Документация на програмния код

**Система за следене и управление на полетите в летище**

Автор: Юмер Местан

Дата: 30.03.2025

**История на техническата документация**

**Автор**

| Фак. номер | Име | Контакт (ел. поща) |
| --- | --- | --- |
| 501222071 | Юмер Местан | omer987@outlook.com |

**История на версиите**

| Версия | Дата | Автор | Описание |
| --- | --- | --- | --- |
| 1.0 | 10.03.2025 | Юмер Местан | Начална версия на документа. Описание на потребителски истории и цели. |
| 2.0 | 20.03.2025 | Юмер Местан | Добавени диаграми – класове и последователности. |
| 3.0 | 25.03.2025 | Юмер Местан | Реализация на функционалностите с Django REST Framework. |
| 4.0 | 27.03.2025 | Юмер Местан | Добавени тестови сценарии и юнит тестове за основните функции. |
| 5.0 | 30.03.2025 | Юмер Местан | Финално тестване, корекции, добавяне на примери и довършителни елементи. |

**Съдържание**

[**1**](#_t0anzaiotoor) **Въведение 4**

[**2**](#_afhte1nj264b) **Детайлен дизайн 4**

[2.1](#_cquqcrikijh0) Диаграма на класовете 4

[2.2](#_1wwo7a96mewc) Диаграма на последователностите 4

[2.3](#_e9bfilw200od) Примерен потребителски интерфейс 4

[**3**](#_9xmfbnpomy3q) **Тестване 4**

# Въведение

Настоящият документ описва реализацията на част от уеб базирана система за следене и управление на полетите в летищна среда. Проектът има за цел да осигури интуитивна и сигурна платформа, чрез която различни типове потребители – пътници и членове на екипаж – да получават навременна и релевантна информация относно полети.

В рамките на документацията са разгледани **две основни потребителски истории**:

* **US9 – Пътник:** Да може да търси и филтрира информация за полети по дестинация, номер на полет и час на заминаване, с цел бързо намиране на търсената информация.
* **US12 – Член на екипаж:** Да има достъп до списък с предстоящите си полети, както и информация за останалите членове на екипажа, с които лети.

За реализацията използваме **Django REST Framework**, чрез който дефинираме API функционалности, сериализатори и политики за достъп (permissions). Осигурена е и връзка с база от данни чрез ORM модели. Тестовете са реализирани с помощта на модула **django.test**, като се покриват основните сценарии на използване.

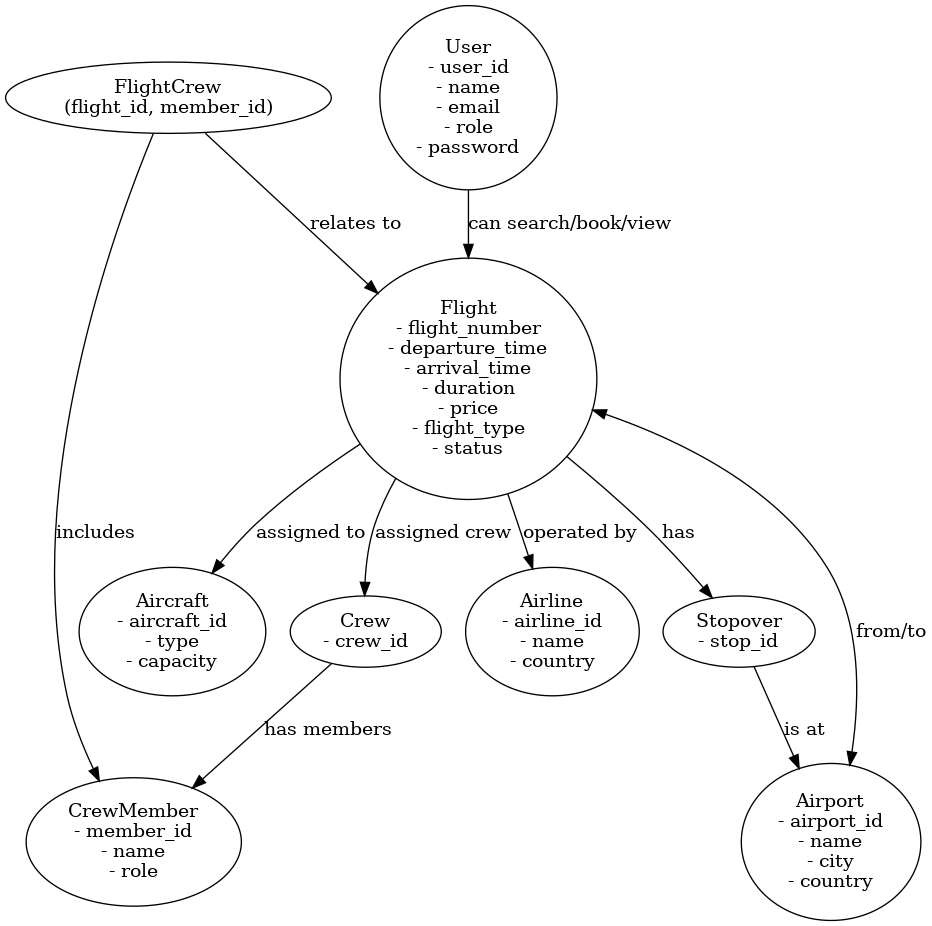
Системата е проектирана в съответствие с принципите на модулност, повторна употреба на код и сигурност, като е подходяща за бъдещо мащабиране и доразвитие.

