# Sprint Completion Status Report

\*\*Student Name:\*\* Yonghao Lin

\*\*Sprint Number:\*\* [Sprint 0]

\*\*Duration:\*\* [09.07.2025] – [09.14.2025]

\*\*Report Date:\*\* [09.14.2025]

## 1. Sprint Goal 🎯

\*\*Defined Goal:\*\*

1. Clone Professor Ferguson’s *Simple Microservices Repository.*
2. Create a project that is my version using two different resources (**Pet** and **Owner**).
   1. Copy the structure of Professor Ferguson’s repository
   2. Define two models (**Pet** and **Owner**).
   3. Implement “API first” definition by implementing placeholder routes for each resource:
      1. GET /<resource>
      2. POST /<resource>
      3. GET /<resource>/{id}
      4. PUT /<resource>/{id}
      5. DELETE /<resource>/{id}
   4. Annotate models and paths to autogenerate OpenAPI document.
   5. Tested OpenAPI document dispatching to methods.

\*\*Outcome:\*\* Achieved

\*\*Notes:\*\* Fully Achieved

## 2. Completed Work ✅

### Owner

class OwnerBase(BaseModel):  
 first\_name: str = Field(  
 ...,  
 description="Owner given name.",  
 json\_schema\_extra={"example": "Ada"},  
 )  
 last\_name: str = Field(  
 ...,  
 description="Owner family name.",  
 json\_schema\_extra={"example": "Lovelace"},  
 )  
 email: EmailStr = Field(  
 ...,  
 description="Primary email address.",  
 json\_schema\_extra={"example": "ada@example.com"},  
 )  
 phone: Optional[str] = Field(  
 None,  
 description="Contact phone number in any reasonable format.",  
 json\_schema\_extra={"example": "+1-317-555-0123"},  
 )  
 government\_id: Optional[str] = Field(  
 None,  
 description="Optional government-issued ID or number.",  
 json\_schema\_extra={"example": "NY-123-456-789"},  
 )  
  
 *# Embed addresses (each with persistent ID)* addresses: List[AddressBase] = Field(  
 default\_factory=list,  
 description="Addresses linked to this person (each carries a persistent Address ID).",  
 json\_schema\_extra={  
 "example": [  
 {  
 "id": "550e8400-e29b-41d4-a716-446655440000",  
 "street": "123 Main St",  
 "city": "London",  
 "state": None,  
 "postal\_code": "SW1A 1AA",  
 "country": "UK",  
 }  
 ]  
 },  
 )  
  
 model\_config = {  
 "json\_schema\_extra": {  
 "examples": [  
 {  
 "first\_name": "Leslie",  
 "last\_name": "Knope",  
 "email": "leslie.knope@example.com",  
 "phone": "+1-317-555-0123",  
 "government\_id": "NY-123-456-789",  
 "addresses": [  
 {  
 "id": "550e8400-e29b-41d4-a716-446655440000",  
 "street": "123 Main St",  
 "city": "London",  
 "state": None,  
 "postal\_code": "SW1A 1AA",  
 "country": "UK",  
 }  
 ],  
 }  
 ]  
 }  
 }  
  
  
class OwnerCreate(OwnerBase):  
 *"""Creation payload for an Owner."""* model\_config = {  
 "json\_schema\_extra": {  
 "examples": [  
 {  
 "first\_name": "April",  
 "last\_name": "Ludgate",  
 "email": "april@example.com",  
 "phone": "+1-317-555-0987",  
 "government\_id": None,  
 "addresses": [],  
 }  
 ]  
 }  
 }  
  
  
class OwnerUpdate(BaseModel):  
 *"""Partial update for an Owner; supply only fields to change."""* first\_name: Optional[str] = Field(None, json\_schema\_extra={"example": "Ann"})  
 last\_name: Optional[str] = Field(None, json\_schema\_extra={"example": "Perkins"})  
 email: Optional[EmailStr] = Field(None, json\_schema\_extra={"example": "ann.perkins@example.com"})  
 phone: Optional[str] = Field(None, json\_schema\_extra={"example": "+1-317-555-0000"})  
 government\_id: Optional[str] = Field(None, json\_schema\_extra={"example": "CA-987-654-321"})  
 addresses: Optional[List[AddressBase]] = Field(  
 None,  
 description="Replace the entire set of addresses with this list.",  
 json\_schema\_extra={  
 "example": [  
 {  
 "id": "bbbbbbbb-bbbb-4bbb-8bbb-bbbbbbbbbbbb",  
 "street": "10 Downing St",  
 "city": "London",  
 "state": None,  
 "postal\_code": "SW1A 2AA",  
 "country": "UK",  
 }  
 ]  
 },  
 )  
  
