DRF에 DCI 패턴 적용해보기

ESMOND

목차

1. DCI 패턴이란

1. DCI 패턴이란

DATA - CONTEXT - INTERACTION

1. DCI 패턴이란

위키피디아 왈

Data, context, and interaction (DCI) is a paradigm used in computer software to program systems of communicating objects

1. DCI 패턴이란

위키피디아 왈

Its goals are:

1. To improve the readability of object-oriented code by giving system behavior first-class status;

1. DCI 패턴이란

위키피디아 왈

Its goals are:

1. To improve the readability of object-oriented code by giving system behavior first-class status;

2. To cleanly separate code for rapidly changing system behavior (what a system *does*) versus slowly changing domain knowledge (what a system *is*), instead of combining both in one class interface

1. DCI 패턴이란

위키피디아 왈

Its goals are:

1. To improve the readability of object-oriented code by giving system behavior first-class status;
2. To cleanly separate code for rapidly changing system behavior (what a system *does*) versus slowly changing domain knowledge (what a system *is*), instead of combining both in one class interface
3. To help software developers reason about system-level state and behavior instead of only object state and behavior

1. DCI 패턴이란

위키피디아 왈

Its goals are:

1. To improve the readability of object-oriented code by giving system behavior first-class status;
2. To cleanly separate code for rapidly changing system behavior (what a system *does*) versus slowly changing domain knowledge (what a system *is*), instead of combining both in one class interface
3. To help software developers reason about system-level state and behavior instead of only object state and behavior
4. To support an object style of thinking that is close to programmers' mental models, rather than the class style of thinking that overshadowed object thinking early in the history of object-oriented programming languages.

1. DCI 패턴이란

The DCI Architecture: A New Vision of Object-Oriented Programming

https://www.artima.com/articles/dci_vision.html

2. DRF 에서 어떻게 적용할까

2. DRF 에서 어떻게 적용할까

기본 DRF의 구조

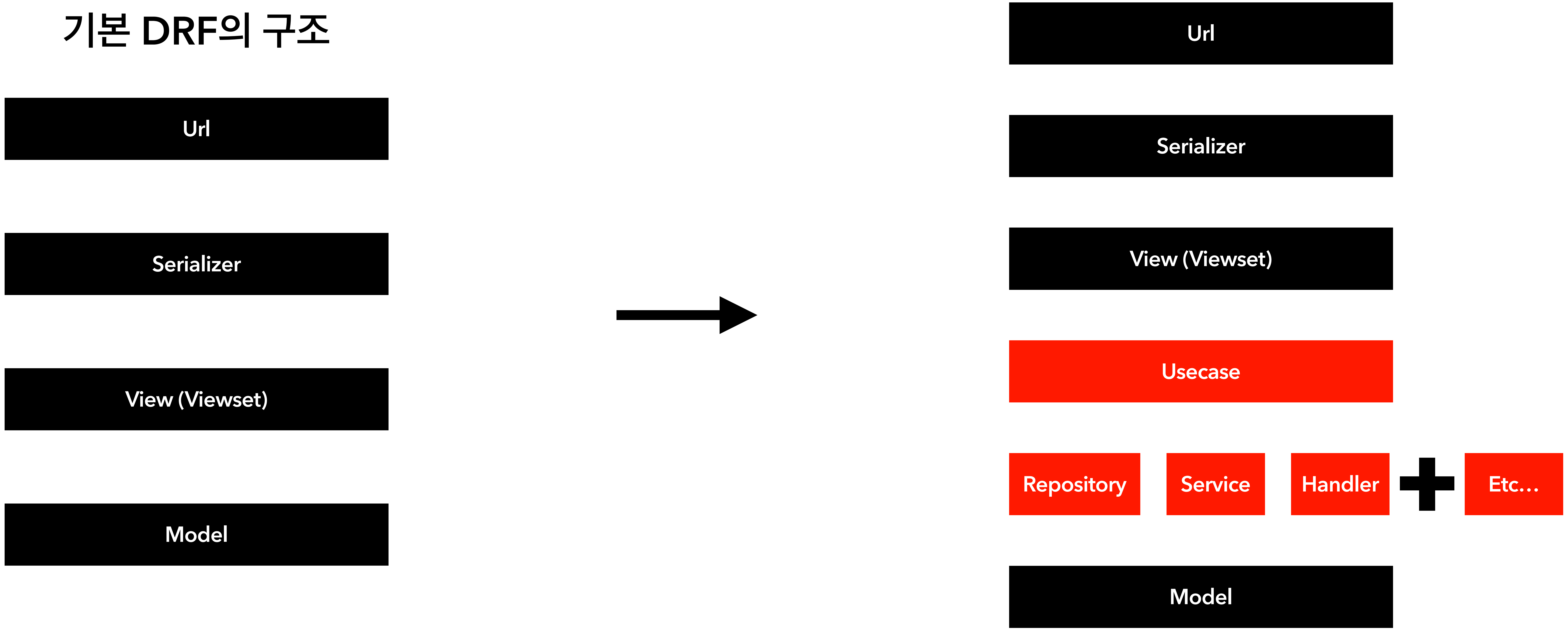
Url

Serializer

View (Viewset)

Model

2. DRF 에서 어떻게 적용할까



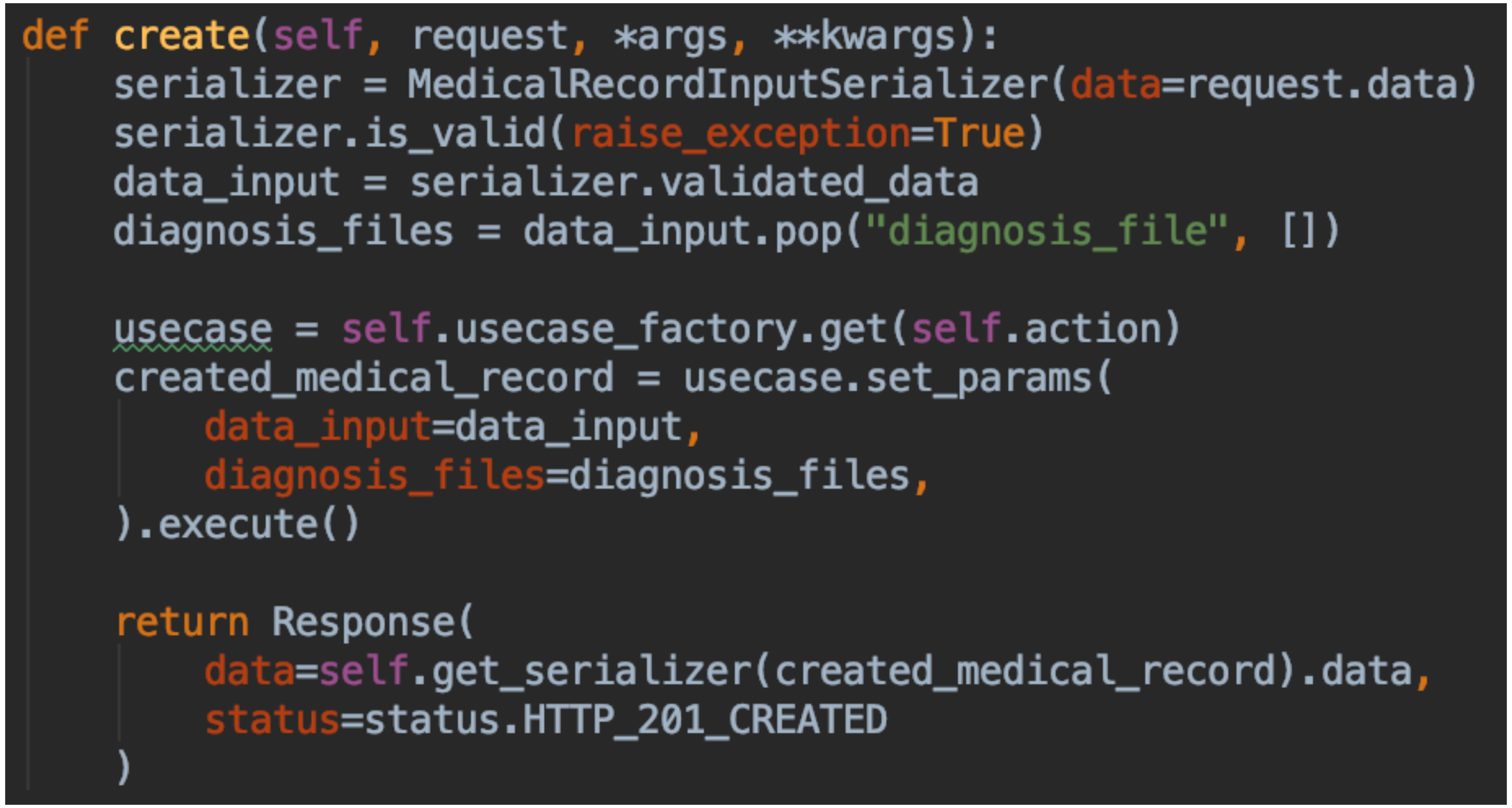
2. DRF 에서 어떻게 적용할까

예시 시나리오: 강아지의 의료기록 업로드 프로세스

펫 객체 쿼리	Pet Repo
의료기록 파일 S3 업로드	S3 Service
S3 object key와 함께 의료기록 모델 저장	Medical Record Repo
Worker에게 이미지 파일을 pdf로 바꿀 것을 지시	Celery Worker

