NAME: PRATEEK

SMART STREAM

OVERVIEW

SmartStream is an in-memory greedy scheduler that selects content to fill homepage slots over multiple days while enforcing business rules and supporting interchangeable greedy strategies (by revenue, by engagement, balanced). All logic runs in-memory; the catalog is hard-coded.

Key features - Greedy scheduling strategies: by revenue, by engagement, balanced (weighted). - Centralized scoring (kids/prime-time/weekend/diversity rules). - Hard filters: release window, max runs, per-day genre cap, per-week unique-day limit.

REPOSITORY LAYOUT

- app->main.py (to run scheduler and generates report)
- model->content.py (core domain models)
- scoring->rules.py (business logic for computing revenue, engagement)
- engine->scheduler.py (scheduler base class and greedy scheduler implement)
- engine->strategies.py (greedy strategies function:by_revenue,by_engagement etc)
- util->helpers.py (shared utility functions like clamp)

Business Logic and Implementation

Content Domain and Validations (model/content.py)

- Defines the Content dataclass representing each piece of content, including attributes like title, monetization, base revenue/engagement, release windows, and max runs.
- Uses Monetization enum to define monetization types (SUBSCRIPTION, AD, PPV).
- Custom exceptions ContentNotInWindow and MaxRunsExceeded to signal violations of business constraints during scheduling.
- Includes helper methods to check if content is in release window, is kids content, or is PPV.

Rule Context and Scoring Adjustments (scoring/rules.py)

- RuleContext encapsulates per-day state including weekend status, prime-time and kids' hours, genre counts, and counters for remaining runs.
- Methods include checks for weekend, prime-time, kids' daytime, and if genre caps are exceeded.
- Scoring class applies all business rules to base revenue and engagement:
    - PPV prime-time revenue bonus (×1.25)

    - Weekend PPV revenue multiplier (×1.15)
    - Kids daytime engagement bonus (+10, clamped)
    - Weekend engagement boost (+8, clamped)
    - Diversity soft penalty for exceeding genre cap (-8 engagement, clamped)
- Rule order and clamping is consistently enforced for accurate adjustments.

Scheduler Engine and Strategies (engine/scheduler.py + engine/strategies.py)

- Abstract base Scheduler enforces the interface and shared properties.
- GreedyScheduler implements the daily scheduling logic as a greedy algorithm:
    - Filters candidates by release window and max runs.
    - Iteratively assigns slots picking highest-ranked item per the current strategy.
    - Enforces a hard cap on the number of items per genre per day (no more than 4).
    - Tracks and decrements global run limits per content.
- Strategy key functions define different sort orders based on adjusted revenue, engagement, or a weighted balanced score (0.6 revenue, 0.4 engagement).
- Switching strategies requires changing only one line specifying the key function, demonstrating polymorphism and composition.

Utilities (util/helpers.py)

- Contains helper utilities such as a clamp function to keep numeric values within specified minimum and maximum bounds. This is essential for safely applying engagement boosts and penalties.

Application Entry Point and Reporting (app/main.py)

- Initializes the hardcoded content catalog and weekend day mapping.
- Configures scheduling strategy by selecting one key function.
- Runs the greedy scheduler over the fixed 5-day schedule, 10 slots per day.
- Aggregates and prints a detailed console report showing:
    - Scheduled content for each day.
    - Daily and weekly totals for adjusted revenue and engagement.
    - Rejected content per day due to rule violations.
- Uses fixed hours for time-dependent rule application during reporting.

