Software Design and Analysis (SE-2002)

Assignment 4

Software: AnkiDroid

Group#10

Sana Idrees(23i-2039), Manahil(23i-3000),

Maryam Fatima(23i-3007) and Ayesha Khan(23i-3037)

BS-Software Engineering(4B)

Submitted to: Dr. Behjat Zuhaira

Client: Group # 1

GRASP:

I. Principles Applied:

- Creator
- Information Expert
- High Cohesion
- Low coupling
- Polymorphism
- Pure fabrication
- Controller
- Protected variations
- Indirection

II. Where are the principles applied?

III. Justification of Principles applied:

Principle	Classes	Justification
Creator	• **Deck** creates **Flashcards** • **Learner** creates **Deck**	Object creation was assigned to the classes that logically own or aggregate other objects. For example, Deck creates flashcards, Learner creates decks. This reduces dependency on external factories
Information Expert	• Learner • Deck • Flashcard • Progress Tracker	Assigned responsibilities to classes that have the data needed. For instance, the deck knows about flashcards, and it is responsible for adding and deleting cards. This keeps behavior near the data and avoids way too much coupling
High Cohesion	• Progress Tracker • Flashcard review session	Classes are designed to perform well-defined tasks or a set of tasks. E.g., Progress Tracker focuses only on tracking and updating performance metrics, which enhances clarity and maintainability.
Low coupling	• Learner interacts with the Flashcard battle • Deck manages Flashcards	Interaction between classes is minimal and done through an interface. For example, Learner uses the AI Recommendations engine

	- Ai Recommendation Engine	indirectly ensuring changes in recommendation logic don’t ripple through unrelated classes Ai recommendation indirectly via Facade
Polymorphism	- Flashcard review Session - Shared Study Session Functions like startSession() and endSession() can be over ridden	Session related classes like **FlashcardReviewSession** and **SharedStudySession** provide polymorphic implementations for common method like StartSession etc.
Indirection	**AI recommendation engine** is the one that recommends flashcards instead of learner doing it directly.	As it is used for managing complex responsibilities So AI Recommendation Engine and sync operation is included in it .This reduces direct dependences between the core entities and external services.
Controller	- Flashcard Battle - Flashcard review Session	Workflow heavy classes like Flashcard battles and review session are handled by controller.This centralizes coordination logic, seperates it from UI or data class and also enhance modularity
Protected variations	**SyncOperations** manages changes independently from other classes	Classes like Sync Ops isolate potential variation points such as cloud storage API integration and sync logic.
Pure fabrication	**AI recommendation engine** **SyncOperations** **ProgressTracker**	**AI recommendation engine, SyncOperations, ProgressTracker** are fabricated classes to improve separation of concerns and high cohesion will be achieved this way.

GOF

1. Observer Pattern (Behavioral)

Category: Behavioral

- **Problem Solved:**

  When a flashcard is reviewed or a battle ends, progress, leaderboard, and streaks should update automatically. Observer decouples those updates from core logic.

- **Classes/Interfaces Modified or Introduced:**

  `Observer` and `Subject` interfaces:

  - `ProgressTracker`, `Leaderboard` as observers
  - `ReviewSession`, `FlashcardBattle` as subjects

FOR EXAMPLE

Interface: Subject

- attach(observer: Observer): void
- detach(observer: Observer): void
- notifyObservers(): void

Interface: Observer

- update(): void

Class: Leaderboard

- update(): void
- displayRankings(): void

****

2. Strategy Pattern (Behavioral)

Category: Behavioral

- **Problem Solved:**

  Different review behaviors (e.g., spaced repetition, battle quiz, manual review) need to be swappable without rewriting ReviewSession logic.

- **Classes/Interfaces Modified or Introduced:**
  - `ReviewStrategy` (interface)
  - `SpacedRepetitionStrategy`, `BattleReviewStrategy`, `ManualReviewStrategy` (implementations)
  - Modified `ReviewSession` to include a `strategy` field (ReviewSession.setStrategy())

FOR EXAMPLE

Interface: ReviewStrategy

- review(deck: Deck): void

Class: SpacedRepetitionStrategy

- review(deck: Deck): void

Class: BattleReviewStrategy

- review(deck: Deck): void

Class: ManualReviewStrategy

- review(deck: Deck): void

Class: ReviewSession

- strategy: ReviewStrategy
- setStrategy(strategy: ReviewStrategy): void
- startReview(): void
- attach(observer: Observer): void
- notifyObservers(): void

****

3. Singleton Pattern (Creational)

Category: Creational

- **Problem Solved:**

  Ensures only one instance of core managers like `DeckManager`, `SyncManager`, `ProgressTracker`, so global state remains consistent and controlled.

- **Classes/Interfaces Modified or Introduced:**
  - Singleton version of `DeckManager`, `SyncManager`, `ProgressTracker`, `ChallengeManager`, `FlashcardManager`, `ReviewSchedular`, `ShareStudySessionManager`
  - `getInstance()` methods added
  - Constructors made private

FOR EXAMPLE

Class: DeckManager

- instance: DeckManager
- DeckManager() // private constructor
- getInstance(): DeckManager
- addDeck(deck: Deck): void
- removeDeck(deckID: String): void

Class: ProgressTracker

- instance: ProgressTracker
- ProgressTracker() // private constructor
- getInstance(): ProgressTracker
- updateProgress(): void
- displayStats(): void

- update(): void // from Observer

Class: SyncManager

- instance: SyncManager
- SyncManager() // private constructor
- getInstance(): SyncManager
- syncWithAnkiWeb(): void

****

4. Factory Method (Creational)

Category: Creational

- **Problem Solved:**

  Simplifies creation of complex objects like `Flashcard`, `Deck`, `DailyChallenge`, etc., while centralizing validation and construction logic. Factory centralizes and manages creation logic, especially if attributes like difficulty or category grow in the future.

- **Classes/Interfaces Modified or Introduced:**
  - `FlashcardFactory`
  - `DeckFactory`
  - `ChallengeFactory`
  - Use in `DeckManager`, `FlashcardManager,ChallengeManager`

FOR EXAMPLE

Class: FlashcardFactory

- createFlashcard(question: String, answer: String, difficulty: String): Flashcard

Class: DeckFactory

- createDeck(title: String): Deck

Class: ChallengeFactory

- createChallenge(name: String, cardLimit: int): DailyChallenge

****

5. Facade Pattern (Structural)

Category: Structural

- **Problem Solved:**
  UI previously had to interact with multiple manager classes directly. Facade simplifies system interaction by exposing one clean interface.
- **Classes/Interfaces Modified or Introduced:**
  - `FlashcardFacade`
    - Methods: `startReview()`, `addDeck()`, `sync()`, `getProgress(),getLeaderboard(),SyncWithCloud()`
    - `Manages:DeckManager,FlashCardManager,ReviewSession,SyncOperation,Progressmanager,ChallengeManager,AiRecommendationEngine`
  - UI now interacts **only** with this facade

