Tugas pertemuan pemrograman server 10 – 12

Nama Uut Prasetia

Nim a11.2019.11867

Pertemuan 10 PEMBUATAN REST API MENGGUNAKAN DJANGO NINJA PADA APLIKASI SIMPLE LMS

1. **Pendahuluan**
   Pada pertemuan ke-10 praktikum Pemrograman Server dilakukan pengembangan backend aplikasi Simple Learning Management System (LMS) dengan fokus pada pembuatan REST API menggunakan Django Ninja. REST API ini berfungsi sebagai penghubung antara backend dengan frontend agar data dapat diakses secara terstruktur.

2. **Dasar Teori**

   REST API adalah layanan berbasis web yang memungkinkan pertukaran data menggunakan protokol HTTP. Django Ninja merupakan framework berbasis Django yang mempermudah pembuatan API dengan performa tinggi dan dokumentasi otomatis.

   Metode HTTP yang digunakan antara lain GET dan POST untuk mengambil dan menambahkan data.

3. **Langkah-nya pratikum**

   Pembuatan Schema

   Schema digunakan untuk mendefinisikan format data yang dikirim dan diterima oleh API. Schema dibuat pada file schemas.py.

```python
from ninja import Schema
from datetime import datetime


class UserSchema(Schema):
    id: int
    username: str
    email: str
    role: str


class CourseSchema(Schema):
    id: int
    title: str
    description: str
    instructor_id: int
```

```python
class CourseCreateSchema(Schema):
    title: str
    description: str
    instructor_id: int


class LessonSchema(Schema):
    id: int
    title: str
    content: str
    course_id: int


class LessonCreateSchema(Schema):
    title: str
    content: str
    course_id: int


class AssignmentSchema(Schema):
    id: int
    title: str
    deadline: datetime
    course_id: int


class AssignmentCreateSchema(Schema):
    title: str
    deadline: datetime
    course_id: int


class SubmissionSchema(Schema):
    id: int
    answer: str
    grade: int | None
    student_id: int
    assignment_id: int


class SubmissionCreateSchema(Schema):
    answer: str
    student_id: int
    assignment_id: int
```

Pembuatan File api.py

```python
from ninja import Router
```

```python
from ninja.security import HttpBearer
from django.shortcuts import get_object_or_404
from django.contrib.auth import authenticate
from ninja.errors import HttpError

from .models import User, Course, Lesson, Assignment, Submission
from .schemas import (
    UserSchema,
    CourseSchema, CourseCreateSchema,
    LessonSchema, LessonCreateSchema,
    AssignmentSchema, AssignmentCreateSchema,
    SubmissionSchema, SubmissionCreateSchema
)
from .auth import create_token, JWTAuth

router = Router()


# ======================
# RBAC HELPER
# ======================
def allow_roles(*roles):
    def checker(request):
        user = request.user
        if not user:
            raise HttpError(401, "Unauthorized")

        if user.role not in roles:
            raise HttpError(403, "Forbidden: insufficient role")

        return True
    return checker


# ======================
# AUTH / LOGIN (JWT)
# ======================
@router.post("/login")
def login(request, username: str, password: str):
    user = authenticate(username=username, password=password)
    if not user:
        return {"error": "Invalid username or password"}

    token = create_token(user)
    return {
        "access_token": token,
        "token_type": "bearer",
        "role": user.role,
    }
```

```python
# ======================
# USERS (ADMIN ONLY)
# ======================
@router.get("/users", response=list[UserSchema], auth=JWTAuth())
def list_users(request):
    allow_roles("admin")(request)
    return User.objects.all()


# ======================
# COURSES
# ======================
@router.get("/courses", response=list[CourseSchema])
def list_courses(request):
    return Course.objects.all()


@router.post("/courses", response=CourseSchema, auth=JWTAuth())
def create_course(request, data: CourseCreateSchema):
    allow_roles("admin", "dosen")(request)

    instructor = get_object_or_404(User, id=data.instructor_id)
    return Course.objects.create(
        title=data.title,
        description=data.description,
        instructor=instructor
    )


# ======================
# LESSONS
# ======================
@router.get("/lessons", response=list[LessonSchema])
def list_lessons(request):
    return Lesson.objects.all()


@router.post("/lessons", response=LessonSchema, auth=JWTAuth())
def create_lesson(request, data: LessonCreateSchema):
    allow_roles("admin", "dosen")(request)

    course = get_object_or_404(Course, id=data.course_id)
    return Lesson.objects.create(
        title=data.title,
        content=data.content,
        course=course
    )
```

```python
# ======================
# ASSIGNMENTS
# ======================
@router.get("/assignments", response=list[AssignmentSchema])
def list_assignments(request):
    return Assignment.objects.all()


@router.post("/assignments", response=AssignmentSchema, auth=JWTAuth())
def create_assignment(request, data: AssignmentCreateSchema):
    allow_roles("admin", "dosen")(request)

    course = get_object_or_404(Course, id=data.course_id)
    return Assignment.objects.create(
        title=data.title,
        deadline=data.deadline,
        course=course
    )


# ======================
# SUBMISSIONS
# ======================
@router.get("/submissions", response=list[SubmissionSchema], auth=JWTAuth())
def list_submissions(request):
    allow_roles("admin", "dosen")(request)
    return Submission.objects.all()


@router.post("/submissions", response=SubmissionSchema, auth=JWTAuth())
def create_submission(request, data: SubmissionCreateSchema):
    allow_roles("mahasiswa")(request)

    student = get_object_or_404(User, id=data.student_id)
    assignment = get_object_or_404(Assignment, id=data.assignment_id)

    return Submission.objects.create(
        answer=data.answer,
        student=student,
        assignment=assignment
    )
```