2. DRF 에서 어떻게 적용할까

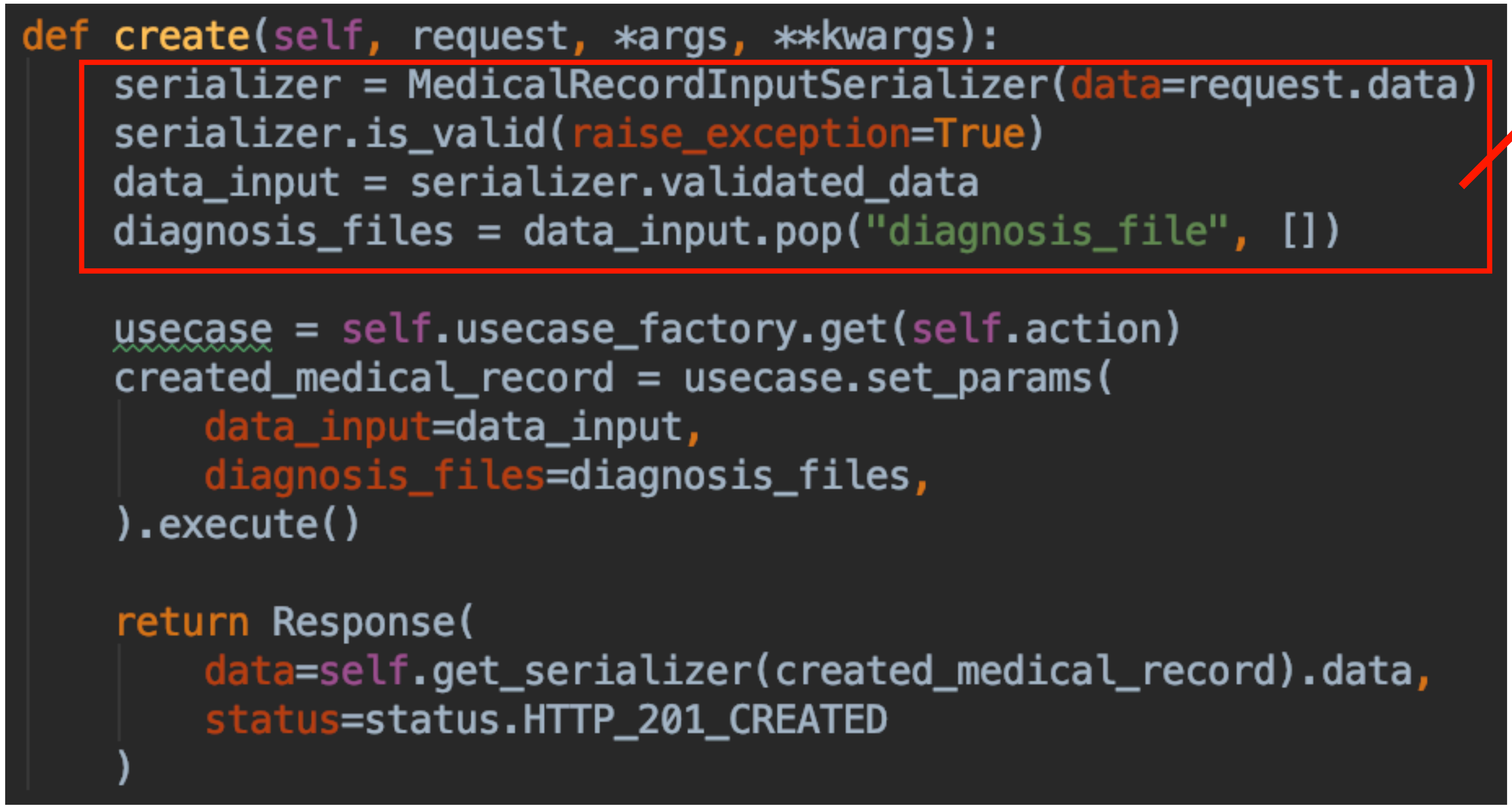
- View 메서드



```python
def create(self, request, *args, **kwargs):
    serializer = MedicalRecordInputSerializer(data=request.data)
    serializer.is_valid(raise_exception=True)
    data_input = serializer.validated_data
    diagnosis_files = data_input.pop("diagnosis_file", [])

    usecase = self.usecase_factory.get(self.action)
    created_medical_record = usecase.set_params(
        data_input=data_input,
        diagnosis_files=diagnosis_files,
    ).execute()

    return Response(
        data=self.get_serializer(created_medical_record).data,
        status=status.HTTP_201_CREATED
    )
```

2. DRF 에서 어떻게 적용할까

- View 메서드

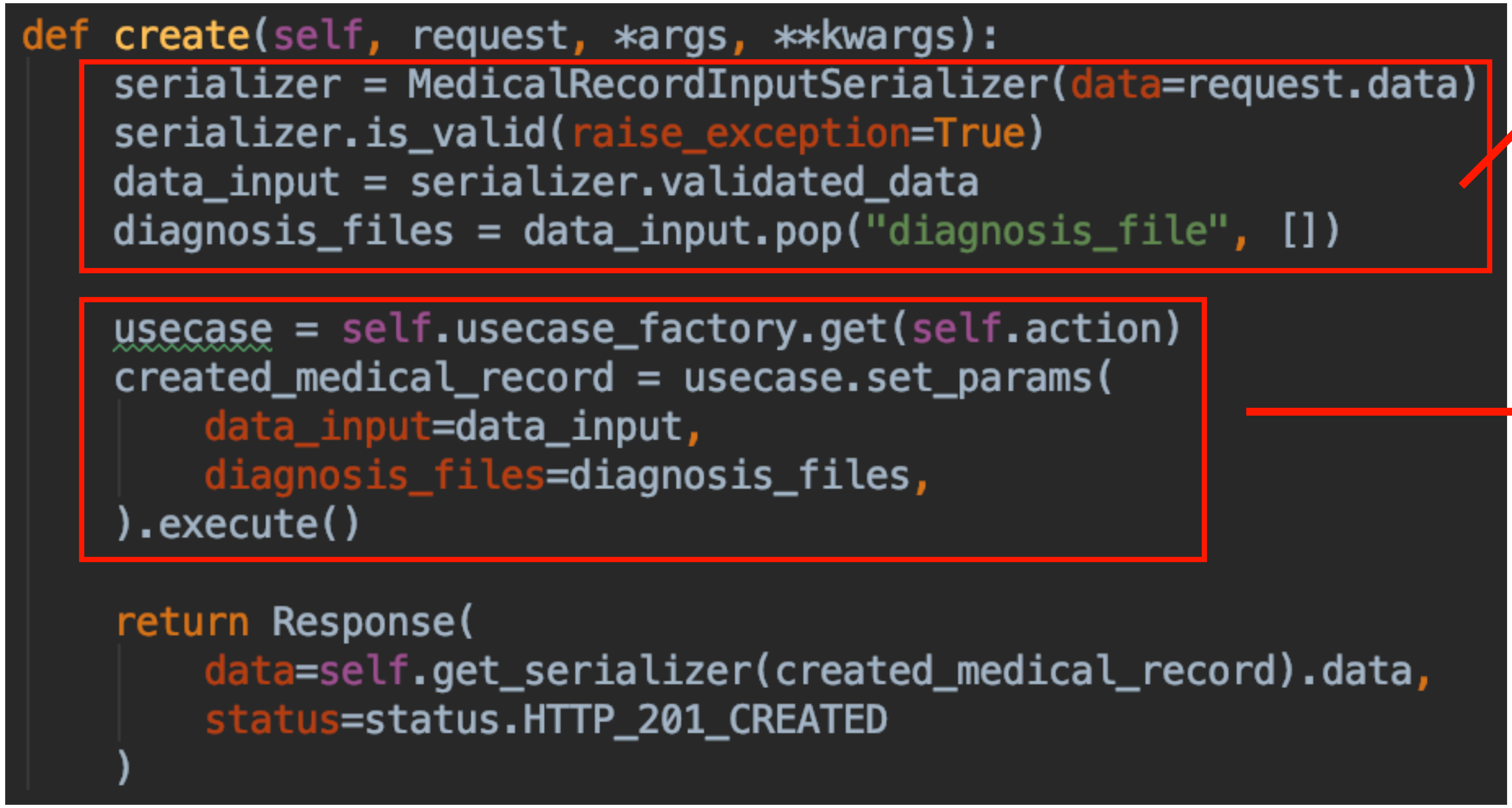


```
def create(self, request, *args, **kwargs):
    serializer = MedicalRecordInputSerializer(data=request.data)
    serializer.is_valid(raise_exception=True)
    data_input = serializer.validated_data
    diagnosis_files = data_input.pop("diagnosis_file", [])

    usecase = self.usecase_factory.get(self.action)
    created_medical_record = usecase.set_params(
        data_input=data_input,
        diagnosis_files=diagnosis_files,
    ).execute()

    return Response(
        data=self.get_serializer(created_medical_record).data,
        status=status.HTTP_201_CREATED
    )
```