FOR EXAMPLE

Class: FlashcardFacade

- createDeck(title: String): void
- startReview(deckID: String): void
- completeReview(deckID: String): void
- getProgress(userID: String): void
- sync(): void
- ManageCards(cardId:int):void
- GetLearnerProgress(Learner:Learner):void

****

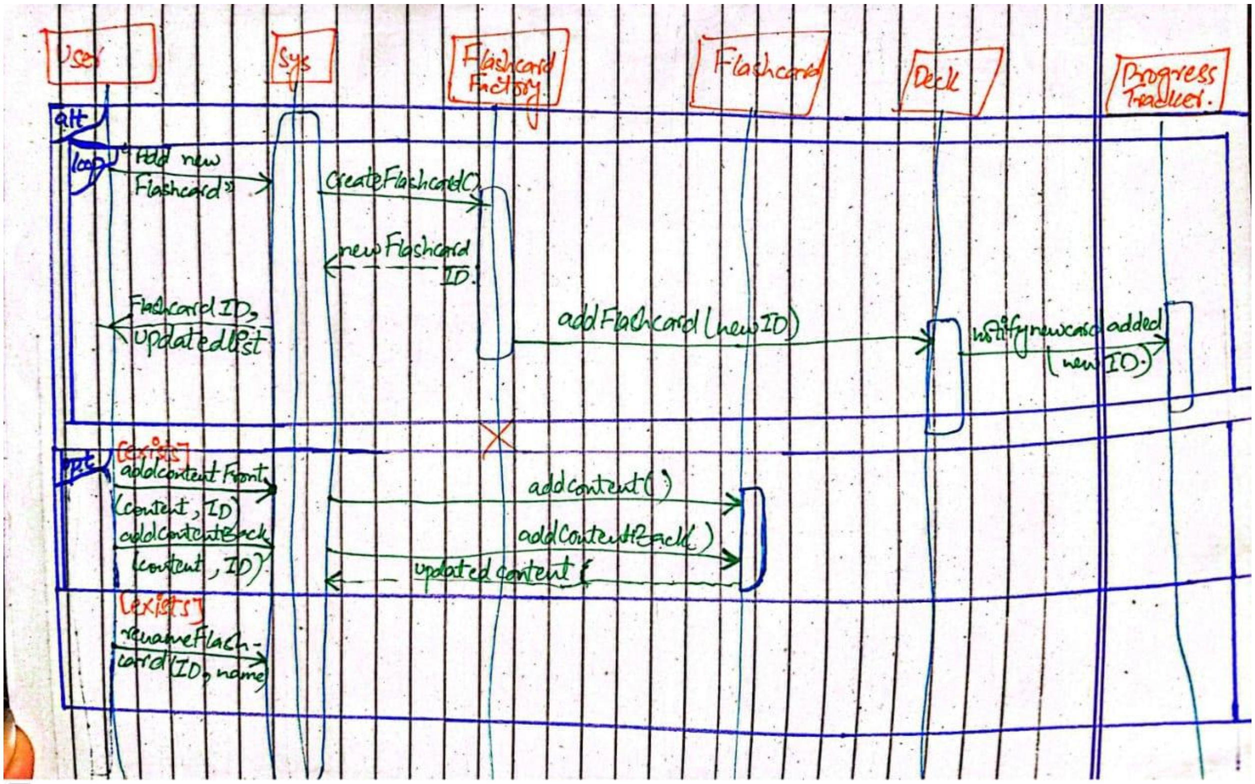
Overall Summary:

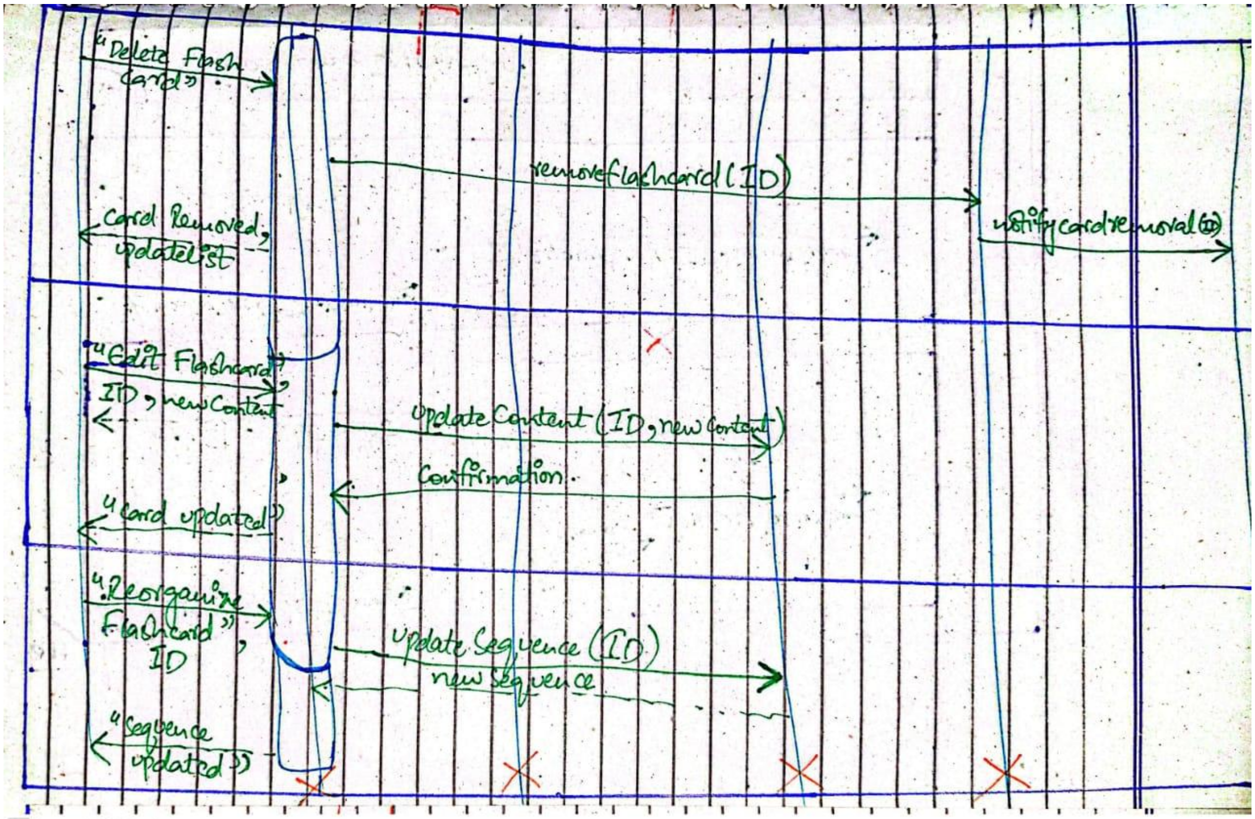
Pattern	**Category**	**Problem Solved**	**Classes/Interfaces Modified/Introduced**
Factory Method	Creational	Centralized object creation	`FlashcardFactory`, `DeckFactory`, `ChallengeFactory`
Strategy	Behavioral	Flexible review behavior	`ReviewStrategy`, strategy implementations, `ReviewSession`
Singleton	Creational	Single-instance managers	`DeckManager`, `SyncManager`, `ProgressTracker,ChallengeManager,FlashCardsManager,ReviewSchedular,SharedStudySessionManager`
Observer	Behavioral	Decoupled real-time updates	`Observer`, `Subject`, `ReviewSession`, `Leaderboard`
Facade	Structural	Unified, simple system access for UI	`FlashcardFacade`