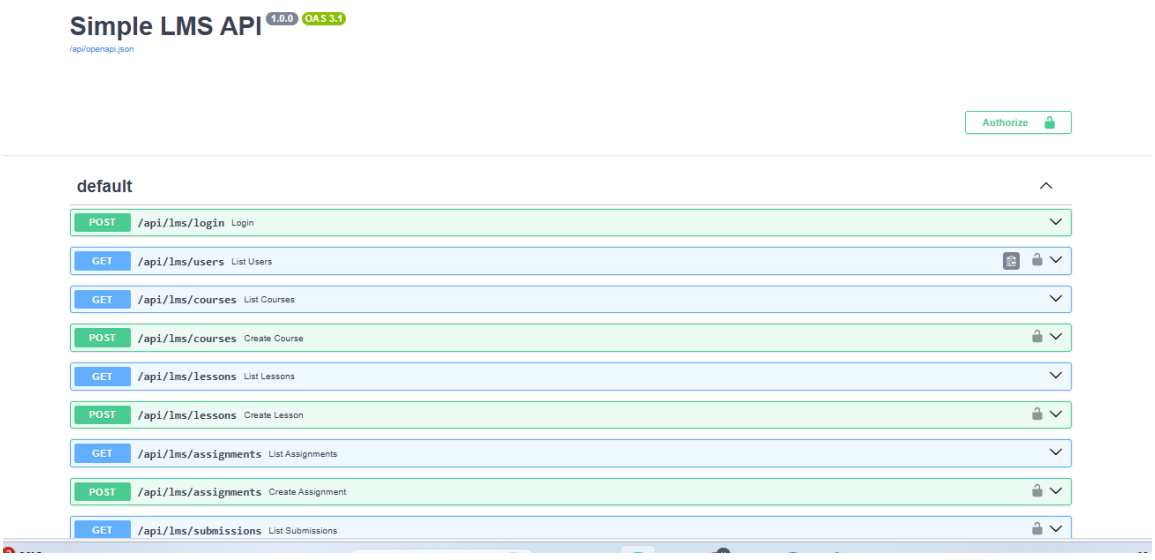
Registrasi Api ke projek urls.py

```python
"""
URL configuration for simple_lms project.
```

```
The `urlpatterns` list routes URLs to views. For more information please see:
https://docs.djangoproject.com/en/6.0/topics/http/urls/
"""

from django.contrib import admin
from django.urls import path
from ninja import NinjaAPI
from lms.api import router as lms_router

# Inisialisasi API
api = NinjaAPI(title="Simple LMS API")

# Register router LMS
api.add_router("/lms/", lms_router)

urlpatterns = [
    path("admin/", admin.site.urls),
    path("api/", api.urls),
]
```

