Radhika Patel

rpp2142

🚀 Sprint Completion Status Report

Student Name: Radhika Patel

Sprint Number: 0

Duration: [09/01] – [09/14]

Report Date: [09/14]

1. Sprint Goal 🎯

Defined Goal:

1. Clone Professor Ferguson's SimpleMicroservices Repository.
2. Create a project that is my version using two different resources.
   a. Copy the structure of Professor Ferguson's repository
   b. Define two models: City and Restaurants
   c. Implement "API first" definition by implementing placeholder routes for each resource: GET/<resource>, POST/<resource>, GET/<resource>/{id}, PUT /<resource>/{id} v. DELETE /<resource>/{id}
   d. Annotate models and paths to autogenerate OpenAPI document

Outcome: [Achieved]

2. Completed Work ✅

Resource 1: City

```
from __future__ import annotations

from typing import Optional

from uuid import UUID, uuid4

from datetime import datetime

from pydantic import BaseModel, Field



class CityBase(BaseModel):

   name: str = Field(..., description="Name of the city",
json_schema_extra={"example": "New York"})
```

```
   country: str = Field(..., description="Country of the city",
json_schema_extra={"example": "USA"})
   population: Optional[int] = Field(None, description="Population of the
city", json_schema_extra={"example": 8419000})

   model_config = {
       "json_schema_extra": {
           "examples": [
               {"name": "New York", "country": "USA", "population": 8419000},
               {"name": "Paris", "country": "France", "population": 2148000},
           ]
       }
   }

class CityCreate(CityBase):
   model_config = CityBase.model_config

class CityUpdate(BaseModel):
   name: Optional[str] = Field(None, json_schema_extra={"example": "San
Francisco"})
   country: Optional[str] = Field(None, json_schema_extra={"example": "USA"})
   population: Optional[int] = Field(None, json_schema_extra={"example":
870000})

   model_config = {
       "json_schema_extra": {
           "examples": [
               {"name": "Los Angeles"},

               {"population": 4000000},

               {"country": "Germany"},

           ]

       }

   }



class CityRead(CityBase):

   id: UUID = Field(default_factory=uuid4, description="Server-generated City
ID", json_schema_extra={"example": "aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa"})

   created_at: datetime = Field(default_factory=datetime.utcnow,
description="Creation timestamp", json_schema_extra={"example":
"2025-01-15T10:20:30Z"})

   updated_at: datetime = Field(default_factory=datetime.utcnow,
description="Last update timestamp", json_schema_extra={"example":
"2025-01-16T12:00:00Z"})



   model_config = CityBase.model_config
```

Resource 2: Restaurant

```
from __future__ import annotations

from typing import Optional

from uuid import UUID, uuid4

from datetime import datetime

from pydantic import BaseModel, Field
```

Radhika Patel

rpp2142

```python
class RestaurantBase(BaseModel):
    name: str = Field(..., description="Name of the restaurant",
json_schema_extra={"example": "Joe's Pizza"})
    cuisine: str = Field(..., description="Type of cuisine",
json_schema_extra={"example": "Italian"})
    city_id: UUID = Field(..., description="UUID of the city where the
restaurant is located", json_schema_extra={"example":
"aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa"})
    rating: Optional[float] = Field(None, description="Average rating (0-5)",
json_schema_extra={"example": 4.5})


    model_config = {
        "json_schema_extra": {
            "examples": [
                {"name": "Joe's Pizza", "cuisine": "Italian", "city_id":
"aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa", "rating": 4.5},
                {"name": "Sushi Place", "cuisine": "Japanese", "city_id":
"bbbbbbbb-bbbb-4bbb-8bbb-bbbbbbbbbbbb", "rating": 4.8},
            ]
        }
    }


class RestaurantCreate(RestaurantBase):
    model_config = RestaurantBase.model_config


class RestaurantUpdate(BaseModel):
    name: Optional[str] = Field(None, json_schema_extra={"example": "Pizzeria
Uno"})
```

```
    cuisine: Optional[str] = Field(None, json_schema_extra={"example": "American"})
    city_id: Optional[UUID] = Field(None, description="UUID of the city", json_schema_extra={"example": "cccccccc-cccc-4ccc-8ccc-cccccccccccc"})
    rating: Optional[float] = Field(None, description="Average rating", json_schema_extra={"example": 4.7})

    model_config = {
        "json_schema_extra": {
            "examples": [
                {"name": "New Place"},
                {"cuisine": "Mexican"},
                {"rating": 4.9},
            ]
        }
    }


class RestaurantRead(RestaurantBase):
    id: UUID = Field(default_factory=uuid4, description="Server-generated Restaurant ID", json_schema_extra={"example": "cccccccc-cccc-4ccc-8ccc-cccccccccccc"})
    created_at: datetime = Field(default_factory=datetime.utcnow, description="Creation timestamp", json_schema_extra={"example": "2025-01-15T10:20:30Z"})
    updated_at: datetime = Field(default_factory=datetime.utcnow, description="Last update timestamp", json_schema_extra={"example": "2025-01-16T12:00:00Z"})

    model_config = RestaurantBase.model_config
```