CODING

app->main.py

```
from model.content import Content, Monetization
from engine.scheduler import GreedyScheduler
from engine.strategies import by_revenue, by_engagement, balanced
from scoring.rules import RuleContext, Scoring

# HARDCODING THE CATALOG
catalog = [
    Content('C1', 'Blockbuster Movie', 120, Monetization.SUBSCRIPTION, 200, 95,
True, set([1,2,3]), 2, 'movie'),
    Content('C2', 'Sports Match (Live)', 180, Monetization.PPV, 400, 90, True,
set([2]), 1, 'sports'),
    Content('C3', 'Original Series Ep1', 45, Monetization.SUBSCRIPTION, 150, 85,
True, set(range(1,6)), 2, 'originals'),
    Content('C4', 'Kids Cartoon Hour', 60, Monetization.AD, 60, 70, False,
set(range(1,6)), 5, 'kids'),
```

```
    Content('C5', 'Music Concert Special', 90, Monetization.PPV, 300, 88, True,
set([4,5]), 1, 'music'),
    Content('C6', 'Regional Drama', 60, Monetization.AD, 80, 75, False,
set(range(1,6)), 3, 'regional'),
    Content('C7', 'Documentary Feature', 75, Monetization.SUBSCRIPTION, 120, 78,
True, set(range(2,6)), 2, 'documentary'),
    Content('C8', 'Stand-up Comedy Night', 60, Monetization.AD, 90, 82, False,
set(range(3,6)), 2, 'originals'),
    Content('C9', 'Originals: Ep2', 45, Monetization.SUBSCRIPTION, 160, 86, True,
set(range(2,6)), 2, 'originals'),
    Content('C10', 'Regional Music Show', 45, Monetization.AD, 70, 72, False,
set(range(1,6)), 3, 'regional')
]
#SETTING DAY 2 AND 3 AS WEEKEND DAYS
weekend_days = {2, 3}
#CHOOSING THE STRATEGY
key_func = lambda scoring, ctx, c, hour: balanced(scoring, ctx, c, hour)

scheduler = GreedyScheduler(catalog, weekend_days, key_func)
schedule, rejected = scheduler.run()

weekly_rev_total = 0
weekly_eng_total = 0

print("=== SmartStream Scheduling Report ===")
for day in range(1, 6):
    ctx = RuleContext(day, weekend_days)
    scoring = Scoring(ctx)
    day_schedule = schedule.get(day, [])
    day_revenue = 0
    day_engagement = 0
    print(f"Day {day} ({'Weekend' if day in weekend_days else 'Weekday'}):")
    print(f" Scheduled: {day_schedule}")
    for cid in day_schedule:
        content = next((c for c in catalog if c.id == cid), None)
        if content:
            day_revenue += scoring.adjusted_revenue(content, 10)
            day_engagement += scoring.adjusted_engagement(content, 10)
    weekly_rev_total += day_revenue
    weekly_eng_total += day_engagement
    #TOTAL REVENUE AND ENGAGEMENT
    print(f" Total Revenue = ₹{int(day_revenue)}")
    print(f" Total Engagement = {int(day_engagement)}")
```

```
    print()

print("-------------------------------------")
print("WEEKLY TOTALS")
#TOTAL REVENUE AND ENGAGEMENT
print(f"Total Revenue = ₹{int(weekly_rev_total)}")
print(f"Total Engagement = {int(weekly_eng_total)}")
print()
print("Rejected Items per Day:")
#PRINTS THE REJECTED ITEMS
for day in range(1, 6):
    rejected_items = rejected.get(day, [])
    if rejected_items:
        print(f"Day {day}: {rejected_items}")
    else:
        print(f"Day {day}: None")
```