 model\_config = {  
 "json\_schema\_extra": {  
 "examples": [  
 {"first\_name": "Ann", "last\_name": "Perkins"},  
 {"phone": "+1-317-555-0000"},  
 {  
 "addresses": [  
 {  
 "id": "aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa",  
 "street": "742 Evergreen Terrace",  
 "city": "Springfield",  
 "state": "IL",  
 "postal\_code": "62704",  
 "country": "USA",  
 }  
 ]  
 },  
 ]  
 }  
 }  
  
  
class OwnerRead(OwnerBase):  
 *"""Server representation returned to clients."""* id: UUID = Field(  
 default\_factory=uuid4,  
 description="Server-generated Owner ID.",  
 json\_schema\_extra={"example": "99999999-9999-4999-8999-999999999999"},  
 )  
 created\_at: datetime = Field(  
 default\_factory=datetime.utcnow,  
 description="Creation timestamp (UTC).",  
 json\_schema\_extra={"example": "2025-01-15T10:20:30Z"},  
 )  
 updated\_at: datetime = Field(  
 default\_factory=datetime.utcnow,  
 description="Last update timestamp (UTC).",  
 json\_schema\_extra={"example": "2025-01-16T12:00:00Z"},  
 )  
  
 model\_config = {  
 "json\_schema\_extra": {  
 "examples": [  
 {  
 "first\_name": "Leslie",  
 "last\_name": "Knope",  
 "email": "leslie.knope@example.com",  
 "phone": "+1-317-555-0123",  
 "government\_id": "NY-123-456-789",  
 "addresses": [  
 {  
 "id": "550e8400-e29b-41d4-a716-446655440000",  
 "street": "123 Main St",  
 "city": "London",  
 "state": None,  
 "postal\_code": "SW1A 1AA",  
 "country": "UK",  
 }  
 ],  
 "created\_at": "2025-01-15T10:20:30Z",  
 "updated\_at": "2025-01-16T12:00:00Z",  
 }  
 ]  
 }  
 }