4. **Bukti Akses Dokumentasi API**

   http://localhost:8000/api/docs



Pertemuan 11 IMPLEMENTASI AUTENTIKASI DAN OTORISASI MENGGUNAKAN JWT PADA APLIKASI SIMPLE LMS

1. **Pendahuluan**

   Pemrograman Server dilakukan pengembangan lanjutan pada aplikasi backend Simple Learning Management System (LMS) dengan menambahkan fitur autentikasi dan otorisasi pengguna menggunakan JSON Web Token (JWT). Fitur ini diperlukan untuk mengamankan endpoint API agar hanya dapat diakses oleh user yang telah login.

   Autentikasi digunakan untuk memverifikasi identitas pengguna, sedangkan otorisasi digunakan untuk menentukan hak akses pengguna berdasarkan perannya.

2. **Dasar Teori**

   JSON Web Token (JWT) adalah standar autentikasi berbasis token yang digunakan untuk pertukaran data secara aman. JWT terdiri dari tiga bagian utama, yaitu header, payload, dan signature. Token ini dikirimkan melalui header Authorization pada setiap request API.

   JWT banyak digunakan pada REST API karena bersifat stateless dan lebih aman dibandingkan autentikasi berbasis session.

3. **Langkah Praktikum**

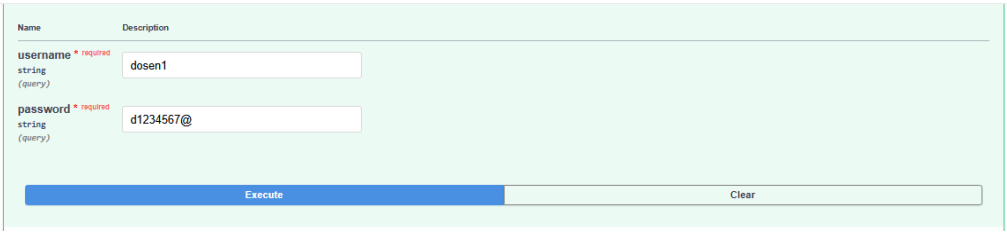
   Fungsi login dibuat untuk memverifikasi username dan password pengguna. Jika login berhasil, sistem akan menghasilkan token JWT.

   Di auth.py

```python
import jwt
from datetime import datetime, timedelta
from django.conf import settings
from ninja.security import HttpBearer
from django.contrib.auth import authenticate
from .models import User

JWT_SECRET = settings.SECRET_KEY
JWT_ALGORITHM = "HS256"
JWT_EXPIRE_MINUTES = 60


def create_token(user: User):
    payload = {
        "user_id": user.id,
        "username": user.username,
        "role": user.role,
        "exp": datetime.utcnow() + timedelta(minutes=JWT_EXPIRE_MINUTES),
        "iat": datetime.utcnow(),
```

```
    }
    return jwt.encode(payload, JWT_SECRET, algorithm=JWT_ALGORITHM)


class JWTAuth(HttpBearer):
    def authenticate(self, request, token):
        try:
            payload = jwt.decode(
                token, JWT_SECRET, algorithms=[JWT_ALGORITHM]
            )
            user = User.objects.get(id=payload["user_id"])
            request.user = user
            return user
        except Exception:
            return None
```

4. **Bukti Login menggunakan JWT**

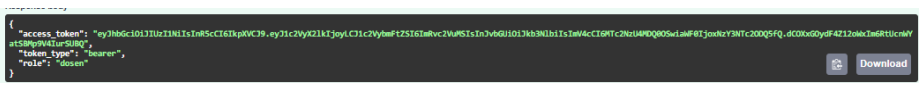


Endpoint login diuji melalui Swagger UI dengan memasukkan username dan password. Sistem berhasil mengembalikan token JWT sebagai bukti autentikasi berhasil.