view는 request를 처음 받아서 usecase에 들어갈 데이터들을 serializing(+validation) 해서 usecase로 인자들을 넘겨줌

2. DRF 에서 어떻게 적용할까

- View 메서드



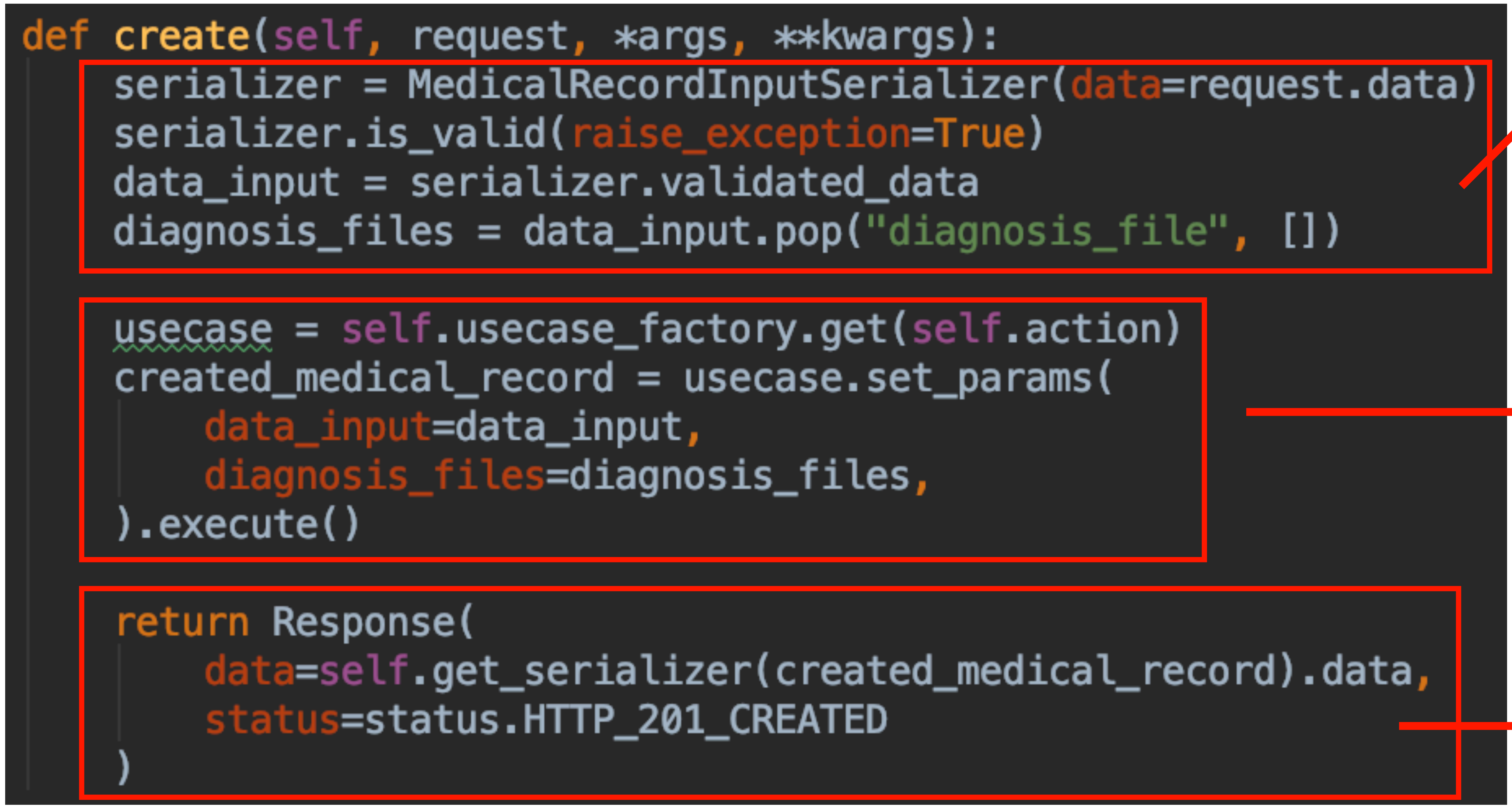
```python
def create(self, request, *args, **kwargs):
    serializer = MedicalRecordInputSerializer(data=request.data)
    serializer.is_valid(raise_exception=True)
    data_input = serializer.validated_data
    diagnosis_files = data_input.pop("diagnosis_file", [])

    usecase = self.usecase_factory.get(self.action)
    created_medical_record = usecase.set_params(
        data_input=data_input,
        diagnosis_files=diagnosis_files,
    ).execute()

    return Response(
        data=self.get_serializer(created_medical_record).data,
        status=status.HTTP_201_CREATED
    )
```

view는 request를 처음 받아서 usecase에 들어갈 데이터들을 serializing(+validation) 해서 usecase로 인자들을 넘겨줌

usecase_factory에서 생성된 usecase를 받아오고
set_params()로 인자를 설정하고
execute()로 실행하며 결과값을 받아옴

2. DRF 에서 어떻게 적용할까

- View 메서드



```
def create(self, request, *args, **kwargs):
    serializer = MedicalRecordInputSerializer(data=request.data)
    serializer.is_valid(raise_exception=True)
    data_input = serializer.validated_data
    diagnosis_files = data_input.pop("diagnosis_file", [])

    usecase = self.usecase_factory.get(self.action)
    created_medical_record = usecase.set_params(
        data_input=data_input,
        diagnosis_files=diagnosis_files,
    ).execute()

    return Response(
        data=self.get_serializer(created_medical_record).data,
        status=status.HTTP_201_CREATED
    )
```

