KISHKINDA UNIVERSITY

MINI PROJECT REPORT

PROJECT TITLE: FILM PROJECT FUNDING PLATFORM(POC)

PROJECT TEAM MEMBERS

THABITHA K	**BTech-CSE083**
DEEPIKA S KHOKALE	**BTech-CSE029**
GOWTHAMI H	**BTech-CSE036**
K ASMA SIDDIQUA	**BTech-CSE046**
SHAILAJA P	**BTech-CSE072**

TABLE OF CONTENTS

Introduction to the Film Project Funding Platform

This Python code represents a **Film Project Funding Platform**, designed to simulate how filmmakers and investors can connect and collaborate to fund creative film projects. It includes the core functionality needed to manage film projects, handle investor funding, process transactions, and match filmmakers with potential investors.

The code is structured around four main components: **Investor**, **FilmProject**, **FundingRequest**, and **FundingPlatform**. Together, these classes form the backbone of the platform, enabling various operations like adding projects and investors, creating funding requests, and managing funding transactions.

System Overview:

1. **Investor Class:**
    - Represents an investor with attributes:
        - name: Investor's name.
        - available_funds: The amount of money the investor has available to invest in projects.
    - **Usage**: Investors are added to the platform, and their funds are used to back film projects.
2. **FilmProject Class:**
    - Represents a film project with attributes:
        - project_id: A unique identifier for the film project.
        - project_name: The name of the film project.
        - required_funds: The total amount of funds required to complete the project.
        - current_funds: Tracks how much funding the project has received so far.
    - **Methods**:
        - add_funds(amount): Adds the given amount to the current funds and checks if the project is fully funded.
    - **Usage**: Film projects are listed on the platform, and investors can contribute to their funding.
3. **FundingRequest Class:**
    - Represents a funding request with attributes:
        - request_id: A unique identifier for the request.

        - project: The film project for which funds are being requested.
        - amount: The amount requested.
        - status: The status of the request (open, funded, or canceled).
    - **Usage**: When filmmakers need funding, they create requests that investors can fulfill.

4. **FundingPlatform Class:**
    - The core of the platform, responsible for managing projects, investors, requests, and transactions.
    - **Methods**:
        - add_project(project_id, project_name, required_funds): Adds a film project to the platform.
        - remove_project(project_id): Removes a film project from the platform.
        - add_investor(investor_name, funds): Adds an investor to the platform.
        - remove_investor(investor_name): Removes an investor from the platform.
        - create_request(project_id, amount): Creates a funding request for a given project.
        - cancel_request(request_id): Cancels a specific funding request.
        - connect_filmmakers_with_investors(project_id): Matches projects with investors who have sufficient funds to back them.
        - manage_funding_transactions(transaction_data): Processes transactions, transferring funds from investors to projects.
    - **Usage**: This class manages all the interactions between filmmakers, investors, and funding activities on the platform.

Problem Statement: Film Project Funding Platform

Context:

In the film industry, securing funding is a major challenge, particularly for independent filmmakers and smaller production houses. Traditional funding routes such as studio financing, government grants, or private investors are often

difficult to access and involve lengthy approval processes. This leads to many promising film projects being shelved due to a lack of financial backing.

Problem:

There is a need for a streamlined, efficient, and transparent way to connect filmmakers with investors who are willing to fund their projects. Current systems are fragmented, inefficient, and often limit the opportunity for smaller or independent filmmakers to reach potential investors.

Filmmakers need a platform to:

- Present their film projects to a wide audience.
- Seek funding based on the specific needs of their projects.
- Engage with potential investors in a transparent and efficient manner.

Investors need a platform where they can:

- Easily browse film projects seeking funding.
- Understand how their funds will be used.
- Have clear visibility into the progress of projects they support.

Proposed Solution:

Develop a **Film Project Funding Platform** that facilitates connections between filmmakers and investors. The platform will allow filmmakers to list their projects with funding goals and timelines. Investors can view available projects and contribute funds based on their interest and financial capacity. The platform will manage funding requests, process transactions, and provide transparency in the flow of funds. Additionally, filmmakers will be able to attract multiple investors, making the process of gathering funds quicker and more scalable.

Objectives of the Platform:

1. **For Filmmakers**:
    - Provide a space to showcase film projects and seek funding.
    - Simplify the process of raising funds by connecting with multiple investors.
    - Ensure transparency in how funds are received and managed.

2. **For Investors**:
    - Provide easy access to a variety of film projects looking for financial backing.
    - Enable a streamlined and transparent method for funding projects.
    - Receive updates on project progress after funds have been invested.
3. **For Both Parties**:
    - Establish trust by ensuring financial transparency and accountability.
    - Create a simplified user experience for tracking funds and project milestones.
    - Reduce time and complexity in securing and managing project funding.

- **e Code**:
    - Preventing the platform from crashing when an operation like accessing a non-existing project or investor is attempted.
    - Example: If a transaction fails due to insufficient funds, the code returns False, preventing the transaction from proceeding.

6. Command-Line Interface (CLI)

- **Description**: The current proof-of-concept implementation operates through a simple command-line interface (CLI) where functions are triggered programmatically without a graphical user interface (GUI).
- **Purpose in the Code**:
    - The CLI allows developers to manually interact with the platform's core functionalities, such as adding projects, processing transactions, and creating funding requests during testing.