****

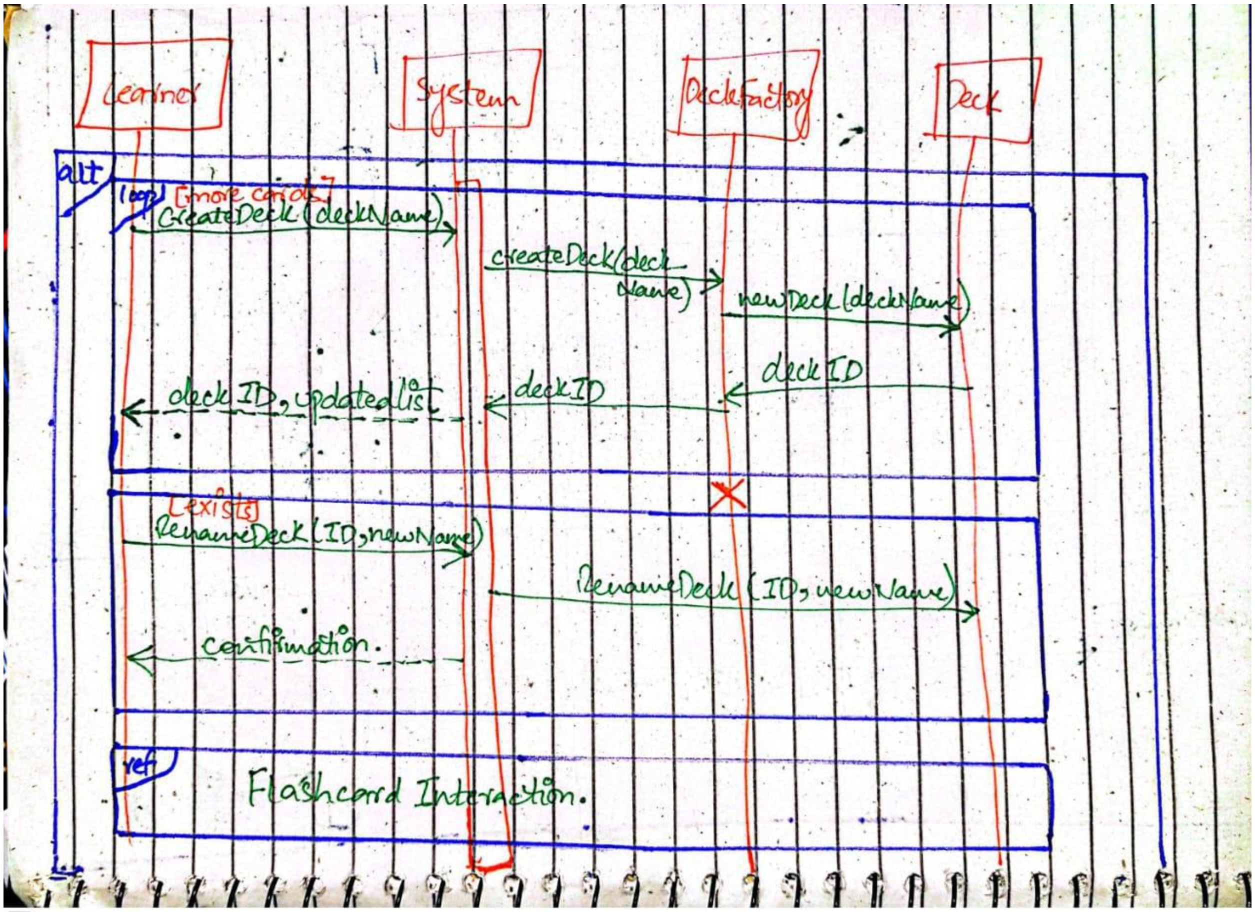
Sequence Diagrams:

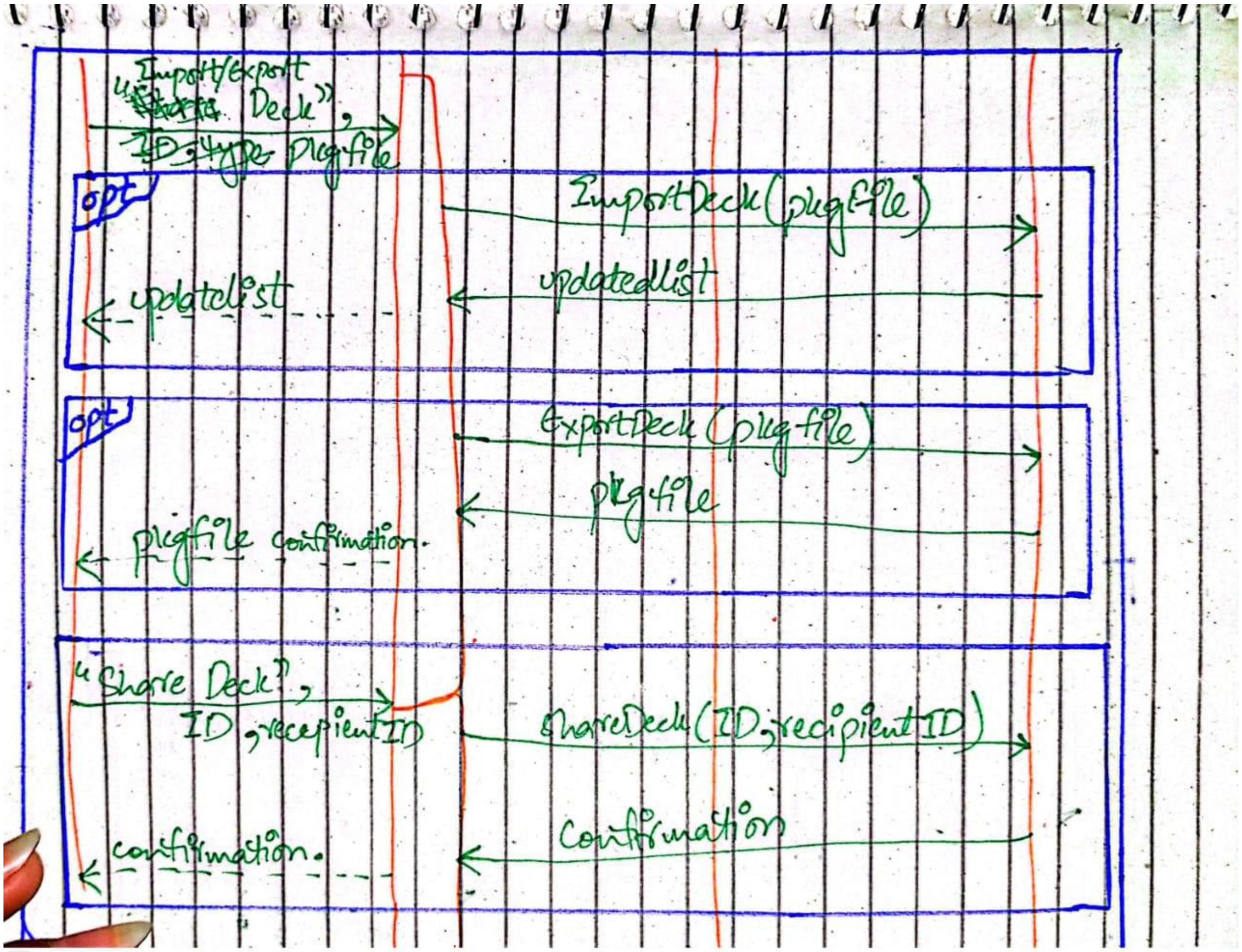
→ **Manage Flashcards:**

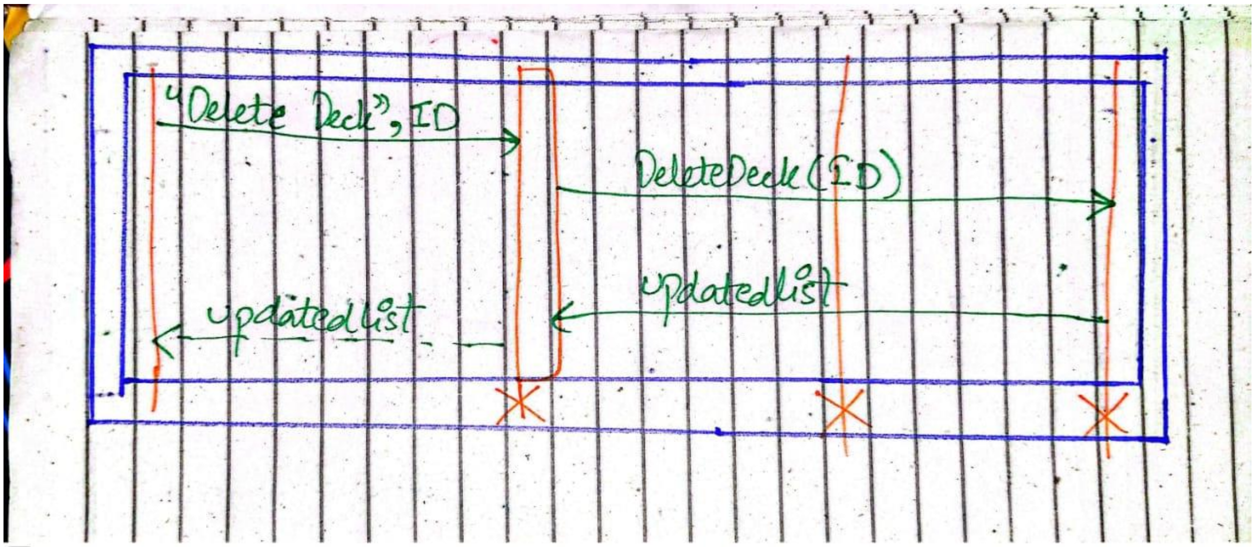

User
Sys
Flashcard Factory
Flashcard
Deck
Progress Tracker.
alt
loop
"Add new Flashcard"
createFlashcard()
new Flashcard ID
Flashcard ID, updatedlist
addFlashcard(newID)
notifynewcard added (new ID.)
alt
[exists]
addcontentFront (content, ID)
addContentBack (content, ID)
addContent()
addContentBack()
updated content
[exists]
renameFlashcard(ID, name)


"Delete Flash card"
removeFlashcard(ID)
card Removed, updatelist
notifycardremoval(ID)
"Edit Flashcard"
ID, newContent
updateContent(ID, new Content)
Confirmation
"card updated"
"Reorganize Flashcard", ID
updateSequence(ID)
new Sequence
"Sequence updated"

