PsychSync - FastAPI API Framework Setup Guide

Epic: Core Infrastructure API Framework Setup

Assignee: Backend Developer 1 | **Effort:** 12h | **Priority:** High

Prerequisites

- Python 3.9+
- Git
- Virtual environment tool (venv, conda, or virtualenv)
- Code editor (VS Code, PyCharm)

Step 1: Organize Existing Project Structure (1-2 hours)

Looking at your current structure, you have a good foundation but need to organize it properly. Your current structure shows:

```
psychsync/
 ├── backend/
|   ├── app/
|  |   ├── api/
|  |   ├── core/
|  |   ├── db/
|  |   └── schemas/
|   └── main.py
```

1.1 Complete Missing Directory Structure

From your `backend/` directory, run:

```bash
```

```
cd backend/app

# Complete the missing API structure
mkdir -p api/dependencies
mkdir -p api/v1/endpoints

# Ensure core security exists
touch core/security.py

# Complete services structure
mkdir -p services/{auth,team,assessment}

# Ensure tests directory is properly set up
mkdir -p tests/{unit,integration}
```

1.2 Update Your Existing Requirements

Looking at your existing `backend/requirements.txt`, you'll need to add missing packages. Update your `backend/requirements.txt`:

```text
# Add these to your existing requirements.txt
python-jose[cryptography]==3.3.0
passlib[bcrypt]==1.7.4
python-multipart==0.0.6
```

Create `backend/requirements-dev.txt` if it doesn't exist:

```text
-r requirements.txt
black==23.10.1
isort==5.12.0
flake8==6.1.0
mypy==1.7.1
pytest==7.4.3
pytest-asyncio==0.21.1
```

1.3 Install Missing Dependencies

From your `backend/` directory:

```bash
```

```
# Ensure you're in the backend directory
cd backend

# Install new dependencies
pip install python-jose[cryptography] passlib[bcrypt] python-multipart
pip install -r requirements-dev.txt
```

1.4 Verify Your Environment

Create `requirements.txt`:

```text
fastapi==0.104.1
uvicorn[standard]==0.24.0
pydantic==2.4.2
pydantic-settings==2.0.3
python-jose[cryptography]==3.3.0
passlib[bcrypt]==1.7.4
python-multipart==0.0.6
sqlalchemy==2.0.23
alembic==1.12.1
psycopg2-binary==2.9.9
redis==5.0.1
celery==5.3.4
httpx==0.25.2
pytest==7.4.3
pytest-asyncio==0.21.1
```

Create `requirements-dev.txt`:

```text
-r requirements.txt
black==23.10.1
isort==5.12.0
flake8==6.1.0
mypy==1.7.1
pre-commit==3.5.0
```

Install dependencies:

```bash
pip install -r requirements-dev.txt
```

Step 2: Configure Your Existing Config.py (1 hour)

Your existing `backend/app/core/config.py` needs to be updated with authentication settings. Here's the enhanced version:

python

```python
from pydantic_settings import BaseSettings
from typing import Optional, List
import secrets
import os


class Settings(BaseSettings):
    # API Settings
    API_V1_STR: str = "/api/v1"
    PROJECT_NAME: str = "PsychSync API"
    PROJECT_DESCRIPTION: str = "Team Psychology & Optimization Platform API"
    VERSION: str = "1.0.0"

    # Security
    SECRET_KEY: str = secrets.token_urlsafe(32)
    ACCESS_TOKEN_EXPIRE_MINUTES: int = 30
    REFRESH_TOKEN_EXPIRE_DAYS: int = 7
    ALGORITHM: str = "HS256"

    # Database - Use your existing database URL
    DATABASE_URL: str = os.getenv("DATABASE_URL", "postgresql://user:password@localhost/psychsync")

    # Redis
    REDIS_URL: str = "redis://localhost:6379"

    # CORS
    BACKEND_CORS_ORIGINS: List[str] = [
        "http://localhost:3000",  # Your frontend
        "http://localhost:8000",
        "http://localhost:5173"   # Vite default port
    ]

    # Email (for future use)
    SMTP_TLS: bool = True
    SMTP_PORT: Optional[int] = None
    SMTP_HOST: Optional[str] = None
    SMTP_USER: Optional[str] = None
    SMTP_PASSWORD: Optional[str] = None

    # Environment
    ENVIRONMENT: str = "development"
    DEBUG: bool = True

    class Config:
        env_file = ".env"
        case_sensitive = True
```

```
settings = Settings()
```