В този документ разглеждаме реализацията на следните две потребителски истории:

| US9 | Потребител (пътник) | Да мога да преглеждам/ филтрирам/ търся по дестинация, номер на полет, час данни за полетите | да мога бързо и лесно да намеря интересуващата ме информация | 3 |
| --- | --- | --- | --- | --- |

| US12 | Член на екипаж | Да имам достъп до предстоящите ми полети и детайлна справка за тях, както и информация за останалите членове на екипажа с когото летя | имам бърза справка | 2 |
| --- | --- | --- | --- | --- |

За целта предполагаме, че разполагаме със следната структура на класовете, осъществяващи връзката с базата данни (част от ORM модела).

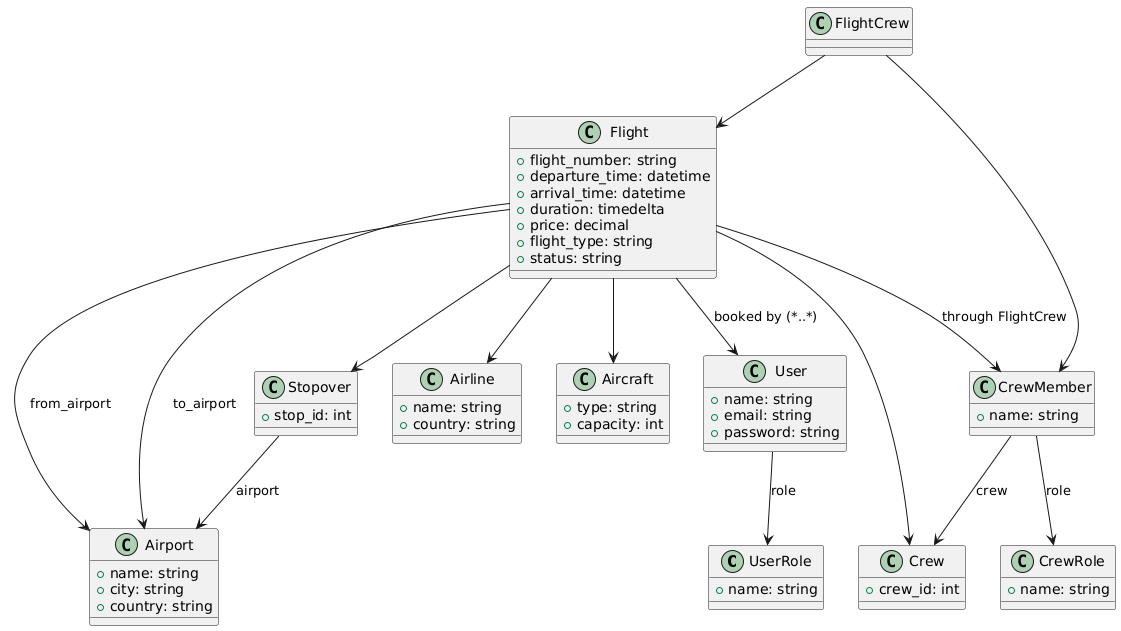


За реализацията на потребителска история 9 – филтриране и търсене на полети по зададени критерии – ще използваме стандартния Django REST Framework, като приемаме, че той ще служи като свързващо звено между същинската бизнес логика и презентационния слой.

