Project Report #2

Quiz of Fury

Daniel Seip
Daniel Terreros
Julia Rogers

Github URL: https://github.com/DanielTKC/quiz_of_fury/

CSCI 441

Table of Contents

All Team Members Contributed Equally To This Report

Summary of Changes:

- Added reference to Django querysets
- Added reference and explanation of SM-2 Algorithm
- Algorithms & Data Structures section included
- User Interface Design and implementation added
- Test Design section

---

1. Analysis and Domain Modeling

A. Conceptual Model

I. Concept definitions

Responsibility Description	Type	Concept Name
Manage all operations of retrieving, storing, and sharing deck objects. Includes calls to spaced repetition algorithm	D	Controller
Storage system for decks (contains definitions of the decks and the associated cards). This includes the data models for the card and deck type.	K	Database
Storage system for user accounts and access from the authentication system (although part of main database, this is listed separately as it is implemented by Django)	K	Profile database
Accesses the user storage system to sign in users and retrieve decks that belong to them. (implemented by Django)	D	Authenticator
Builds HTML templates to display the interface in the user's browser (implemented by Django)	D	Template engine
System forms that allow the user to create and edit forms in Deck Storage	D	Editor
Displays statistics tied to the user account and the card model	D	Dashboard page
Gives the user a form to do a flash card study session	D	Quiz form

II. Association definitions

Concept pair	Association description	Association name
Controller <-> Template engine	Handle requests from the browser, provide HTML. Provide card order and deck information to the template engine. Gather feedback from user - allow access to edit page and flash card practice	Conveys requests
Template engine <-> Dashboard page	Provide HTML to the browser to render the dashboard page	Show dashboard page
Controller <-> deck storage	Gather data from deck storage, optionally calculate study order if user is opted into spaced repetition	Gather flashcards
Editor <-> deck storage	Create and edit flash card fields.	Create, edit flashcards
Editor <-> Template engine	Generates HTML to show user the edit/create card interface	Show create/edit page
Quiz form <-> Template engine	Generates HTML to show user the quiz interface	Show quiz page
Controller <-> Authenticator	The user sign in/out request is sent to the authenticator for authentication. Sign out doesn't need further verification, sign in does	Log on/off
Authenticator <-> Profile database	The authenticator searched the profile database to confirm identity of user	Authenticate

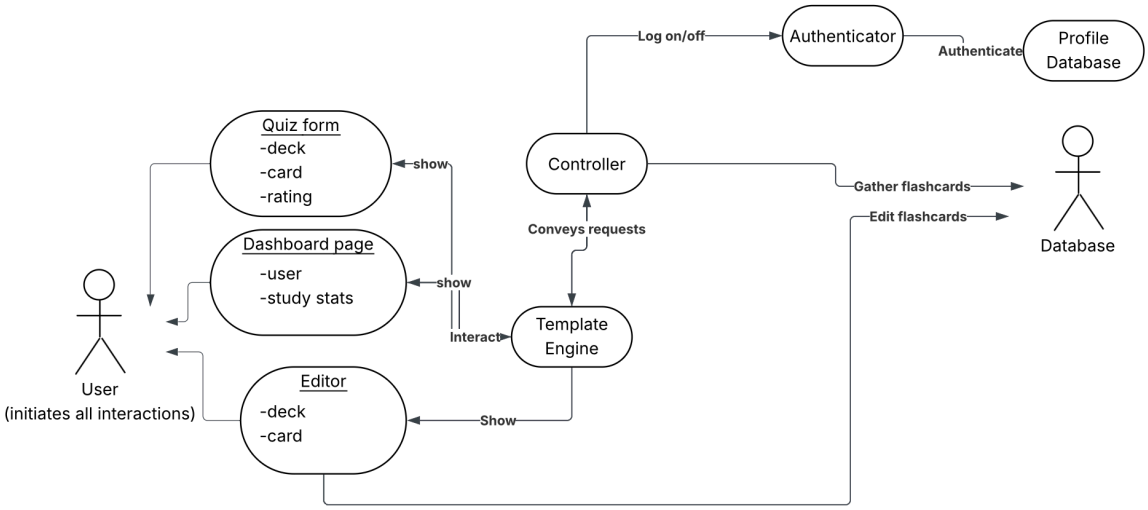
III. Attribute definitions

Concept	Attributes	Attribute Description
Quiz form	deck	The deck that is currently being quizzed on, ordered by the controller before being passed to the template engine
	card	The current card being tested
	rating	A form element allowing the user to rate their card performance. Scale of 1-5
Dashboard page	user	The current user profile
	Study statistics	Statistics aggregate or from the most recent study session
Editor	deck	Currently selected deck
	card	Card currently being modified

IV. Traceability matrix

	Cont-roller	Data-base	Profile Data-base	Authen-ticator	Temp-late engine	Editor	Dash-board Page	Quiz Form
UC1		X			X	X		
UC2	X	X			X			X
UC3		X			X		X	
UC4		X			X	X		
UC5	X		X	X				
UC6	X	X			X			X
UC7			X	X				
UC8	X	X	X					

V. Domain Model Diagram



B. System Operation Contracts

conveysRequests()

- **Cross Reference**: Controller <-> Template Engine
- **Preconditions**:
  - A user request is received by the controller (e.g., view deck, study flashcards, edit card).
- **Postconditions**:
  - The controller routes the request to the appropriate handler.
  - The template engine is called to generate the corresponding HTML interface (dashboard, quiz, or edit form).

showDashboardPage()

- **Cross Reference**: Template Engine <-> Dashboard Page
- **Preconditions**:
  - A valid authenticated user has requested the dashboard.
- **Postconditions**:
  - The template engine renders the dashboard page using the user's statistics and data.

gatherFlashcards()

- **Cross Reference**: Controller <-> Deck Storage
- **Preconditions**:
    - A deck has been selected.
    - The user is authenticated.
- **Postconditions**:
    - The controller retrieves flashcards from deck storage.
    - If applicable, cards are ordered based on spaced repetition logic.

createEditFlashcard()

- **Cross Reference**: Editor <-> Deck Storage
- **Preconditions**:
    - The user is authenticated and has permission to edit the deck.
- **Postconditions**:
    - A new flashcard is created or an existing one is updated in the storage system.

showEditPage()

- **Cross Reference**: Editor <-> Template Engine
- **Preconditions**:
    - The user has navigated to the card editing interface.
- **Postconditions**:
    - The template engine renders the appropriate form for creating or editing a card.

showQuizPage()

- **Cross Reference**: Quiz Form <-> Template Engine
- **Preconditions**:
    - A study session has been initiated by the controller.
- **Postconditions**:
    - The quiz form is rendered and presented via the template engine.

logOnOff()

- **Cross Reference**: Controller <-> Authenticator
- **Preconditions**:
    - A login or logout request has been made.
- **Postconditions**:
    - For login: credentials are passed to the authenticator.
    - For logout: the user's session is terminated.

authenticateUser()

- **Cross Reference**: Authenticator <-> Profile Database

- **Preconditions**:
    - Login credentials have been provided.
- **Postconditions**:
    - The profile database is queried.
    - If valid, a session is created otherwise an error message is returned.