engine->scheduler.py

```
from abc import ABC, abstractmethod
from typing import List, Dict, Callable
from collections import defaultdict
from model.content import Content
from scoring.rules import RuleContext, Scoring
from model.content import MaxRunsExceeded

class Scheduler(ABC):
    def __init__(self, catalog: List[Content], weekend_days: set):
        self.catalog = catalog
        self.weekend_days = weekend_days
        self.day_slots = 10
        self.days = 5
```

```
    #INITIALIZING WITH DAY AND SLOTS
    @abstractmethod
    def run(self):
        pass

class GreedyScheduler(Scheduler):
    def __init__(self, catalog: List[Content], weekend_days: set, key_func:
Callable):
        super().__init__(catalog, weekend_days)
        self.key_func = key_func

    def run(self):
        #INITIALIZING SCHEDULE AND REJECTED LISTS
        schedule: Dict[int, List[str]] = {day: [] for day in range(1,
self.days+1)}
        rejected: Dict[int, List[str]] = {day: [] for day in range(1,
self.days+1)}
        #REMAINING ALLOWED RUNS LEFT
        global_runs_left: Dict[str,int] = {c.id: c.max_runs for c in
self.catalog}

        #PREPARING A RULE CONTEXT
        #REJECTS THOSE WHO ARE NOT IN THE RELEASE WINDOW OR WITH ZERO RUNS LEFT
        #HERE RUNS IS REFERRING TO MAX_RUNTIME RUNS
        #ONLY WHO PASS THESE CHECKS ARE CONSIDERED FOR SCHEDULING
        for day in range(1, self.days+1):
            ctx = RuleContext(day, self.weekend_days)
            scoring = Scoring(ctx)

            candidates = []
            for c in self.catalog:
                if not c.in_window(day):
                    rejected[day].append(c.id + ' (Not in window)')
                    continue
                if global_runs_left[c.id] <= 0:
                    rejected[day].append(c.id + ' (Max runs reached)')
                    continue
                candidates.append(c)

            picks = []
            genre_count = defaultdict(int)
            slot_hours = [10]*self.day_slots
            #
```

```
            for slot in range(self.day_slots):
                hour = slot_hours[slot]
                filtered_candidates = [c for c in candidates if
global_runs_left[c.id] > 0]
                if not filtered_candidates:
                    break
                filtered_candidates.sort(key=lambda c: self.key_func(scoring,
ctx, c, hour), reverse=True)

                chosen = None
                for item in filtered_candidates:
                    if genre_count[item.genre.lower()] >= ctx.max_genre_per_day:
                        continue
                    chosen = item
                    break
                if chosen is None:
                    break

                picks.append(chosen.id)
                genre_count[chosen.genre.lower()] += 1
                global_runs_left[chosen.id] -= 1

            schedule[day] = picks

        return schedule, rejected
```

engine->strategies.py

```
from model.content import Content
from scoring.rules import Scoring, RuleContext

def by_revenue(scoring: Scoring, ctx: RuleContext, content: Content, hour: int) -
> float:
    return scoring.adjusted_revenue(content, hour)

def by_engagement(scoring: Scoring, ctx: RuleContext, content: Content, hour:
int) -> float:
    return scoring.adjusted_engagement(content, hour)
```

```
def balanced(scoring: Scoring, ctx: RuleContext, content: Content, hour: int,
wR=0.6, wE=0.4) -> float:
    rev = scoring.adjusted_revenue(content, hour)
    eng = scoring.adjusted_engagement(content, hour)
    return wR * rev + wE * eng

#INTERCHANGABLE KEY FUNCTIONS
```

model->content.py

```
from dataclasses import dataclass
from enum import Enum
from typing import Set

class Monetization(Enum):#SETTING MONETIZATION TYPES USING ENUMERATOR
    SUBSCRIPTION = 1
    AD = 2
    PPV = 3
```

```
class ContentNotInWindow(Exception):
    pass

class MaxRunsExceeded(Exception):
    pass

@dataclass(frozen=True)#TO CREATE IMMUTABLE DATAOBJECTS
class Content:
    id: str
    title: str
    duration_min: int
    monetization: Monetization
    base_revenue: int
    base_engagement: int
    premium: bool
    release_window: Set[int]
    max_runs: int
    genre: str#ADDED EXPLICITLY TO ADHERE TO THE CONDITION GIVE IN MANUAL

    def in_window(self, day: int) -> bool:#TO CHECK WHETHER IN THE RELEASE WINDOW
        return day in self.release_window

    def is_kids(self) -> bool:#TO CHECK IF IT FALLS UNDER KIDS CATEGORY
        return self.genre.lower() == 'kids'
    def is_ppv(self)->bool:#TO CHECK IF IS A PAY PER VIEW
        return self.monetization==Monetization
```

