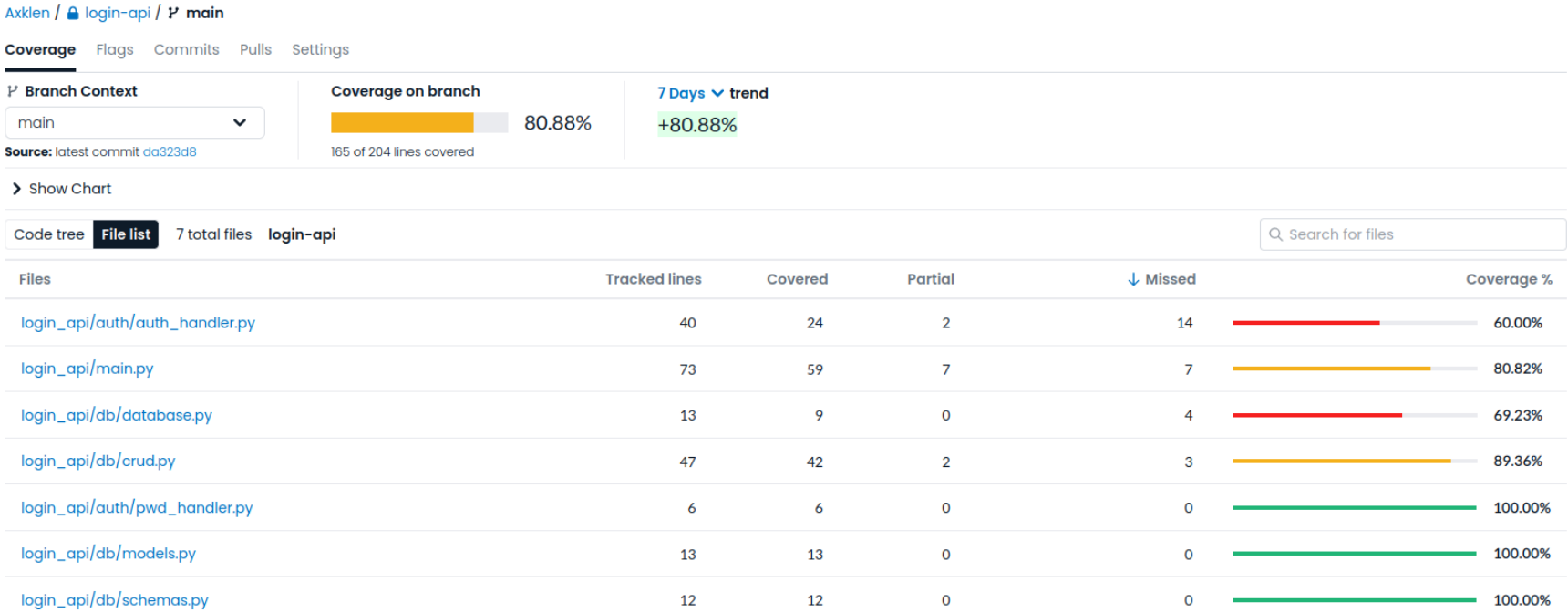
```python
from datetime import datetime

from sqlalchemy import Boolean, DateTime, Integer, String
from sqlalchemy.orm import DeclarativeBase, mapped_column

class Base(DeclarativeBase):
    pass

class User(Base):
    __tablename__ = "users"

    id = mapped_column(Integer, primary_key=True, index=True)
    email = mapped_column(String, unique=True)
    username = mapped_column(String, unique=True)
    hashed_pwd = mapped_column(String)
    is_activated = mapped_column(Boolean, default=False)
    created_at = mapped_column(DateTime, default=datetime.now())

```

Unit-Tests - `test_main.py`

```python
def test_change_email_no_auth():
    response = client.put("/user/change_email", json={"new_email": "roland@peter.de"})
    assert response.json() == {"detail": "Not authenticated"}
    assert response.status_code == 401


# test Userendpoints
def test_get_all_users_with_auth():
    response = client.get("/users", headers=access_token_header)
    assert response.status_code == 200
    response_data = response.json()
    assert response_data[-1]["email"] == TEST_EMAIL


def test_get_user_by_mail_with_auth():
    url = f"/user/by_mail?email={TEST_EMAIL}"
    response = client.get(url, headers=access_token_header)
    assert response.status_code == 200
    response_data = response.json()
    assert response_data["email"] == TEST_EMAIL
    assert response_data["username"] == TEST_USERNAME