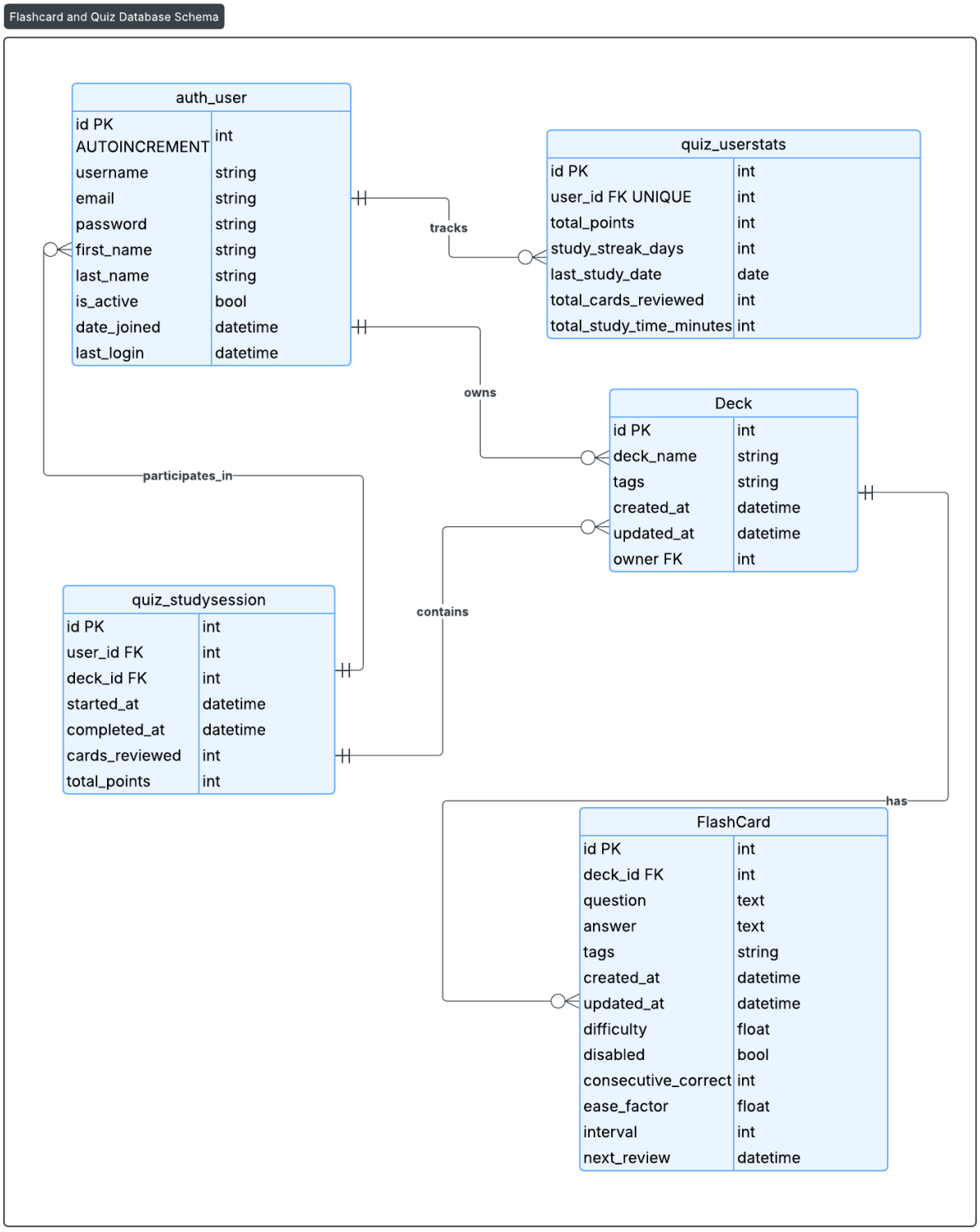
C. Data Model and Persistent Data Storage

Storage Management Strategy
Selected Strategy: Relational Database using SQLite with Django ORM

Justification:

- Users, decks, flashcards, and study sessions are persistent objects that have natural relationships
- Django ORM just works. Between data validation, easy migration management, and easy to understand queries, our team gets a lot of functionality with very little overhead
- SQLite is lightweight and ideal for MVP.

Database Schema:



The above represents the entity relationship diagram of our planned database. Currently we have implemented the deck and the flashcard models. Our current schema output can be seen by visiting the document schema tab, or clicking here

Key Relationships and Constraints:

- One User owns many decks
- One Deck contains many cards
- One User has many StudySessions
- One StudySession has many card reviews

D. Mathematical Model

The spaced repetition model relies on the following variables:
Interval: the number of days until the next review (calculated by algorithm)
EaseFactor: how difficult the card is (starts at 2.5, grow or shrink based on user performance)
Quality: how the user ranks their performance on a card (1 to 5, 3 or lower considered failure, determined by user self-reporting)
Repetitions: how many times in a row the user has gotten the card correct

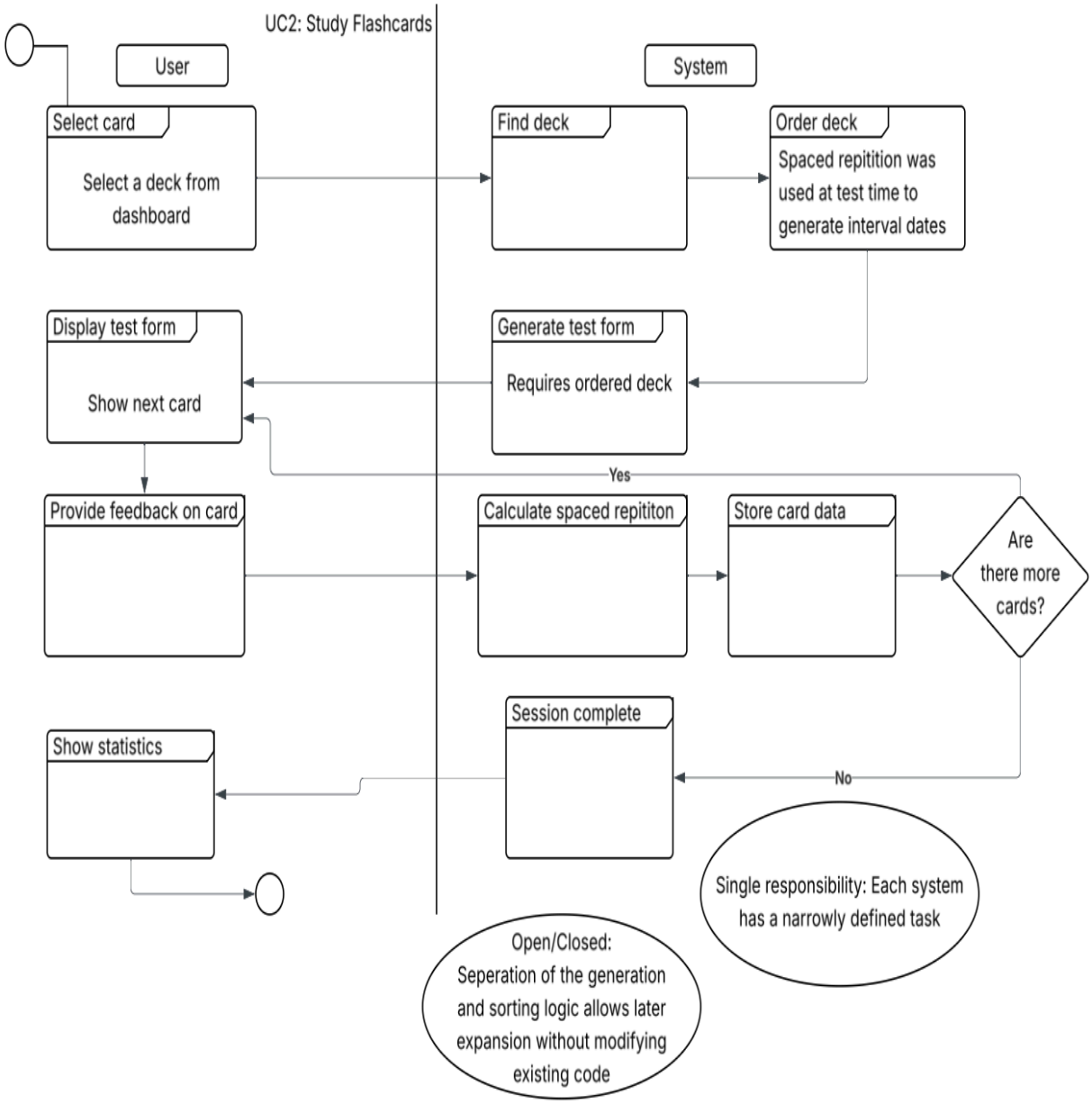
The spaced repetition model follows these rules:

