Coverage for **app/services/readinglist_service.py**: 85%

114 statements | 97 run | 17 missing | 0 excluded


```python
from pathlib import Path
from app.models.request import Request
from app.schemas.readinglist import ReadingListCreate, ReadingListDetail, ReadingListSummary
from app.models.readinglist import ReadingList
from app.utils.data_manager import CSVRepository

class ReadingListService:
    def __init__(self):
        self.repo = CSVRepository()
        self.path = Path(__file__).resolve().parents[1] / "data" / "ReadingLists.csv"
        self.fields = ["ListID", "UserID", "Name", "ISBNs", "IsPublic"]

    def __generate_next_id(self) -> int:
        rows = self.repo.read_all(self.path)
        if not rows:
            return 1
        ids = [int(r["ListID"]) for r in rows if r["ListID"].isdigit()]
        return max(ids, default=0) + 1

    def __already_added(self, list_id: int, isbn: str) -> bool:
        rows = self.repo.read_all(self.path)

        found = False
        for row in rows:
            if row["ListID"] == str(list_id):
                books = row["ISBNs"].split("|") if row["ISBNs"] else []
                if isbn in books:
                    found = True
                break
        return found

    def __number_of_readinglist(self, user_id)-> int:
        rows = self.repo.read_all(self.path)
        count = 0
```

```
        for row in rows:
            if str(user_id) == row["UserID"]:
                count += 1

        return count

    def get_all_readinglist(self, user_id: int):
        rows = self.repo.read_all(self.path)
        result = []
        for r in rows:
            if r["UserID"] == str(user_id):
                rl = ReadingList.from_dict(r)
                result.append(
                    ReadingListSummary(
                        list_id=rl.list_id,
                        name=rl.name,
                        total_books=len(rl.books),
                        is_public=rl.is_public
                    )
                )

        return result

    def create_list(self, data: ReadingListCreate, user_id: int) -> ReadingListDetail:
        next_id = self.__generate_next_id()
        num_readinglist = self.__number_of_readinglist(user_id=user_id)
        if num_readinglist >= 10:
            raise ValueError("You can only have 10 reading lists")

        rows = self.repo.read_all(self.path)
        for r in rows:
            if r["UserID"] == str(user_id) and r["Name"].lower() == data.name.lower():
                raise ValueError(f'A reading list named "{data.name}" already exists.')

        readinglist = ReadingList(list_id=next_id,
                                  user_id=user_id,
                                  name=data.name)

        self.repo.append_row(self.path, self.fields, readinglist.to_csv_dict())
        return ReadingListDetail(**readinglist.to_api_dict())

```

```python
    def delete_list(self, list_id: int, user_id: int):

        rows = self.repo.read_all(self.path)
        original_count = len(rows)

        updated_rows = [
            r for r in rows if not (int(r["ListID"]) == list_id and int(r["UserID"]) == user_id)
        ]
        if len(updated_rows) == original_count:
            return False

        for i, row in enumerate(updated_rows, start=1):
            row["ListID"] = str(i)

        self.repo.write_all(self.path, self.fields, updated_rows)

        return True

    def rename(self, list_id: int, user_id: int, new_name: str) -> bool:
        rows = self.repo.read_all(self.path)
        all_names = [r["Name"].lower() for r in rows
                     if r["UserID"] == str(user_id) and r["ListID"] != str(list_id)]

        if new_name.lower() in all_names:
            raise ValueError(f'A reading list named "{new_name}" already exists.')

        renamed = False
        for r in rows:
            if r["UserID"] == str(user_id) and r["ListID"] == str(list_id):
                readinglist = ReadingList.from_dict(r)
                readinglist.rename(new_name)
                r.update(readinglist.to_csv_dict())
                renamed = True
                break

        if not renamed:
            return False
        self.repo.write_all(self.path, self.fields, rows)
        return True

    def toggle_visibility(self, list_id: int, user_id: int):
```

```
        rows = self.repo.read_all(self.path)

        for r in rows:
            if r["ListID"] == str(list_id) and r["UserID"] == str(user_id):
                rl = ReadingList.from_dict(r)
                rl.is_public = not rl.is_public
                r.update(rl.to_csv_dict())
                self.repo.write_all(self.path, self.fields, rows)
                return {
                    "list_id": list_id,
                    "is_public": rl.is_public,
                    "message": "Visibility toggled successfully",
                }
        return False

    def add_book(self, list_id: int, user_id: int, isbn: str) -> bool:
        rows = self.repo.read_all(self.path)

        for r in rows:
            if r["ListID"] == str(list_id) and r["UserID"] == str(user_id):

                rl = ReadingList.from_dict(r)

                if isbn in rl.books:
                    raise ValueError(f"Book {isbn} already in the reading list.")

                rl.add_book(isbn)

                r.update(rl.to_csv_dict())

                self.repo.write_all(self.path, self.fields, rows)
                return True

        return False

    def remove_book(self, list_id: int, user_id: int, isbn: str) -> bool:
        rows = self.repo.read_all(self.path)

        for r in rows:
            if r["ListID"] == str(list_id) and r["UserID"] == str(user_id):
                rl = ReadingList.from_dict(r)
```

```
158
159                 if isbn not in rl.books:
160                     raise ValueError(f"Book {isbn} not found in the reading list.")
161
162                 rl.remove_book(isbn)
163
164                 r.update(rl.to_csv_dict())
165
166                 self.repo.write_all(self.path, self.fields, rows)
167                 return True
168
169         return False
170
```