https://fastapi.tiangolo.com/

INSTALAR LIBRERIAS

FAST API

1. Instalar la libreria en la terminal

$ pip install "fastapi[standard]"

1. Pydantic\_setting
2. pip install pydantic\_settings

GIT COMMANDS

Git add . todo

Git commit -m “mensaje”

git push origin main

ACTUALIZAR EL PROYECYO

git pull origin main

INSTALAR LAS DEPENDENCIAS QUE ESTAN EN EL ARCHIVO .text

pip install -r appSale/requirements.txt

DOCUMENTACION DEL PROYECTO HOUSE LY

To structure your FastAPI project with proper separation of concerns and integrate it with MongoDB, we can follow a clean architecture approach. Here's an example of how you can organize your code:

### 1. Project Structure

```

project/

│

├── app/

│   ├── main.py

│   ├── models/

│   │   └── property.py

│   ├── schemas/

│   │   └── property.py

│   ├── database/

│   │   ├── \_\_init\_\_.py

│   │   └── connection.py

│   ├── crud/

│   │   └── property.py

│   ├── api/

│   │   └── v1/

│   │       └── endpoints/

│   │           └── property.py

│   ├── core/

│   │   └── config.py

│   └── dependencies/

│       └── \_\_init\_\_.py

│

└── requirements.txt

```

### 2. `requirements.txt`

```plaintext

fastapi

pydantic

motor

pymongo

uvicorn

```

### 3. Code Structure

#### `app/main.py`

```python

from fastapi import FastAPI

from app.api.v1.endpoints import property

app = FastAPI()

app.include\_router(property.router, prefix="/properties", tags=["properties"])

@app.get("/")

async def read\_root():

    return {"Hello": "World"}

```

#### `app/models/property.py`

```python

from pydantic import BaseModel, HttpUrl

from typing import Optional

class Property(BaseModel):

    name: str

    image: HttpUrl

    description: str

    maxPrice: float

    minPrice: float

    offer: bool

    city: str

    area: float

    address: str

```

#### `app/schemas/property.py`

```python

from pydantic import BaseModel, HttpUrl

class PropertyCreate(BaseModel):

    name: str

    image: HttpUrl

    description: str

    maxPrice: float

    minPrice: float

    offer: bool

    city: str

    area: float

    address: str

class Property(BaseModel):

    id: str

    name: str

    image: HttpUrl

    description: str

    maxPrice: float

    minPrice: float

    offer: bool

    city: str

    area: float

    address: str

```

#### `app/database/connection.py`

```python

import motor.motor\_asyncio

from app.core.config import settings

client = motor.motor\_asyncio.AsyncIOMotorClient(settings.MONGO\_URI)

database = client[settings.MONGO\_DB]

```

#### `app/crud/property.py`

```python

from [app.database.connection.py](http://app.database.connection.py/) import database

from app.models.property import Property

from typing import List

collection = database.get\_collection("properties")

async def create\_property(property: Property) -> Property:

    result = await collection.insert\_one(property.dict())

    return Property(\*\*property.dict(), id=str(result.inserted\_id))

async def get\_properties() -> List[Property]:

    properties = await collection.find().to\_list(length=100)

    return [Property(\*\*prop) for prop in properties]

# Add more CRUD operations as needed

```

#### `app/api/v1/endpoints/property.py`

```python

from fastapi import APIRouter, HTTPException

from typing import List

from app.crud import property as crud

from app.models.property import Property

from app.schemas.property import PropertyCreate, Property as PropertyResponse

router = APIRouter()

@[router.post](http://router.post/)("/", response\_model=PropertyResponse)

async def create\_property(property: PropertyCreate):

    return await crud.create\_property(Property(\*\*property.dict()))

@router.get("/", response\_model=List[PropertyResponse])

async def read\_properties():

    return await crud.get\_properties()

```

#### `app/core/config.py`

```python

from pydantic import BaseSettings

class Settings(BaseSettings):

    MONGO\_URI: str

    MONGO\_DB: str

    class Config:

        env\_file = ".env"

settings = Settings()

```

#### `app/dependencies/\_\_init\_\_.py`

```python

# Add dependency overrides and other common dependencies here

```

### .env

```plaintext

MONGO\_URI=mongodb://localhost:27017

MONGO\_DB=mydatabase

```

### Running the Application

1. Install the dependencies:

    ```bash

    pip install -r requirements.txt

    ```

2. Run the application:

    ```bash

    uvicorn app.main:app --reload

    ```

### Explanation

- \*\*Project Structure\*\*: Organizes the project into different folders for models, schemas, database connections, CRUD operations, and API endpoints.

