UDACITY

‹ Return to Classroom

DISCUSS ON STUDENT HUB ›

UdaConnect

REVIEW

CODE REVIEW

HISTORY

Requires Changes

3 specifications require changes

Excellent Job on the very first submission 👏👏

Amazing work throughout and I was truly happy to see your learnings and the progress you have made so far in this project. 💪💪

- `Dockerfile` for all the services are correctly created 👍👍
- `Kubernetes` manifests for all the services are correctly created 👍👍
- Good work on documenting design decisions and creating Architecture diagram 👍👍

However, there are few changes needed -

- Provide **setup instructions** or **Kubernetes manifests** for **Kafka** and **Zookeeper**
- Provide **OpenApi Spec** or **Swagger Documentation** for all the API endpoints defined in **controller.py**
- Refactor **location_api** module to remove duplicate code

Please follow the suggestions given in the rubric to make adjustments accordingly and I am confident that you will get this done next time. 💪💪

Please revisit lessons in case you are stuck or post your doubts on the Knowledge Hub.

Looking forward to your next submission.
Keep Learning

All the best 👍👍

Additional Resource

- A Practical Guide to Building an Event Streaming Platform
- Python Microservices with gRPC
- Communication in a microservice architecture

Architecture Design

✓

Each module includes a 1-2 sentence justification for the module design choice either as a label on the design diagram or in a separate document

The justifications indicate that the decisions about the chosen protocols and technologies are based on business requirements including required scale. Your supervisor wants to be able to launch this project in 2 weeks with a limited budget. At the same time, the project needs to be able to scale to handle large volumes of location data being ingested.

The project is designed as an MVP and does not include any unnecessary features

- Cost and development time are minimized
- Services should run in containers
- Design should be able to handle ingress of a large volume of data

Awesome 💯

Your architecture decision looks great justifying various Message Passing techniques, their uses in various services and advantages with reasoning 👍👍

Useful Resource

Below are some resources to help you better understand and make design decision between `gRPC` vs `REST` -

- When to use gRPC over REST
- Understanding gRPC, OpenAPI and REST and when to use them in API design
- GRPC VS REST: COMPARING APIS ARCHITECTURAL STYLES

✓

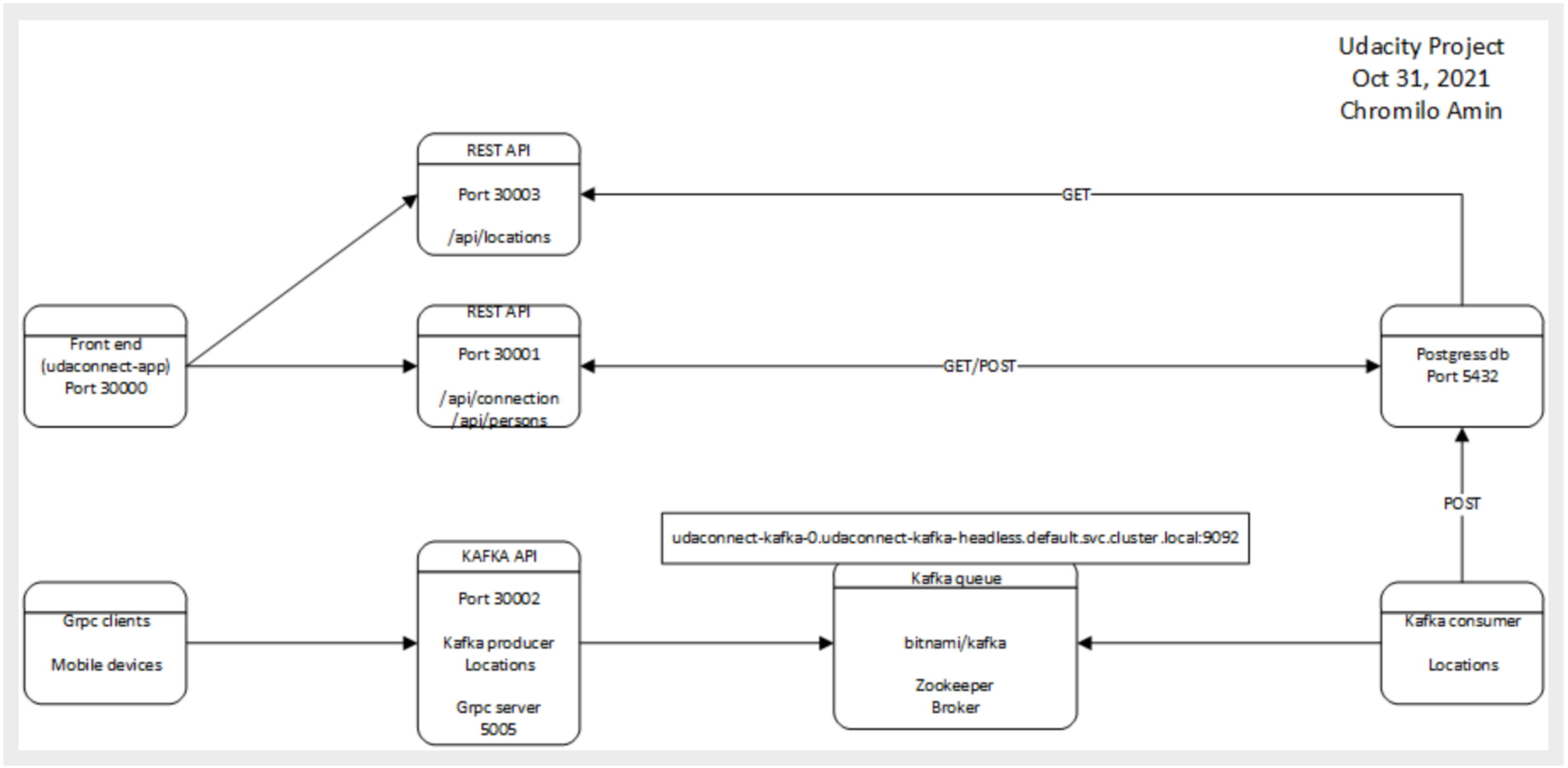
Architecture diagram shows the design of the system as individual services. It should show the relationship between the frontend, API's, databases, and Kafka.