5. **Bukti Akses Endpoint Terproteksi**

Endpoint yang dilindungi JWT tidak dapat diakses tanpa token. Setelah token dimasukkan ke menu Authorize pada Swagger UI, endpoint dapat diakses dengan normal.

6. **Bukti Role User**



Response login menampilkan role user seperti admin, dosen, atau mahasiswa. Hal ini membuktikan bahwa sistem otorisasi berbasis role berjalan dengan baik.

7. KESIMPULAN

Dari praktikum pertemuan ke-11 dapat disimpulkan bahwa JWT merupakan metode autentikasi yang efektif untuk REST API. Implementasi JWT pada aplikasi Simple LMS berhasil mengamankan endpoint API dan mengatur hak akses pengguna.

Pertemuan 12 IMPLEMENTASI DOCKER DAN PENGUJIAN APLIKASI SIMPLE LMS

1. Pendahuluan

Pemrograman Server dilakukan tahap akhir pengembangan aplikasi backend Simple Learning Management System (LMS), yaitu dengan menerapkan Docker dan Docker Compose untuk menjalankan aplikasi secara terisolasi serta melakukan pengujian aplikasi secara keseluruhan.

Penggunaan Docker bertujuan agar aplikasi dapat dijalankan dengan mudah di berbagai lingkungan tanpa perlu melakukan konfigurasi ulang secara manual.

2. Dasar Teori

Docker adalah platform yang digunakan untuk menjalankan aplikasi di dalam container. Container bersifat ringan, terisolasi, dan portable. Docker Compose digunakan untuk mengatur dan menjalankan beberapa container secara bersamaan, seperti aplikasi web dan database atau service pendukung lainnya.

Dengan Docker, aplikasi dapat dijalankan di lingkungan yang konsisten, baik di laptop pengembang maupun di laptop pengguna lain.

3. Langkah praktikum

Membuat Dockerfile

```
FROM python:3.12-slim

ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1

WORKDIR /app

# dependensi OS minimal
```

```
RUN apt-get update && apt-get install -y --no-install-recommends \
    bash curl \
  && rm -rf /var/lib/apt/lists/*

COPY requirements.txt /app/requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

COPY docker/entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh

# copy project code (nanti folder app/)
COPY app /app

EXPOSE 8000

ENTRYPOINT ["/entrypoint.sh"]
```

4. **Konfigurasi menggunakan docker-compose.yml**

```
services:
  web:
    build:
      context: .
      dockerfile: docker/Dockerfile
    container_name: simple_lms_web
    env_file:
      - .env
    volumes:
      - ./app:/app
    ports:
      - "8000:8000"
    depends_on:
      - redis

  redis:
    image: redis:7
    container_name: simple_lms_redis
    ports:
      - "6379:6379"
```