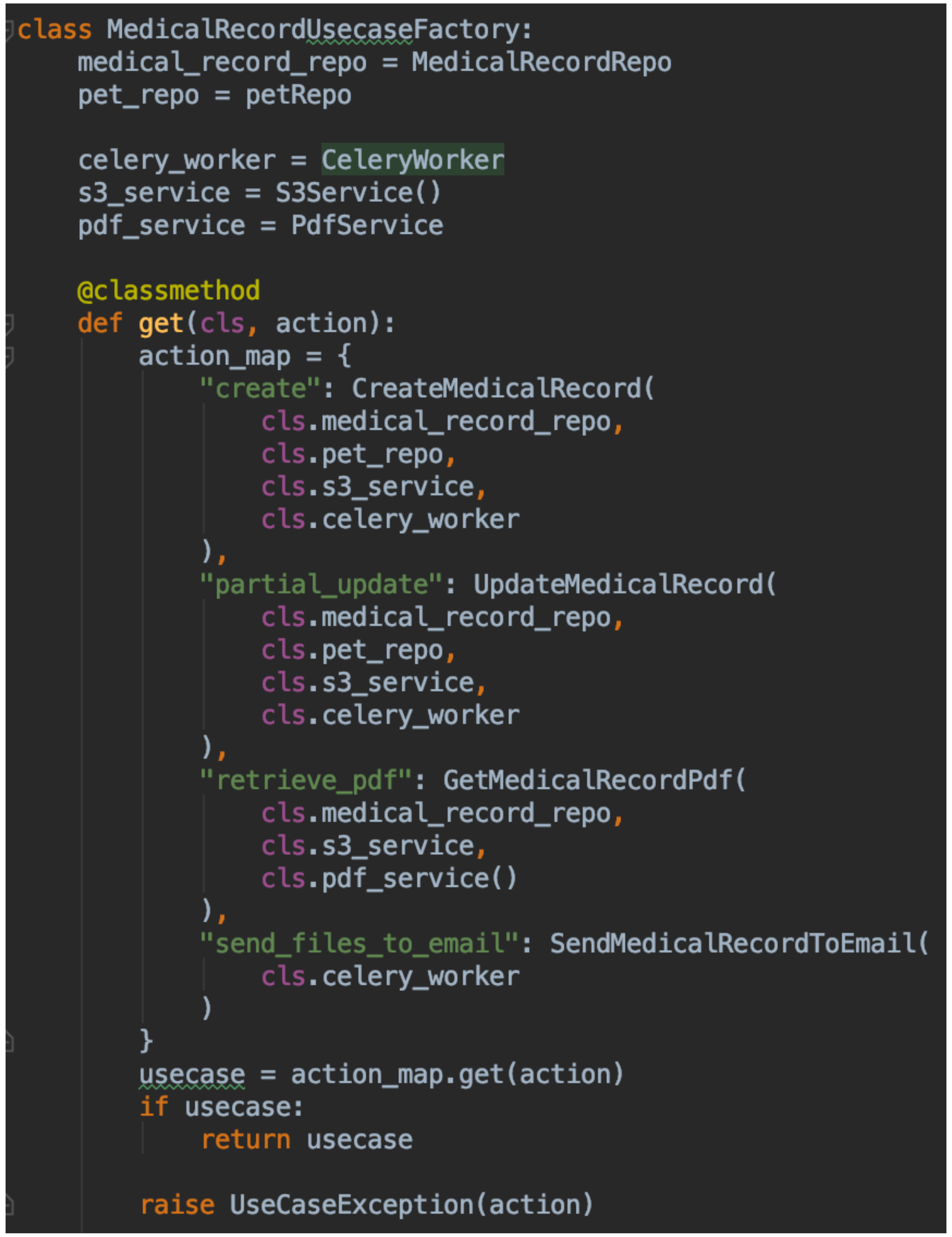
view는 request를 처음 받아서 usecase에 들어갈 데이터들을 serializing(+validation) 해서 usecase로 인자들을 넘겨줌

usecase_factory에서 생성된 usecase를 받아오고
set_params()로 인자를 설정하고
execute()로 실행하며 결과값을 받아옴

usecase가 반환한 값을 바탕으로 Response

2. DRF 에서 어떻게 적용할까

- Medical Record Usecase factory

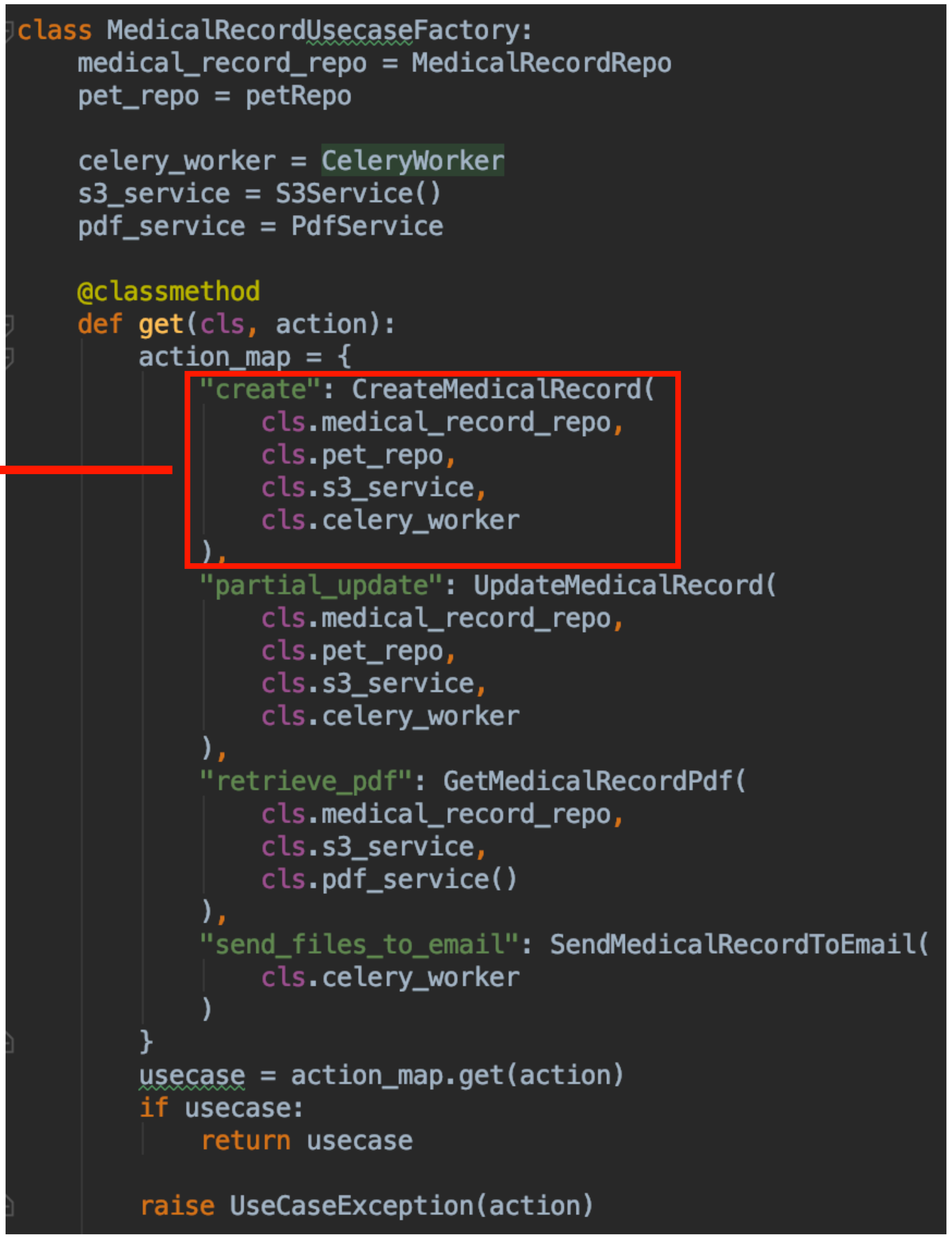


```python
class MedicalRecordUsecaseFactory:
    medical_record_repo = MedicalRecordRepo
    pet_repo = petRepo

    celery_worker = CeleryWorker
    s3_service = S3Service()
    pdf_service = PdfService

    @classmethod
    def get(cls, action):
        action_map = {
            "create": CreateMedicalRecord(
                cls.medical_record_repo,
                cls.pet_repo,
                cls.s3_service,
                cls.celery_worker
            ),
            "partial_update": UpdateMedicalRecord(
                cls.medical_record_repo,
                cls.pet_repo,
                cls.s3_service,
                cls.celery_worker
            ),
            "retrieve_pdf": GetMedicalRecordPdf(
                cls.medical_record_repo,
                cls.s3_service,
                cls.pdf_service()
            ),
            "send_files_to_email": SendMedicalRecordToEmail(
                cls.celery_worker
            )
        }
        usecase = action_map.get(action)
        if usecase:
            return usecase

        raise UseCaseException(action)
```