Create or update `backend/.env`:

```env
SECRET_KEY=your-super-secret-key-here-change-this-in-production
DATABASE_URL=postgresql://your_existing_db_url_here
REDIS_URL=redis://localhost:6379
ENVIRONMENT=development
DEBUG=true
```

Create `app/core/database.py`:

```python
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from app.core.config import settings

engine = create_engine(settings.DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()


def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Step 3: Authentication Middleware Setup (3-4 hours)

3.1 Create Security Utilities

Create `app/core/security.py`:

```python
```

```python
from datetime import datetime, timedelta
from typing import Any, Union, Optional
from jose import jwt, JWTError
from passlib.context import CryptContext
from fastapi import HTTPException, status
from app.core.config import settings

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")


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), "type": "access"}
    encoded_jwt = jwt.encode(to_encode, settings.SECRET_KEY, algorithm=settings.ALGORITHM)
    return encoded_jwt


def create_refresh_token(subject: Union[str, Any]) -> str:
    expire = datetime.utcnow() + timedelta(days=settings.REFRESH_TOKEN_EXPIRE_DAYS)
    to_encode = {"exp": expire, "sub": str(subject), "type": "refresh"}
    encoded_jwt = jwt.encode(to_encode, settings.SECRET_KEY, algorithm=settings.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)


def verify_token(token: str, token_type: str = "access") -> Optional[str]:
    try:
        payload = jwt.decode(
            token, settings.SECRET_KEY, algorithms=[settings.ALGORITHM]
        )
        user_id: str = payload.get("sub")
        token_type_payload: str = payload.get("type")
        if user_id is None or token_type_payload != token_type:
            return None
        return user_id
    except JWTError:
        return None
```

3.2 Create Authentication Dependencies

Create `app/api/dependencies/auth.py`:

```
python
```

```python
from typing import Optional
from fastapi import Depends, HTTPException, status
from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials
from sqlalchemy.orm import Session
from app.core.database import get_db
from app.core.security import verify_token
from app.models.user import User

security = HTTPBearer()


async def get_current_user(
    credentials: HTTPAuthorizationCredentials = Depends(security),
    db: Session = Depends(get_db)
) -> User:
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )

    token = credentials.credentials
    user_id = verify_token(token, "access")

    if user_id is None:
        raise credentials_exception

    user = db.query(User).filter(User.id == user_id).first()
    if user is None:
        raise credentials_exception

    return user


async def get_current_active_user(
    current_user: User = Depends(get_current_user),
) -> User:
    if not current_user.is_active:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user


# Optional admin user dependency
async def get_current_superuser(
    current_user: User = Depends(get_current_user),
) -> User:
    if not current_user.is_superuser:
        raise HTTPException(
            status_code=400, detail="The user doesn't have enough privileges"
        )
    return current_user
```

3.3 Create User Model (Basic)

Create `app/models/__init__.py`:

```python
# Empty file to make models a package
```

Create `app/models/user.py`:

```python
from sqlalchemy import Boolean, Column, Integer, String, DateTime
from sqlalchemy.sql import func
from app.core.database import Base


class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    email = Column(String, unique=True, index=True, nullable=False)
    hashed_password = Column(String, nullable=False)
    first_name = Column(String)
    last_name = Column(String)
    is_active = Column(Boolean, default=True)
    is_superuser = Column(Boolean, default=False)
    created_at = Column(DateTime(timezone=True), server_default=func.now())
    updated_at = Column(DateTime(timezone=True), onupdate=func.now())
```

Step 4: FastAPI Application Setup (2-3 hours)

4.1 Create Pydantic Schemas

Create `app/schemas/__init__.py`:

```python
# Empty file to make schemas a package
```

Create `app/schemas/user.py`:

```python
from typing import Optional
from pydantic import BaseModel, EmailStr
from datetime import datetime


class UserBase(BaseModel):
    email: EmailStr
    first_name: Optional[str] = None
    last_name: Optional[str] = None
    is_active: bool = True


class UserCreate(UserBase):
    password: str


class UserUpdate(UserBase):
    password: Optional[str] = None


class UserInDBBase(UserBase):
    id: int
    created_at: datetime
    updated_at: Optional[datetime] = None

    class Config:
        from_attributes = True


class User(UserInDBBase):
    pass


class UserInDB(UserInDBBase):
    hashed_password: str
```

Create `app/schemas/auth.py`:

```python
```

```python
from pydantic import BaseModel
from typing import Optional


class Token(BaseModel):
    access_token: str
    refresh_token: str
    token_type: str


class TokenPayload(BaseModel):
    sub: Optional[str] = None


class LoginRequest(BaseModel):
    email: str
    password: str


class RefreshTokenRequest(BaseModel):
    refresh_token: str
```

4.2 Create Main FastAPI Application

Create `app/main.py`:

```
python
```

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from app.core.config import settings
from app.api.v1.api import api_router

app = FastAPI(
    title=settings.PROJECT_NAME,
    description=settings.PROJECT_DESCRIPTION,
    version=settings.VERSION,
    openapi_url=f"{settings.API_V1_STR}/openapi.json" if settings.DEBUG else None,
    docs_url=f"{settings.API_V1_STR}/docs" if settings.DEBUG else None,
    redoc_url=f"{settings.API_V1_STR}/redoc" if settings.DEBUG else None,
)

# Set up CORS middleware
app.add_middleware(
    CORSMiddleware,
    allow_origins=settings.BACKEND_CORS_ORIGINS,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Include API router
app.include_router(api_router, prefix=settings.API_V1_STR)


@app.get("/")
async def root():
    return {
        "message": "Welcome to PsychSync API",
        "version": settings.VERSION,
        "docs": f"{settings.API_V1_STR}/docs" if settings.DEBUG else "Not available in production"
    }


@app.get("/health")
async def health_check():
    return {"status": "healthy", "service": "psychsync-api"}
```

4.3 Create API Router Structure

Create `app/api/__init__.py`:

```
python
```

```
# Empty file to make api a package
```

Create `app/api/v1/__init__.py`:

```python
# Empty file to make v1 a package
```

Create `app/api/v1/api.py`:

```python
from fastapi import APIRouter
from app.api.v1.endpoints import auth, users

api_router = APIRouter()

api_router.include_router(auth.router, prefix="/auth", tags=["authentication"])
api_router.include_router(users.router, prefix="/users", tags=["users"])
```

4.4 Create Auth Endpoints

Create `app/api/v1/endpoints/__init__.py`:

```python
# Empty file to make endpoints a package
```

Create `app/api/v1/endpoints/auth.py`:

```python
```

```
from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.orm import Session
from app.core.database import get_db
from app.core.security import verify_password, create_access_token, create_refresh_token, verify_token
from app.models.user import User
from app.schemas.auth import Token, LoginRequest, RefreshTokenRequest

router = APIRouter()


@router.post("/login", response_model=Token)
async def login(
    login_data: LoginRequest,
    db: Session = Depends(get_db)
):
    user = db.query(User).filter(User.email == login_data.email).first()
    if not user or not verify_password(login_data.password, user.hashed_password):
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect email or password",
        )
    if not user.is_active:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Inactive user"
        )

    access_token = create_access_token(subject=user.id)
    refresh_token = create_refresh_token(subject=user.id)

    return {
        "access_token": access_token,
        "refresh_token": refresh_token,
        "token_type": "bearer"
    }


@router.post("/refresh", response_model=Token)
async def refresh_token(
    refresh_data: RefreshTokenRequest,
    db: Session = Depends(get_db)
):
    user_id = verify_token(refresh_data.refresh_token, "refresh")
    if not user_id:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid refresh token",
        )

    user = db.query(User).filter(User.id == user_id).first()
    if not user or not user.is_active:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid user",
        )

    access_token = create_access_token(subject=user.id)
    refresh_token = create_refresh_token(subject=user.id)

    return {
        "access_token": access_token,
        "refresh_token": refresh_token,
        "token_type": "bearer"
    }
```

Create `app/api/v1/endpoints/users.py`:

```
python
```

```python
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from typing import List
from app.core.database import get_db
from app.api.dependencies.auth import get_current_active_user
from app.models.user import User
from app.schemas.user import User as UserSchema

router = APIRouter()


@router.get("/me", response_model=UserSchema)
async def read_user_me(
    current_user: User = Depends(get_current_active_user)
):
    return current_user


@router.get("/", response_model=List[UserSchema])
async def read_users(
    skip: int = 0,
    limit: int = 100,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_active_user)
):
    users = db.query(User).offset(skip).limit(limit).all()
    return users
```

Step 5: OpenAPI Documentation Configuration (1-2 hours)