5. **Menjalankan Aplikasi dengan**

   docker compose up –build

tunggu proses build sampai selesai

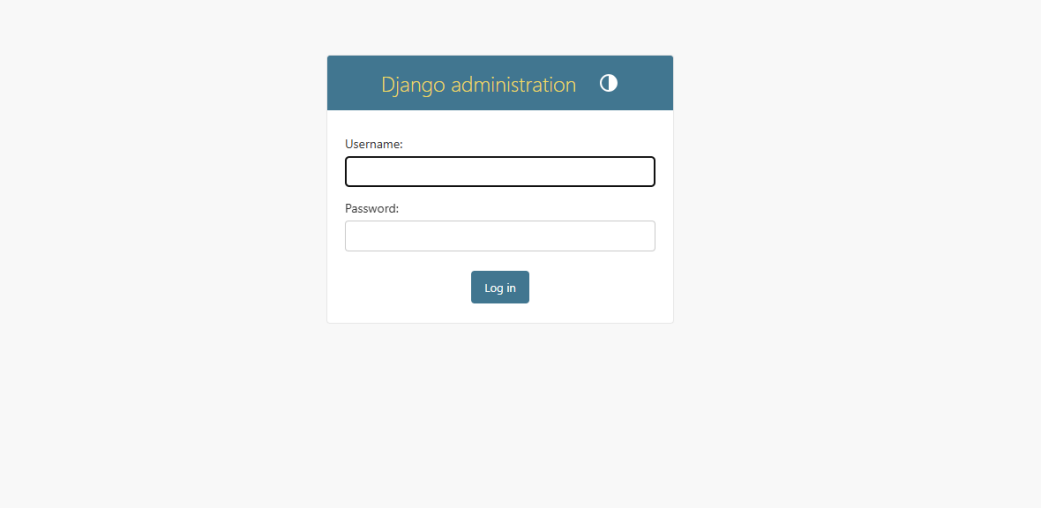
6. **Bukti Praktikum**

   Ini local host yg di pakai :

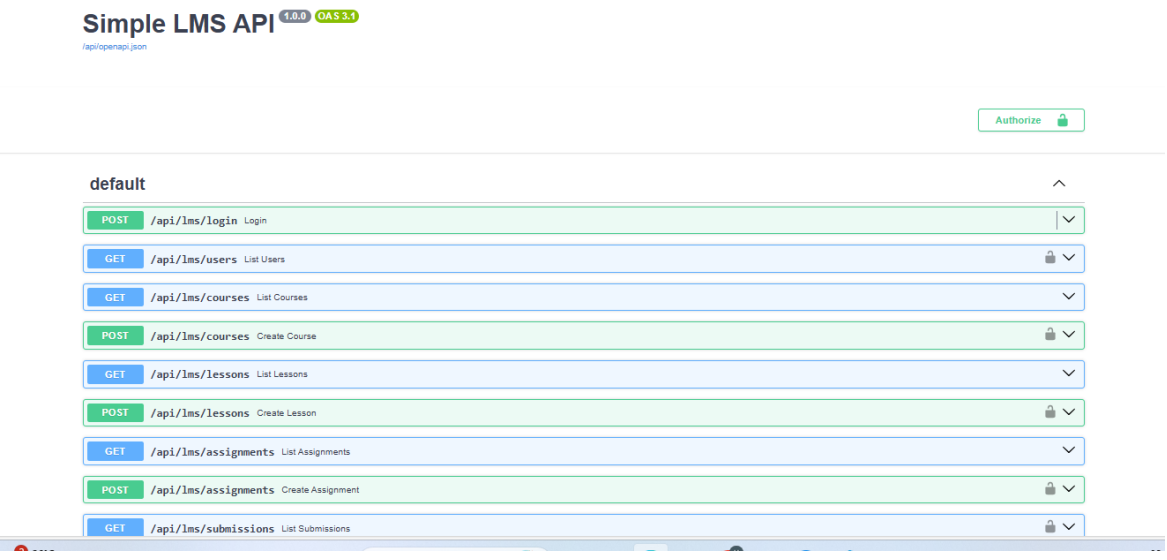
   http://localhost:8000

7. **Bukti akses admin dan api**

   http://localhost:8000/admin



   http://localhost:8000/api/docs



8. **Pengujian system di terminal**

   python manage.py check

9. Hasil dan pembahasan

Hasil dari praktikum pertemuan ke-12 adalah aplikasi backend Simple LMS berhasil dijalankan menggunakan Docker dan Docker Compose. Seluruh fitur utama seperti autentikasi JWT, API, dan admin Django dapat berjalan dengan baik.
Dengan menggunakan Docker, aplikasi dapat dijalankan kembali di perangkat lain tanpa konfigurasi tambahan.