- \*\*Dependency Injection\*\*: Uses FastAPI's dependency injection system for clean and maintainable code.

- \*\*MongoDB Integration\*\*: Uses `motor`, an async MongoDB driver, to interact with the database.

- \*\*Configuration\*\*: Loads environment variables using `pydantic`'s `BaseSettings`.

This structure ensures a clean separation of concerns and makes the project easy to extend and maintain.

CORRER EL PROYECTO

uvicorn appSale.main:app –reload

DOCUMENTACION PYCHARM PARA SER PRO

¡Claro! Aquí tienes una lista extensa de atajos de teclado útiles para PyCharm en Windows 10. Esto te ayudará a navegar más rápido y ser más eficiente al codificar:

### Atajos de Navegación

- \*\*Ir a la Clase\*\*: `Ctrl + N`

- \*\*Ir a Archivo\*\*: `Ctrl + Shift + N`

- \*\*Ir a Símbolo\*\*: `Ctrl + Shift + A`

- \*\*Ir a Línea\*\*: `Ctrl + G`

- \*\*Ir a la Declaración\*\*: `Ctrl + B` o `Ctrl + Click Izquierdo`

- \*\*Ir a la Definición\*\*: `F12`

- \*\*Ir a la Implementación\*\*: `Ctrl + Alt + B`

- \*\*Navegar hacia Adelante\*\*: `Ctrl + Alt + Right Arrow`

- \*\*Navegar hacia Atrás\*\*: `Ctrl + Alt + Left Arrow`

### Edición de Código

- \*\*Deshacer\*\*: `Ctrl + Z`

- \*\*Rehacer\*\*: `Ctrl + Shift + Z`

- \*\*Cortar Línea Actual\*\*: `Ctrl + X`

- \*\*Copiar Línea Actual\*\*: `Ctrl + C`

- \*\*Pegar\*\*: `Ctrl + V`

- \*\*Duplicar Línea Actual\*\*: `Ctrl + D`

- \*\*Eliminar Línea Actual\*\*: `Ctrl + Y`

- \*\*Ajustar la sangría\*\*: `Ctrl + ]` (aumentar), `Ctrl + [` (disminuir)

- \*\*Reformatear Código\*\*: `Ctrl + Alt + L`

- \*\*Comentar/Descomentar Línea\*\*: `Ctrl + /` (comentario de una sola línea), `Ctrl + Shift + /` (comentario de bloque)

### Búsqueda y Reemplazo

- \*\*Buscar en el Archivo\*\*: `Ctrl + F`

- \*\*Buscar en el Proyecto\*\*: `Ctrl + Shift + F`

- \*\*Reemplazar en el Archivo\*\*: `Ctrl + R`

- \*\*Reemplazar en el Proyecto\*\*: `Ctrl + Shift + R`

### Control de Versiones

- \*\*Mostrar el Historial de Cambios\*\*: `Alt + Shift + C`

- \*\*Commit Cambios\*\*: `Ctrl + K`

- \*\*Actualizar Proyecto\*\*: `Ctrl + T`

- \*\*Ver Cambios Locales\*\*: `Ctrl + D`

### Herramientas y Ventanas

- \*\*Mostrar/Ocultar la Terminal\*\*: `Alt + F12`

- \*\*Mostrar/Ocultar la Barra de Navegación\*\*: `Alt + 1`

- \*\*Mostrar/Ocultar la Vista de Problemas\*\*: `Alt + 6`

- \*\*Mostrar/Ocultar la Vista de Ejecutar\*\*: `Alt + 4`

- \*\*Mostrar/Ocultar el Panel de Debug\*\*: `Alt + 5`

### Refactorización

- \*\*Refactorizar\*\*: `Ctrl + Alt + Shift + T`

- \*\*Renombrar\*\*: `Shift + F6`

- \*\*Cambiar Firma del Método\*\*: `Ctrl + F6`

### Navegación en el Código

- \*\*Ir a la Última Modificación\*\*: `Ctrl + Shift + Backspace`

- \*\*Ir a la Anterior\*\*: `Ctrl + Alt + Left Arrow`

- \*\*Ir a la Siguiente\*\*: `Ctrl + Alt + Right Arrow`

- \*\*Ir a la Última Edición\*\*: `Ctrl + Shift + Backspace`

### Otros Atajos Útiles

- \*\*Abrir la Paleta de Comandos\*\*: `Ctrl + Shift + A`

- \*\*Mostrar la Ayuda de Código\*\*: `Ctrl + P`

- \*\*Mostrar Documentación Rápida\*\*: `Ctrl + Q`