Radhika Patel

rpp2142

main.py Routes:

```python
from fastapi import FastAPI, HTTPException, Query, Path

from typing import Dict, List, Optional

from uuid import UUID

from datetime import datetime


import socket

from models.health import Health

from models.city import CityCreate, CityRead, CityUpdate

from models.restaurant import RestaurantCreate, RestaurantRead,
RestaurantUpdate


app = FastAPI(title="City/Restaurant API", version="0.1.0")


cities: Dict[UUID, CityRead] = {}

restaurants: Dict[UUID, RestaurantRead] = {}


# ----------------- Health Endpoint -----------------

def make_health(echo: Optional[str], path_echo: Optional[str] = None) ->
Health:

   return Health(

       status=200,

       status_message="OK",

       timestamp=datetime.utcnow().isoformat() + "Z",

       ip_address=socket.gethostbyname(socket.gethostname()),
       echo=echo,

       path_echo=path_echo

   )

@app.get("/health", response_model=Health)

def get_health_no_path(echo: str | None = Query(None, description="Optional
echo string")):

   return make_health(echo=echo)


@app.get("/health/{path_echo}", response_model=Health)

def get_health_with_path(

   path_echo: str = Path(..., description="Required echo in the URL path"),

   echo: str | None = Query(None, description="Optional echo string"),

):

   return make_health(echo=echo, path_echo=path_echo)


# ----------------- City Endpoints -----------------

@app.post("/cities", response_model=CityRead, status_code=201)

def create_city(city: CityCreate):

   city_read = CityRead(**city.model_dump())

   cities[city_read.id] = city_read

   return city_read


@app.get("/cities", response_model=List[CityRead])

def list_cities(

   name: Optional[str] = Query(None, description="Filter by city name"),

   country: Optional[str] = Query(None, description="Filter by country"),
   min_population: Optional[int] = Query(None, description="Filter by minimum population"),

   max_population: Optional[int] = Query(None, description="Filter by maximum population"),

):

   results = list(cities.values())

   if name is not None:

       results = [c for c in results if c.name == name]

   if country is not None:

       results = [c for c in results if c.country == country]

   if min_population is not None:

       results = [c for c in results if c.population and c.population >= min_population]

   if max_population is not None:

       results = [c for c in results if c.population and c.population <= max_population]

   return results



@app.get("/cities/{city_id}", response_model=CityRead)

def get_city(city_id: UUID):

   if city_id not in cities:

       raise HTTPException(status_code=404, detail="City not found")

   return cities[city_id]



@app.put("/cities/{city_id}", response_model=CityRead)

def update_city(city_id: UUID, city_update: CityCreate):

   if city_id not in cities:
```

