Authentication and Flashcard App Flow Sequences

Below are the mermaid diagrams showing the flow sequences for user registration, login, and flashcard functionality.

User Registration Flow

```
sequenceDiagram
    actor User
    participant Client
    participant AuthController
    participant UserService
    participant EmailService
    participant Database

    User->>Client: Enter registration details
    Client->>AuthController: POST /api/auth/signup
    AuthController->>UserService: registerUser(username, email, password)
    UserService->>Database: Check if username/email exists
    Database-->>UserService: Return result

    alt Username or Email already exists
        UserService-->>AuthController: Return error
        AuthController-->>Client: 400 Bad Request
        Client-->>User: Display error message
    else Registration successful
        UserService->>Database: Create new user
        Database-->>UserService: Return user data
        UserService->>EmailService: generateVerificationToken(userId)
        EmailService->>Database: Store verification token
        EmailService->>EmailService: Send verification email
        UserService-->>AuthController: Return success
        AuthController-->>Client: 201 Created with userId
        Client-->>User: Display success message & prompt to check email
    end
```

Email Verification Flow

```
sequenceDiagram
    actor User
    participant EmailClient
    participant Client
    participant AuthController
    participant UserService
    participant Database

    EmailClient->>User: Receive verification email
```

```
    User->>EmailClient: Click verification link
    EmailClient->>Client: Open app with token
    Client->>AuthController: POST /api/auth/verify-email
    AuthController->>UserService: verifyEmail(token)
    UserService->>Database: Find token & validate

    alt Token invalid or expired
        Database-->>UserService: Token not found/expired
        UserService-->>AuthController: Return error
        AuthController-->>Client: 400 Bad Request
        Client-->>User: Display error message
    else Token valid
        Database-->>UserService: Return token data
        UserService->>Database: Update user.isVerified = true
        UserService->>Database: Delete used token
        Database-->>UserService: Confirmation
        UserService-->>AuthController: Return success
        AuthController-->>Client: 200 OK
        Client-->>User: Display success & redirect to login
    end
```

Login Flow

```
sequenceDiagram
    actor User
    participant Client
    participant AuthController
    participant UserService
    participant JWTService
    participant Database

    User->>Client: Enter login credentials
    Client->>AuthController: POST /api/auth/login
    AuthController->>UserService: authenticateUser(username, password)
    UserService->>Database: Find user
    Database-->>UserService: Return user data

    alt User not found
        UserService-->>AuthController: User not found
        AuthController-->>Client: 401 Unauthorized
        Client-->>User: Display error message
    else Password incorrect
        UserService->>UserService: Compare password hash
        UserService-->>AuthController: Invalid credentials
        AuthController-->>Client: 401 Unauthorized
        Client-->>User: Display error message
    else Email not verified
        UserService-->>AuthController: Email not verified
        AuthController-->>Client: 401 Unauthorized
        Client-->>User: Prompt to verify email
    else Authentication successful
```

```
        UserService->>JWTService: generateTokens(userId, role)
        JWTService->>Database: Store refresh token
        JWTService-->>UserService: Return tokens
        UserService-->>AuthController: Return user data & tokens
        AuthController-->>Client: 200 OK with user data & tokens
        Client->>Client: Store tokens securely
        Client-->>User: Redirect to dashboard
    end
```

Token Refresh Flow

```
sequenceDiagram
    actor User
    participant Client
    participant AuthController
    participant JWTService
    participant Database

    Client->>Client: Detect expired access token
    Client->>AuthController: POST /api/auth/refresh
    AuthController->>JWTService: refreshToken(refreshToken)
    JWTService->>Database: Validate refresh token

    alt Refresh token invalid/expired
        Database-->>JWTService: Invalid token
        JWTService-->>AuthController: Invalid refresh token
        AuthController-->>Client: 401 Unauthorized
        Client-->>User: Redirect to login
    else Refresh token valid
        Database-->>JWTService: Token valid
        JWTService->>JWTService: Generate new access token
        JWTService-->>AuthController: Return new access token
        AuthController-->>Client: 200 OK with new access token
        Client->>Client: Update stored access token
        Client-->>User: Continue with original request
    end
```