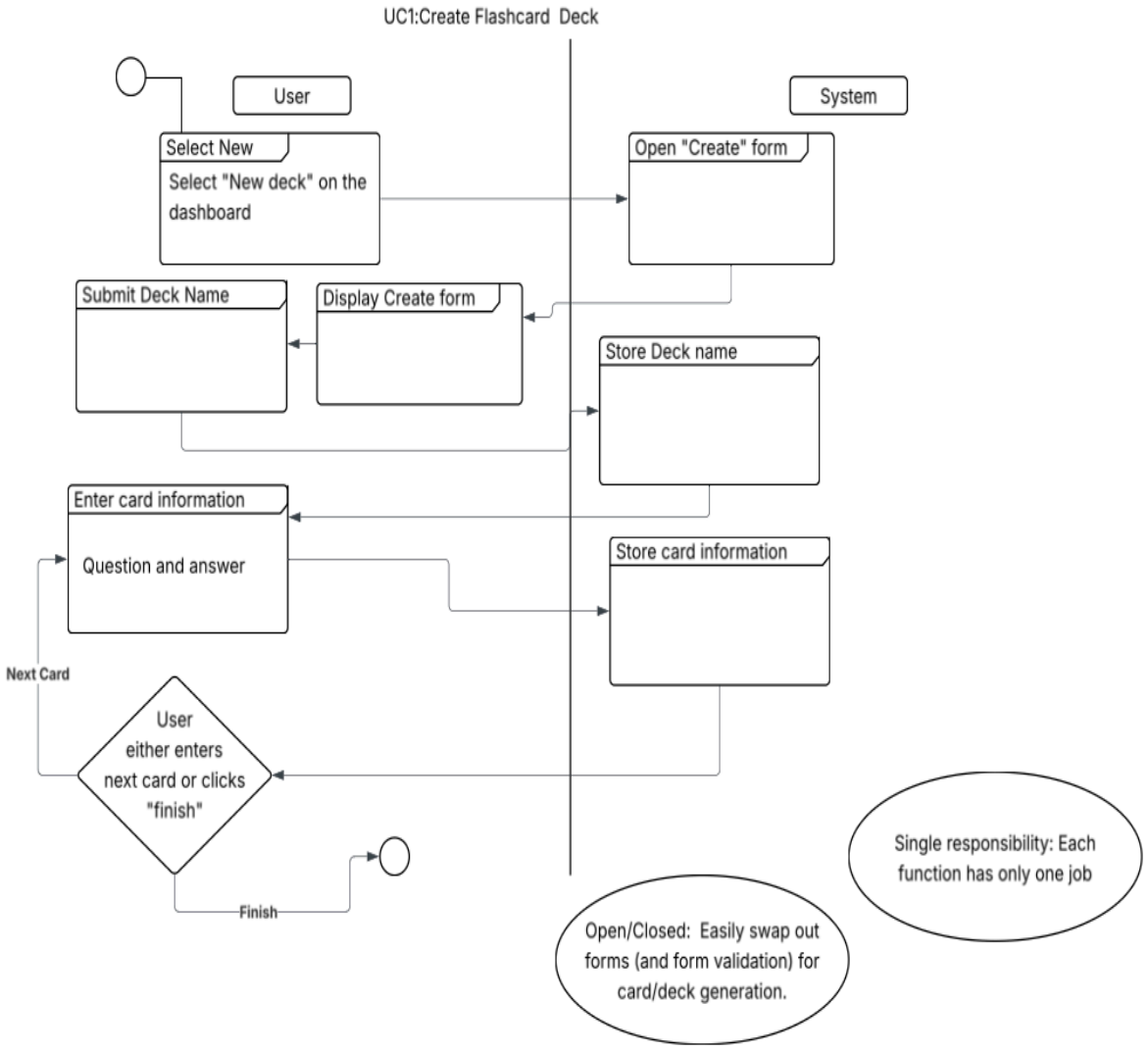
- If user gets card wrong, repetitions = 0
- Repetitions += 1 each time the user attempts a card
- If repetitions = 1, then interval = 1
- If repetitions = 2, then interval = 6
- If repetitions > 2, then interval = Floor(interval * EaseFactor)
- EaseFactor adjusted by EaseFactor += (0.1 - (5 - quality) * (0.08 + (5 - quality) * 0.02)) after each answer on a specific card

---

.