Arrows or lines connect the individual services to represent request/response relationships between services.

All of the necessary services and protocols are included as modules in the architectural design

Awesome 💯

Your architecture diagram correctly demonstrating overall flow of the project showing all the required services, databases, message passing protocols and relationship between them.



The direction of arrows correctly demonstrating the request flow 👏👏

Useful Resource

Below are some good resources to keep in mind when creating `MicroService Architecture` -

- Microservice Architecture and its 10 Most Important Design Patterns
- Microservices Architecture Pattern

Below are some example architecture diagrams for your reference -

- Microservice Architecture Examples and Diagram

Microservice Development

✓

- Services have their own `Dockerfile`'s
- Services are deployable with Kubernetes through Kubernetes `Deployment`'s and `Service`'s.

- **You have created `Dockerfile` for all the required Services** ☑ Your `Dockerfile` contains command for copying required files, installing dependencies, exposing Port and appropriate start-up CMD 👍👍
- **You have created `Kubernetes` manifests for all the required Services** ☑ All the yaml files contains `Deployment` and `Service` resources 👍👍

Well done 🎉🎉

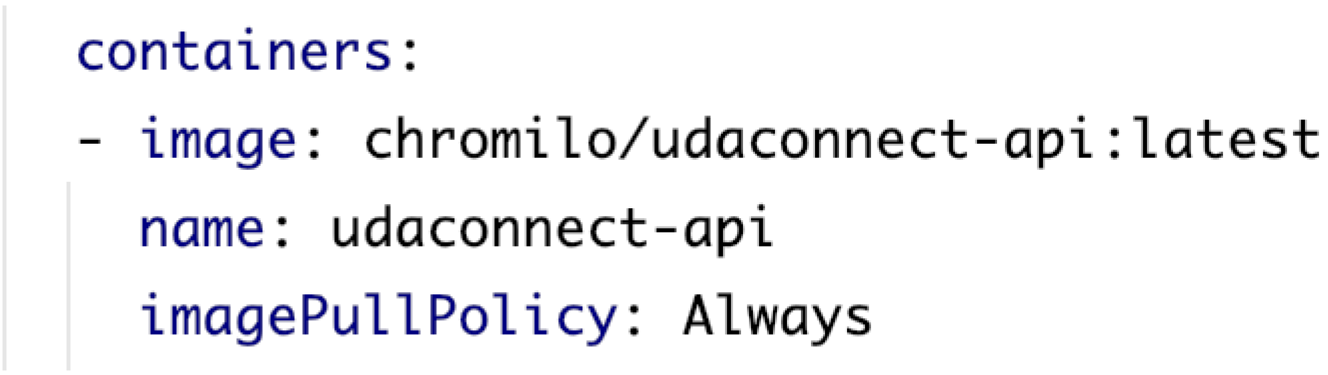
Further Reading

- Best practices for writing Dockerfiles
- Docker development best practices
- Deployments in K8S
- Services in K8S