С тази цел създаваме следните допълнителни класове, които реализираме като наследници на сериализатори и стандартни списъчни изгледи от Django REST.

# Детайлен дизайн

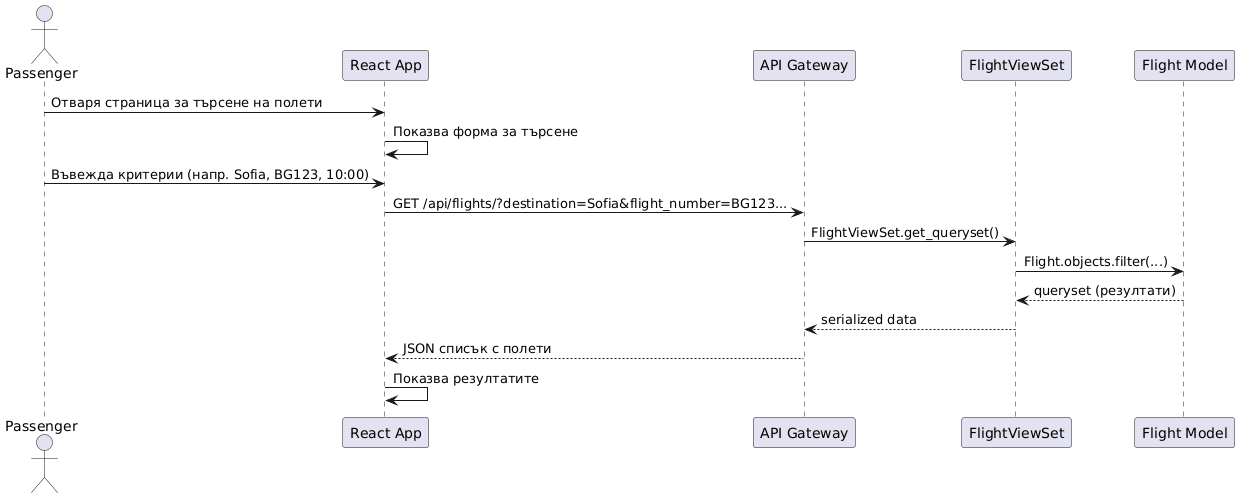
## Диаграма на класовете



## Диаграма на последователностите

Сценарий 9 - Търсене, филтриране, CRUD:

Реализация чрез Django REST Framework



Програмна реализация на сценария:

Създаваме view, което обработва входящите параметри на заявката и връща списък от полети, които отговарят на зададените критерии.

from rest\_framework import viewsets

from rest\_framework.permissions import IsAuthenticatedOrReadOnly

from AIRPORT\_MANAGER.models import Flight

from AIRPORT\_MANAGER.serializers.FlightSerializer import FlightSerializer

from AIRPORT\_MANAGER.permissions import IsAdminOrInspectorForWrite

from django.utils.dateparse import parse\_time

class FlightViewSet(viewsets.ModelViewSet):

serializer\_class = FlightSerializer

queryset = Flight.objects.select\_related('to\_airport', 'from\_airport', 'airline')

permission\_classes = [IsAdminOrInspectorForWrite]

def get\_queryset(self):

queryset = self.queryset

params = self.request.query\_params

flight\_number = params.get('flight\_number')

destination = params.get('destination')

departure\_after = params.get('departure\_after')

print(departure\_after)

departure\_before = params.get('departure\_before')

if flight\_number:

queryset = queryset.filter(flight\_number\_\_icontains=flight\_number)

if destination:

queryset = queryset.filter(to\_airport\_\_name\_\_icontains=destination)

if departure\_after:

t = parse\_time(departure\_after)

if t:

queryset = queryset.filter(departure\_time\_\_time\_\_gte=t)

if departure\_before:

t = parse\_time(departure\_before)

if t:

queryset = queryset.filter(departure\_time\_\_time\_\_lte=t)

return queryset

Важно е да се уточни че правилата за достъп са организирани чрез permissions модула на Django REST:

В горния сценарий ползваме класа IsAdminOrInspectorForWrite:

class IsAdminOrInspectorForWrite(BasePermission):