2. DRF 에서 어떻게 적용할까

- Medical Record Usecase factory

CreateMedicalRecord Usecase 객체 미리 생성



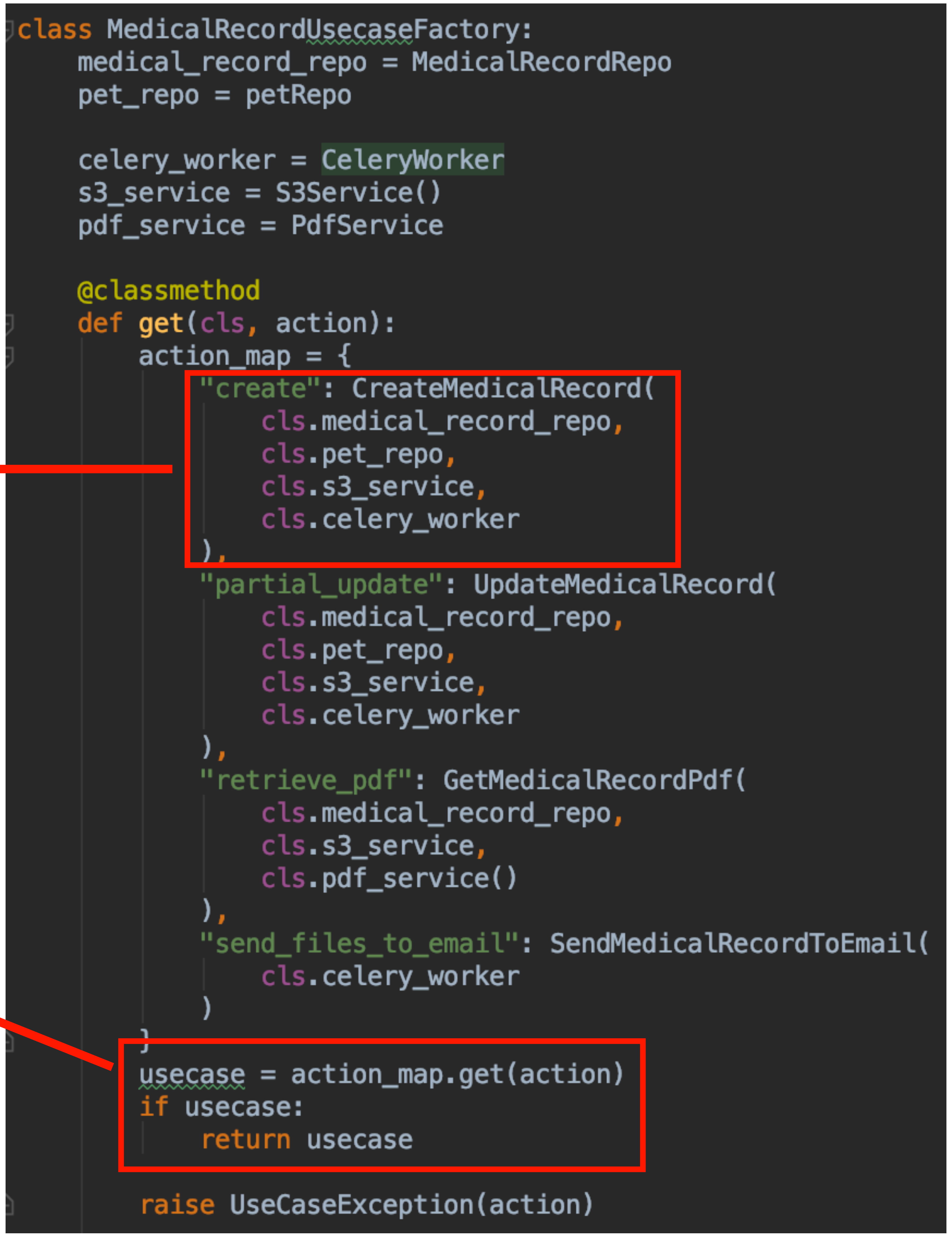
```
class MedicalRecordUsecaseFactory:
    medical_record_repo = MedicalRecordRepo
    pet_repo = petRepo

    celery_worker = CeleryWorker
    s3_service = S3Service()
    pdf_service = PdfService

    @classmethod
    def get(cls, action):
        action_map = {
            "create": CreateMedicalRecord(
                cls.medical_record_repo,
                cls.pet_repo,
                cls.s3_service,
                cls.celery_worker
            ),
            "partial_update": UpdateMedicalRecord(
                cls.medical_record_repo,
                cls.pet_repo,
                cls.s3_service,
                cls.celery_worker
            ),
            "retrieve_pdf": GetMedicalRecordPdf(
                cls.medical_record_repo,
                cls.s3_service,
                cls.pdf_service()
            ),
            "send_files_to_email": SendMedicalRecordToEmail(
                cls.celery_worker
            )
        }
        usecase = action_map.get(action)
        if usecase:
            return usecase

        raise UseCaseException(action)
```

2. DRF 에서 어떻게 적용할까

- Medical Record Usecase factory

CreateMedicalRecord Usecase 객체 미리 생성

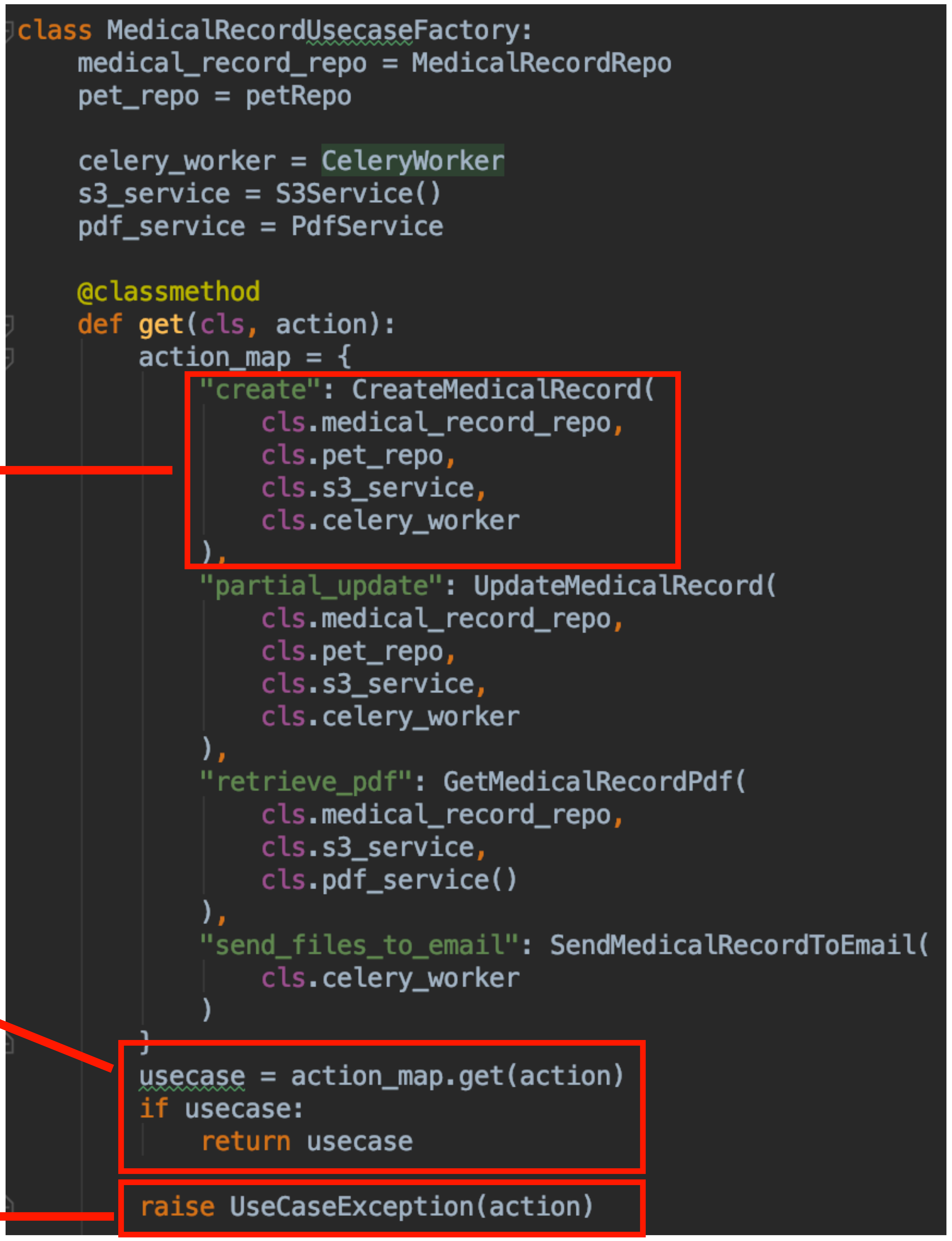
action명을 기반으로 usecase 반환



```
class MedicalRecordUsecaseFactory:
    medical_record_repo = MedicalRecordRepo
    pet_repo = petRepo

    celery_worker = CeleryWorker
    s3_service = S3Service()
    pdf_service = PdfService

    @classmethod
    def get(cls, action):
        action_map = {
            "create": CreateMedicalRecord(
                cls.medical_record_repo,
                cls.pet_repo,
                cls.s3_service,
                cls.celery_worker
            ),
            "partial_update": UpdateMedicalRecord(
                cls.medical_record_repo,
                cls.pet_repo,
                cls.s3_service,
                cls.celery_worker
            ),
            "retrieve_pdf": GetMedicalRecordPdf(
                cls.medical_record_repo,
                cls.s3_service,
                cls.pdf_service()
            ),
            "send_files_to_email": SendMedicalRecordToEmail(
                cls.celery_worker
            )
        }
        usecase = action_map.get(action)
        if usecase:
            return usecase

        raise UseCaseException(action)
```