scoring->rules.py

```
from typing import Dict, Set, Tuple
from collections import defaultdict
from model.content import Content
from util.helpers import clamp

class RuleContext:#HERE AS SAID WE HAVE INCLUDED GENRE MAX 4 PER DAY OF SAME
GENRE
```

```
    #EXPLICITLY INCLUDED TIME IN THE INIT AS MENTIONED IN THE PDF (9-15 IS KIDS
TIME)
    def __init__(self, day: int, weekend_days: Set[int], kids_hours:
Tuple[int,int] = (9,15),
                 prime_time_hours: Tuple[int,int] = (19,22), max_genre_per_day:
int = 4):
        self.day = day
        self.is_weekend_day = day in weekend_days#TRACK OF DAYS TO DETERMINE
WEEKEND
        #THE NEXT THREE LINES ARE USED TO CHECK GENRE DIVERSITY GAP
        self.kids_hours = kids_hours
        self.prime_time_hours = prime_time_hours
        self.max_genre_per_day = max_genre_per_day
        self.genre_counts: Dict[str,int] = defaultdict(int)
        self.runs_left: Dict[str,int] = {}
    #HELPER METHODS
    def is_weekend(self) -> bool:#TO CHECK FOR WEEKEND
        return self.is_weekend_day

    def is_prime_time(self, hour: int) -> bool:#TO CHECK FOR PRIME TIME
        return self.prime_time_hours[0] <= hour < self.prime_time_hours[1]

    def is_kids_daytime(self, hour: int) -> bool:#TO CHECK KIDS TIME
        return self.kids_hours[0] <= hour < self.kids_hours[1]

    def genre_over_cap(self, genre: str) -> bool:#ASSIGNMENT OF ANOTHER GENRE IF
EXCEEDS 4
        return self.genre_counts[genre.lower()] >= self.max_genre_per_day

    def increment_genre(self, genre: str):#INCREMENT GENRE COUNT(MAX 4)
        self.genre_counts[genre.lower()] += 1

class Scoring:
    def __init__(self, context: RuleContext):
        self.ctx = context
    #TO CALCULATE ADJUSTING REVENUE
    def adjusted_revenue(self, content: Content, hour: int) -> float:
        revenue = content.base_revenue
        if content.is_ppv() and self.ctx.is_prime_time(hour):
            revenue *= 1.25#MONETIZATION->PPV AND PRIME TIME
        if self.ctx.is_weekend():
            if content.is_ppv():
                revenue *= 1.15#WEEKEND MULTIPLIER
```

```
        return revenue

    def adjusted_engagement(self, content: Content, hour: int) -> int:
        engagement = content.base_engagement
        if content.is_kids() and self.ctx.is_kids_daytime(hour):
            engagement += 10#+10 FOR DAYTIME
        engagement = clamp(engagement, 0, 100)
        if self.ctx.is_weekend():
            engagement += 8#+8 FOR ALL CONTENT(WEEKEND)
        engagement = clamp(engagement, 0, 100)
        if self.ctx.genre_over_cap(content.genre):
            engagement -= 8#IF EXCEED GENRE OVER CAP -8
        engagement = clamp(engagement, 0, 100)
        return engagement
```