7. Modular Code Structure

- **Description**: The code is broken into several modular components using Python classes, making it easy to maintain, extend, and reuse.
- **Purpose in the Code**:
    - Each class, such as Investor, FilmProject, FundingRequest, and FundingPlatform, is responsible for handling specific tasks, allowing the platform to scale easily.
    - This design structure promotes code reusability and separation of concerns.

Project Plan for Film Project Funding Platform

Objective: To build a Film Project Funding Platform that connects filmmakers seeking financial support with potential investors. The platform should handle project listings, manage investors, process funding transactions, and provide transparency.

Phases of Development:

Phase 1: Requirement Analysis and Planning

- **Objective**: Define the core functionality and project scope.
  - Gather requirements for how filmmakers and investors will interact.
  - Determine the specific features, such as project listing, investor matching, and transaction processing.
- **Deliverables**:
  - Project requirements document.
  - Technology stack decision (e.g., Python for the backend, databases, etc.).
  - Detailed project timeline and milestones.

Phase 2: System Design

- **Objective**: Architect the overall system and define the structure of the platform.
  - **Class design**: Define the classes (Investor, FilmProject, FundingRequest, FundingPlatform).
  - **Database design** (if necessary for future phases): Design a schema for projects, investors, and transactions.
- **Deliverables**:
  - UML diagrams for class structure.
  - Data flow diagrams for interaction between classes.
  - Database schema (if moving to persistent storage in future iterations).

Phase 3: Core Backend Development

- **Objective**: Implement core functionality in Python.
  - **Task 1**: Implement the Investor class.

        - Attributes: name, available_funds.
    - **Task 2**: Implement the FilmProject class.
        - Attributes: project_id, project_name, required_funds, current_funds.
        - Methods: add_funds(amount) for funding progress.
    - **Task 3**: Implement the FundingRequest class.
        - Attributes: request_id, project, amount, status.
    - **Task 4**: Implement the FundingPlatform class.
        - Manage the list of projects, investors, and requests.
        - Implement project addition and removal, investor management, funding request creation, and processing transactions.
        - Match investors to projects based on available funds.
- **Deliverables**:
    - Working backend code for the platform.
    - Integration of all classes.
    - Basic command-line interface (CLI) for manual interaction.

Phase 4: Testing

- **Objective**: Ensure all features are working correctly.
    - **Unit Testing**: Write unit tests for all core functionality using unittest.
        - Test project addition, investor management, funding transactions, and investor matching.
    - **Test Coverage**: Ensure key components (add project, add investor, transactions, etc.) are thoroughly tested.
    - **Bug Fixes**: Address any issues found during testing.
- **Deliverables**:
    - Unit test scripts.
    - Test results for all major functionalities.
    - Bug report and fixes.

Phase 5: Feature Enhancements

- **Objective**: Expand the platform with additional functionality based on future requirements.
    - **Web Interface**: Implement a web interface for easy user interaction.
    - **Database Integration**: Add persistent storage for projects, investors, and transactions using a database like SQLite or PostgreSQL.

    - **Payment Gateway**: Integrate a payment gateway for handling real financial transactions.
- **Deliverables**:
    - Fully functional web platform (in later phases).
    - Integration with payment and other third-party services.

Code Implementation:

Python

```
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
class FundingRequest:
    def __init__(self, request_id, project, amount):
        self.request_id = request_id
        self.project = project
        self.amount = amount
        self.status = 'open'

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 []
        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

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):
        self.platform.add_investor("charlie",100000)
        potential_investors = self.platform.connect_filmmakers_with_investors(1)
        self.assertEqual(len(potential_investors),1)

if __name__ == '__main__':
    unittest.main()
```

Summary of Results

All tests pass successfully, confirming that the funding platform behaves as expected:

- Projects are correctly added and can be found.
- Investors are correctly added and tracked.
- Funding transactions correctly update both project and investor states.
- The connection function accurately identifies eligible investors based on their available funds.

Sample Output

....

Ran 4 tests in 0.000s

OK

- **Result:**

The code and accompanying unit tests demonstrate a well-structured funding platform that performs its intended functions effectively. The tests ensure that core functionalities are validated, paving the way for potential extensions and enhancements.

Conclusion:

The Film Project Funding Platform is primarily built using Python and Object-Oriented Programming principles, ensuring that it is modular, scalable, and easy to test. The use of Python's unittest framework ensures that the core functionalities are reliable, while basic data structures like dictionaries and lists make it easy to manage projects, investors, and transactions. The platform serves as a solid foundation that can be expanded with additional technologies like web frameworks, databases, and payment gateways in future iterations.

Potential Future Enhancements (Technology Stack Expansion):

1. **Web Framework**:
    - **Flask** or **Django** could be used to build a full-fledged web application with user interfaces for filmmakers and investors.
2. **Database Integration**:
    - **SQLite**, **PostgreSQL**, or **MongoDB** could be used to store persistent data related to projects, investors, and transactions.
3. **Payment Gateway Integration**:
    - **Stripe** or **PayPal** could be integrated for secure financial transactions between investors and filmmakers.

4. **Authentication and Authorization**:
    - **JWT (JSON Web Token)** or OAuth for secure login and identity management.
5. **Frontend Technologies**:
    - **React.js** or **Vue.js** for building interactive user interfaces and improving user experience.