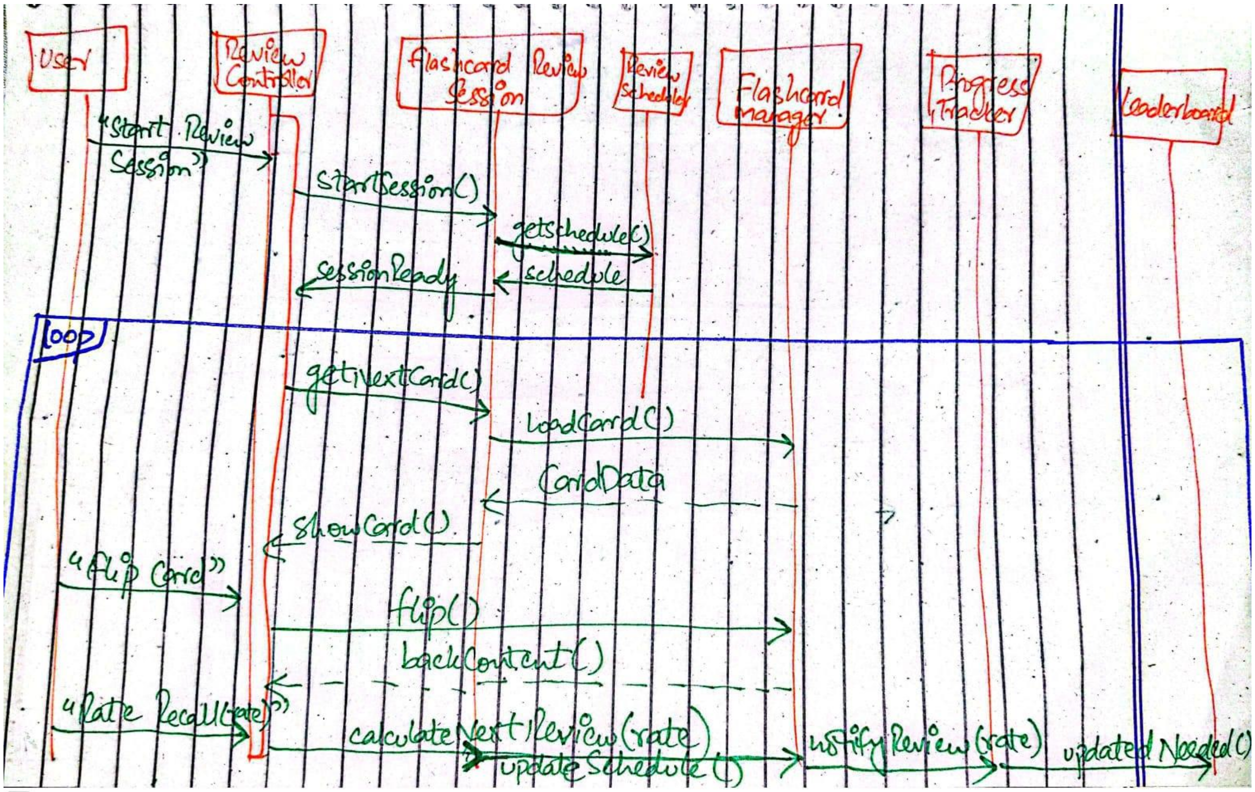
→ **Manage Deck:**

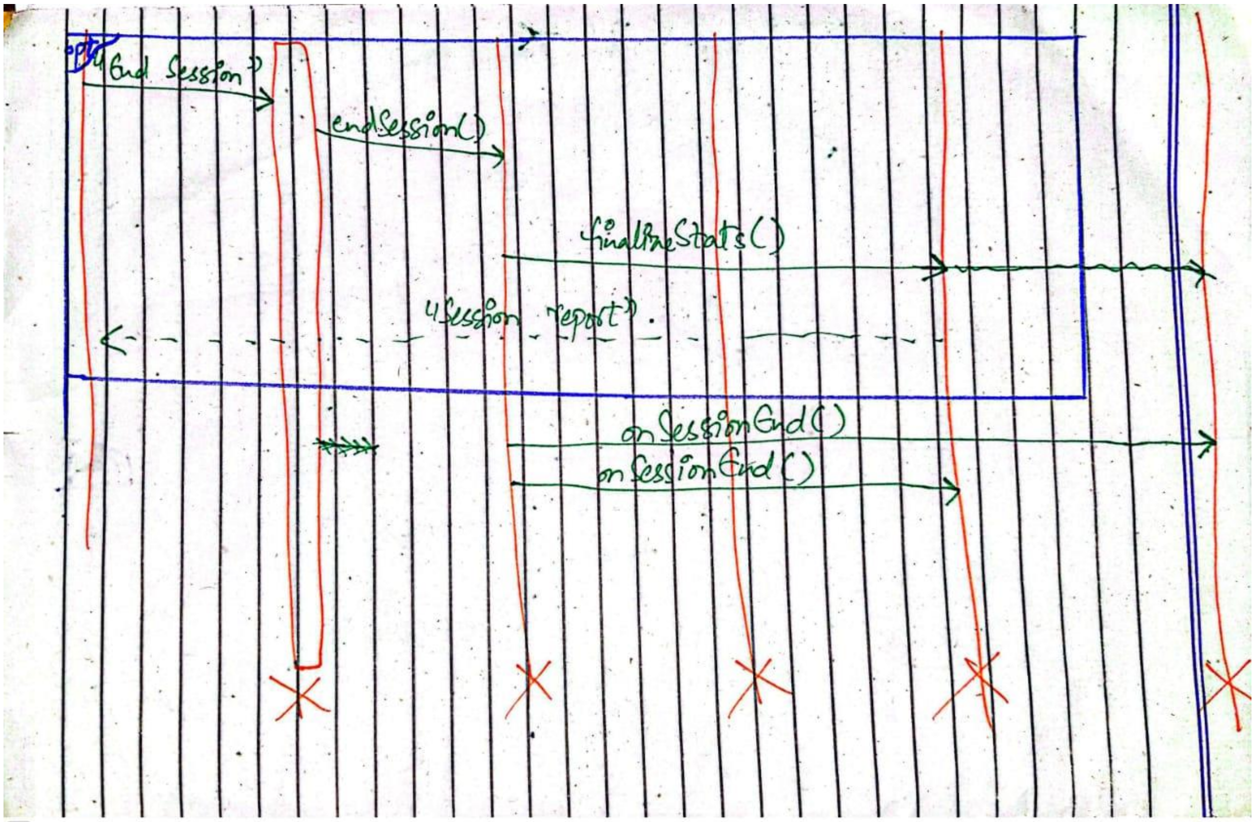

Learner
System
DeckFactory
Deck
alt
loop
[more cards]
CreateDeck(deckName)
createDeck(deck Name)
newDeck(deckName)
deckID
deckID
deckID, updatedlist
[exists]
RenameDeck(ID,newName)
RenameDeck(ID,newName)
confirmation.
ref
Flashcard Interaction.



