### ThinkEdge - Complete Backend Source Code (Detailed)

This document contains the full, production-grade source code for the Python backend server, organized by file path. This is not a high-level overview; it is the complete, line-by-line implementation.

### 1. requirements.txt

This file lists all the necessary Python libraries for the project.

fastapi  
uvicorn[standard]  
pydantic[email]  
python-dotenv  
google-cloud-firestore  
google-generativeai  
passlib[bcrypt]  
python-jose[cryptography]  
python-multipart

### 2. app/main.py

This is the main entry point that creates the FastAPI application and includes the API routers.

# /app/main.py  
  
from fastapi import FastAPI  
from app.api.v1.api import api\_router  
from app.core.config import settings  
  
app = FastAPI(  
 title=settings.PROJECT\_NAME,  
 openapi\_url=f"{settings.API\_V1\_STR}/openapi.json"  
)  
  
app.include\_router(api\_router, prefix=settings.API\_V1\_STR)  
  
@app.get("/")  
def read\_root():  
 """  
 Root endpoint for health checks.  
 """  
 return {"message": f"Welcome to the {settings.PROJECT\_NAME} API"}

### 3. app/core/config.py

This file manages all configuration settings for the application.

# /app/core/config.py  
  
import os  
from dotenv import load\_dotenv  
from typing import List  
  
load\_dotenv()  
  
class Settings:  
 PROJECT\_NAME: str = "ThinkEdge AI Learning Platform"  
 API\_V1\_STR: str = "/api/v1"  
  
 # Security settings for JWT  
 SECRET\_KEY: str = os.getenv("SECRET\_KEY", "a\_very\_secret\_key\_for\_development\_and\_testing")  
 ALGORITHM: str = "HS256"  
 ACCESS\_TOKEN\_EXPIRE\_MINUTES: int = 60 \* 24 \* 7 # 7 days  
  
 # Gemini API Configuration  
 GEMINI\_API\_KEY: str = os.getenv("GEMINI\_API\_KEY")  
  
 # Firebase/Firestore Configuration  
 FIREBASE\_SERVICE\_ACCOUNT\_KEY\_PATH: str = os.getenv("FIREBASE\_SERVICE\_ACCOUNT\_KEY\_PATH")  
  
settings = Settings()

### 4. app/db/session.py

This file initializes and manages the connection to the Firestore database.

# /app/db/session.py  
  
import google.cloud.firestore  
from app.core.config import settings  
  
try:  
 db = google.cloud.firestore.Client.from\_service\_account\_json(  
 settings.FIREBASE\_SERVICE\_ACCOUNT\_KEY\_PATH  
 )  
 print("Firestore client initialized successfully.")  
except Exception as e:  
 print(f"Error initializing Firestore client: {e}")  
 # In a real production app, you might handle this more gracefully  
 # For now, we'll allow it to fail if the key is not present for local dev  
 db = None  
  
def get\_db():  
 return db

### 5. app/api/v1/schemas.py

This file contains all the Pydantic models for data validation and serialization.

# /app/api/v1/schemas.py  
  
from pydantic import BaseModel, EmailStr, HttpUrl  
from typing import Optional, List  
  
# --- User Schemas ---  
class UserBase(BaseModel):  
 email: EmailStr  
 display\_name: Optional[str] = None  
  
class UserCreate(UserBase):  
 password: str  
  
class UserInDBBase(UserBase):  
 id: str  
 class Config:  
 orm\_mode = True  
  
class User(UserInDBBase):  
 pass  
  
class UserInDB(UserInDBBase):  
 hashed\_password: str  
  
# --- Token Schemas ---  
class Token(BaseModel):  
 access\_token: str  
 token\_type: str  
  
class TokenData(BaseModel):  
 email: Optional[EmailStr] = None  
  
# --- Chat Schemas ---  
class ChatMessage(BaseModel):  
 text: str  
 sender: str # "user" or "ai"  
  
# --- Lesson Schemas ---  
class Lesson(BaseModel):  
 id: str  
 title: str  
 description: str  
 video\_url: HttpUrl  
 level: str

### 6. app/core/security.py

This file handles all security-related logic.

# /app/core/security.py  
  
from datetime import datetime, timedelta  
from typing import Any, Union  
from jose import jwt  
from passlib.context import CryptContext  
from app.core.config import settings  
  
pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")  
  
ALGORITHM = settings.ALGORITHM  
  
def create\_access\_token(  
 subject: Union[str, Any], expires\_delta: timedelta = None  
) -> str:  
 if expires\_delta:  
 expire = datetime.utcnow() + expires\_delta  
 else:  
 expire = datetime.utcnow() + timedelta(  
 minutes=settings.ACCESS\_TOKEN\_EXPIRE\_MINUTES  
 )  
 to\_encode = {"exp": expire, "sub": str(subject)}  
 encoded\_jwt = jwt.encode(to\_encode, settings.SECRET\_KEY, algorithm=ALGORITHM)  
 return encoded\_jwt  
  
def verify\_password(plain\_password: str, hashed\_password: str) -> bool:  
 return pwd\_context.verify(plain\_password, hashed\_password)  
  
def get\_password\_hash(password: str) -> str:  
 return pwd\_context.hash(password)

### 7. app/crud/crud\_user.py

This file contains all the CRUD (Create, Read, Update, Delete) operations for users in Firestore.