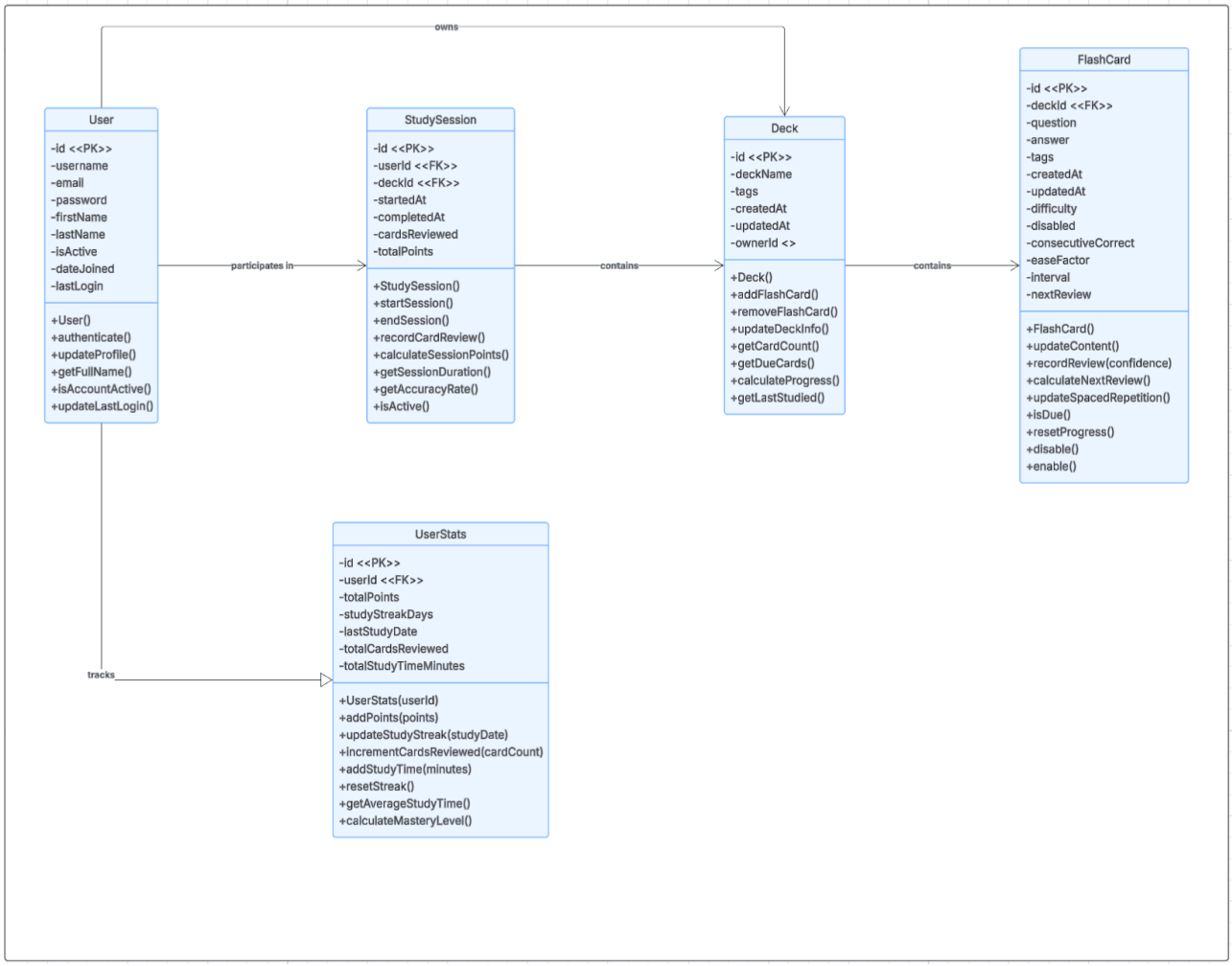
2. Interaction Diagrams




UC1:Create Flashcard Deck
User
System
Select New
Select "New deck" on the dashboard
Open "Create" form
Submit Deck Name
Display Create form
Store Deck name
Enter card information
Question and answer
Store card information
Next Card
User either enters next card or clicks "finish"
Finish
Single responsibility: Each function has only one job
Open/Closed: Easily swap out forms (and form validation) for card/deck generation.

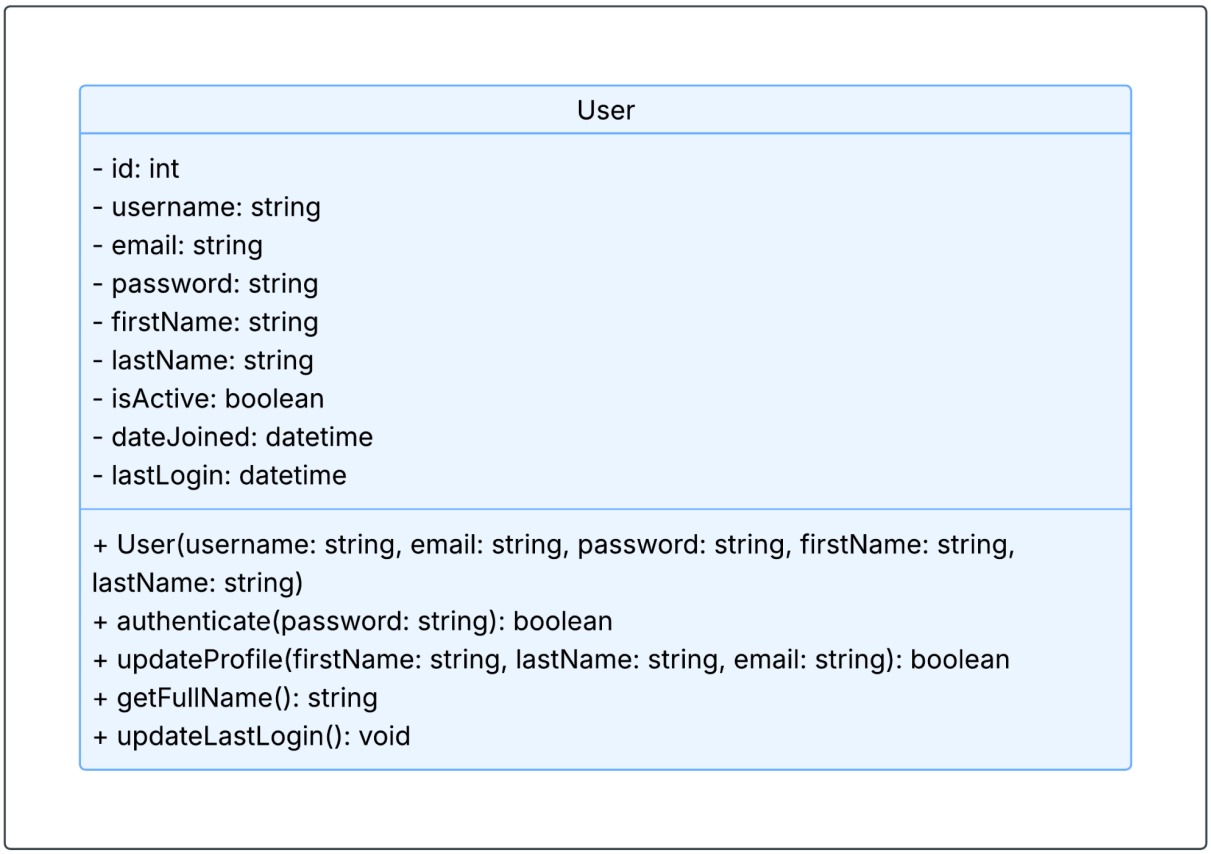
3. Class Diagrams and Interface Specification

a. Class Diagram



b. Data Types and Operation Signature

User Class UML Diagram



Definition: Represents a student or learner who uses the flashcard application to study and and track their progress in learning on a subject

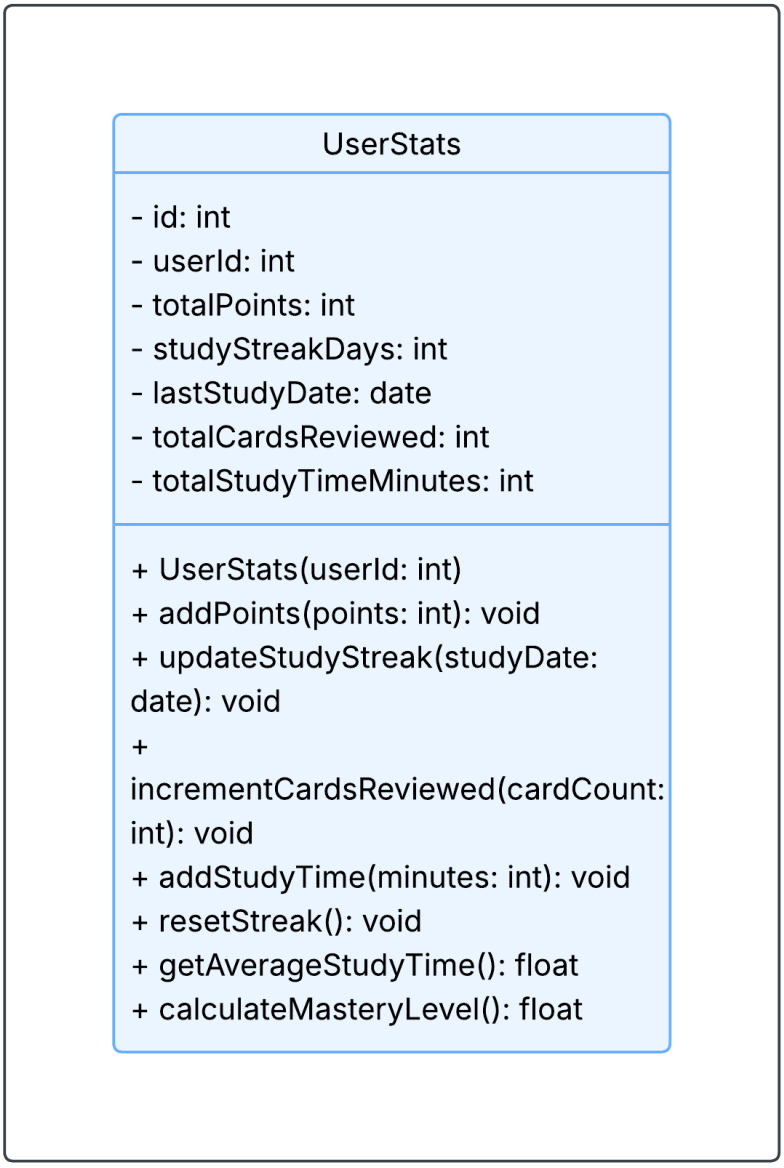
Attribute Definitions:

- id: Unique identifier for the user account
- username: Unique display name chosen by the user
- email: User's email address for authentication and communication
- password: Encrypted password for account security
- firstName: User's given name
- lastName: User's family name
- isActive: Flag indicating whether the account is currently active
- dateJoined: Timestamp when the user created their account
- lastLogin: Timestamp of the user's most recent login

Operation Definitions:

- User(): Constructor that creates a new user account with provided credentials and personal information
- authenticate(): Verifies if the provided password matches the user's stored password for login purposes
- updateProfile(): Modifies the user's personal information including name and email address
- getFullName(): Combines first and last name into a single formatted string for display purposes
- isAccountActive(): Checks and returns whether the user's account is currently active and able to log in
- updateLastLogin(): Records the current timestamp as the user's most recent login time

UserStats Class UML Diagram



Definition: Tracks learning statistics and gamification metrics for each user's study progress.