✓

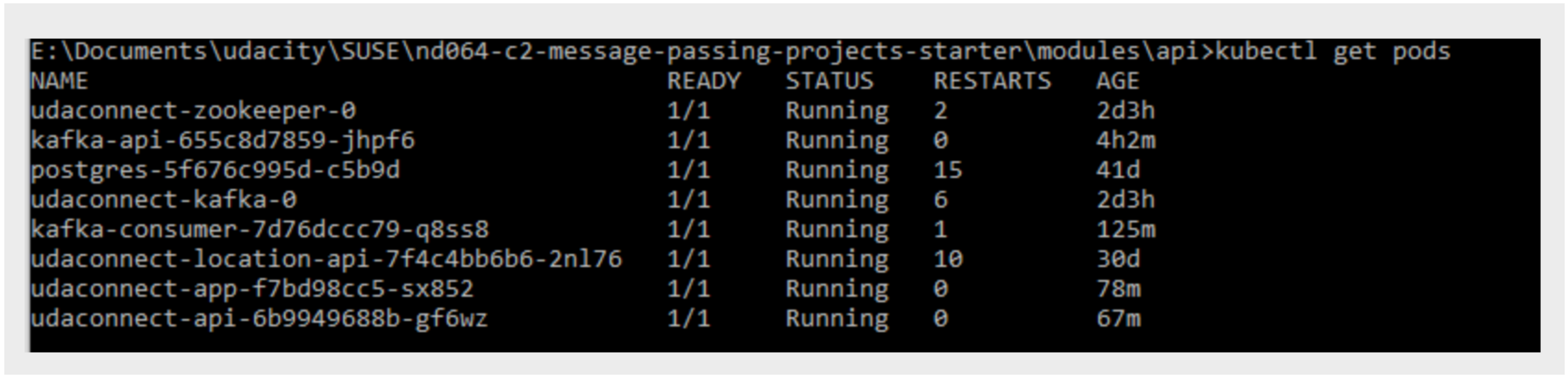
- Kubernetes deployments use Docker images built from the students' final solution and not from the starter code.

All the `Kubernetes` yaml files are using Docker images built from your solution ☑



```
containers:
- image: chromilo/udaconnect-api:latest
  name: udaconnect-api
  imagePullPolicy: Always
```

You have correctly deployed all the `Kubernetes` manifests 👍👍



```
E:\Documents\udacity\SUSE\nd064-c2-message-passing-projects-starter\modules\api>kubectl get pods
NAME                                          READY   STATUS    RESTARTS   AGE
udaconnect-zookeeper-0                        1/1     Running   2          2d3h
kafka-api-655c8d7859-jhpf6                    1/1     Running   0          4h2m
postgres-5f676c995d-c5b9d                     1/1     Running   15         41d
udaconnect-kafka-0                            1/1     Running   6          2d3h
kafka-consumer-7d76dccc79-q8ss8               1/1     Running   1          125m
udaconnect-location-api-7f4c4bb6b6-2nl76      1/1     Running   10         30d
udaconnect-app-f7bd98cc5-sx852                1/1     Running   0          78m
udaconnect-api-6b9949688b-gf6wz               1/1     Running   0          67m
```

Well done 🎉

Further Reading

- Deploying K8S cluster using k3s

- Deploying K8s cluster using k3s

Message Passing

- Project implements a Kafka queue in a container in a `Dockerfile` or Kubernetes deployment file. This can be configured manually in a base image or using a pre-built Kafka Docker image.
- `requirements.txt` file installs the `kafka-python` Kafka library.
- `kafka` deployment runs successfully without errors.

- `requirements.txt` **file installs the** `kafka-python` **Kafka library** ☑
- `KafkaProducer` **and** `KafkaConsumer` **is used correctly** ☑
- `kafka` **deployment runs successfully without errors** ☑

Well done 🎉

Requires Changes

- **Project implements a Kafka queue in a container in a Dockerfile or Kubernetes deployment file** ✗ *I can see the* `pods` *and* `services` *for* `kafka` *and* `zookeeper` *in the provided screenshot but their* ***set up instructions*** *or* `Kubernetes` ***manifests*** *are missing in your submission. Please provide either of them to pass this rubric*

✓

- Project contains a `*.proto` file showing that they mapped a message and service into a protobuf format. The `*.proto` file should have at least one `message` and one `service` declared in it.
- Code should contain a `*_pb2` and `*_pb2_grpc` file generated from the `*.proto` file.
- Project contains a gRPC client. If using Python with the standard `grpc` library, code should open a gPRC channel.
- Project contains a gPRC host. The standard `gprc` library code should contain a `grpc.server()` instantiation.
- `requirements.txt` file should define the `grpcio` package.