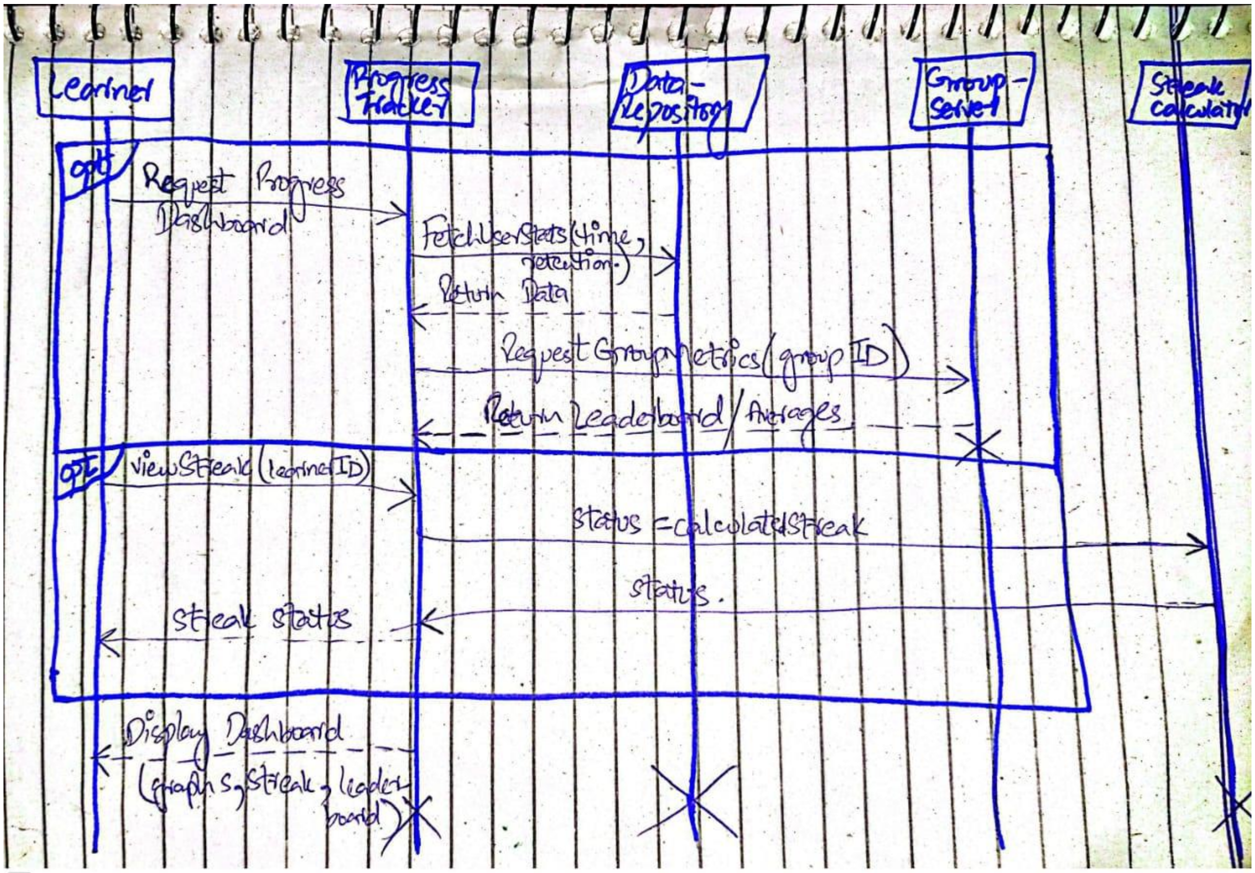


→ **Review Flashcards:**

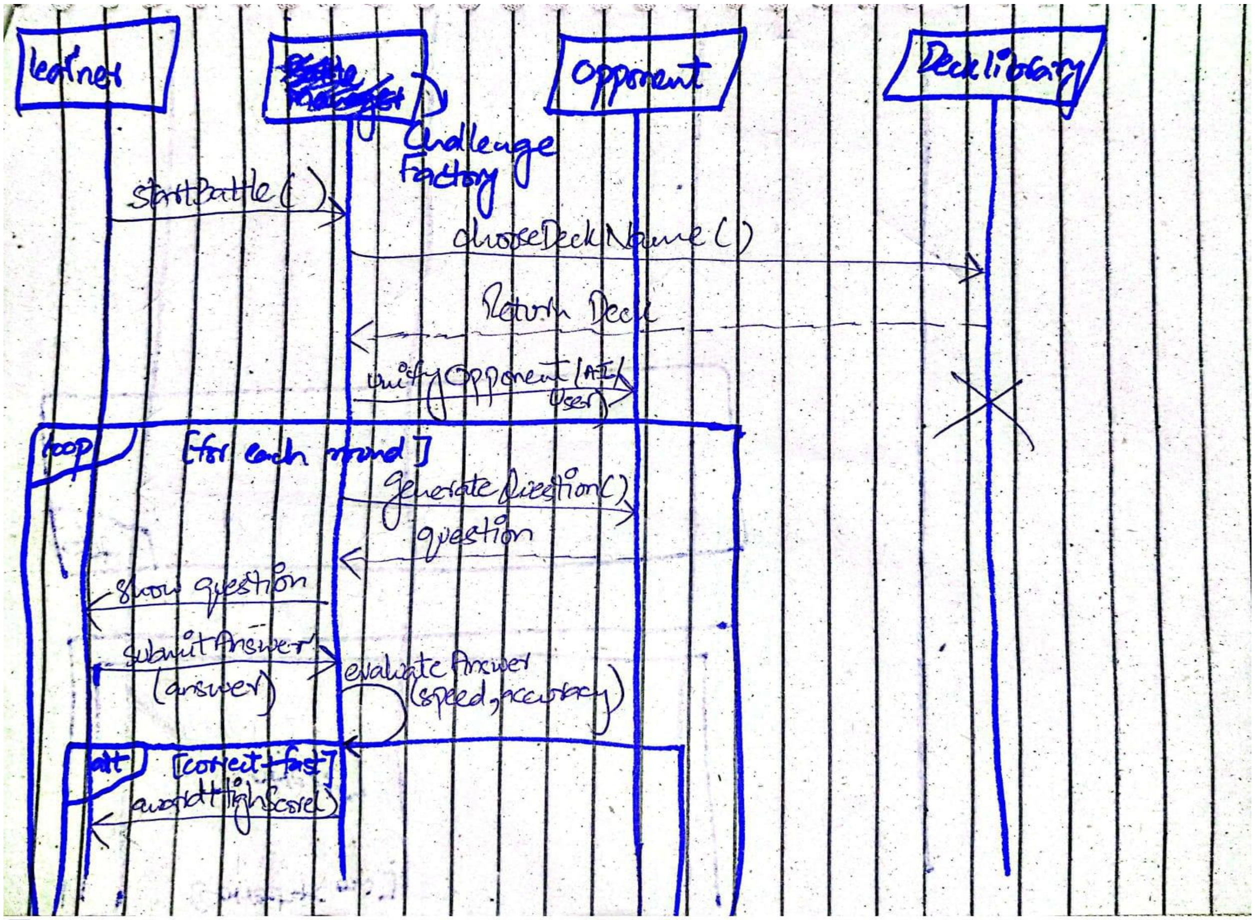

User
Review Controller
flashcard Review Session
Review Scheduler
Flashcard manager
Progress Tracker
Leaderboard
"Start Review Session"
startSession()
getSchedule()
schedule
sessionReady
loop
getNextCard()
LoadCard()
CardData
showCard()
"flip Card"
flip()
backContent()
"Rate Recall(rate)"
calculateNextReview(rate)
updateSchedule()
notifyReview(rate)
updatedNeeded()