### Pet

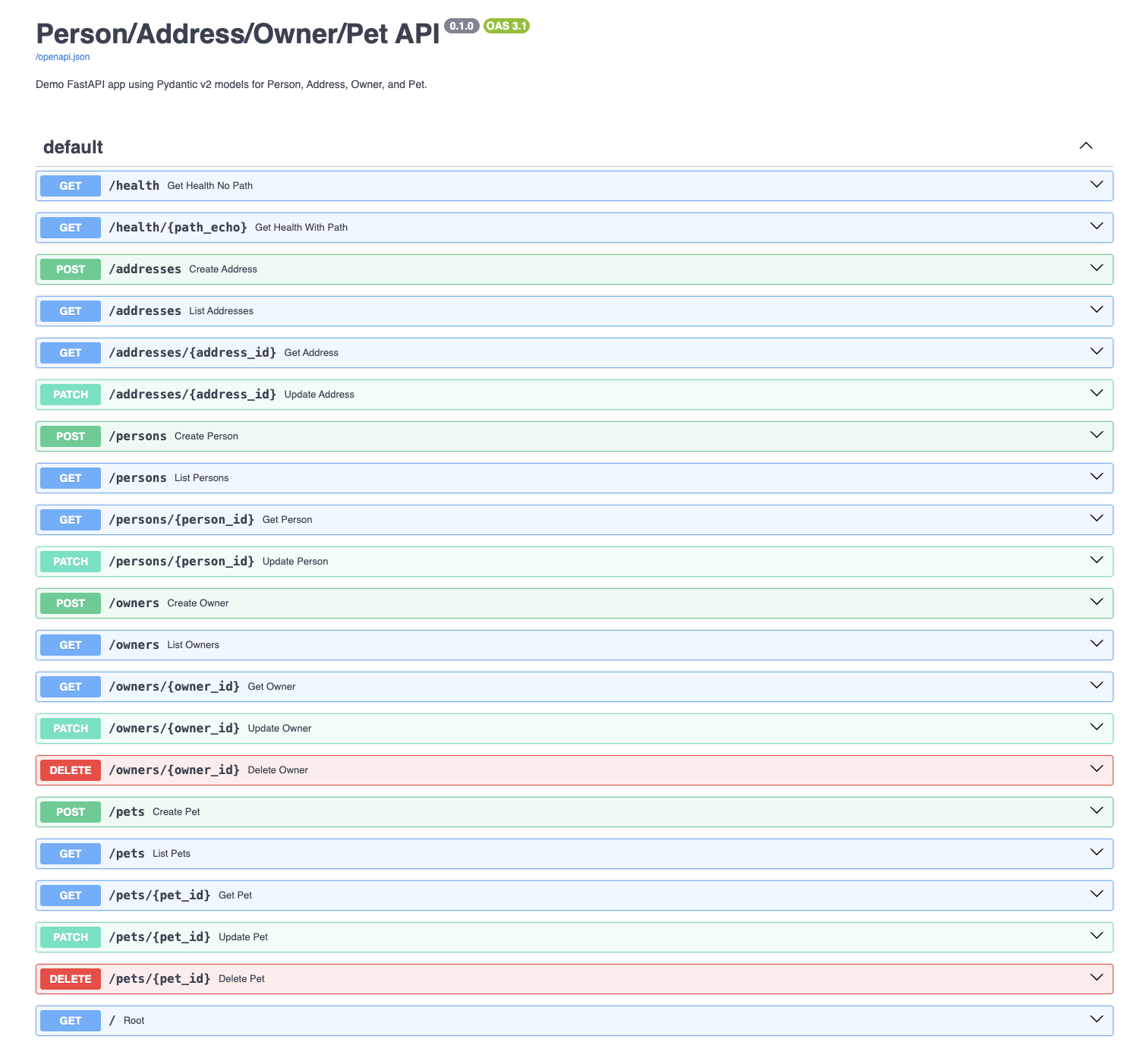
class PetBase(BaseModel):  
 name: str = Field(  
 ...,  
 description="Pet's given name.",  
 json\_schema\_extra={"example": "Buddy"},  
 )  
 species: str = Field(  
 ...,  
 description="Type of animal.",  
 json\_schema\_extra={"example": "Dog"},  
 )  
 breed: Optional[str] = Field(  
 None,  
 description="Specific breed if applicable.",  
 json\_schema\_extra={"example": "Golden Retriever"},  
 )  
 birth\_date: Optional[date] = Field(  
 None,  
 description="Date of birth (YYYY-MM-DD).",  
 json\_schema\_extra={"example": "2020-05-10"},  
 )  
 color: Optional[str] = Field(  
 None,  
 description="Primary color of the pet.",  
 json\_schema\_extra={"example": "Golden"},  
 )  
  
 model\_config = {  
 "json\_schema\_extra": {  
 "examples": [  
 {  
 "name": "Buddy",  
 "species": "Dog",  
 "breed": "Golden Retriever",  
 "birth\_date": "2020-05-10",  
 "color": "Golden",  
 }  
 ]  
 }  
 }  
  
  
class PetCreate(PetBase):  
 *"""Creation* *payload for a Pet."""* owner\_id: UUID = Field(  
 ...,  
 description="The Owner ID this pet belongs to.",  
 json\_schema\_extra={"example": "99999999-9999-4999-8999-999999999999"},  
 )  
  
 model\_config = {  
 "json\_schema\_extra": {  
 "examples": [  
 {  
 "name": "Whiskers",  
 "species": "Cat",  
 "breed": "Siamese",  
 "birth\_date": "2021-07-04",  
 "color": "Cream",  
 "owner\_id": "99999999-9999-4999-8999-999999999999",  
 }  
 ]  
 }  
 }  
  