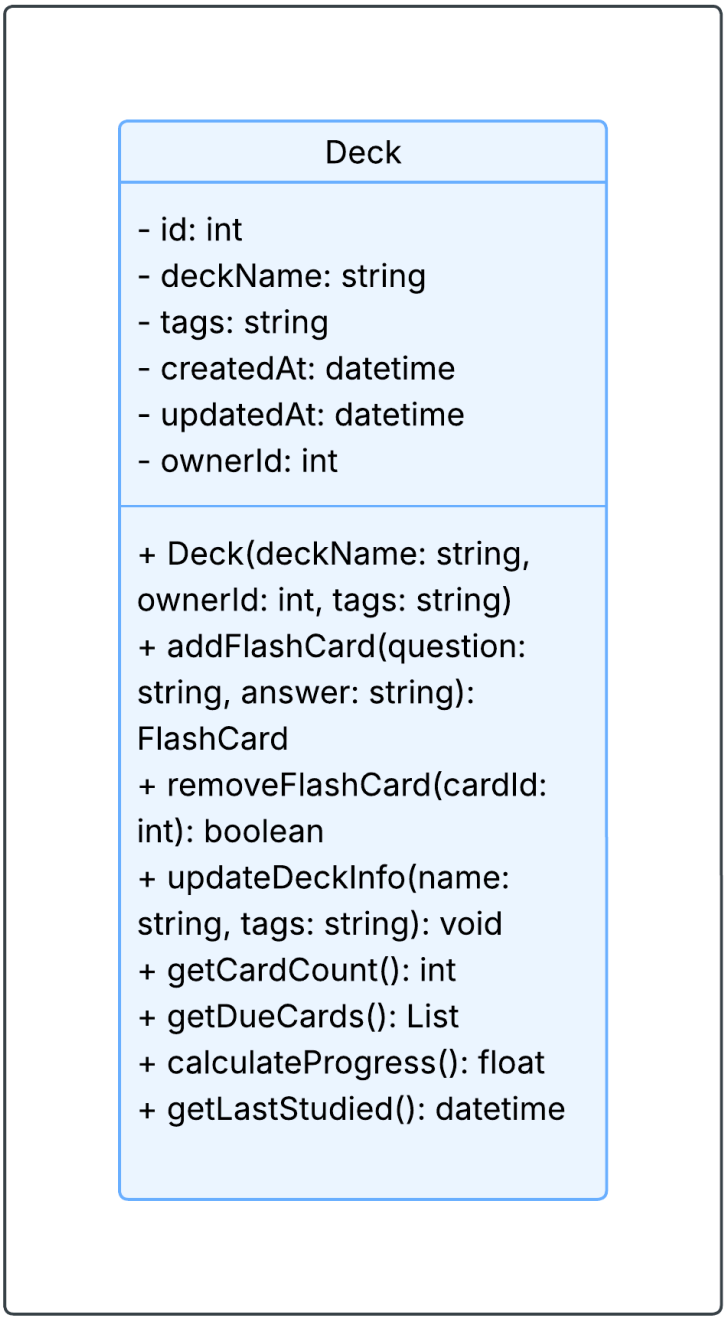
Attribute Definitions:

- id: Unique identifier for the statistics record
- userId: Foreign key linking to the User who owns these statistics
- totalPoints: Cumulative points earned through study activities
- studyStreakDays: Number of consecutive days the user has studied
- lastStudyDate: Most recent date when the user completed a study session
- totalCardsReviewed: Lifetime count of flashcards the user has reviewed
- totalStudyTimeMinutes: Total time spent studying across all sessions

Operation Definitions:

- UserStats(): Constructor that creates a new statistics record for a user with initial values set to zero
- addPoints(): Increases the user's total point balance by the specified amount earned from study activities
- updateStudyStreak(): Calculates and updates the consecutive day streak based on the provided study date
- incrementCardsReviewed(): Adds the specified number of cards to the user's lifetime review count
- addStudyTime(): Increases the total study time by the specified number of minutes from a session
- resetStreak(): Sets the study streak back to zero when the user breaks their consecutive study pattern
- getAverageStudyTime(): Calculates and returns the average time spent per study session
- calculateMasteryLevel(): Computes an overall learning progress percentage based on user's study statistics

Deck Class UML Diagram



Definition: A collection of related flashcards organized around a specific topic or subject area for focused studying.

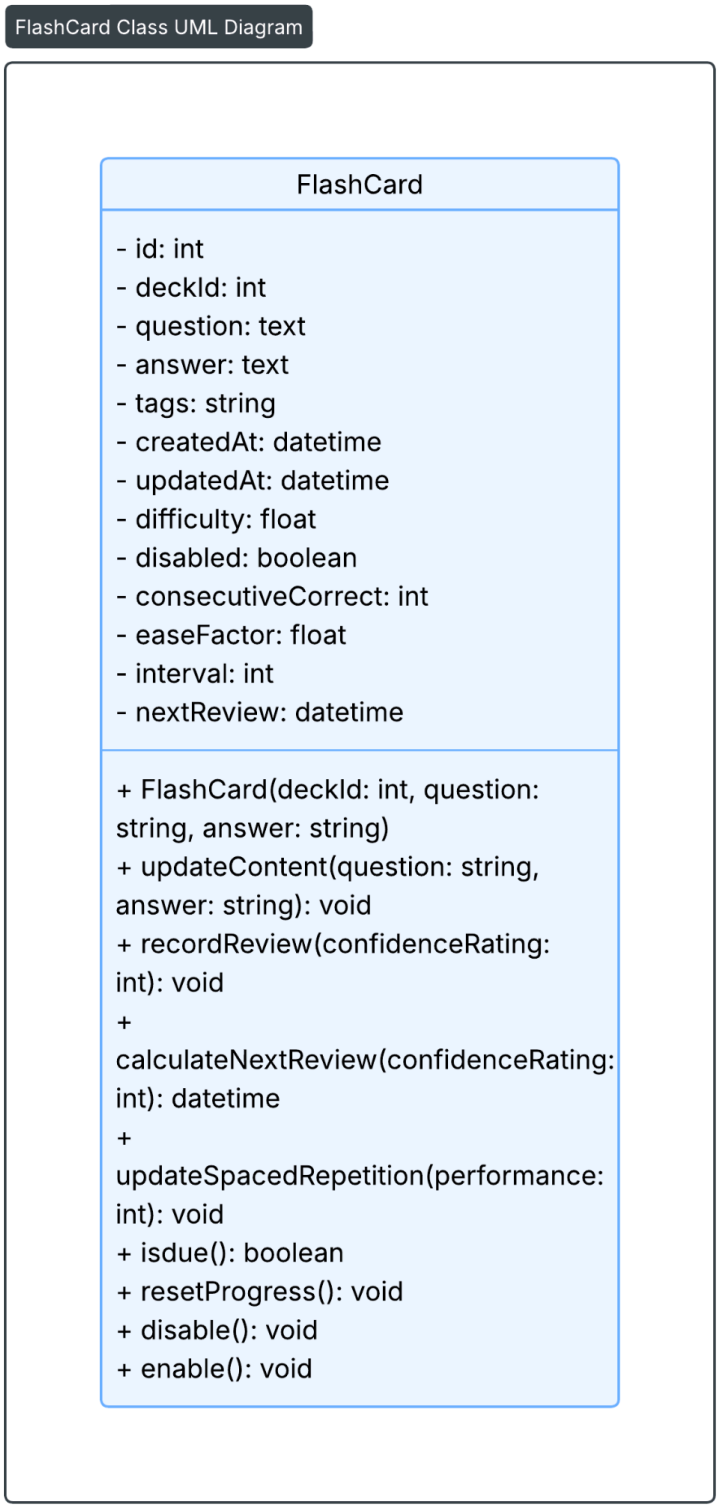
Attribute Definitions:

- id: Unique identifier for the deck
- deckName: Descriptive title for the flashcard collection

- tags: Comma-separated keywords for organizing and searching decks
- createdAt: Timestamp when the deck was first created
- updatedAt: Timestamp of the most recent modification to the deck
- ownerId: Foreign key linking to the User who created this deck

Operation Definitions:

- Deck(): Constructor that creates a new flashcard deck with specified name, owner, and optional tags
- addFlashCard(): Creates and adds a new flashcard to this deck with the provided question and answer
- removeFlashCard(): Deletes the specified flashcard from the deck and returns success status
- updateDeckInfo(): Modifies the deck's name and tags while updating the modification timestamp
- getCardCount(): Returns the total number of flashcards currently in this deck
- getDueCards(): Retrieves a list of all flashcards that are scheduled for review today
- calculateProgress(): Computes the percentage of cards in the deck that have been mastered
- getLastStudied(): Returns the timestamp when any card in this deck was last reviewed



Definition: An individual study item containing a question-answer pair with spaced repetition scheduling data.

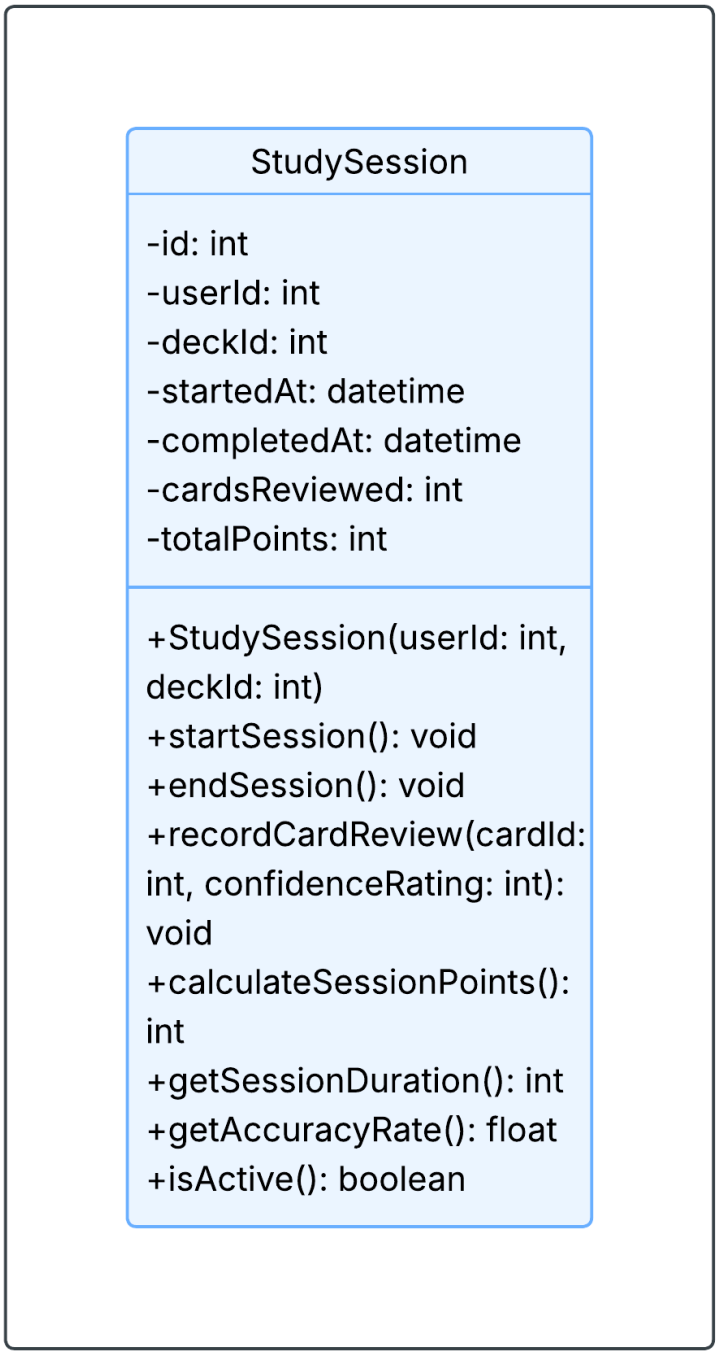
Attribute Definitions:

- id: Unique identifier for the flashcard
- deckId: Foreign key linking to the parent Deck
- question: The prompt or question text displayed to the user
- answer: The correct response or answer text
- tags: Optional keywords for additional organization within the deck
- createdAt: Timestamp when the card was created
- updatedAt: Timestamp of the most recent content modification
- difficulty: Calculated difficulty rating based on user performance (0.0-1.0)
- disabled: Flag to temporarily hide the card from study sessions
- consecutiveCorrect: Count of consecutive correct responses for spacing calculations
- easeFactor: Spaced repetition algorithm parameter affecting review intervals
- interval: Current number of days until next scheduled review
- nextReview: Calculated datetime when card should next appear for study

Operation Definitions:

- FlashCard(): Constructor that creates a new flashcard with question and answer in the specified deck
- updateContent(): Modifies the question and answer text while updating the modification timestamp
- recordReview(): Logs a study attempt with the user's confidence ratiXng and updates spaced repetition data
- calculateNextReview(): Computes when this card should next appear based on spaced repetition algorithm and performance
- updateSpacedRepetition(): Adjusts the ease factor, interval, and consecutive correct count based on user performance
- isDue(): Checks if the current date/time is past the card's scheduled next review time
- resetProgress(): Clears all spaced repetition data to start the learning process over
- disable(): Marks the card as disabled so it won't appear in study sessions
- enable(): Reactivates a disabled card so it can appear in study sessions again

UML Diagram for StudySession Class



Definition: Represents a single study period where a user reviews flashcards, tracking performance and time spent.

Attribute Definitions:

- id: Unique identifier for the study session
- userId: Foreign key linking to the User conducting the session
- deckId: Foreign key linking to the Deck being studied
- startedAt: Timestamp when the study session began
- completedAt: Timestamp when the session was finished (null if ongoing)
- cardsReviewed: Number of flashcards reviewed during this session
- totalPoints: Points earned during this specific session

Operation Definitions:

- StudySession(): Constructor that creates a new study session for a user and deck with start time recorded
- startSession(): Initializes the session by recording the start time and preparing for card reviews
- endSession(): Finalizes the session by recording completion time and calculating final statistics
- recordCardReview(): Logs that a specific card was reviewed with a given confidence rating during this session
- calculateSessionPoints(): Computes the total points earned based on cards reviewed and performance ratings
- getSessionDuration(): Calculates and returns the time elapsed between session start and end in minutes
- getAccuracyRate(): Determines the percentage of cards that were rated as "easy" or "good" during the session
- isActive(): Checks whether the session is currently ongoing (has start time but no completion time)

c. Traceability Matrix (Detailed Concept Transformations)

1. Student to User

Rationale: The original domain identified “students” as primary users, but we want to target a larger audience. The User class includes college students, high school students, and even adult learners.