Awesome 💯

- `.proto` **file have at least one** `message` **and one** `service` **declared in it** ✅ You have correctly defined a `protobuf` message and used the same in the Service. 👍👍
- **Code contains a** `_pb2` **and** `_pb2_grpc` **file generated from the** `.proto` **file** ✅ You have correctly generated the `_pb2` and `_pb2_grpc` python files. 👍👍
- **a** `gRPC server` **is created** ✅ You have correctly created server using `add_insecure_port` method 👍👍
- **a** `gRPC client` **is created and should open a** `gPRC channel` ✅ You have correctly created client using `grpc.insecure_channel` method 👍👍
- `requirements.txt` **file should define the** `grpcio` **package** ✅ Your `requirements.txt` file contains `grpcio` library 👍👍

Further Reading

- gRPC Performance Best Practices
- Protocol Buffers
- gRPC services vs HTTP APIs

🔄

- Project either created a new API endpoint(s) or a modification to existing Flask API.
- All new API endpoints use proper HTTP request types.
- `GET` or `DELETE` request does not contain an HTTP payload
- REST API's have live Swagger documentation in an external library (or manually written an OpenAPI spec.

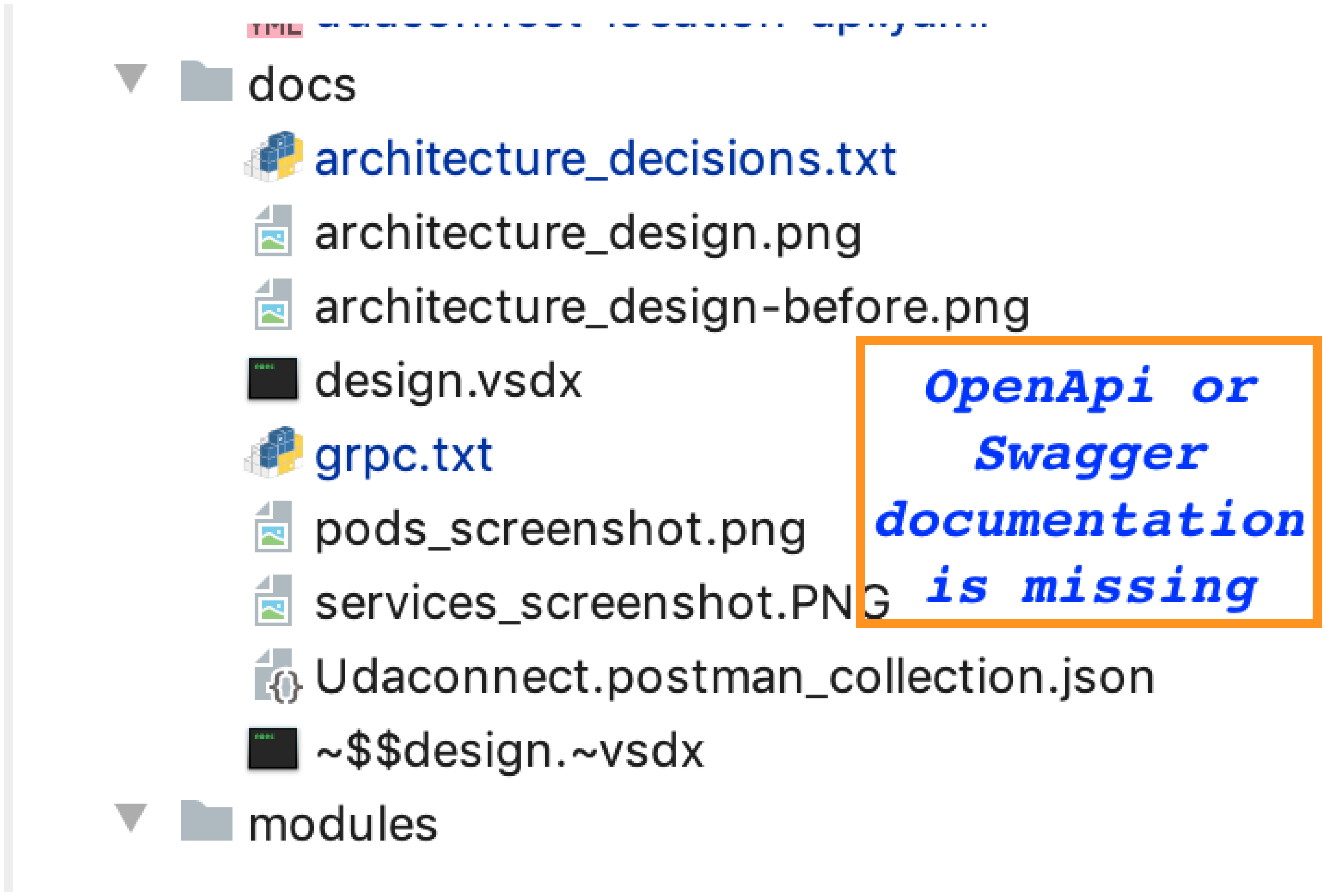
- **All the APIs are correctly configured with proper HTTP types in the** `controllers.py` ✅
- `GET` **or** `DELETE` **request does not contain an** `HTTP` **payload** ✅