  
class PetUpdate(BaseModel):  
 *"""Partial update for a Pet; supply only fields to change."""* name: Optional[str] = Field(None, json\_schema\_extra={"example": "Max"})  
 species: Optional[str] = Field(None, json\_schema\_extra={"example": "Dog"})  
 breed: Optional[str] = Field(None, json\_schema\_extra={"example": "Labrador"})  
 birth\_date: Optional[date] = Field(None, json\_schema\_extra={"example": "2019-12-25"})  
 color: Optional[str] = Field(None, json\_schema\_extra={"example": "Black"})  
  
 model\_config = {  
 "json\_schema\_extra": {  
 "examples": [  
 {"name": "Max"},  
 {"breed": "Labrador", "color": "Black"},  
 ]  
 }  
 }  
  
  
class PetRead(PetBase):  
 *"""Server representation returned to clients."""* id: UUID = Field(  
 default\_factory=uuid4,  
 description="Server-generated Pet ID.",  
 json\_schema\_extra={"example": "aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa"},  
 )  
 owner: Optional[OwnerRead] = Field(  
 None,  
 description="The Owner record this pet belongs to.",  
 )  
 created\_at: datetime = Field(  
 default\_factory=datetime.utcnow,  
 description="Creation timestamp (UTC).",  
 json\_schema\_extra={"example": "2025-01-15T10:20:30Z"},  
 )  
 updated\_at: datetime = Field(  
 default\_factory=datetime.utcnow,  
 description="Last update timestamp (UTC).",  
 json\_schema\_extra={"example": "2025-01-16T12:00:00Z"},  
 )

### main.py Routes

*# -----------------------------------------------------------------------------  
# Owner endpoints  
# -----------------------------------------------------------------------------*@app.post("/owners", response\_model=OwnerRead, status\_code=201)  
def create\_owner(owner: OwnerCreate):  
 *# Each owner gets its own UUID; stored as OwnerRead* owner\_read = OwnerRead(\*\*owner.model\_dump())  
 owners[owner\_read.id] = owner\_read  
 return owner\_read  
  
@app.get("/owners", response\_model=List[OwnerRead])  
def list\_owners(  
 first\_name: Optional[str] = Query(None, description="Filter by first name"),  
 last\_name: Optional[str] = Query(None, description="Filter by last name"),  
 email: Optional[str] = Query(None, description="Filter by email"),  
 phone: Optional[str] = Query(None, description="Filter by phone number"),  
 city: Optional[str] = Query(None, description="Filter by city of at least one address"),  
 country: Optional[str] = Query(None, description="Filter by country of at least one address"),  
):  
 results = list(owners.values())  
  
 if first\_name is not None:  
 results = [o for o in results if o.first\_name == first\_name]  
 if last\_name is not None:  
 results = [o for o in results if o.last\_name == last\_name]  
 if email is not None:  
 results = [o for o in results if o.email == email]  
 if phone is not None:  
 results = [o for o in results if o.phone == phone]  
  
 *# nested address filtering (same style as persons)* if city is not None:  
 results = [o for o in results if any(addr.city == city for addr in o.addresses)]  
 if country is not None:  
 results = [o for o in results if any(addr.country == country for addr in o.addresses)]  
  
 return results  
  
@app.get("/owners/{owner\_id}", response\_model=OwnerRead)  
def get\_owner(owner\_id: UUID):  
 if owner\_id not in owners:  
 raise HTTPException(status\_code=404, detail="Owner not found")  
 return owners[owner\_id]  
  
@app.patch("/owners/{owner\_id}", response\_model=OwnerRead)  
def update\_owner(owner\_id: UUID, update: OwnerUpdate):  
 if owner\_id not in owners:  
 raise HTTPException(status\_code=404, detail="Owner not found")  
 stored = owners[owner\_id].model\_dump()  
 stored.update(update.model\_dump(exclude\_unset=True))  
 owners[owner\_id] = OwnerRead(\*\*stored)  
 return owners[owner\_id]  
  
@app.delete("/owners/{owner\_id}", status\_code=204)  
def delete\_owner(owner\_id: UUID):  
 if owner\_id not in owners:  
 raise HTTPException(status\_code=404, detail="Owner not found")  
 owners.pop(owner\_id)  
 return None