Key Changes:

- Generic Terminology
- Attributes for personalization and security
- Added isActive flag for account management beyond the original concept

2. FlashCard

Rationale: The question-answer card concept was improved to support spaced repetition learning. The domain concept was simple, but implementing it has lead us to scheduling and performance tracking.

Key Additions:

- Spaced repetition algorithm parameters (easeFactor, interval, nextReview)
- Performance tracking (consecutiveCorrect, difficulty)
- Lifecycle management (disabled, timestamps)

Justification: Simple flashcards are not optimized for learning. The enhanced class implements proven learning strategies

3. Deck

Rationale: "Deck" provides domain-specific language familiar to flashcard users while adding organizational and ownership abilities not present in the initial domain model.

Key Enhancements:

- Ownership tracking for multi-user system
- Metadata for organization (tags) and auditing (timestamps)
- Methods for progress calculation and due card identification

4. Studying Activity to StudySession

Rationale: The domain described "studying" as an activity, but we discovered the need f or trackable sessions. This class is all about how engagement, time and studying fit together

Why This Class Was Added:

- Enables gamification through session-based point systems
- Provides data for learning analytics and progress visualization
- Supports interrupted study sessions with state management

5. Progress/Statistics to UserStats

Rationale: Originally, user progress was attached as part of the student's profile. However, the gamification requirements (streaks, points, analytics) was creating some tightly coupled code that was bloating quickly

Design Decision Benefits:

- Separation of concerns: User identity vs. learning analytics
- Ease of adding new statistical measures

Classes Not Directly Traceable to Original Domain

StudySession

Why Added: While "studying" was a clear domain activity, it wasn't explicitly modeled in the original domain analysis. Requirements analysis revealed that session tracking is good for:

- Meaningful gamification
- Providing progress feedback
- Supporting learning analytics
- Resuming sessions

UserStats

Why Separated: Originally created as part of the User profile,our gamification of studying had a clear conflict of interest with its current location. While having a fat model in Django can be desirable, it became clear that it needed to be split. Separating these concerns:

- Keeps User class focused on identity and authentication
- Allows for complex analytics without bloating the User class
- Enables future enhancements like comparative statistics and leaderboards

Validation Against Use Cases

The class design directly supports the identified use cases:

- UC-1 (Create Deck): Deck and FlashCard classes provide creation and management abilities
- UC-2 (Study Flashcards): StudySession handles the learning process while FlashCard manages spaced repetition
- UC-3 (View Dashboard): UserStats aggregates progress data for display

- UC-4 (Edit Flashcards): FlashCard and Deck classes support content modification

---

4. Algorithms and Data Structures

a. Algorithms

Spaced Repetition Algorithm Implementation

Our system implements a version of the **SM-2 (SuperMemo 2) algorithm** for spaced repetition, which is a well-established algorithm for memory retention through calculated review intervals.

Algorithm Description: The spaced repetition algorithm calculates when a flashcard should next be reviewed based on:

- User performance rating (1-5 scale)
- Previous review history
- Card difficulty (ease factor)

Algorithm Flow:

Input: quality (1-5), repetitions, easeFactor, interval

1. IF quality < 3 THEN

   - repetitions = 0

   - interval = 1

   ELSE

   - repetitions = repetitions + 1

2. IF repetitions = 1 THEN

   - interval = 1

   ELSE IF repetitions = 2 THEN

   - interval = 6

ELSE

- interval = floor(interval * easeFactor)

3. easeFactor = easeFactor + (0.1 - (5 - quality) * (0.08 + (5 - quality) * 0.02))

IF easeFactor < 1.3 THEN easeFactor = 1.3

4. nextReview = currentDate + interval (days)

Output: updated interval, easeFactor, repetitions, nextReview

Implementation Rationale:

- We chose SM-2 because
    - Well-documented and proven effective
    - Simpler to implement and debug in our MVP timeframe

Other Algorithms:

- **Card Selection Algorithm**: Orders due cards by priority (overdue cards first, then by difficulty)
- **Progress Calculation**: Computes mastery percentage based on consecutive correct answers and review frequency

b. Data Structures

Our system uses several data structures, primarily leveraging Django's ORM which abstracts underlying database structures:

Primary Data Structures:

1. Django QuerySets (Database-backed)

- **Purpose**: Efficiently retrieve and filter flashcards, study sessions, and user data
- **Criteria**: Chosen for built-in optimization, lazy evaluation, and database integration
- **Performance**: O(1) for indexed lookups, O(log n) for sorted queries

2. Python Lists and Dictionaries

- **Purpose**: Temporary storage for study session data and algorithm calculations
- **Example**: due_cards = list(Deck.objects.filter(nextReview__lte=today))
- **Criteria**: Native Python structures for simplicity and readability

3. Session Storage (Dictionary-based)

- **Purpose**: Track current study session state without database writes on every interaction
- **Implementation**: Django session framework using signed cookies
- **Criteria**: Balances performance (no DB writes) with data persistence

Data Structure Decision Criteria:

Requirement	**Structure Choice**	**Rationale**
Persistent storage	Django Models/QuerySets	Built-in ORM, automatic SQL optimization
Temporary calculations	Python lists/dicts	Simple, readable, sufficient performance
User session state	Session storage	Stateless HTTP handling, automatic cleanup
Card ordering	Python lists with custom sorting	Flexibility for algorithm experimentation

c. Concurrency

Current Implementation: Single-threaded Django application
Rationale: For our MVP with expected <100 concurrent users, single-threaded processing is more than enough

---

5. User Interface Design and Implementation

Implementation Changes from Initial Mockups

We haven't made many changes to the implementation of the UI as the goal was to keep our MVP as simple yet engaging as possible.

Ease of Use Implementation:

- **Minimal Clicks**: Study session requires maximum 2 clicks per card (show answer then rate)
- **Visual Hierarchy**: Primary actions use high-contrast colors and larger fonts
- **Error Prevention**: Form validation with clear error messages
- **Consistency**: Same interaction patterns across all forms (create deck, add cards, study)

Specific User Effort Optimizations:

- Deck creation: Auto-focus on first form field
- Navigation: Breadcrumb navigation shows current location

---

6. Design of Tests

a. Unit Testing Strategy

Core Model Tests (test_models.py):

FlashCard Model Tests:

```
class FlashCardTestCase(TestCase):

    def test_calculate_next_review_easy_rating(self):

        Test that rating 5 increases interval significantly



    def test_is_due_functionality(self):

        Test due date calculation accuracy
```

Deck Model Tests:

```
class DeckTestCase(TestCase):

    def test_get_due_cards(self):

        Verify correct cards returned for study



    def test_card_count(self):

        Ensure accurate card counting
```

User Model Tests:

```
class UserTestCase(TestCase):
```

def test_user_stats_calculation(self):

Verify statistics aggregation

Algorithm-Specific Tests:

Spaced Repetition Algorithm Tests:

- Test edge cases (minimum/maximum ease factors)
- Test algorithm with various quality ratings (1-5)
- Validate that failed cards reset appropriately

b. Test Coverage Analysis

Target Coverage: 80% minimum for core functionality

Coverage Priorities:

1. **Critical (90%+ coverage)**: Deck/card CRUD operations, study session logic, spaced repetition algorithm
2. **Important Features (80%+ coverage)**: User authentication, statistics calculation
3. **Supporting Features (60%+ coverage)**: UI form validation, error handling