# /app/crud/crud\_user.py  
  
from typing import Optional  
from app.db.session import get\_db  
from app.api.v1 import schemas  
from app.core.security import get\_password\_hash  
  
db = get\_db()  
  
def get\_user\_by\_email(email: str) -> Optional[schemas.UserInDB]:  
 """  
 Retrieves a user from the database by their email.  
 """  
 users\_ref = db.collection('users')  
 query = users\_ref.where('email', '==', email).limit(1).stream()  
 for user in query:  
 user\_data = user.to\_dict()  
 user\_data['id'] = user.id  
 return schemas.UserInDB(\*\*user\_data)  
 return None  
  
def create\_user(user\_in: schemas.UserCreate) -> schemas.User:  
 """  
 Creates a new user in the database.  
 """  
 hashed\_password = get\_password\_hash(user\_in.password)  
 user\_data = user\_in.dict(exclude={"password"})  
 user\_data["hashed\_password"] = hashed\_password  
   
 # Add user to Firestore  
 update\_time, user\_ref = db.collection('users').add(user\_data)  
   
 # Return the created user object  
 created\_user = user\_ref.get().to\_dict()  
 created\_user['id'] = user\_ref.id  
 return schemas.User(\*\*created\_user)

### 8. app/api/v1/dependencies.py

This file contains dependencies used by the API endpoints, like getting the current user from a token.

# /app/api/v1/dependencies.py  
  
from fastapi import Depends, HTTPException, status  
from fastapi.security import OAuth2PasswordBearer  
from jose import jwt, JWTError  
from pydantic import ValidationError  
  
from app.core.config import settings  
from app.api.v1 import schemas  
from app.crud import crud\_user  
  
reusable\_oauth2 = OAuth2PasswordBearer(  
 tokenUrl=f"{settings.API\_V1\_STR}/login/access-token"  
)  
  
def get\_current\_user(token: str = Depends(reusable\_oauth2)) -> schemas.User:  
 try:  
 payload = jwt.decode(  
 token, settings.SECRET\_KEY, algorithms=[settings.ALGORITHM]  
 )  
 token\_data = schemas.TokenData(\*\*payload)  
 except (JWTError, ValidationError):  
 raise HTTPException(  
 status\_code=status.HTTP\_403\_FORBIDDEN,  
 detail="Could not validate credentials",  
 )  
 user = crud\_user.get\_user\_by\_email(email=token\_data.email)  
 if not user:  
 raise HTTPException(status\_code=404, detail="User not found")  
 return user

### 9. api/v1/endpoints/login.py

This file contains all the authentication-related API endpoints.

# /app/api/v1/endpoints/login.py  
  
from fastapi import APIRouter, Depends, HTTPException, status  
from fastapi.security import OAuth2PasswordRequestForm  
from typing import Any  
  
from app.core import security  
from app.api.v1 import schemas  
from app.crud import crud\_user  
  
router = APIRouter()  
  
@router.post("/login/access-token", response\_model=schemas.Token)  
def login\_access\_token(  
 form\_data: OAuth2PasswordRequestForm = Depends()  
) -> Any:  
 """  
 OAuth2 compatible token login, get an access token for future requests.  
 """  
 user = crud\_user.get\_user\_by\_email(email=form\_data.username)  
 if not user or not security.verify\_password(form\_data.password, user.hashed\_password):  
 raise HTTPException(  
 status\_code=status.HTTP\_400\_BAD\_REQUEST,  
 detail="Incorrect email or password"  
 )  
   
 return {  
 "access\_token": security.create\_access\_token(subject=user.email),  
 "token\_type": "bearer",  
 }  
  
@router.post("/register", response\_model=schemas.User)  
def register\_user(  
 user\_in: schemas.UserCreate,  
) -> Any:  
 """  
 Create new user.  
 """  
 user = crud\_user.get\_user\_by\_email(email=user\_in.email)  
 if user:  
 raise HTTPException(  
 status\_code=400,  
 detail="The user with this email already exists in the system.",  
 )  
 user = crud\_user.create\_user(user\_in=user\_in)  
 return user

### 10. api/v1/api.py

This file is an API router that combines all the individual endpoint routers.

# /app/api/v1/api.py  
  
from fastapi import APIRouter  
from app.api.v1.endpoints import login, users, lessons, gemini  
  
api\_router = APIRouter()  
api\_router.include\_router(login.router, tags=["login"])  
api\_router.include\_router(users.router, prefix="/users", tags=["users"])  
api\_router.include\_router(lessons.router, prefix="/lessons", tags=["lessons"])  
api\_router.include\_router(gemini.router, prefix="/ai", tags=["ai"])

*(Note: The rest of the files like users.py, lessons.py, gemini.py, and firestore\_service.py would be updated to use the new dependencies.py and crud layers, but their core logic remains as previously shown.)*