2. DRF 에서 어떻게 적용할까

- Medical Record Usecase factory



```
class MedicalRecordUsecaseFactory:
    medical_record_repo = MedicalRecordRepo
    pet_repo = petRepo

    celery_worker = CeleryWorker
    s3_service = S3Service()
    pdf_service = PdfService

    @classmethod
    def get(cls, action):
        action_map = {
            "create": CreateMedicalRecord(
                cls.medical_record_repo,
                cls.pet_repo,
                cls.s3_service,
                cls.celery_worker
            ),
            "partial_update": UpdateMedicalRecord(
                cls.medical_record_repo,
                cls.pet_repo,
                cls.s3_service,
                cls.celery_worker
            ),
            "retrieve_pdf": GetMedicalRecordPdf(
                cls.medical_record_repo,
                cls.s3_service,
                cls.pdf_service()
            ),
            "send_files_to_email": SendMedicalRecordToEmail(
                cls.celery_worker
            )
        }
        usecase = action_map.get(action)
        if usecase:
            return usecase
        raise UseCaseException(action)
```

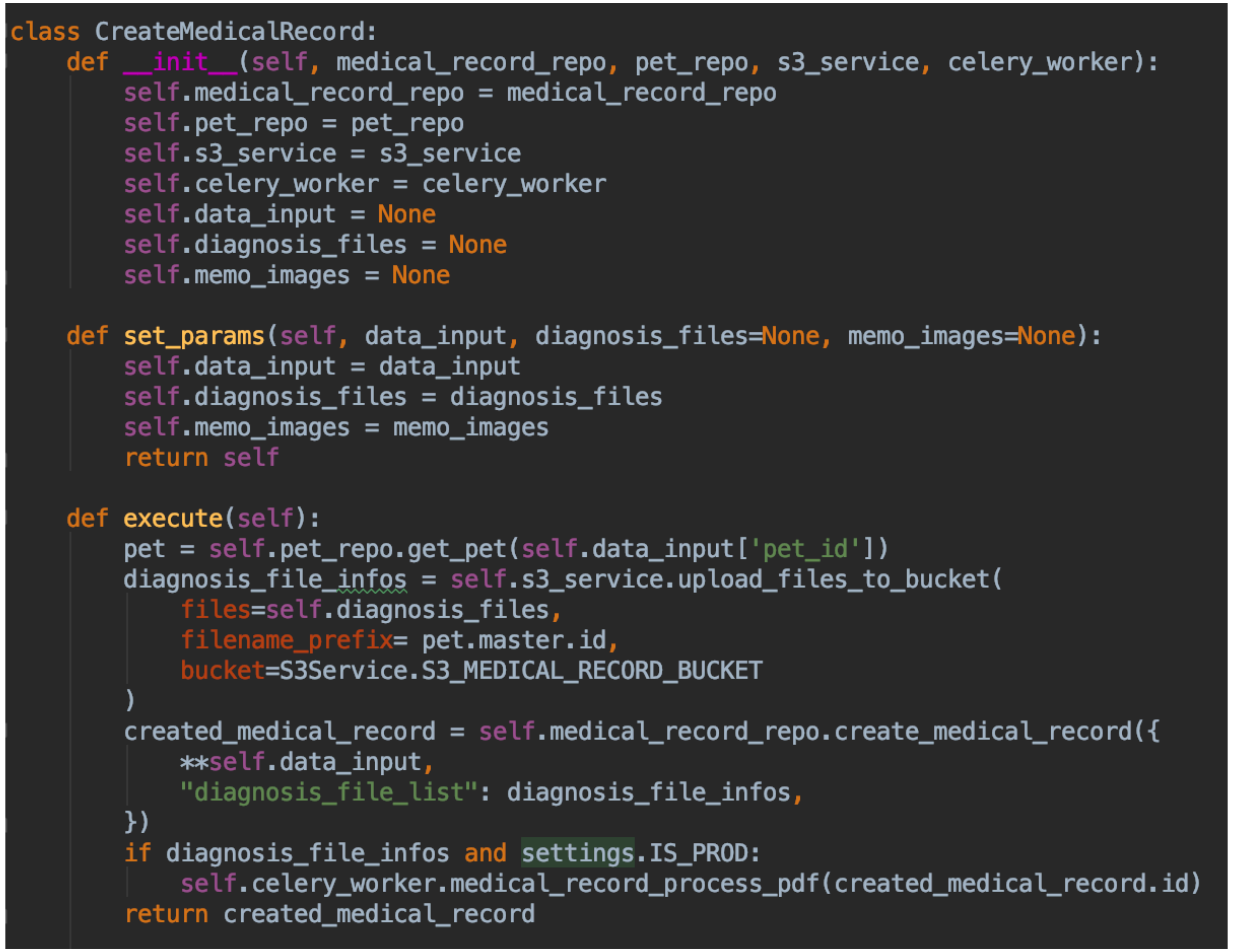
CreateMedicalRecord Usecase 객체 미리 생성

action명을 기반으로 usecase 반환

usecase 없으면 에러 발생

2. DRF 에서 어떻게 적용할까

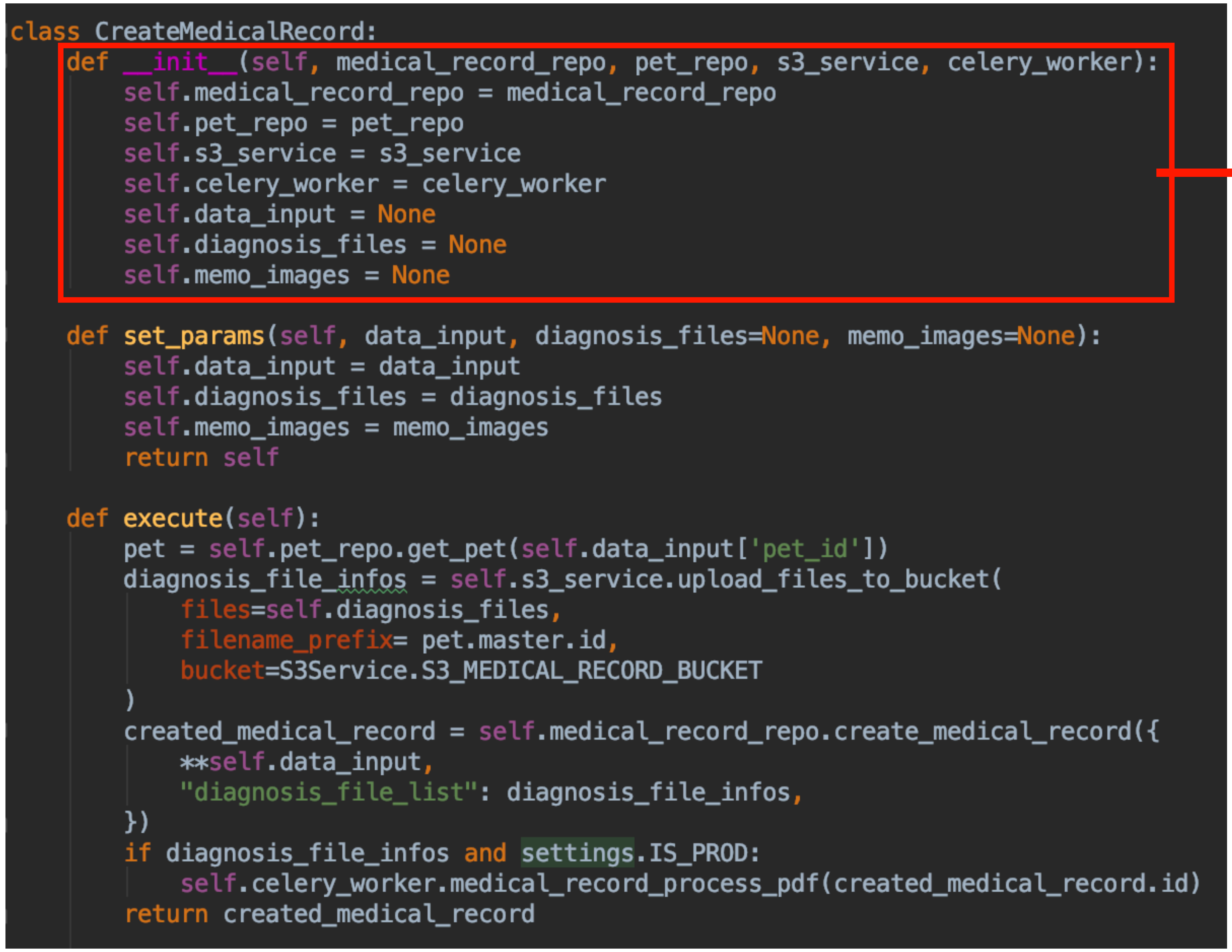
- Create Medical Record Usecase



```python
class CreateMedicalRecord:
    def __init__(self, medical_record_repo, pet_repo, s3_service, celery_worker):
        self.medical_record_repo = medical_record_repo
        self.pet_repo = pet_repo
        self.s3_service = s3_service
        self.celery_worker = celery_worker
        self.data_input = None
        self.diagnosis_files = None
        self.memo_images = None

    def set_params(self, data_input, diagnosis_files=None, memo_images=None):
        self.data_input = data_input
        self.diagnosis_files = diagnosis_files
        self.memo_images = memo_images
        return self

    def execute(self):
        pet = self.pet_repo.get_pet(self.data_input['pet_id'])
        diagnosis_file_infos = self.s3_service.upload_files_to_bucket(
            files=self.diagnosis_files,
            filename_prefix= pet.master.id,
            bucket=S3Service.S3_MEDICAL_RECORD_BUCKET
        )
        created_medical_record = self.medical_record_repo.create_medical_record({
            **self.data_input,
            "diagnosis_file_list": diagnosis_file_infos,
        })
        if diagnosis_file_infos and settings.IS_PROD:
            self.celery_worker.medical_record_process_pdf(created_medical_record.id)
        return created_medical_record
```