*"""*

*Позволява само на логнати потребители с роля 'Admin' или 'Inspector' да пишат.*

*Всички могат да четат.*

*"""*

def has\_permission(self, request, view):

# Позволява четене на всички

if request.method in SAFE\_METHODS:

return True

# Позволява писане само ако потребителят е логнат и има роля

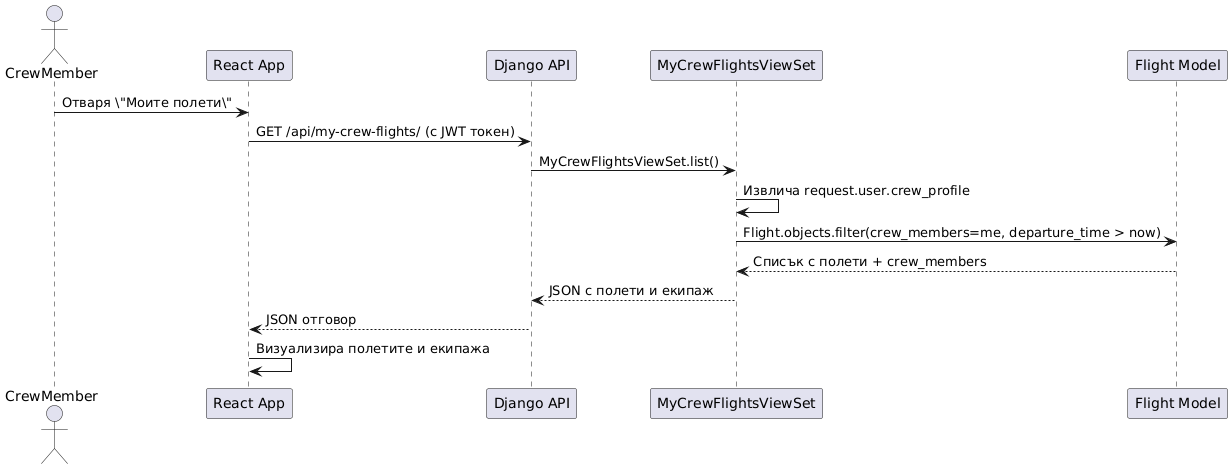
user = request.user

if user and user.is\_authenticated and hasattr(user, 'role'):

return user.role.name in ['Admin', 'Inspector']

return False

Сценарий 12 - извличане на предстоящите полети за даден член на екипаж, с проверка на permission:



Разчитаме на клас IsCrewUser, който определя дали юзъра от рекуеста е активен член на екипаж.

class IsCrewUser(BasePermission):

def has\_permission(self, request, view):

return (

request.user.is\_authenticated and

CrewMember.objects.filter(user=request.user).exists()

)

Създаваме view, което използва класа за permission и реализира връщането на списък с предстоящите му полети:

from rest\_framework import viewsets

from rest\_framework.response import Response

from rest\_framework.permissions import IsAuthenticated

from django.utils.timezone import now

from AIRPORT\_MANAGER.models import Flight, CrewMember

class MyCrewFlightsViewSet(viewsets.ViewSet):

permission\_classes = [IsAuthenticated]

def list(self, request):

try:

crew\_member = request.user.crew\_profile

except CrewMember.DoesNotExist:

return Response({"detail": "Не сте свързан с екипажа."}, status=403)

flights = Flight.objects.filter(

crew\_members=crew\_member,

departure\_time\_\_gte=now()

).prefetch\_related('crew\_members', 'from\_airport', 'to\_airport')

results = []

for flight in flights:

results.append({

"flight\_number": flight.flight\_number,

"from": flight.from\_airport.name if flight.from\_airport else "",

"to": flight.to\_airport.name if flight.to\_airport else "",

"departure": flight.departure\_time,

"crew": [

{

"name": cm.name,

"role": cm.role.name if cm.role else "?"