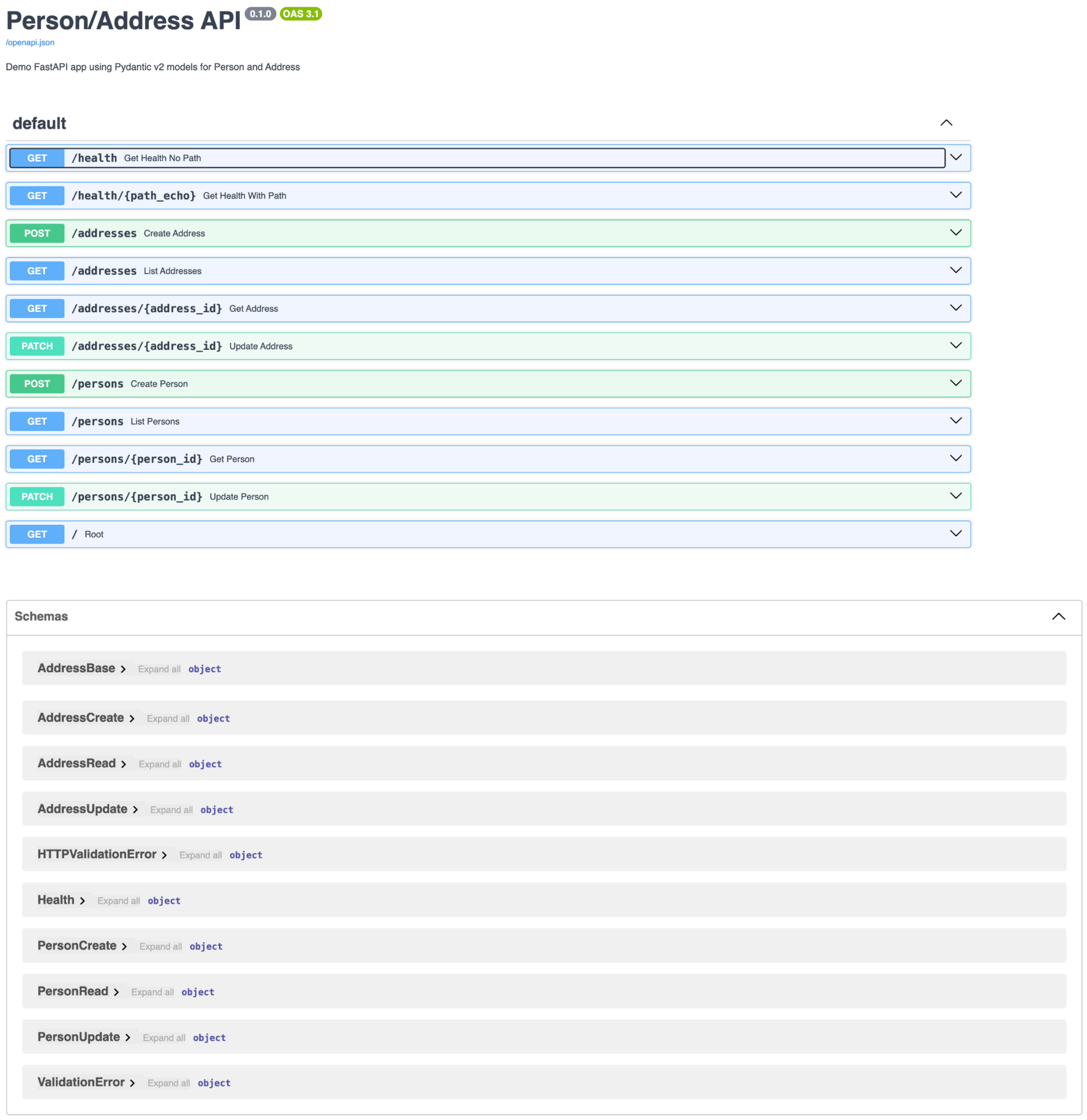
```python
        raise HTTPException(status_code=404, detail="City not found")
    cities[city_id] = CityRead(id=city_id, **city_update.model_dump())
    return cities[city_id]


@app.delete("/cities/{city_id}", status_code=204)
def delete_city(city_id: UUID):
    if city_id not in cities:
        raise HTTPException(status_code=404, detail="City not found")
    del cities[city_id]
    return


# ----------------- Restaurant Endpoints -----------------
@app.post("/restaurants", response_model=RestaurantRead, status_code=201)
def create_restaurant(restaurant: RestaurantCreate):
    if restaurant.city_id not in cities:
        raise HTTPException(status_code=400, detail="City does not exist")
    restaurant_read = RestaurantRead(**restaurant.model_dump())
    restaurants[restaurant_read.id] = restaurant_read
    return restaurant_read


@app.get("/restaurants", response_model=List[RestaurantRead])
def list_restaurants(
    name: Optional[str] = Query(None, description="Filter by restaurant name"),
    cuisine: Optional[str] = Query(None, description="Filter by cuisine type"),
    city_id: Optional[UUID] = Query(None, description="Filter by city UUID"),
    min_rating: Optional[float] = Query(None, description="Filter by minimum rating"),
    max_rating: Optional[float] = Query(None, description="Filter by maximum rating"),
):
    results = list(restaurants.values())
    if name is not None:
        results = [r for r in results if r.name == name]
    if cuisine is not None:
        results = [r for r in results if r.cuisine == cuisine]
    if city_id is not None:
        results = [r for r in results if r.city_id == city_id]
    if min_rating is not None:
        results = [r for r in results if r.rating and r.rating >= min_rating]
    if max_rating is not None:
        results = [r for r in results if r.rating and r.rating <= max_rating]
    return results


@app.get("/restaurants/{restaurant_id}", response_model=RestaurantRead)
def get_restaurant(restaurant_id: UUID):
    if restaurant_id not in restaurants:
        raise HTTPException(status_code=404, detail="Restaurant not found")