2. DRF 에서 어떻게 적용할까

- Create Medical Record Usecase

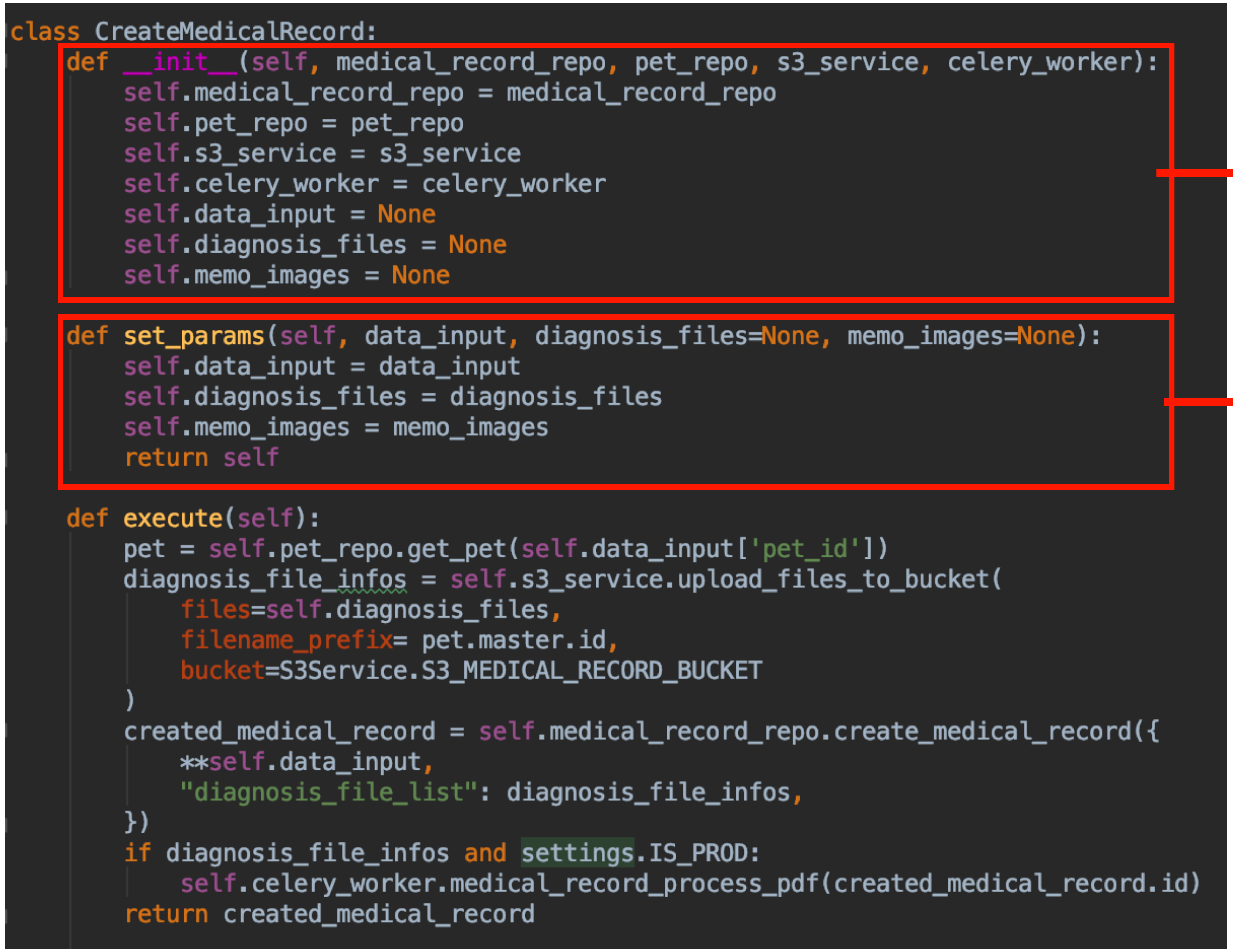


```python
class CreateMedicalRecord:
    def __init__(self, medical_record_repo, pet_repo, s3_service, celery_worker):
        self.medical_record_repo = medical_record_repo
        self.pet_repo = pet_repo
        self.s3_service = s3_service
        self.celery_worker = celery_worker
        self.data_input = None
        self.diagnosis_files = None
        self.memo_images = None

    def set_params(self, data_input, diagnosis_files=None, memo_images=None):
        self.data_input = data_input
        self.diagnosis_files = diagnosis_files
        self.memo_images = memo_images
        return self

    def execute(self):
        pet = self.pet_repo.get_pet(self.data_input['pet_id'])
        diagnosis_file_infos = self.s3_service.upload_files_to_bucket(
            files=self.diagnosis_files,
            filename_prefix= pet.master.id,
            bucket=S3Service.S3_MEDICAL_RECORD_BUCKET
        )
        created_medical_record = self.medical_record_repo.create_medical_record({
            **self.data_input,
            "diagnosis_file_list": diagnosis_file_infos,
        })
        if diagnosis_file_infos and settings.IS_PROD:
            self.celery_worker.medical_record_process_pdf(created_medical_record.id)
        return created_medical_record
```