5.1 Enhanced OpenAPI Configuration

Update `app/main.py` to include comprehensive OpenAPI configuration:

```
python
```

```python
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from fastapi.openapi.utils import get_openapi
from app.core.config import settings
from app.api.v1.api import api_router


def custom_openapi():
    if app.openapi_schema:
        return app.openapi_schema

    openapi_schema = get_openapi(
        title=settings.PROJECT_NAME,
        version=settings.VERSION,
        description="""
        ## PsychSync API

        The PsychSync API provides endpoints for team psychology analysis and optimization.

        ### Features
        * **Authentication**: JWT-based authentication system
        * **Team Management**: Create and manage teams and members
        * **Assessments**: Psychological assessment frameworks (MBTI, Big Five, DISC)
        * **Analytics**: Team compatibility and performance analytics
        * **Optimization**: AI-powered team optimization recommendations

        ### Authentication
        Most endpoints require authentication. To authenticate:
        1. Use `/api/v1/auth/login` to get access and refresh tokens
        2. Include the access token in the Authorization header as `Bearer <token>`
        3. Use `/api/v1/auth/refresh` to refresh expired tokens

        ### Rate Limiting
        API endpoints are rate-limited to ensure fair usage and system stability.
        """,
        routes=app.routes,
    )

    # Add security scheme
    openapi_schema["components"]["securitySchemes"] = {
        "HTTPBearer": {
            "type": "http",
            "scheme": "bearer",
            "bearerFormat": "JWT",
        }
    }
    # Add global security requirement
    openapi_schema["security"] = [{"HTTPBearer": []}]

    app.openapi_schema = openapi_schema
    return app.openapi_schema


app = FastAPI(
    title=settings.PROJECT_NAME,
    description=settings.PROJECT_DESCRIPTION,
    version=settings.VERSION,
    openapi_url=f"{settings.API_V1_STR}/openapi.json" if settings.DEBUG else None,
    docs_url=f"{settings.API_V1_STR}/docs" if settings.DEBUG else None,
    redoc_url=f"{settings.API_V1_STR}/redoc" if settings.DEBUG else None,
    contact={
        "name": "PsychSync Team",
        "email": "api@psychsync.com",
    },
    license_info={
        "name": "MIT License",
    },
)

app.openapi = custom_openapi

# Set up CORS middleware
app.add_middleware(
    CORSMiddleware,
    allow_origins=settings.BACKEND_CORS_ORIGINS,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Include API router
app.include_router(api_router, prefix=settings.API_V1_STR)


@app.get("/")
async def root():
    return {
        "message": "Welcome to PsychSync API",
        "version": settings.VERSION,
        "docs": f"{settings.API_V1_STR}/docs" if settings.DEBUG else "Not available in production"
    }
```

```
@app.get("/health")
async def health_check():
    return {"status": "healthy", "service": "psychsync-api"}
```

Step 6: Database Setup (1 hour)

6.1 Initialize Alembic

```bash
# Initialize Alembic
alembic init alembic

# Edit alembic.ini to use your database URL
# Replace sqlalchemy.url = driver://user:pass@localhost/dbname
# with your actual database URL or use env variable
```

Edit `alembic/env.py`:

```python
from logging.config import fileConfig
from sqlalchemy import engine_from_config, pool
from alembic import context
from app.core.config import settings
from app.core.database import Base
from app.models import user  # Import all models here

# this is the Alembic Config object
config = context.config

# Set the SQLAlchemy URL
config.set_main_option("sqlalchemy.url", settings.DATABASE_URL)

# Interpret the config file for Python logging
fileConfig(config.config_file_name)

# Set target metadata
target_metadata = Base.metadata

# Rest of the file remains the same...
```

Create initial migration:

```bash
```

```
alembic revision --autogenerate -m "Initial migration"
alembic upgrade head
```

Step 7: Testing Setup (1 hour)

7.1 Create Test Configuration

Create `app/tests/__init__.py`:

```python
# Empty file to make tests a package
```

Create `app/tests/conftest.py`:

```python
```

```
import pytest
from fastapi.testclient import TestClient
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from app.core.database import Base, get_db
from app.main import app

# Test database URL (use SQLite for tests)
SQLALCHEMY_DATABASE_URL = "sqlite:///./test.db"

engine = create_engine(
    SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread": False}
)
TestingSessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base.metadata.create_all(bind=engine)


def override_get_db():
    try:
        db = TestingSessionLocal()
        yield db
    finally:
        db.close()


app.dependency_overrides[get_db] = override_get_db

client = TestClient(app)


@pytest.fixture
def test_client():
    return client
```