Create Flashcard Deck Flow

```
sequenceDiagram
    actor User
    participant Client
    participant DeckController
    participant DeckService
    participant AuthService
    participant Database

    User->>Client: Enter deck details
    Client->>DeckController: POST /api/decks
```

```
    DeckController->>AuthService: validateToken(accessToken)

    alt Token invalid
        AuthService-->>DeckController: Unauthorized
        DeckController-->>Client: 401 Unauthorized
        Client-->>User: Redirect to login
    else Token valid
        AuthService-->>DeckController: User authenticated
        DeckController->>DeckService: createDeck(name, description, userId)
        DeckService->>Database: Save new deck
        Database-->>DeckService: Return deck data
        DeckService-->>DeckController: Return created deck
        DeckController-->>Client: 201 Created with deck data
        Client-->>User: Display success & show new deck
    end
```

Create Flashcard Flow

```
sequenceDiagram
    actor User
    participant Client
    participant CardController
    participant CardService
    participant DeckService
    participant AuthService
    participant Database

    User->>Client: Enter card front/back content
    Client->>CardController: POST /api/decks/{deckId}/cards
    CardController->>AuthService: validateToken(accessToken)

    alt Token invalid
        AuthService-->>CardController: Unauthorized
        CardController-->>Client: 401 Unauthorized
        Client-->>User: Redirect to login
    else Token valid
        AuthService-->>CardController: User authenticated
        CardController->>DeckService: checkDeckOwnership(deckId, userId)

        alt User doesn't own deck
            DeckService-->>CardController: Forbidden
            CardController-->>Client: 403 Forbidden
            Client-->>User: Display error message
        else User owns deck
            DeckService-->>CardController: Deck verified
            CardController->>CardService: createCard(deckId, front, back,
notes)
            CardService->>Database: Save new card
            Database-->>CardService: Return card data
            CardService-->>CardController: Return created card
            CardController-->>Client: 201 Created with card data
```

```
            Client-->>User: Display success & show new card
        end
    end
```

Study Session Flow

```
sequenceDiagram
    actor User
    participant Client
    participant ReviewController
    participant ReviewService
    participant CardService
    participant AuthService
    participant Database

    User->>Client: Start study session for deck
    Client->>ReviewController: GET /api/decks/{deckId}/review
    ReviewController->>AuthService: validateToken(accessToken)

    alt Token invalid
        AuthService-->>ReviewController: Unauthorized
        ReviewController-->>Client: 401 Unauthorized
        Client-->>User: Redirect to login
    else Token valid
        AuthService-->>ReviewController: User authenticated
        ReviewController->>ReviewService: getDueCards(deckId, userId,
limit)
        ReviewService->>Database: Retrieve due cards
        Database-->>ReviewService: Return cards
        ReviewService->>ReviewService: Create session ID
        ReviewService-->>ReviewController: Return cards and sessionId
        ReviewController-->>Client: 200 OK with cards and sessionId
        Client-->>User: Display first flashcard

        loop For each card
            User->>Client: Review card and submit result
            Client->>ReviewController: POST /api/review
            ReviewController->>ReviewService: recordReview(cardId,
sessionId, result, timeSpent)
            ReviewService->>CardService: calculateNextReview(card, result)
            CardService-->>ReviewService: Return new review date &
difficulty
            ReviewService->>Database: Update card data
            Database-->>ReviewService: Confirmation
            ReviewService-->>ReviewController: Return updated card data
            ReviewController-->>Client: 200 OK with next review date
            Client-->>User: Show next card or session summary
        end

        User->>Client: Complete session
        Client->>ReviewController: POST /api/review/complete
```

```
        ReviewController->>ReviewService: completeSession(sessionId,
cardsReviewed, totalTime)
        ReviewService->>Database: Save session statistics
        Database-->>ReviewService: Confirmation
        ReviewService-->>ReviewController: Return session stats
        ReviewController-->>Client: 200 OK with stats summary
        Client-->>User: Display session results
    end
```