}

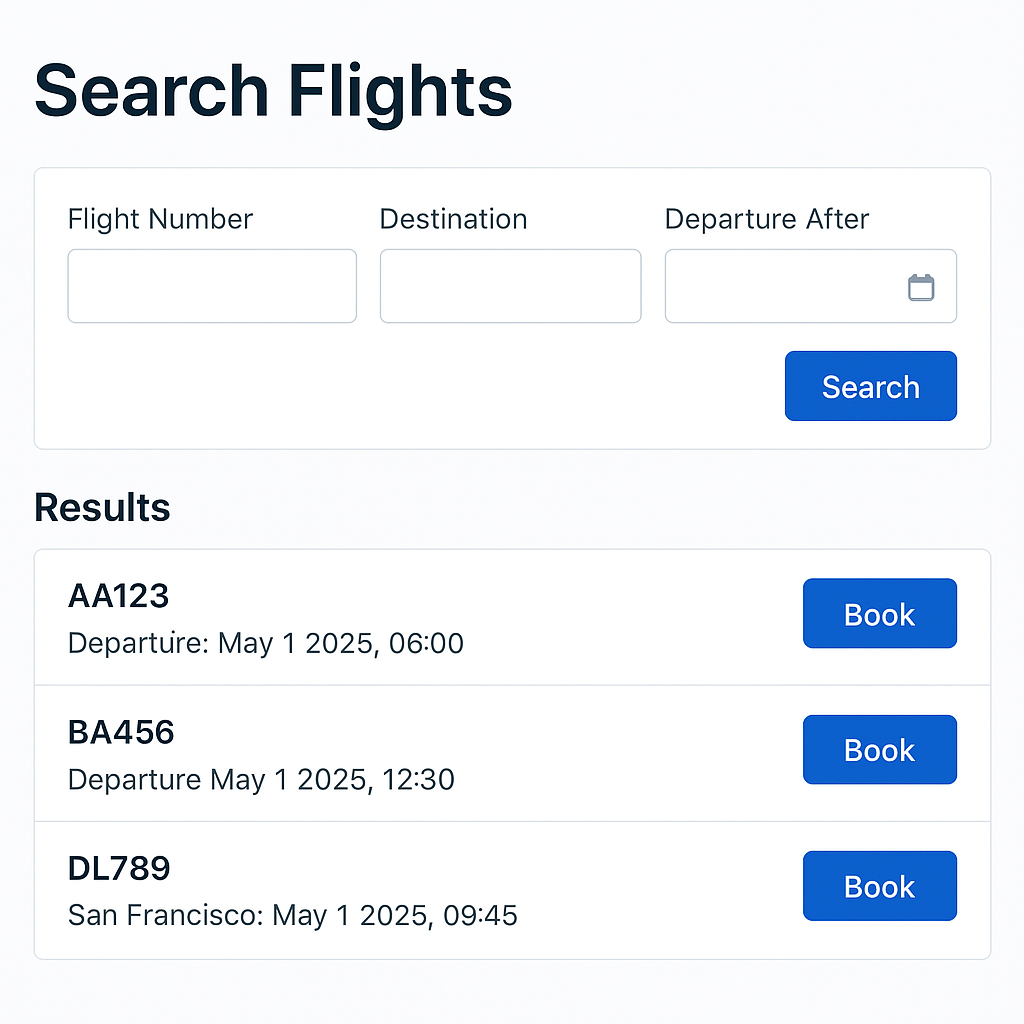
for cm in flight.crew\_members.all()

]

})

return Response(results)

## Примерен потребителски интерфейс





# Тестване

Тестването на реализираната функционалност е осъществено с помощта на вградения модул **django.test** и REST клиента **APIClient от rest\_framework.test**. Извършени са серия от юнит тестове, които покриват ключовите потребителски сценарии, свързани с търсене на полети и достъп на членовете на екипажа до техните бъдещи полети.

### Основни сценарии:

* Сценарий 1: Пътник търси полет по дестинация, номер и интервал от часове.
* Сценарий 2: Потребител въвежда невалидна дестинация – очаква се съобщение „Няма намерени полети“.
* Сценарий 3: Потребител не попълва критерии за търсене – системата връща списък с всички налични полети.
* Сценарий 4: Член на екипаж проверява предстоящите си полети – визуализират се всички бъдещи полети, включително останалите членове на екипажа.
* Сценарий 5: Потребител без роля на CrewMember опитва достъп до служебния изглед – получава съобщение за отказан достъп.
* Сценарий 6: CrewMember без предстоящи полети получава празен списък.