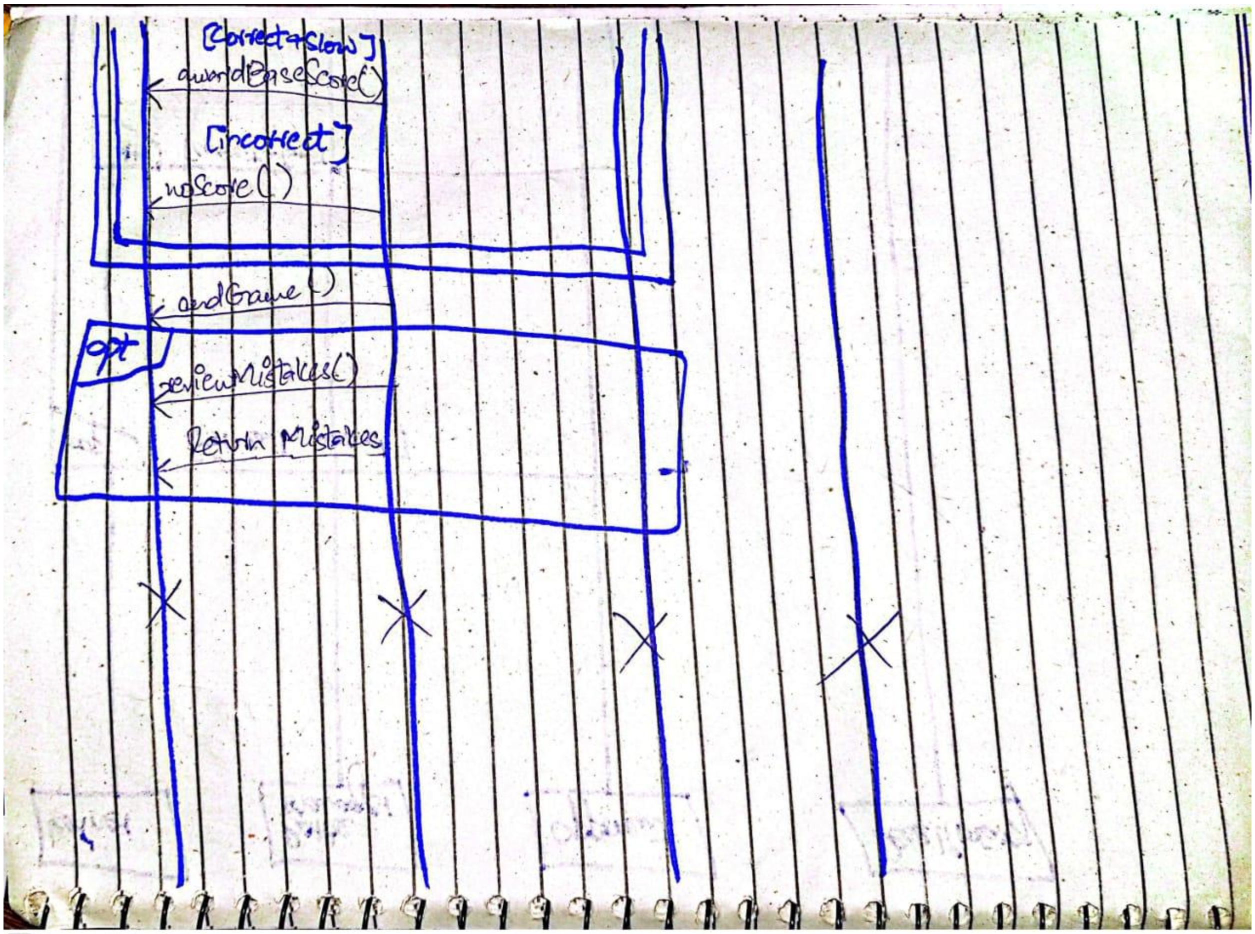

opt
"End Session"
endSession()
finalizeStats()
"Session report"
onSessionEnd()
onSessionEnd()

→ **Progress Tracker:**

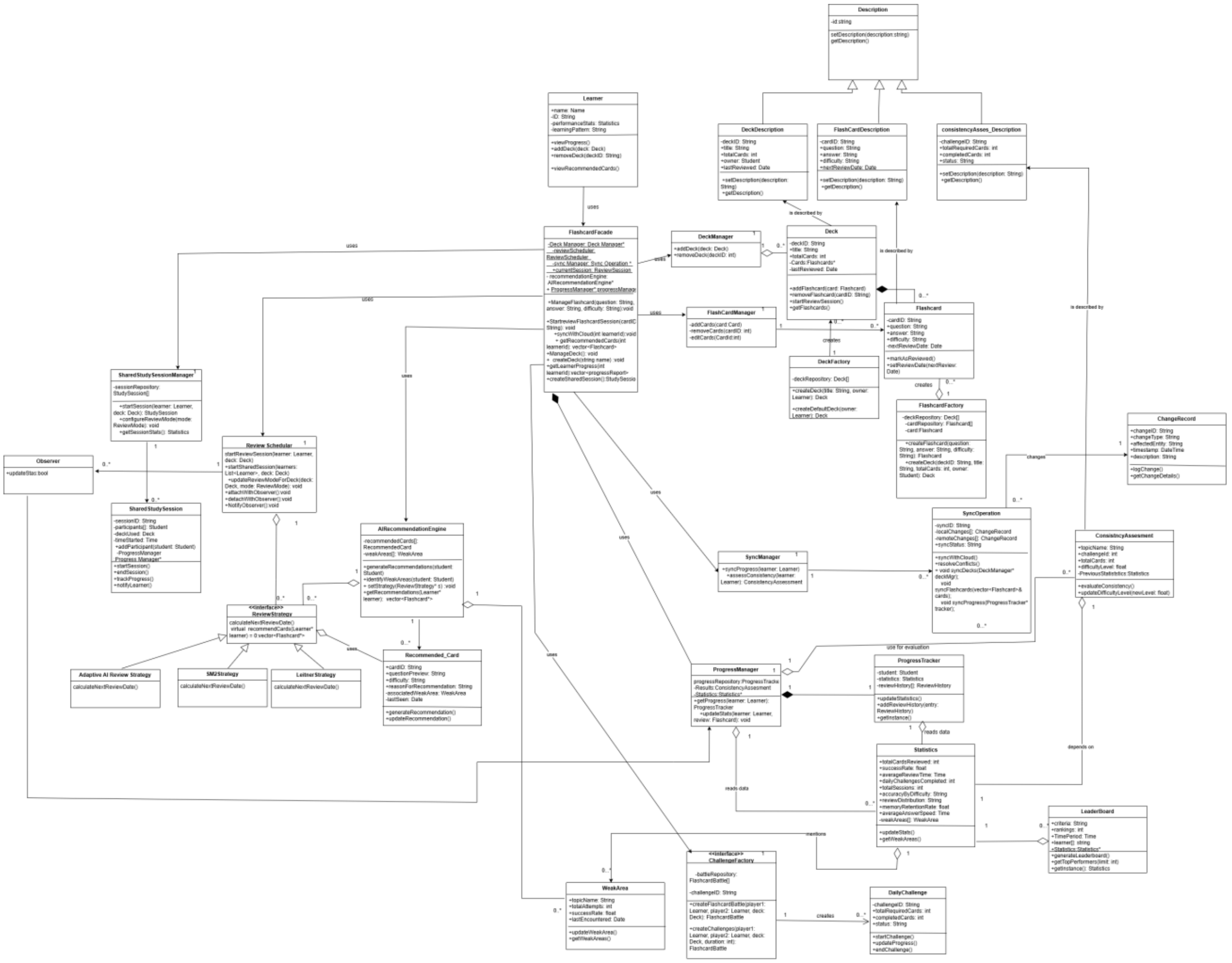

Learner
Progress Tracker
Data-Repository
Group-Server
Streak calculator
opt
Request Progress Dashboard
FetchUserStats(time, retention.)
Return Data
Request GroupMetrics(group ID)
Return Leaderboard/Averages
opt
viewStreak(learnerID)
status =calculatedStreak
status.
streak status
Display Dashboard (graphs, streak, leader board)

→ **Manage Flashcard Battles:**


learner
~~Battle Manager~~
Challenge Factory
Opponent
DeckLibrary
startBattle()
chooseDeckName()
Return Deck
notifyOpponent(AI/User)
loop
[for each round]
GenerateQuestion()
question
show question
submitAnswer(answer)
evaluateAnswer(speed, accuracy)
alt
[correct+fast]
awardHighScore()



