CODE

```
class Investor:
    def __init__(self, name, available_funds):
        self.name = name
        self.available_funds = available_funds
class FilmProject:
    def __init__(self, project_id, project_name, required_funds):
        self.project_id = project_id
        self.project_name = project_name
        self.required_funds = required_funds
        self.current_funds = 0
    def add_funds(self, amount):
        self.current_funds += amount
        return self.current_funds >= self.required_funds  # Returns True if
fully funded
class FundingRequest:
    def __init__(self, request_id, project, amount):
        self.request_id = request_id
        self.project = project
        self.amount = amount
        self.status = 'open'  # Possible statuses: open, funded, cancelled
class FundingPlatform:
    def __init__(self):
        self.projects = {}
        self.investors = {}
        self.requests = {}
        self.transactions = []
    def add_project(self, project_id, project_name, required_funds):
        self.projects[project_id] = FilmProject(project_id, project_name,
required_funds)
    def remove_project(self, project_id):
        if project_id in self.projects:
            del self.projects[project_id]
    def add_investor(self, investor_name, funds):
        new_investor = Investor(investor_name, funds)
        self.investors[investor_name] = new_investor
    def remove_investor(self, investor_name):
        if investor_name in self.investors:
            del self.investors[investor_name]
    def create_request(self, project_id, amount):
        project = self.projects.get(project_id)
        if project:
            request = FundingRequest(len(self.requests) + 1, project, amount)
            self.requests[request.request_id] = request
            return request.request_id
    def cancel_request(self, request_id):
        if request_id in self.requests:
```

```
            self.requests[request_id].status = 'cancelled'
    def connect_filmmakers_with_investors(self, project_id):
        project = self.projects.get(project_id)
        if not project:
            return []
        # Match investors based on available funds and project fund needs
        potential_investors = [
            inv for inv in self.investors.values() if inv.available_funds >=
project.required_funds - project.current_funds]
        return potential_investors
    def manage_funding_transactions(self, transaction_data):
        project_id, investor_name, amount = transaction_data
        project = self.projects.get(project_id)
        investor = self.investors.get(investor_name)
        if project and investor and investor.available_funds >= amount:
            project.add_funds(amount)
            investor.available_funds -= amount
            self.transactions.append(transaction_data)
            return True
        return False
### Unit Tests Using Python's unittest
import unittest
class TestFundingPlatform(unittest.TestCase):
    def setUp(self):
        self.platform = FundingPlatform()
        self.platform.add_project(1, "Epic Space Opera", 100000)
        self.platform.add_investor("Alice", 50000)
        self.platform.add_investor("Bob", 75000)
    def test_project_addition(self):
        self.assertIn(1, self.platform.projects)
    def test_investor_addition(self):
        self.assertIn("Alice", self.platform.investors)
        self.assertIn("Bob", self.platform.investors)
    def test_funding_transaction(self):
        self.platform.manage_funding_transactions((1, "Alice", 50000))
        self.assertEqual(self.platform.projects[1].current_funds, 50000)
    def test_connect_filmmakers_with_investors(self):
        potential_investors =
self.platform.connect_filmmakers_with_investors(1)
        self.assertEqual(len(potential_investors),0)  # Both investors have
sufficient funds
if __name__ == '__main__':
    unittest.main()
```

OUTPUT:-

....

```
----------------------------------------------------------------------
Ran 4 tests in 0.001s

OK
```