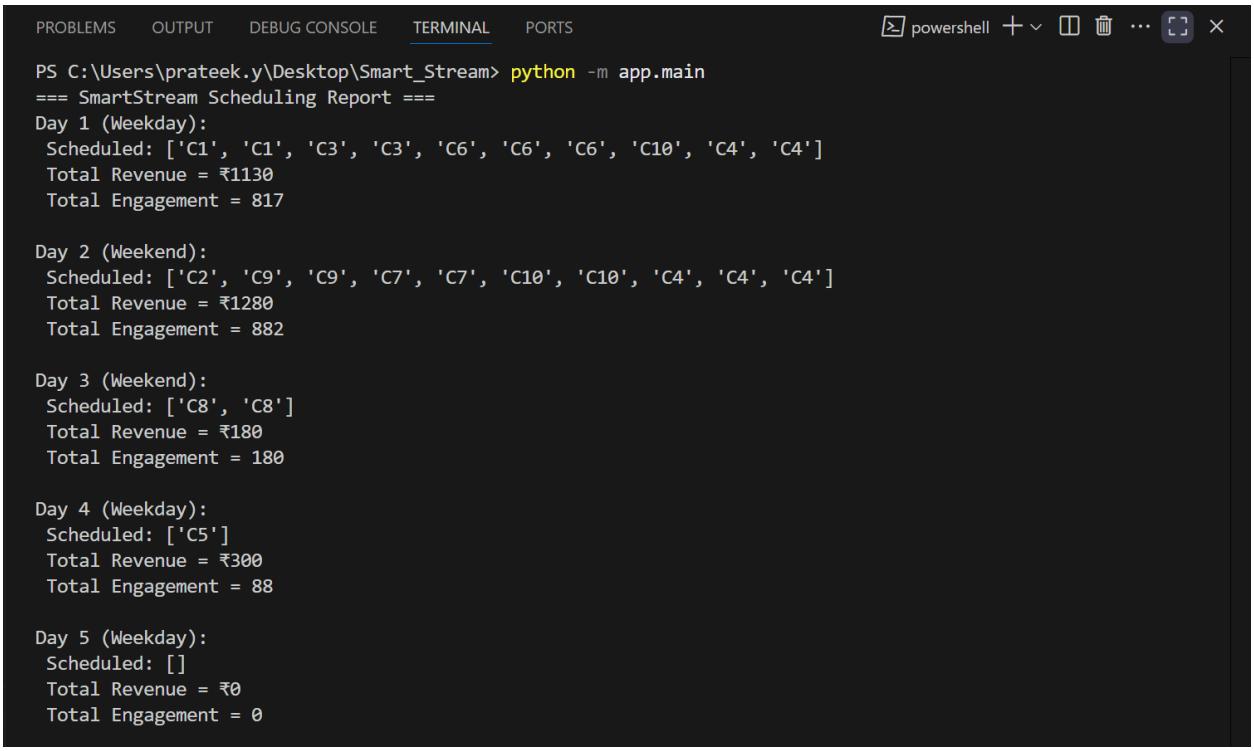
util->helpers.py

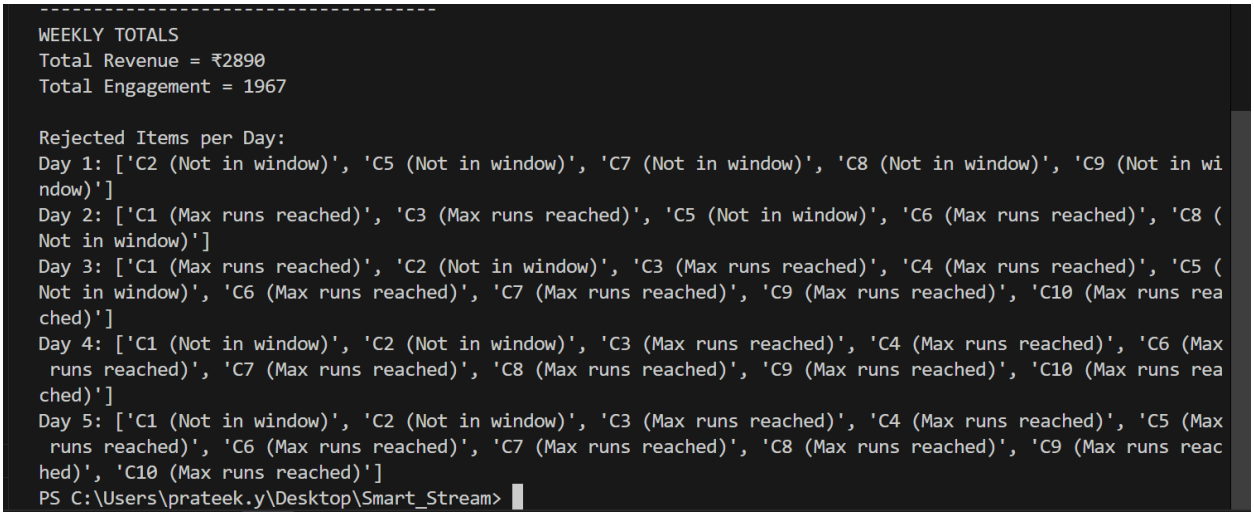
```
def clamp(value:int,min_value:int,max_value:int)->int:

    return max(min_value,min(value,max_value))
```

```
#THE SOLE PURPOSE HERE IS TO SET THE CLAMP VALUES
#FOR EXAMPLE IF IT IS 105 IT WILL BE SET TO 100 OR IF IT IS -10 IT WILL BE SET TO
0
#AS GIVEN IN THE INSTRUCTION(SET BETWEEN 0 AND 100)
```

OUTPUT:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
powershell
PS C:\Users\prateek.y\Desktop\Smart_Stream> python -m app.main
=== SmartStream Scheduling Report ===
Day 1 (Weekday):
Scheduled: ['C1', 'C1', 'C3', 'C3', 'C6', 'C6', 'C6', 'C10', 'C4', 'C4']
Total Revenue = ₹1130
Total Engagement = 817
Day 2 (Weekend):
Scheduled: ['C2', 'C9', 'C9', 'C7', 'C7', 'C10', 'C10', 'C4', 'C4', 'C4']
Total Revenue = ₹1280
Total Engagement = 882
Day 3 (Weekend):
Scheduled: ['C8', 'C8']
Total Revenue = ₹180
Total Engagement = 180
Day 4 (Weekday):
Scheduled: ['C5']
Total Revenue = ₹300
Total Engagement = 88
Day 5 (Weekday):
Scheduled: []
Total Revenue = ₹0
Total Engagement = 0