Create `app/tests/test_main.py`:

```
python
```

```python
def test_read_root(test_client):
    response = test_client.get("/")
    assert response.status_code == 200
    assert "message" in response.json()


def test_health_check(test_client):
    response = test_client.get("/health")
    assert response.status_code == 200
    assert response.json() == {"status": "healthy", "service": "psychsync-api"}
```

Step 8: Development Scripts (30 minutes)

8.1 Create Development Scripts

Create `run_dev.py`:

```python
import uvicorn

if __name__ == "__main__":
    uvicorn.run(
        "app.main:app",
        host="0.0.0.0",
        port=8000,
        reload=True,
        log_level="info"
    )
```

Create `Makefile`:

```makefile
```

```
.PHONY: install dev test lint format migration upgrade-db

install:
  pip install -r requirements-dev.txt

dev:
  python run_dev.py

test:
  pytest app/tests/ -v

lint:
  flake8 app/
  mypy app/

format:
  black app/
  isort app/

migration:
  alembic revision --autogenerate -m "$(message)"

upgrade-db:
  alembic upgrade head

clean:
  find . -type f -name "*.pyc" -delete
  find . -type d -name "__pycache__" -delete
```

Step 9: Docker Configuration (Optional - 30 minutes)

Create `Dockerfile`:

```
dockerfile
```

```
FROM python:3.9-slim

WORKDIR /app

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

COPY . .

EXPOSE 8000

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Create `docker-compose.yml`:

```yaml
```

```
version: '3.8'

services:
  api:
    build: .
    ports:
      - "8000:8000"
    environment:
      - DATABASE_URL=postgresql://psychsync_user:psychsync_pass@db:5432/psychsync_db
      - REDIS_URL=redis://redis:6379
    depends_on:
      - db
      - redis
    volumes:
      - .:/app
    command: uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload

  db:
    image: postgres:13
    environment:
      - POSTGRES_USER=psychsync_user
      - POSTGRES_PASSWORD=psychsync_pass
      - POSTGRES_DB=psychsync_db
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:6-alpine
    ports:
      - "6379:6379"

volumes:
  postgres_data:
```

Step 10: Running and Testing (30 minutes)

10.1 Start the Development Server

```bash
```

```
# Install dependencies
make install

# Start development server
make dev

# Or alternatively
python run_dev.py
```

10.2 Test the API

1. Open your browser and go to `http://localhost:8000/api/v1/docs`
2. You should see the Swagger UI documentation
3. Test the health check endpoint: `GET /health`
4. Test the root endpoint: `GET /`

10.3 Run Tests

```bash
make test
```

Final Project Structure

```
psychsync-api/
├── app/
│   ├── api/
│   │   ├── dependencies/
│   │   │   └── auth.py
│   │   └── v1/
│   │       ├── endpoints/
│   │       │   ├── auth.py
│   │       │   └── users.py
│   │       └── api.py
│   ├── core/
│   │   ├── config.py
│   │   ├── database.py
│   │   └── security.py
│   ├── models/
│   │   └── user.py
│   ├── schemas/
│   │   ├── auth.py
│   │   └── user.py
│   ├── tests/
│ │   ├── conftest.py
│ │   └── test_main.py
│   └── main.py
├── alembic/
├── requirements.txt
├── requirements-dev.txt
├── .env
├── run_dev.py
├── Makefile
├── Dockerfile
└── docker-compose.yml
```

Next Steps

After completing this setup, you should:

1. **Test all endpoints** using the Swagger UI at `/api/v1/docs`
2. **Set up your database** and run migrations
3. **Create additional models** for teams, assessments, etc.
4. **Implement the remaining authentication endpoints** (register, password reset)
5. **Add comprehensive tests** for all endpoints
6. **Set up CI/CD pipeline** with GitHub Actions
7. **Configure logging and monitoring**

This foundation provides a solid base for the PsychSync API with authentication middleware, OpenAPI documentation, and a scalable project structure.

Troubleshooting

- If you encounter database connection issues, ensure PostgreSQL is running and credentials are correct
- For JWT token issues, verify your SECRET_KEY is properly set
- If CORS errors occur, check your BACKEND_CORS_ORIGINS setting
- For import errors, ensure your Python path includes the app directory