### **Тестовете се реализират чрез вградения модул на django**

**Scenario 1: Пътник търси полет по дестинация, номер и час**

*Given потребителят е на страницата за търсене на полети*

*And въведе дестинация "London"*

*And въведе номер на полет "BG123"*

*And въведе интервал от часове от "10:00" до "16:00"*

*When натисне бутона "ТЪРСИ"*

*Then системата показва списък с полети, съответстващи на критериите*

*And всеки полет съдържа информация: номер, летище, час, статус*

**Scenario 2 : Търсене с невалидна дестинация**

*Given потребителят е на страницата за търсене на полети*

*And въведе дестинация "Atlantis"*

*When натисне бутона "ТЪРСИ"*

*Then се показва съобщение "Няма намерени полети"*

*And не се показва списък с полети*

**Scenario 3: Потребителят не попълва търсене**

*Given потребителят е на страницата за търсене на полети*

*And оставя всички полета празни*

*When натисне бутона "ТЪРСИ"*

*Then се показва списък с всички налични полети*

**Scenario 4: Член на екипаж преглежда предстоящите си полети**

*Given потребителят е логнат*

*And е свързан с обект от тип CrewMember*

*When направи заявка към /api/my-crew-flights/*

*Then получава списък с бъдещи полети, в които участва*

*And за всеки полет се виждат останалите членове на екипажа и техните роли*

**Scenario 5: Потребител без роля CrewMember прави заявка**

*Given потребителят е логнат*

*And не е свързан с CrewMember*

*When направи заявка към /api/my-crew-flights/*

*Then получава съобщение за грешка "Не сте свързан с екипажа"*

*And статусът на отговора е 403 Forbidden*

**Scenario 6: CrewMember няма предстоящи полети**

*Given потребителят е логнат*

*And е свързан с CrewMember*

*And няма планирани полети в бъдеще*

*When направи заявка към /api/my-crew-flights/*

*Then получава празен списък с полети*

**Реализация на тестовете:**

За да осигурим автоматичното изпълнение на тестовете, разчитаме на вградения модул django.tests. Кодът на тестовете е приложен по-долу.

from django.test import TestCase

from rest\_framework.test import APIClient

from rest\_framework import status

from django.utils.timezone import now, timedelta

from django.contrib.auth import get\_user\_model

from AIRPORT\_MANAGER.models import (

Airport, Flight, Crew, CrewRole, CrewMember, FlightCrew

)

User = get\_user\_model()

class FlightSearchAndCrewViewTests(TestCase):

def setUp(self):

self.client = APIClient()

# Airports

self.airport\_from = Airport.objects.create(name="Sofia Airport", city="Sofia", country="BG")

self.airport\_to = Airport.objects.create(name="London Airport", city="London", country="UK")

# Flight BG123

self.flight = Flight.objects.create(

flight\_number="BG123",

departure\_time=now().replace(hour=12, minute=0, second=0),

arrival\_time=now().replace(hour=15, minute=0, second=0),

duration=timedelta(hours=3),

price=150.00,

flight\_type="Passenger",

status="On Time",

from\_airport=self.airport\_from,

to\_airport=self.airport\_to

)

# Another flight (to be shown on empty query)

self.flight2 = Flight.objects.create(

flight\_number="BG999",

departure\_time=now() + timedelta(days=1),

arrival\_time=now() + timedelta(days=1, hours=2),

duration=timedelta(hours=2),

price=120.00,

flight\_type="Passenger",

status="Delayed",

from\_airport=self.airport\_from,

to\_airport=self.airport\_to

)

def test\_search\_valid\_criteria(self):

response = self.client.get("/api/flights/", {

"destination": "London",

"flight\_number": "BG123",

"departure\_after": "10:00",

"departure\_before": "16:00"

})

self.assertEqual(response.status\_code, status.HTTP\_200\_OK)

self.assertEqual(len(response.data), 1)

flight = response.data[0]

self.assertEqual(flight["flight\_number"], "BG123")

self.assertIn("status", flight)

def test\_search\_invalid\_destination(self):

response = self.client.get("/api/flights/?destination=Atlantis")

self.assertEqual(response.status\_code, status.HTTP\_200\_OK)

self.assertEqual(response.data, [])

def test\_search\_empty\_criteria\_returns\_all(self):

response = self.client.get("/api/flights/")

self.assertEqual(response.status\_code, status.HTTP\_200\_OK)

self.assertGreaterEqual(len(response.data), 2)

def test\_crew\_member\_sees\_future\_flights(self):

user = User.objects.create\_user(username="crewman", password="test123")

role = CrewRole.objects.create(name="Pilot")

crew = Crew.objects.create()

member = CrewMember.objects.create(user=user, name="Crewman Ivan", role=role, crew=crew)