WEEKLY TOTALS
Total Revenue = ₹2890
Total Engagement = 1967
Rejected Items per Day:
Day 1: ['C2 (Not in window)', 'C5 (Not in window)', 'C7 (Not in window)', 'C8 (Not in window)', 'C9 (Not in window)']
Day 2: ['C1 (Max runs reached)', 'C3 (Max runs reached)', 'C5 (Not in window)', 'C6 (Max runs reached)', 'C8 (Not in window)']
Day 3: ['C1 (Max runs reached)', 'C2 (Not in window)', 'C3 (Max runs reached)', 'C4 (Max runs reached)', 'C5 (Not in window)', 'C6 (Max runs reached)', 'C7 (Max runs reached)', 'C9 (Max runs reached)', 'C10 (Max runs reached)']
Day 4: ['C1 (Not in window)', 'C2 (Not in window)', 'C3 (Max runs reached)', 'C4 (Max runs reached)', 'C6 (Max runs reached)', 'C7 (Max runs reached)', 'C8 (Max runs reached)', 'C9 (Max runs reached)', 'C10 (Max runs reached)']
Day 5: ['C1 (Not in window)', 'C2 (Not in window)', 'C3 (Max runs reached)', 'C4 (Max runs reached)', 'C5 (Max runs reached)', 'C6 (Max runs reached)', 'C7 (Max runs reached)', 'C8 (Max runs reached)', 'C9 (Max runs reached)', 'C10 (Max runs reached)']
PS C:\Users\prateek.y\Desktop\Smart_Stream>