Updated Class Diagram


Description
-id:string
setDescription(description:string)
getDescription()
DeckDescription
-deckID: String
+title: String
+totalCards: int
+owner: Student
+lastReviewed: Date
+setDescription(description: String)
+getDescription()
FlashCardDescription
-cardID: String
+question: String
+answer: String
+difficulty: String
+nextReviewDate: Date
+setDescription(description: String)
+getDescription()
consistencyAsses_Description
-challengeID: String
+totalRequiredCards: int
+completedCards: int
+status: String
+setDescription(description: String)
+getDescription()
Learner
+name: Name
-ID: String
-performanceStats: Statistics
-learningPattern: String
+viewProgress()
+addDeck(deck: Deck)
+removeDeck(deckID: String)
+viewRecommendedCards()
uses
FlashcardFacade
-Deck Manager: Deck Manager*
-reviewScheduler: ReviewScheduler
-sync Manager: Sync Operation *
+currentSession: ReviewSession
- recommendationEngine: AIRecommendationEngine*
+ ProgressManager*:progressManag
+ManageFlashcard(question: String, answer: String, difficulty: String):void
+StartreviewFlashcardSession(cardID: String): void
+syncWithCloud(int learnerId):void
+ getRecommendedCards(int learnerId): vector<Flashcard>
+ManageDeck(): void
+ createDeck(string name) :void
+getLearnerProgress(int learnerId):vector<progressReport>
+createSharedSession():StudySessio
uses
DeckManager
+addDeck(deck: Deck)
+removeDeck(deckID: int)
Deck
-deckID: String
+title: String
+totalCards: int
-Cards:Flashcards*
-lastReviewed: Date
+addFlashcard(card: Flashcard)
+removeFlashcard(cardID: String)
+startReviewSession()
+getFlashcards()
is described by
is described by
is described by
uses
FlashCardManager
-addCards(card:Card)
-removeCards(cardID: int)
-editCards(CardId:int)
Flashcard
-cardID: String
+question: String
+answer: String
+difficulty: String
-nextReviewDate: Date
+markAsReviewed()
+setReviewDate(nextReview: Date)
creates
DeckFactory
-deckRepository: Deck[]
+createDeck(title: String, owner: Learner): Deck
+createDefaultDeck(owner: Learner): Deck
creates
FlashcardFactory
-deckRepository: Deck[]
-cardRepository: Flashcard[]
-card:Flashcard
+createFlashcard(question: String, answer: String, difficulty: String): Flashcard
+createDeck(deckID: String, title: String, totalCards: int, owner: Student): Deck
ChangeRecord
+changeID: String
+changeType: String
+affectedEntity: String
+timestamp: DateTime
+description: String
+logChange()
+getChangeDetails()
changes
uses
uses
SharedStudySessionManager
-sessionRepository: StudySession[]
+startSession(learner: Learner, deck: Deck): StudySession
+configureReviewMode(mode: ReviewMode): void
+getSessionStats(): Statistics
Observer
+updateStas:bool
Review Schedular
startReviewSession(learner: Learner, deck: Deck)
+startSharedSession(learners: List<Learner>, deck: Deck)
+updateReviewModeForDeck(deck: Deck, mode: ReviewMode): void
+attachWithObserver():void
+detachWithObserver():void
+NotifyObserver():void
SharedStudySession
-sessionID: String
-participants[]: Student
-deckUsed: Deck
-timeStarted: Time
+addParticipant(student: Student)
-ProgressManager :Progress Manager*
+startSession()
+endSession()
+trackProgress()
+notifyLearner()
uses
AIRecommendationEngine
-recommendedCards[]: RecommendedCard
-weakAreas[]: WeakArea
+generateRecommendations(student: Student)
+identifyWeakAreas(student: Student)
+ setStrategy(ReviewStrategy* s) :void
+getRecommendations(Learner* learner): vector<Flashcard*>
<<interface>>
ReviewStrategy
calculateNextReviewDate()
virtual recommendCards(Learner* learner) = 0:vector<Flashcard*>
Adaptive AI Review Strategy
calculateNextReviewDate()
SM2Strategy
calculateNextReviewDate()
LeitnerStrategy
calculateNextReviewDate()
uses
Recommended_Card
+cardID: String
+questionPreview: String
+difficulty: String
+reasonForRecommendation: String
-associatedWeakArea: WeakArea
-lastSeen: Date
+generateRecommendation()
+updateRecommendation()
uses
uses
SyncManager
+syncProgress(learner: Learner)
+assessConsistency(learner: Learner): ConsistencyAssessment
SyncOperation
-syncID: String
-localChanges[]: ChangeRecord
-remoteChanges[]: ChangeRecord
+syncStatus: String
+syncWithCloud()
+resolveConflicts()
+ void syncDecks(DeckManager* deckMgr);
void syncFlashcards(vector<Flashcard>& cards);
void syncProgress(ProgressTracker* tracker);
ConsistncyAssesment
+topicName: String
+challengeId: int
+totalCards: int
+difficultyLevel: float
-PreviousStatistics:Statistics
+evaluateConsistency()
+updateDifficultyLevel(newLevel: float)
is described by
use for evaluation
ProgressManager
progressRepository:ProgressTracke
-Results:ConsistencyAssesment
-Statistics:Statistics*
+getProgress(learner: Learner): ProgressTracker
+updateStats(learner: Learner, review: Flashcard): void
ProgressTracker
-student: Student
-statistics: Statistics
-reviewHistory[]: ReviewHistory
+updateStatistics()
+addReviewHistory(entry: ReviewHistory)
+getInstance()
reads data
Statistics
+totalCardsReviewed: int
+successRate: float
+averageReviewTime: Time
+dailyChallengesCompleted: int
+totalSessions: int
+accuracyByDifficulty: String
+reviewDistribution: String
+memoryRetentionRate: float
+averageAnswerSpeed: Time
-weakAreas[]: WeakArea
+updateStats()
+getWeakAreas()
reads data
depends on
LeaderBoard
+criteria: String
+rankings: int
+TimePeriod: Time
+learner[]: string
+Statistics:Statistics*
+generateLeaderboard()
+getTopPerformers(limit: int)
+getInstance(): Statistics
mentions
WeakArea
+topicName: String
+totalAttempts: int
+successRate: float
+lastEncountered: Date
+updateWeakArea()
+getWeakAreas()
uses
<<interface>>
ChallengeFactory
-battleRepository: FlashcardBattle[]
-challengeID: String
+createFlashcardBattle(player1: Learner, player2: Learner, deck: Deck): FlashcardBattle
+createChallenges(player1: Learner, player2: Learner, deck: Deck, duration: int): FlashcardBattle
creates
DailyChallenge
-challengeID: String
+totalRequiredCards: int
+completedCards: int
+status: String
+startChallenge()
+updateProgress()
+endChallenge()