Import Cards Flow

```
sequenceDiagram
    actor User
    participant Client
    participant ImportController
    participant ImportService
    participant DeckService
    participant AuthService
    participant Database

    User->>Client: Upload CSV file
    Client->>ImportController: POST /api/decks/{deckId}/import/csv
    ImportController->>AuthService: validateToken(accessToken)

    alt Token invalid
        AuthService-->>ImportController: Unauthorized
        ImportController-->>Client: 401 Unauthorized
        Client-->>User: Redirect to login
    else Token valid
        AuthService-->>ImportController: User authenticated
        ImportController->>DeckService: checkDeckOwnership(deckId, userId)

        alt User doesn't own deck
            DeckService-->>ImportController: Forbidden
            ImportController-->>Client: 403 Forbidden
            Client-->>User: Display error message
        else User owns deck
            DeckService-->>ImportController: Deck verified
            ImportController->>ImportService: importCardsFromCSV(file,
deckId)
            ImportService->>ImportService: Parse CSV file
            ImportService->>Database: Batch insert cards
            Database-->>ImportService: Return results
            ImportService-->>ImportController: Return import statistics
            ImportController-->>Client: 200 OK with import stats
            Client-->>User: Display import results
        end
    end
```

View Statistics Flow

```
sequenceDiagram
    actor User
    participant Client
    participant StatsController
    participant StatsService
    participant AuthService
    participant Database

    User->>Client: Request statistics
    Client->>StatsController: GET /api/stats
    StatsController->>AuthService: validateToken(accessToken)

    alt Token invalid
        AuthService-->>StatsController: Unauthorized
        StatsController-->>Client: 401 Unauthorized
        Client-->>User: Redirect to login
    else Token valid
        AuthService-->>StatsController: User authenticated
        StatsController->>StatsService: getUserStats(userId)
        StatsService->>Database: Retrieve user study data
        Database-->>StatsService: Return aggregated stats
        StatsService-->>StatsController: Return formatted statistics
        StatsController-->>Client: 200 OK with statistics
        Client-->>User: Display statistics dashboard
    end
```

Password Reset Flow

```
sequenceDiagram
    actor User
    participant Client
    participant AuthController
    participant UserService
    participant EmailService
    participant Database

    User->>Client: Request password reset
    Client->>AuthController: POST /api/auth/forgot-password
    AuthController->>UserService: initiatePasswordReset(email)
    UserService->>Database: Find user by email

    alt User not found
        Database-->>UserService: User not found
        UserService-->>AuthController: Email sent message (security)
        AuthController-->>Client: 200 OK (same response for security)
        Client-->>User: Check email message
    else User found
        Database-->>UserService: Return user data
        UserService->>EmailService: generateResetToken(userId)
        EmailService->>Database: Store reset token
```

```
        EmailService->>EmailService: Send reset email
        UserService-->>AuthController: Email sent message
        AuthController-->>Client: 200 OK
        Client-->>User: Check email message

        User->>EmailClient: Receive and open reset email
        User->>Client: Enter new password with token
        Client->>AuthController: POST /api/auth/reset-password
        AuthController->>UserService: resetPassword(token, newPassword)
        UserService->>Database: Validate token

        alt Token invalid/expired
            Database-->>UserService: Invalid token
            UserService-->>AuthController: Reset failed
            AuthController-->>Client: 400 Bad Request
            Client-->>User: Display error message
        else Token valid
            Database-->>UserService: Token valid
            UserService->>UserService: Hash new password
            UserService->>Database: Update password
            UserService->>Database: Delete used token
            Database-->>UserService: Confirmation
            UserService-->>AuthController: Reset successful
            AuthController-->>Client: 200 OK
            Client-->>User: Display success & redirect to login
        end
    end
```