Well done 🎉

Requires Changes

Brilliant job so far. I am glad to see your progress. 👏👏

However, As per this rubric, you need to provide ***OpenApi Spec*** *or* ***Swagger Documentation*** *for all the API endpoints defined in* `controller.py` *which is currently missing in your submission.*



```
▼ docs
    architecture_decisions.txt
    architecture_design.png
    architecture_design-before.png
    design.vsdx
    grpc.txt
    pods_screenshot.png
    services_screenshot.PNG
    Udaconnect.postman_collection.json
    ~$$design.~vsdx
▼ modules
```

OpenApi or Swagger documentation is missing

Please provide them to pass this rubric 🙏🙏

Useful Resource

- Automatically Generate OpenAPI Specifications & Documentation with Python

Code Quality

Project runs without error

Code is appropriately documented. Comments are concise, relevant and not excessive

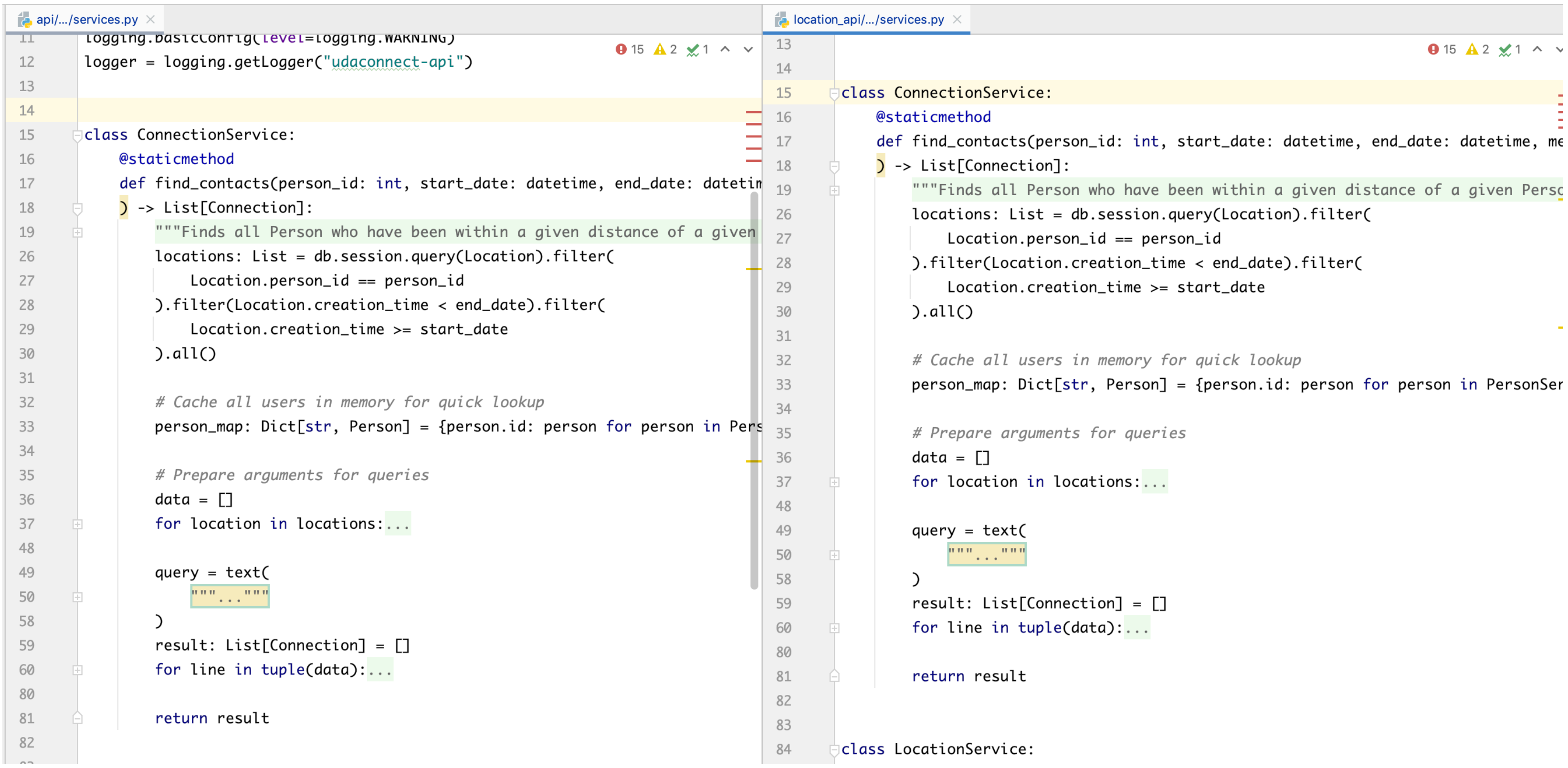
Code is neatly written with proper indenting. Python code follows PEP 8 guidelines.