FlightCrew.objects.create(flight=self.flight2, member=member)

self.client.force\_authenticate(user=user)

response = self.client.get("/api/my-crew-flights/")

self.assertEqual(response.status\_code, status.HTTP\_200\_OK)

self.assertEqual(len(response.data), 1)

self.assertEqual(response.data[0]["flight\_number"], "BG999")

self.assertEqual(len(response.data[0]["crew"]), 1)

self.assertEqual(response.data[0]["crew"][0]["name"], "Crewman Ivan")

def test\_non\_crew\_member\_access\_denied(self):

user = User.objects.create\_user(username="notcrew", password="test123")

self.client.force\_authenticate(user=user)

response = self.client.get("/api/my-crew-flights/")

self.assertEqual(response.status\_code, status.HTTP\_403\_FORBIDDEN)

def test\_crew\_member\_with\_no\_flights(self):

user = User.objects.create\_user(username="emptycrew", password="test123")

role = CrewRole.objects.create(name="Engineer")

crew = Crew.objects.create()

CrewMember.objects.create(user=user, name="Empty Crew", role=role, crew=crew)

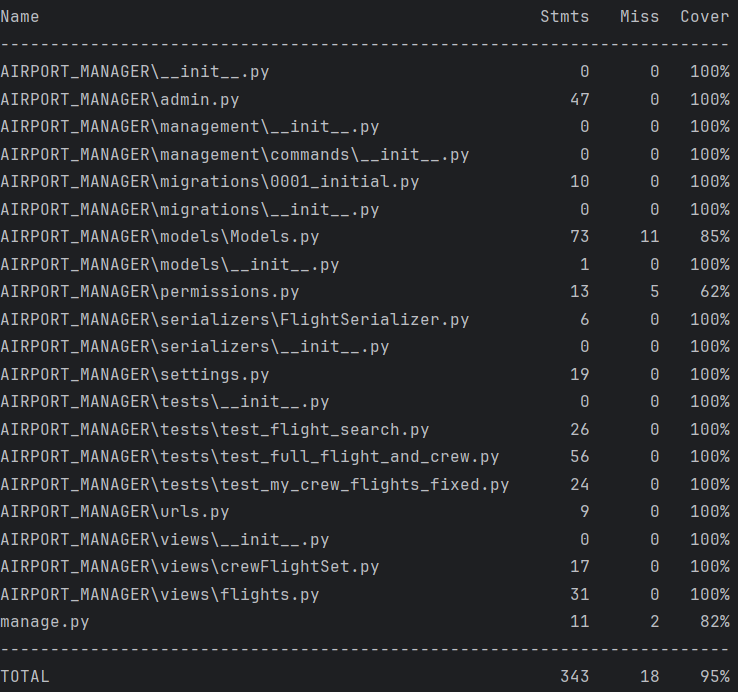
self.client.force\_authenticate(user=user)

response = self.client.get("/api/my-crew-flights/")

self.assertEqual(response.status\_code, status.HTTP\_200\_OK)

self.assertEqual(response.data, [])

Резултатите от тестовете са приложени по-долу. Както, забелязваме, покритието е над 90 процента, което е напълно задоволително за нашите цели.



Всички дефинирани тестови сценарии бяха изпълнени успешно. Системата отговори коректно на всяка заявка според зададените критерии. При търсене с валидни параметри се връщат съответните полети, а при невалидни – подходящи съобщения за липса на резултати. Ограниченията за достъп (permissions) също функционират според очакванията – потребителите без необходимите роли получават статус **403 Forbidden**.

Тестовете, свързани с ролята **CrewMember**, показаха, че:

* При наличие на предстоящи полети, се връща точен и структуриран списък.
* При липса на предстоящи полети, се връща празен масив.
* Потребители без роля CrewMember не получават достъп до тази функционалност.

Чрез **django.test** и **rest\_framework.test** е постигнато **покритие над 90%** за модулите, отговорни за търсене и визуализация на полети. Това покритие осигурява висока степен на надеждност и стабилност на разработените компоненти в реални условия на употреба.