Factory에서 Usecase 객체 생성시 필요한 Repo, Service 등을 주입

2. DRF 에서 어떻게 적용할까

- Create Medical Record Usecase



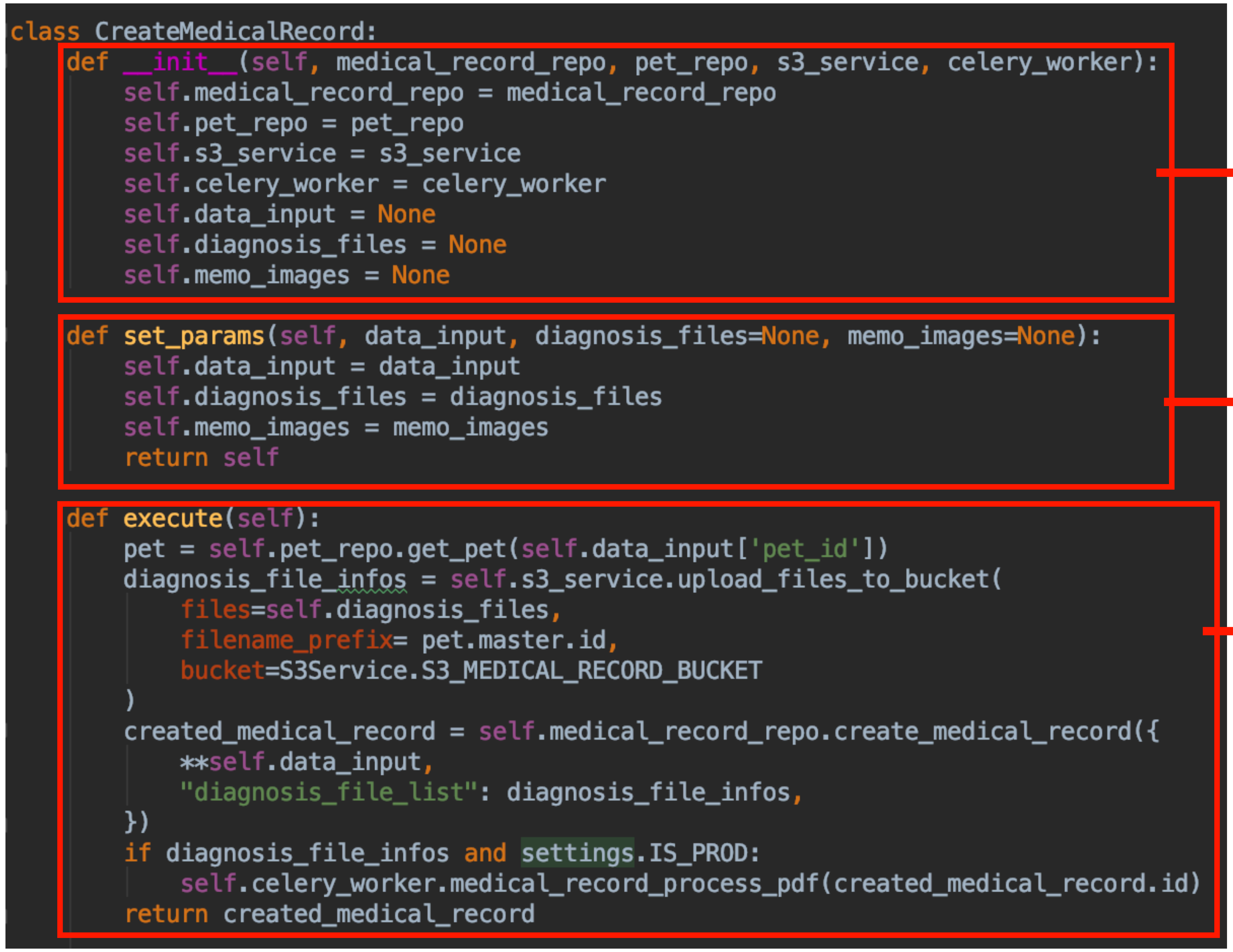
```python
class CreateMedicalRecord:
    def __init__(self, medical_record_repo, pet_repo, s3_service, celery_worker):
        self.medical_record_repo = medical_record_repo
        self.pet_repo = pet_repo
        self.s3_service = s3_service
        self.celery_worker = celery_worker
        self.data_input = None
        self.diagnosis_files = None
        self.memo_images = None

    def set_params(self, data_input, diagnosis_files=None, memo_images=None):
        self.data_input = data_input
        self.diagnosis_files = diagnosis_files
        self.memo_images = memo_images
        return self

    def execute(self):
        pet = self.pet_repo.get_pet(self.data_input['pet_id'])
        diagnosis_file_infos = self.s3_service.upload_files_to_bucket(
            files=self.diagnosis_files,
            filename_prefix= pet.master.id,
            bucket=S3Service.S3_MEDICAL_RECORD_BUCKET
        )
        created_medical_record = self.medical_record_repo.create_medical_record({
            **self.data_input,
            "diagnosis_file_list": diagnosis_file_infos,
        })
        if diagnosis_file_infos and settings.IS_PROD:
            self.celery_worker.medical_record_process_pdf(created_medical_record.id)
        return created_medical_record
```

Factory에서 Usecase 객체 생성시 필요한 Repo, Service 등을 주입

View에서 필요한 params 설정

2. DRF 에서 어떻게 적용할까

- Create Medical Record Usecase



```python
class CreateMedicalRecord:
    def __init__(self, medical_record_repo, pet_repo, s3_service, celery_worker):
        self.medical_record_repo = medical_record_repo
        self.pet_repo = pet_repo
        self.s3_service = s3_service
        self.celery_worker = celery_worker
        self.data_input = None
        self.diagnosis_files = None
        self.memo_images = None

    def set_params(self, data_input, diagnosis_files=None, memo_images=None):
        self.data_input = data_input
        self.diagnosis_files = diagnosis_files
        self.memo_images = memo_images
        return self

    def execute(self):
        pet = self.pet_repo.get_pet(self.data_input['pet_id'])
        diagnosis_file_infos = self.s3_service.upload_files_to_bucket(
            files=self.diagnosis_files,
            filename_prefix= pet.master.id,
            bucket=S3Service.S3_MEDICAL_RECORD_BUCKET
        )
        created_medical_record = self.medical_record_repo.create_medical_record({
            **self.data_input,
            "diagnosis_file_list": diagnosis_file_infos,
        })
        if diagnosis_file_infos and settings.IS_PROD:
            self.celery_worker.medical_record_process_pdf(created_medical_record.id)
        return created_medical_record
```

Factory에서 Usecase 객체 생성시 필요한 Repo, Service 등을 주입

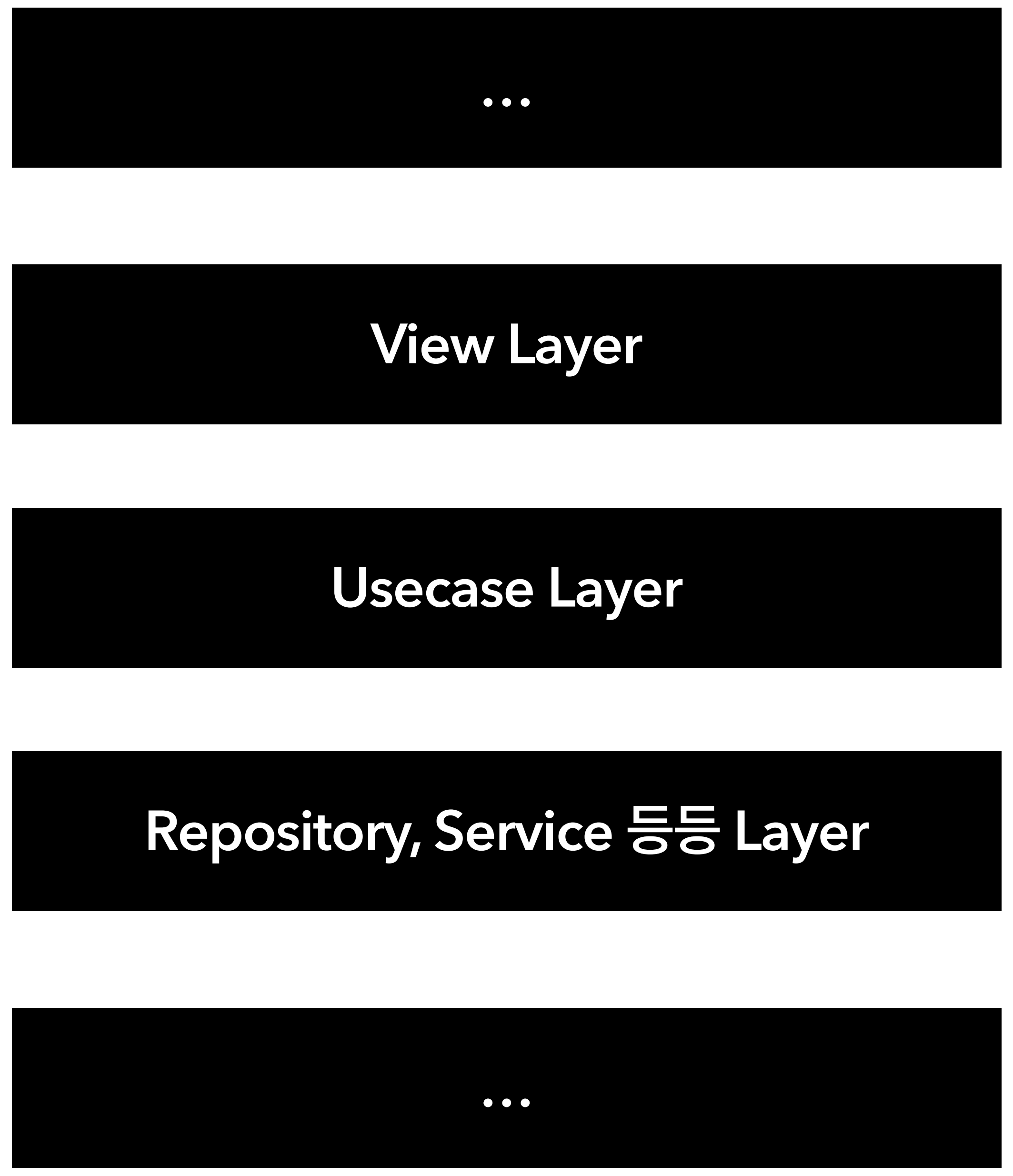
View에서 필요한 params 설정

View에서 실행

3. TEST 용이성

3. TEST 용이성

Multi-Layer



1. 각각의 Layer 별로 명확히 테스트 범위가 나뉘어짐
2. 급한대로 Usecase쪽 테스트만 짜도 나름 많은 부분이 커버됨

4. 실제 적용하고 관리해보면서 느낀 장단점

참고

https://www.artima.com/articles/dci_vision.html

끝