    return restaurants[restaurant_id]


@app.put("/restaurants/{restaurant_id}", response_model=RestaurantRead)
```

```python
def update_restaurant(restaurant_id: UUID, restaurant_update:
RestaurantCreate):
    if restaurant_id not in restaurants:
        raise HTTPException(status_code=404, detail="Restaurant not found")
    restaurants[restaurant_id] = RestaurantRead(id=restaurant_id,
created_at=datetime.utcnow(), updated_at=datetime.utcnow(),
**restaurant_update.model_dump())
    return restaurants[restaurant_id]


@app.delete("/restaurants/{restaurant_id}", status_code=204)
def delete_restaurant(restaurant_id: UUID):
    if restaurant_id not in restaurants:
        raise HTTPException(status_code=404, detail="Restaurant not found")
    del restaurants[restaurant_id]
    return


@app.get("/")
def root():
    return {"message": "Welcome to the City/Restaurant API. See /docs for
OpenAPI UI."}



if __name__ == "__main__":
    import os
    import uvicorn


    port = int(os.environ.get("FASTAPIPORT", 8001))
    uvicorn.run("main:app", host="0.0.0.0", port=port, reload=True)
```

OpenAPI Document (Partial)

Radhika Patel

rpp2142


Person/Address API 0.1.0 OAS 3.1

/openapi.json

Demo FastAPI app using Pydantic v2 models for Person and Address

default

GET /health Get Health No Path

GET /health/{path_echo} Get Health With Path

POST /addresses Create Address

GET /addresses List Addresses

GET /addresses/{address_id} Get Address

PATCH /addresses/{address_id} Update Address

POST /persons Create Person

GET /persons List Persons

GET /persons/{person_id} Get Person

PATCH /persons/{person_id} Update Person

GET / Root

Schemas

AddressBase › Expand all object

AddressCreate › Expand all object

AddressRead › Expand all object

AddressUpdate › Expand all object

HTTPValidationError › Expand all object

Health › Expand all object

PersonCreate › Expand all object

PersonRead › Expand all object

PersonUpdate › Expand all object

ValidationError › Expand all object

Link to GitHub Repo

https://github.com/radhikap0107/MySimpleMicroservices

Link to Recording Demo

https://drive.google.com/file/d/1DEtPE679EnP6Y9mFml8dzXPYRG4d8jGq/view?usp=sharing