Coverage Tools: Django's built-in coverage tools with coverage.py

c. Integration Testing Strategy

Testing Phases:

Phase 1: Component Integration

- Test Model-View integration
- Verify form submission workflows
- Test template rendering with dynamic data

Phase 2: User Workflow Integration

1. User logs in
2. Selects deck
3. Studies all due cards
4. Completes session

5. Views updated statistics

Verify: all database updates, session state management

-

Non-Functional Requirements Testing:

Performance Testing:

- **Load Testing**: Simulate 10 concurrent users studying
- **Response Time**: Verify <1 second page loads

Usability Testing:

- **5-Minute Test**: New users can create deck and start studying within 5 minutes
- **Mobile Usability**: Touch targets meet 44px minimum size
- **Accessibility**: Verify keyboard navigation and screen reader compatibility

Success Criteria for Testing:

- All unit tests pass with >80% coverage
- Integration tests demonstrate complete user workflows
- Performance tests show < 1 second response times

---

7. Project Management

a. Merging Contributions from Individual Team Members

We are an equally distributed team that shares all work and credit equally. For each section of each part of this report, each team member took a section. For example in Part 2, Dan Seip completed the interaction diagrams, Julie Rogers completed the class diagrams, Daniel Terreros did data types and operation signatures, and all three of us convened to contribute to cover the Traceability Matrix section. This pattern of shared load has been repeated throughout the session on this project. The biggest issue has always been time given the short sprints in

between due dates. The way we have overcome this so far is constant communication in our discord and an abundance of encouragement between members.

b. Project Coordination and Progress Report

Use Case 1 is implemented and we have the ability to create a deck and add cards to it. We are very close to having authentication implemented thus allowing the user to have a profile. We will start implementing the algorithm this week.

Project Timeline Overview

Project Duration: 5 weeks (June 12 - July 18, 2025)
Demo Dates: First Demo (July 8), Final Demo (July 18)

Sprint Planning

Sprint 1: Foundation Setup (June 12-19, 2025)

Goal: Establish working development environment and basic user authentication

Week 1 Deliverables:

- Django project structure with Docker setup
- User authentication system (login/logout/register)
- Basic database models defined
- Initial HTML templates and styling framework

Daniel Seip (Algorithm/Backend):

- Research and implement spaced repetition algorithm
- Design database schema for study tracking
- Implement user authentication system
- Create basic URL routing structure

Daniel Terreros (Backend/DevOps):

- Configure Docker development environment
- Set up Django project with proper settings
- Design database schema for study tracking
- Create core model relationships
- Set up database models and migrations

Julia Rogers (Frontend Lead):

- Design responsive CSS framework
- Create base HTML templates
- Implement authentication UI
- Set up static file handling

Sprint 1 Demo: Working login system with basic UI

Current Status: Functional Django app showing base and index templates for the quiz.

Next Steps: Git meeting to go over branching and commit rules to ensure a clean codebase

---

Sprint 2: Core Functionality (June 19-26, 2025)

Goal: Implement deck creation and basic study functionality

Week 2 Deliverables:

- Flashcard deck creation and editing
- Basic study interface (show question/answer)
- Simple confidence rating system
- Dashboard showing user's decks

Daniel Seip (Algorithm/Backend):

- Create study session tracking
- Build confidence rating processing
- Add basic statistics calculations

Daniel Terreros (Backend/DevOps):

- Implement deck creation views and forms
- Create flashcard CRUD operations
- Build basic study session logic
- Add deck management functionality

Julia Rogers (Frontend Lead):

- Create deck creation UI
- Design and implement study interface
- Build dashboard layout
- Add interactive elements and animations

Sprint 2 Demo: Can create decks and study with basic spaced repetition

---

Sprint 3: Polish and Gamification (June 26-July 3, 2025)

Goal: Add motivational features and improve user experience

Week 3 Deliverables:

- Point system and study streaks
- Improved dashboard with statistics
- Session completion feedback
- Mobile responsiveness testing

Daniel Seip (Algorithm/Backend):

- Fine-tune spaced repetition algorithm
- Add advanced statistics calculations
- Implement session management
- Create data backup and recovery

Daniel Terreros (Backend/DevOps):

- Implement points and streak tracking
- Create session statistics and reporting
- Add data validation and error handling
- Performance optimization

Julia Rogers (Frontend Lead):

- Design gamification UI elements
- Improve mobile responsiveness
- Add visual feedback and animations
- Conduct usability testing

Sprint 3 Demo: Engaging study experience with motivational features

---

Sprint 4: Testing and Demo Preparation (July 3-8, 2025)

Goal: Prepare for first demo with thorough testing

Week 4 Deliverables:

- Comprehensive testing (unit, integration, user)
- Bug fixes and stability improvements
- Demo preparation and presentation materials
- Documentation updates

All Team Members:

- **Testing:** Unit tests, integration tests, user acceptance testing
- **Bug Fixes:** Address issues found during testing
- **Demo Prep:** Create demo script, test scenarios, presentation
- **Documentation:** Update README, user guide, code comments

First Demo (July 8): Present working MVP to class

Sprint 5: Final Polish and Presentation (July 8-18, 2025)

Goal: Final refinements and presentation preparation

Week 5 Deliverables:

- Address feedback from first demo
- Final bug fixes and optimizations
- Complete documentation
- Final presentation preparation

Stretch Goals (if time permits):

- Simple leaderboard functionality
- Deck import/export feature
- Enhanced mobile app experience
- Social sharing capabilities
- Spaced repetition algorithm implementation
-

All Team Members:

- **Feedback Integration:** Implement suggestions from first demo
- **Final Polish:** UI improvements, performance tuning
- **Documentation:** Complete technical documentation
- **Presentation:** Prepare final demo and presentation materials

Final Demo (July 18): Present completed project

Team Responsibilities and Coordination

Role Definitions

Daniel Seip - Algorithm/Backend & Project Manager:

- Study session logic and data processing
- Performance optimization and monitoring
- Project coordination and timeline management
- Code review and quality assurance

Daniel Terreros - Algorithm Specialist & DevOps:

-
- Overall project architecture and Django setup
- User authentication and account management
- Database design and model relationships
- Deployment and environment management
- Technical problem-solving and debugging

Julia Rogers - Frontend Lead & UX Designer:

- User interface design and implementation
- Responsive web development
- User experience optimization
- Usability testing and feedback integration
- Visual design and branding

Communication and Coordination

Daily Standups: 15-minute check-ins via Discord/Slack (Monday, Wednesday, Friday) **Sprint Reviews:** End-of-week demos and retrospectives **Code Reviews:** All commits reviewed by at least one other team member **Documentation:** Shared Google Drive for requirements, design decisions, and meeting notes

Risk Mitigation

Technical Risks:

- **Docker setup issues:** Pair programming sessions for environment setup
- **Algorithm complexity:** Start with simple implementation, iterate
- **Integration problems:** Daily integration testing

Timeline Risks:

- **Scope creep:** Strict adherence to MVP feature list
- **Individual delays:** Cross-training on critical components
- **External dependencies:** Buffer time built into each sprint

Revisions

- Changed chart sizing to fit pages
- Added algorithm section
- Added unit testing outline
- Updated project management

References

- Django Software Foundation. (2025). *Django Documentation*. Retrieved from https://docs.djangoproject.com/
- Django QuerySet (2025) *Django Documentation* *https://docs.djangoproject.com/en/5.2/ref/models/querysets/*
- SM-2 Algorithm *Wikipedia https://en.wikipedia.org/wiki/SuperMemo*
- Quizlet. (2025). *Quizlet Help Center*. Retrieved from https://help.quizlet.com/
- Anki. (2025). *Anki Manual*. Retrieved from https://docs.ankiweb.net/