Flashcards APP

This project is a Django-based web application designed to create, manage, and share flashcards. It provides a user-friendly interface for studying. The system also supports RESTful API endpoints for advanced integrations.

Features

User Management

- Custom user model with optional admin privileges.
- User authentication: registration, login, and password management.
- Profile management via a dedicated user profile page.

Flashcards

- Create, view, and delete flashcards.
- Define flashcards with:
    - Question
    - Answer
    - Difficulty levels (easy, medium, hard)
- Rate flashcards (1-5) and view the average rating.
- Toggle sharing for flashcards, enabling collaboration.
- Hide flashcards for a clutter-free experience.
- Sort and filter by difficulty, date, or rating

Collections

- Organize flashcards into collections.
- View flashcards within collections in study mode.
- Add comments to shared collections in a forum-style interaction.
- View all flashcards in shared collections.
- comments set flashcards

Study Workflow

1. Create collections and populate them with flashcards.
2. Enter study mode and select a collection.
3. Review flashcards interactively, tracking performance in real time.
4. Analyze detailed telemetry reports for study progress.

API

- Full support for managing flashcards, collections, and user limits via RESTful APIs.
- API endpoints for flashcard creation, deletion, and sharing.

Admin Interface

- Admin dashboard for managing users, flashcards, and collections.
- Filter and search flashcards and collections by attributes like date and difficulty.
- Manage daily limits for flashcards and collections.

Technologies Used

- **Backend**: Django, Django REST Framework, Python
- **Frontend**: Django Templates, Bootstrap
- **Database**: SQLite (development), PostgreSQL/MySQL (production)
- **Testing**: Pytest, Django Test Framework
- **Deployment**: Compatible with platforms like Heroku, AWS, or DigitalOcean.

Installation

★ **Clone the Repository**

```
git clone https://github.com/AlexxFV/testvar.git
cd testvar
```

Alternatively, using SSH:

```
git clone git@github.com:AlexxFV/Testvar.git
cd testvar
```

Token

```
git clone
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
```

★ **Create a Virtual Environment**

```
python -m venv env
source env/bin/activate  # On Windows: env\Scripts\activate
```

★ **Install Dependencies:**

pip install -r requirements.txt

★ **Set Up the Database**

python manage.py migrate

★ **Create a superuser:**

python manage.py createsuperuser

★ **Run the Development Server**

python manage.py runserver

★ **Access the Application**

Web: Open http://127.0.0.1:8000/

Admin Dashboard: http://127.0.0.1:8000/admin/

APIs

- **Base API URL**: /api/
- **Documentation:** http://127.0.0.1:8000/api/schema/redoc/
- Endpoints:
  - /users/: Manage users
  - /flashcards/: Manage flashcards
  - /collections/: Manage collections

Testing

To run the test suite:

pytest

Includes:

- Twenty six unit tests
- API tests to validate endpoints.
- Integration tests for user flows like registration and study sessions.

Accessibility Test

```python
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By

@pytest.fixture
def driver():
    driver = webdriver.Chrome()
    driver.get("http://localhost:3000")
    yield driver
    driver.quit()

Tabnine | Edit | Test | Fix | Explain | Document | Ask
def test_button_is_accessible(driver):
    speaker_button = driver.find_element(By.ID, "speaker-button")

    aria_label = speaker_button.get_attribute("aria-label")
    assert aria_label == "Read Page"

    speaker_button.send_keys(Keys.TAB)
    focused_element = driver.switch_to.active_element
    assert focused_element == speaker_button
```

(function) active_element: Any

or To run Accessibility Testing

npx pa11y http://localhost:3000

Fuzzing Test

```python
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys

Tabnine | Edit | Test | Explain | Document | Ask
@pytest.fixture
def driver():
    driver = webdriver.Chrome()  # WebDriver configurate
    driver.get("http://localhost:3000")
    yield driver
    driver.quit()

Tabnine | Edit | Test | Explain | Document | Ask
def test_fuzzing_invalid_text(driver):
    speaker_button = driver.find_element(By.ID, "speaker-button")

    driver.execute_script("document.body.innerText = '';")
    speaker_button.click()
    assert True


    driver.execute_script("document.body.innerText = '😀😁💻🔄';")
    speaker_button.click()
    assert True
```

Security Test

```python
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By

Tabnine | Edit | Test | Explain | Document | Ask
@pytest.fixture
def driver():
    driver = webdriver.Chrome()
    driver.get("http://localhost:3000")
    yield driver
    driver.quit()

Tabnine | Edit | Test | Explain | Document | Ask
def test_content_is_sanitized(driver):

    malicious_script = "<script>alert('Hacked!');</script>"
    driver.execute_script(f"document.body.innerHTML = '{malicious_script}';")

    speaker_button = driver.find_element(By.ID, "speaker-button")
    speaker_button.click()


    alert_present = False
    try:
        driver.switch_to.alert
        alert_present = True
    except:
        alert_present = False

    assert not alert_present
```

File Structure

```
testvar/
├── flashcards/
│   ├── __init__.py
│   ├── admin.py          # Admin panel configurations
│   ├── apps.py           # Application configuration
│   ├── models.py         # Database models
│   ├── forms.py          # Django forms
│   ├── serializers.py    # API serializers
│   ├── urls.py           # URL routing
│   ├── views.py          # View functions and classes
│   ├── migrates/         # Migrates for de base
│   ├── templates/        # HTML templates for the app
│   │   ├── flashcards/
│   │   │   ├── login.html
│   │   │   ├── register.html
│   │   │   ├── change_password.html       # change password
│   │   │   ├── index.html      # Home page
│   │   │   ├── flashcard_create.html # Flashcard creation page
│   │   │   ├── flashcard_list.html   # List of flashcards
│   │   │   ├── study_mode.html       # Study mode page
│   │   │   ├── shared_flashcards.html # Shared flashcards page
│   │   │   ├── collection_list.html  # List of collections
│   │   │   ├── collection_create.html # Create a collection
│   │   │   ├── collection_flashcards.html # Flashcards collection
│   │   │   ├── shared_collection_view.html # View shared collections with comments
│   │   │   ├── shared_collection.html # View shared collections with comments
│   │   │   ├── study_question.html    # Study questions
│   │   │   ├── study_results.html     # Study results
│   ├── static/           # Static files (CSS, JS, images)
│   │   ├── js/           # JavaScript files
│   │   ├── images/       # Image assets
│   │   ├── styles.css    # Main stylesheet
├── tests/
│   ├── test_models.py    # Model tests
├── manage.py             # Django management script
├── readme.docx           # Documentation
```

License