*# -----------------------------------------------------------------------------  
# Pet endpoints  
# -----------------------------------------------------------------------------*@app.post("/pets", response\_model=PetRead, status\_code=201)  
def create\_pet(pet: PetCreate):  
 *# Each pet gets its own UUID; stored as PetRead* pet\_read = PetRead(\*\*pet.model\_dump())  
 pets[pet\_read.id] = pet\_read  
 return pet\_read  
  
@app.get("/pets", response\_model=List[PetRead])  
def list\_pets(  
 owner\_id: Optional[UUID] = Query(None, description="Filter by owner ID"),  
 name: Optional[str] = Query(None, description="Filter by pet name"),  
 species: Optional[str] = Query(None, description="Filter by species"),  
 breed: Optional[str] = Query(None, description="Filter by breed"),  
 color: Optional[str] = Query(None, description="Filter by color"),  
 birth\_date: Optional[str] = Query(None, description="Filter by birth date (YYYY-MM-DD)"),  
):  
 results = list(pets.values())  
  
 if owner\_id is not None:  
 results = [p for p in results if p.owner\_id == owner\_id]  
 if name is not None:  
 results = [p for p in results if p.name == name]  
 if species is not None:  
 results = [p for p in results if p.species == species]  
 if breed is not None:  
 results = [p for p in results if p.breed == breed]  
 if color is not None:  
 results = [p for p in results if p.color == color]  
 if birth\_date is not None:  
 results = [p for p in results if str(p.birth\_date) == birth\_date]  
  
 return results  
  
@app.get("/pets/{pet\_id}", response\_model=PetRead)  
def get\_pet(pet\_id: UUID):  
 if pet\_id not in pets:  
 raise HTTPException(status\_code=404, detail="Pet not found")  
 return pets[pet\_id]  
  
@app.patch("/pets/{pet\_id}", response\_model=PetRead)  
def update\_pet(pet\_id: UUID, update: PetUpdate):  
 if pet\_id not in pets:  
 raise HTTPException(status\_code=404, detail="Pet not found")  
 stored = pets[pet\_id].model\_dump()  
 stored.update(update.model\_dump(exclude\_unset=True))  
 pets[pet\_id] = PetRead(\*\*stored)  
 return pets[pet\_id]  
@app.delete("/pets/{pet\_id}", status\_code=204)  
def delete\_pet(pet\_id: UUID):  
 if pet\_id not in pets:  
 raise HTTPException(status\_code=404, detail="Pet not found")  
 pets.pop(pet\_id)  
 return None

### OpenAPI Document (Partial)



### Link to Recording of Demo

https://drive.google.com/file/d/1O9tNVzGa0Q1RJ2XVBxB9y4Gv5Q\_Q5aQe/view?usp=sharing

### Link to GitHub Repository

https://github.com/TheSkyRS/W4153-Fall25-Yonghao/tree/main/SimpleMicroservices

## 3. Incomplete Work ❌

N/A

\*\*Carryover to Next Sprint:\*\* No

## 4. Key Metrics 📊

Note: Ignore this section

\*\*Planned vs. Completed Points:\*\* [e.g., 40 planned / 35 completed]

\*\*Burndown Chart:\*\* [Attach image if available]

\*\*Defects Identified:\*\* [Number + Severity]

## 5. Risks & Blockers ⚠️

Note: Ignore this section

- [Risk/Issue] – [Impact] – [Mitigation/Resolution]

- [Dependency on X team] – [Impact on timeline]

## 6. Team Feedback 💬

Note: Ignore this section

\*\*What Went Well:\*\*

- [Positive note 1]

- [Positive note 2]

\*\*What Could Be Improved:\*\*

- [Improvement area 1]

- [Improvement area 2]

## 7. Next Steps 🔜

Note: Ignore this section

\*\*Upcoming Sprint Goal (Draft):\*\* [Proposed goal]

\*\*Focus Areas:\*\* [e.g., technical debt, new feature, stabilization]

\*\*Planned Dependencies:\*\* [Cross-team items, external blockers]