Awesome 💯

Your code runs without any error 👏🏼👏🏼

Requires Changes

*Currently, your **api** and **location_api** module (**services.py**) contains duplicate code for **ConnectionService**, **LocationService** and **PersonService** implementations as shown below -*

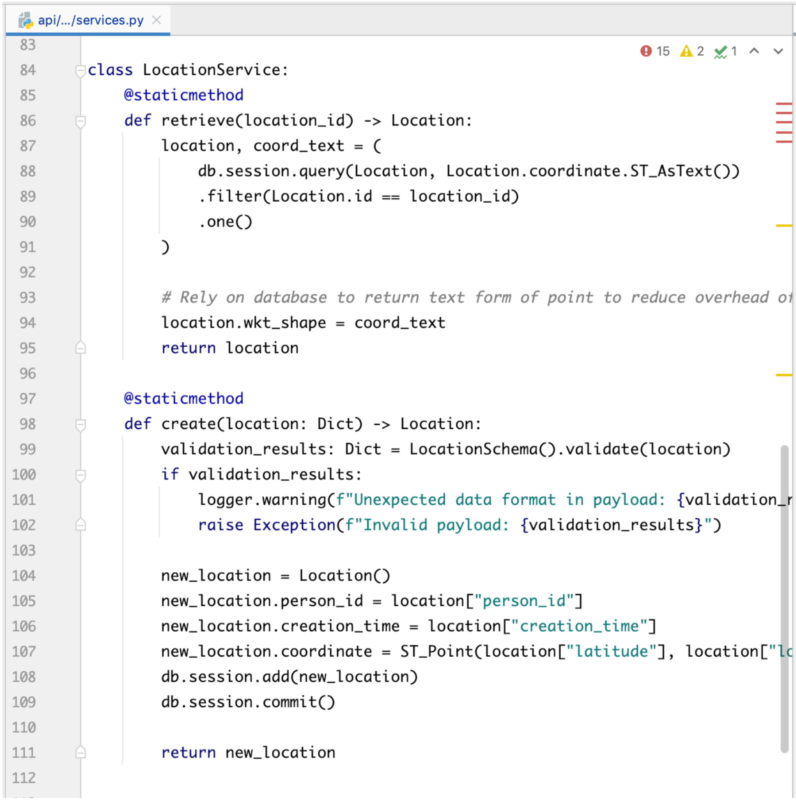


api/.../services.py

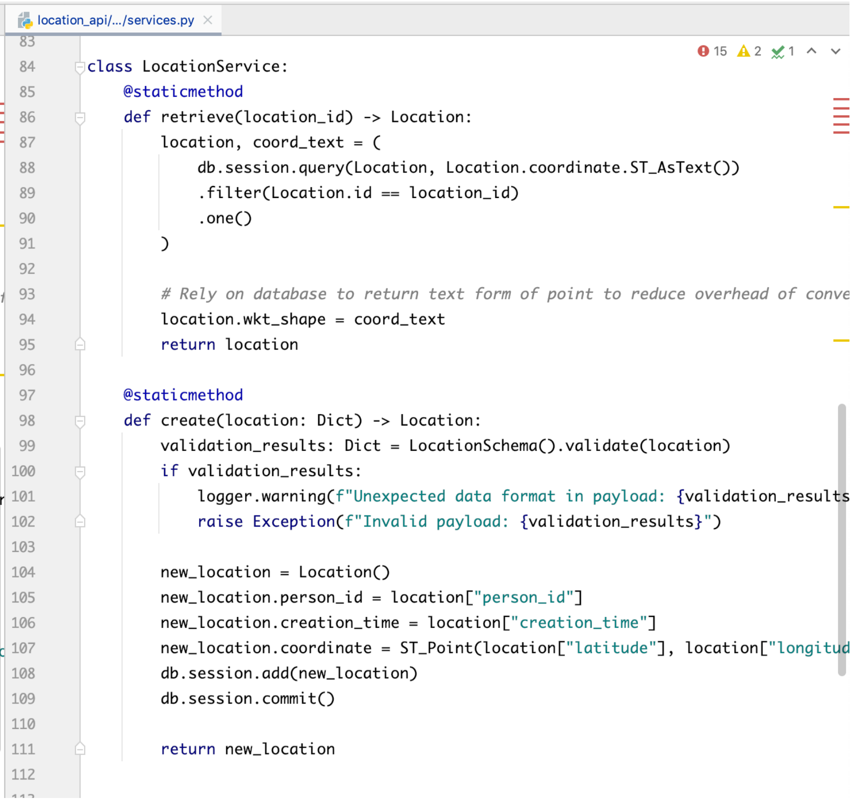
```
logging.basicConfig(level=logging.WARNING)
logger = logging.getLogger("udaconnect-api")


class ConnectionService:
    @staticmethod
    def find_contacts(person_id: int, start_date: datetime, end_date: datetim
    ) -> List[Connection]:
        """Finds all Person who have been within a given distance of a given
        locations: List = db.session.query(Location).filter(
            Location.person_id == person_id
        ).filter(Location.creation_time < end_date).filter(
            Location.creation_time >= start_date
        ).all()

        # Cache all users in memory for quick lookup
        person_map: Dict[str, Person] = {person.id: person for person in Pers

        # Prepare arguments for queries
        data = []
        for location in locations:...