```

Projektsruktur

Codecovarage



Axklen / login-api / main

Coverage | Flags | Commits | Pulls | Settings

Branch Context

main

Source: latest commit da323d8

Coverage on branch

80.88%

165 of 204 lines covered

7 Days trend

+80.88%

Show Chart

Code tree | File list | 7 total files | login-api

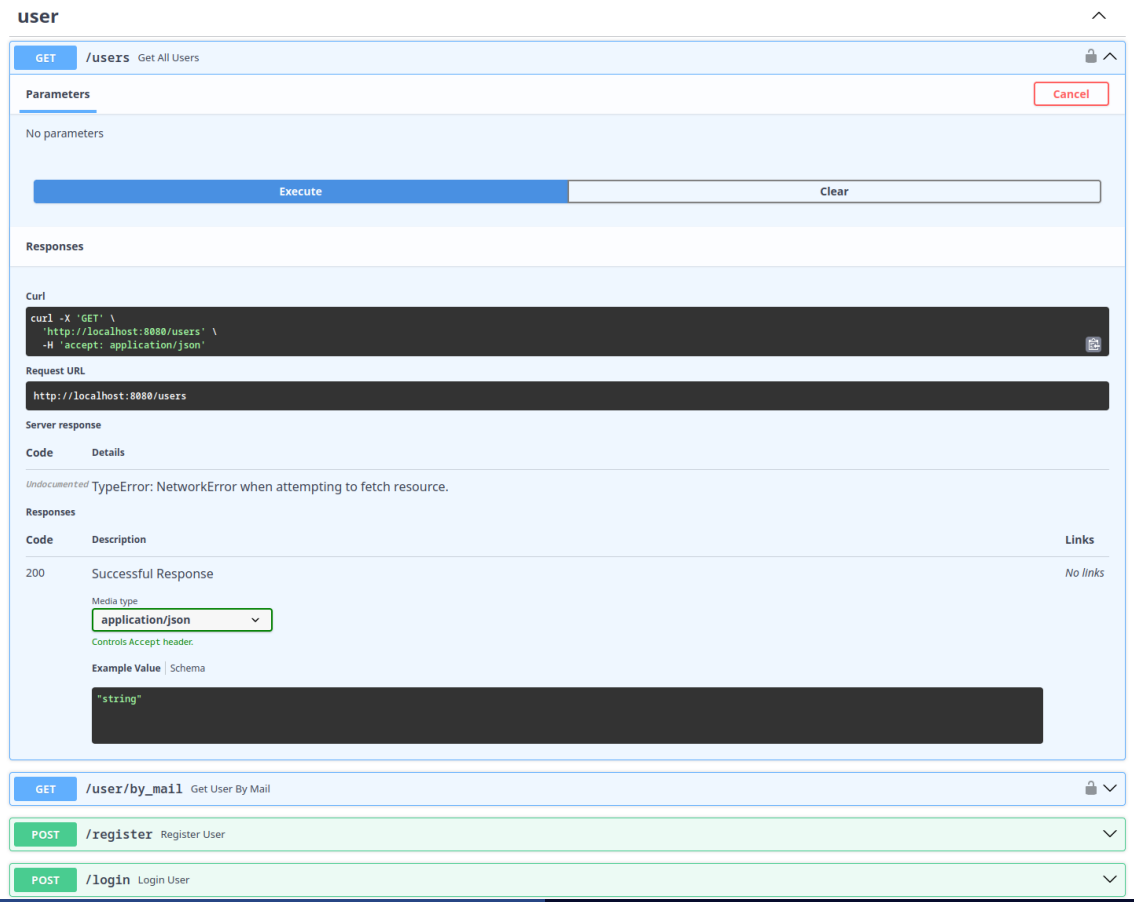
Search for files

Files	Tracked lines	Covered	Partial	Missed	Coverage %
login_api/auth/auth_handler.py	40	24	2	14	60.00%
login_api/main.py	73	59	7	7	80.82%
login_api/db/database.py	13	9	0	4	69.23%
login_api/db/crud.py	47	42	2	3	89.36%
login_api/auth/pwd_handler.py	6	6	0	0	100.00%
login_api/db/models.py	13	13	0	0	100.00%
login_api/db/schemas.py	12	12	0	0	100.00%

Testen & Bereitstellen

Die Zitter-Phase

Manualles Testen - `swagger-UI`



Bereitstellung

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

Uebergabe

Fazit

Danksagung