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**).
    a. Copy the structure of Professor Ferguson's repository
    b. Define two models (**Pet** and **Owner**).
    c. Implement "API first" definition by implementing placeholder routes for each resource:
        i. GET /<resource>
        ii. POST /<resource>
        iii. GET /<resource>/{id}
        iv. PUT /<resource>/{id}
        v. DELETE /<resource>/{id}
    d. Annotate models and paths to autogenerate OpenAPI document.
    e. 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
```

```python
@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:
```

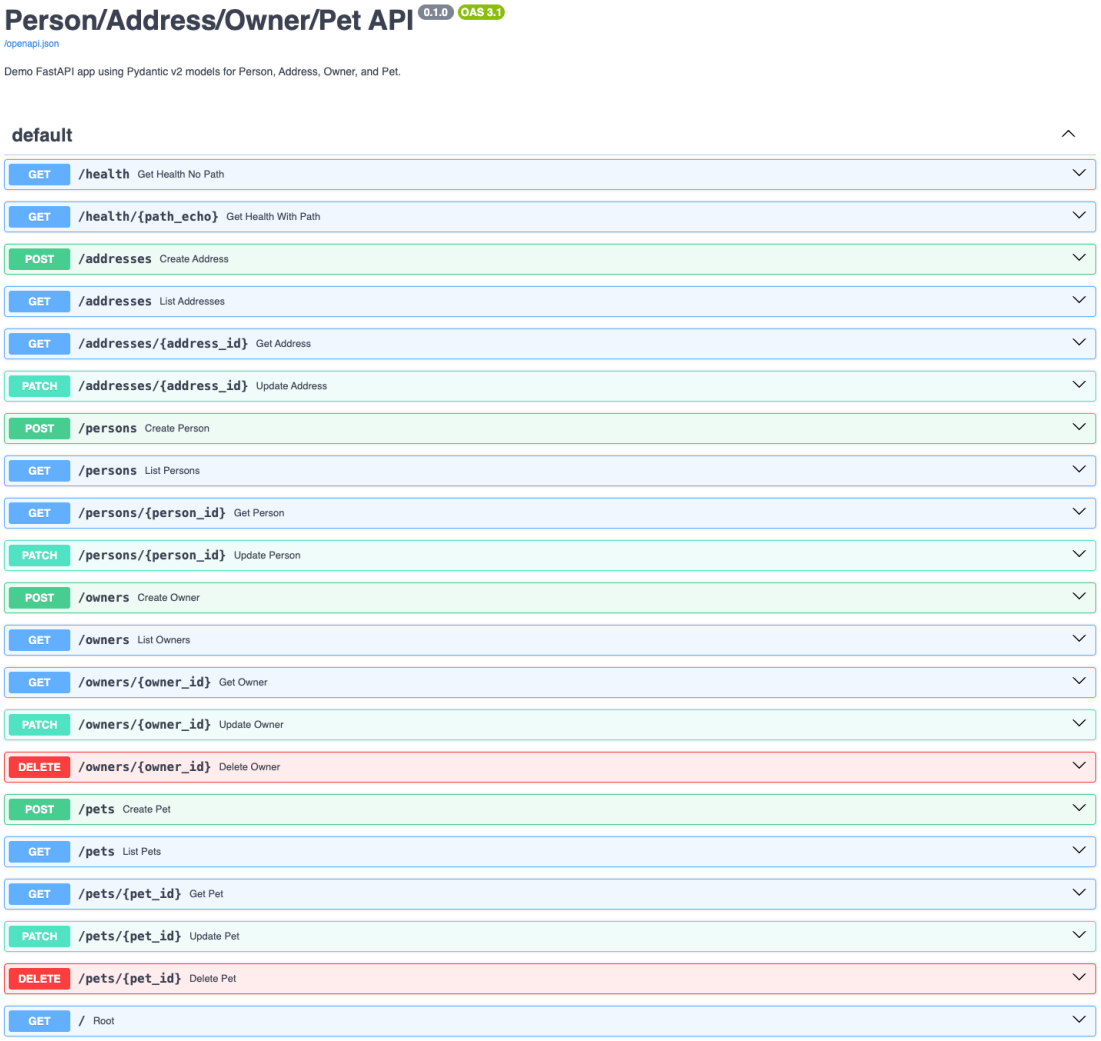
```
        results = [p for p in results if p.breed == breed]
    if color is not None:
        results = [p for p in results if p.color == color]
    if birth_date is not None:
        results = [p for p in results if str(p.birth_date) ==
birth_date]

    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 ➡️SOON

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]