        query = text(
            """..."""
        )
        result: List[Connection] = []
        for line in tuple(data):...

        return result

```

location_api/.../services.py

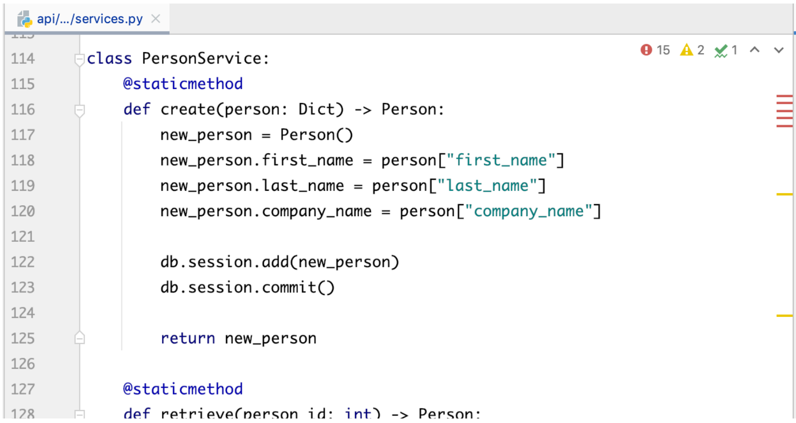
```


class ConnectionService:
    @staticmethod
    def find_contacts(person_id: int, start_date: datetime, end_date: datetime, me
    ) -> List[Connection]:
        """Finds all Person who have been within a given distance of a given Perso
        locations: List = db.session.query(Location).filter(
            Location.person_id == person_id
        ).filter(Location.creation_time < end_date).filter(
            Location.creation_time >= start_date
        ).all()

        # Cache all users in memory for quick lookup
        person_map: Dict[str, Person] = {person.id: person for person in PersonSer

        # Prepare arguments for queries
        data = []
        for location in locations:...

        query = text(
            """..."""
        )
        result: List[Connection] = []
        for line in tuple(data):...

        return result


class LocationService:
```



