Coverage for **app/services/rating_service.py**: 98%

56 statements | 55 run | 1 missing | 0 excluded



```python
# app/services/rating_service.py
from pathlib import Path
from statistics import mean
from app.models.rating import Rating
from app.utils.data_manager import CSVRepository
from app.schemas.rating import RatingRead, AvgRatingRead


class RatingService:
    def __init__(self):
        self.repo = CSVRepository()
        self.ratings_path = str(Path(__file__).resolve().parents[1] / "data" / "Ratings.csv")
        self.fields = ["UserID", "ISBN", "Book-Rating"]

    def __read_rows(self):
        return self.repo.read_all(self.ratings_path)

    def __write_rows(self, rows):
        self.repo.write_all(self.ratings_path, self.fields, rows)

    '''
    This method creates or updates a rating for a given user and book.
    Args:
        user_id (int): The ID of the user creating the rating.
        isbn (str): The ISBN of the book being rated.
        rating_value (int): The rating value to be assigned.
    Returns:
        RatingRead: The created or updated rating.
    '''

    def create_rating(self, user_id: int, isbn: str, rating_value: int) -> RatingRead:
        rows = self.__read_rows()
        uid = str(user_id)
        updated = False
```

```
35
36          for r in rows:
37              if r["UserID"] == uid and r["ISBN"] == isbn:
38                  r["Book-Rating"] = str(rating_value)
39                  updated = True
40                  break
41
42          if updated:
43              self.__write_rows(rows)
44          else:
45              rating = Rating(user_id=user_id, isbn=isbn, rating=rating_value)
46              self.repo.append_row(self.ratings_path, self.fields, rating.to_csv_dict())
47
48          return RatingRead(user_id=user_id, isbn=isbn, rating=rating_value)
49
50      '''
51      This method retrieves a user's rating for a specific book.
52      Args:
53          user_id (int): The ID of the user.
54          isbn (str): The ISBN of the book.
55      Returns:
56          RatingRead | None: The user's rating for the book, or None if not found.
57
58      '''
59      def get_user_rating(self, user_id: int, isbn: str) -> RatingRead | None:
60          for row in self.__read_rows():
61              if row["UserID"] == str(user_id) and row["ISBN"] == isbn:
62                  return RatingRead(user_id=int(row["UserID"]), isbn=row["ISBN"], rating=int(row["Book-Rating"]))
63          return None
64
65      '''
66      This method retrieves all ratings in the system.
67      Returns:
68          list[RatingRead]: A list of all ratings.
69      '''
70      def get_all_ratings(self) -> list[RatingRead]:
71          return [
72              RatingRead(user_id=int(r["UserID"]), isbn=r["ISBN"], rating=int(r["Book-Rating"]))
73              for r in self.__read_rows()
74          ]
75
```

```python
    '''
    This method deletes a user's rating for a specific book.
    Args:
        user_id (int): The ID of the user.
        isbn (str): The ISBN of the book.
    Returns:
        bool: True if the rating was deleted, False if not found.
    '''
    def delete_rating(self, user_id: int, isbn: str) -> bool:
        rows = self.__read_rows()
        filtered = [r for r in rows if not (r["UserID"] == str(user_id) and r["ISBN"] == isbn)]
        if len(filtered) == len(rows):
            return False
        self.__write_rows(filtered)
        return True

    '''
    This method retrieves all ratings for a specific book.
    Args:
        isbn (str): The ISBN of the book.
    Returns:
        list[RatingRead]: A list of ratings for the specified book.
    '''
    def get_ratings_by_isbn(self, isbn: str) -> list[RatingRead]:
        return [
            RatingRead(user_id=int(r["UserID"]), isbn=r["ISBN"], rating=int(r["Book-Rating"]))
            for r in self.__read_rows() if r["ISBN"] == isbn
        ]

    '''
    This method calculates the average rating for a specific book.
    Args:
        isbn (str): The ISBN of the book.
    Returns:
        AvgRatingRead: The average rating and count of ratings for the book.
    '''
    def get_avg_rating(self, isbn: str) -> AvgRatingRead:
        book_ratings = self.get_ratings_by_isbn(isbn)
        if not book_ratings:
            return AvgRatingRead(isbn=isbn, avg_rating=0.0, count=0)
        avg = mean(r.rating for r in book_ratings)
```

```
117         return AvgRatingRead(isbn=isbn, avg_rating=round(avg, 2), count=len(book_ratings))
```