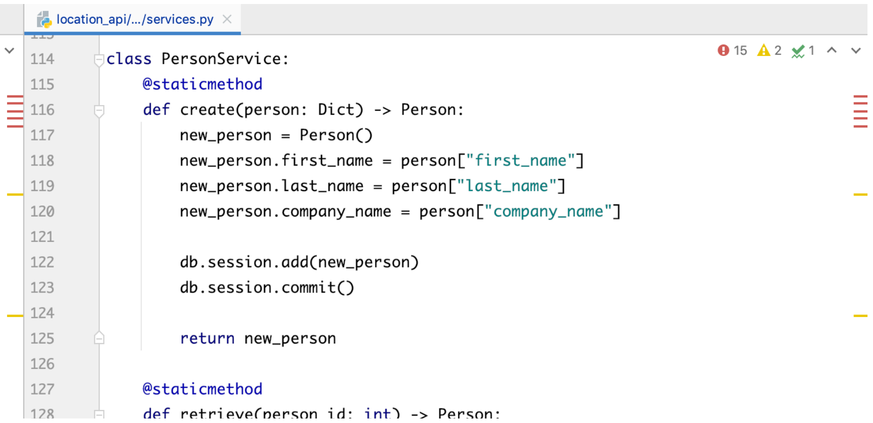
```python
class LocationService:
    @staticmethod
    def retrieve(location_id) -> Location:
        location, coord_text = (
            db.session.query(Location, Location.coordinate.ST_AsText())
            .filter(Location.id == location_id)
            .one()
        )

        # Rely on database to return text form of point to reduce overhead of
        location.wkt_shape = coord_text
        return location

    @staticmethod
    def create(location: Dict) -> Location:
        validation_results: Dict = LocationSchema().validate(location)
        if validation_results:
            logger.warning(f"Unexpected data format in payload: {validation_r
            raise Exception(f"Invalid payload: {validation_results}")

        new_location = Location()
        new_location.person_id = location["person_id"]
        new_location.creation_time = location["creation_time"]
        new_location.coordinate = ST_Point(location["latitude"], location["lo
        db.session.add(new_location)
        db.session.commit()

        return new_location
```



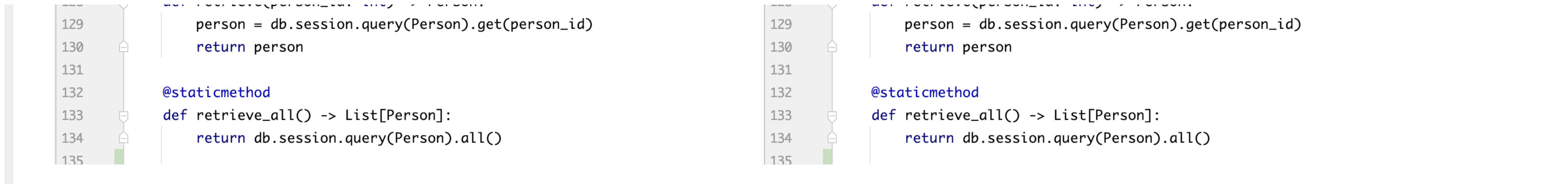
```python
class LocationService:
    @staticmethod
    def retrieve(location_id) -> Location:
        location, coord_text = (
            db.session.query(Location, Location.coordinate.ST_AsText())
            .filter(Location.id == location_id)
            .one()
        )

        # Rely on database to return text form of point to reduce overhead of conve
        location.wkt_shape = coord_text
        return location

    @staticmethod
    def create(location: Dict) -> Location:
        validation_results: Dict = LocationSchema().validate(location)
        if validation_results:
            logger.warning(f"Unexpected data format in payload: {validation_results
            raise Exception(f"Invalid payload: {validation_results}")

        new_location = Location()
        new_location.person_id = location["person_id"]
        new_location.creation_time = location["creation_time"]
        new_location.coordinate = ST_Point(location["latitude"], location["longitud
        db.session.add(new_location)
        db.session.commit()

        return new_location
```



```python
class PersonService:
    @staticmethod
    def create(person: Dict) -> Person:
        new_person = Person()
        new_person.first_name = person["first_name"]
        new_person.last_name = person["last_name"]
        new_person.company_name = person["company_name"]

        db.session.add(new_person)
        db.session.commit()

        return new_person

    @staticmethod
    def retrieve(person_id: int) -> Person:
```



```python
class PersonService:
    @staticmethod
    def create(person: Dict) -> Person:
        new_person = Person()
        new_person.first_name = person["first_name"]
        new_person.last_name = person["last_name"]
        new_person.company_name = person["company_name"]

        db.session.add(new_person)
        db.session.commit()

        return new_person

    @staticmethod
    def retrieve(person_id: int) -> Person:
```



```
        person = db.session.query(Person).get(person_id)
        return person

    @staticmethod
    def retrieve_all() -> List[Person]:
        return db.session.query(Person).all()
```

```
        person = db.session.query(Person).get(person_id)
        return person

    @staticmethod
    def retrieve_all() -> List[Person]:
        return db.session.query(Person).all()
```

Suggestion

*Please refactor your **location_api** module to contain implementation for only **LocationService** and use the same in **api** module **ConnectionService** implementation to retrieve* `Location` *data*

RESUBMIT

DOWNLOAD PROJECT

U
UDACITY

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH

Rate this review

START