Flashcard App

Problem definition

Frameworks to be used

- Flask for application code
- Jinja2 templates + Bootstrap for HTML generation and styling
- SQLite for data storage
- All demos should be possible on a standalone platform like replit.com and should not require setting up new servers for database and frontend management

Flashcards

- Used for memory training
- User can have multiple decks
- System automatically presents one card at a time, and user needs to select a choice based on how well they know the answer
- System will track progress over time and decide which cards need to be reviewed

Terminology

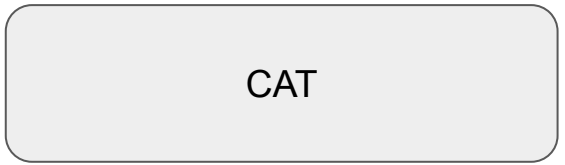
- Front - the part of the flashcard initially shown for review
- Back - the answer or meaning of what is on the front side
- Deck - a collection of related cards (eg. Tamil, Hindi, HTML etc.)

Example

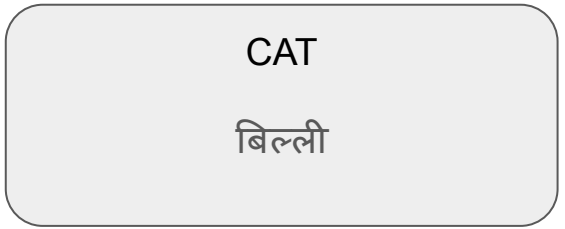
Hindi deck

- Mouse - चूहा
- Cat - बिल्ली
- Horse - घोड़ा

Question view:



Review view:



Core Functionality

- This will be graded
- Base requirements:
  - User login
  - Dashboard
  - Review
  - Deck management

Core - User Login

- Form for username (and optional password)
- You can either use a proper login framework, or just use a field for username - we are not concerned with how secure the login or the app is
- Suitable model for user

Core - Dashboard

- Dashboard with list of decks, last reviewed, score (how well reviewed)
- Time of last review, score on deck (some kind of average of individual reviews etc)
- Scoring method is left up to you as long as you can explain what is done

Core - Review

- Select a deck, then start presenting options one by one;
  - allow user to select from some options like “easy”, “medium”, “difficult” which tells how difficult they found each card
- Update last reviewed time and score, and overall deck score

Core - Deck management

- Create a new deck
  - Add cards to deck - the deck storage should handle multiple languages - usually UTF-8 encoding is sufficient for this
- Edit a deck
- Remove a deck
- Export/Import are optional

Recommended (graded)

- APIs for interaction with decks
  - CRUD on deck, individual cards
  - Additional APIs to retrieve score, or add other features
- Validation
  - All form inputs fields - text, numbers etc. with suitable messages
  - Backend validation before storing / selecting from database

Optional

- Styling and Aesthetics
- Proper login system
- Export/Import decks, compatibility with AnkiWeb etc.

Evaluation

- Report (not more than 2 pages) describing models and overall system design
  - Include as PDF inside submission folder
- All code to be submitted on portal
- A brief (2-3 minute) video explaining how you approached the problem, what you have implemented, and any extra features
  - This will be viewed during or before the viva, so should be a clear explanation of your work
- Viva: after the video explanation, you are required to give a demo of your work, and answer any questions
  - This includes making changes as requested and running the code for a live demo
  - Other questions that may be unrelated to the project itself but are relevant for the course

Instructions

- This is a live document and will be updated with more details and FAQs (possibly including suggested wireframes, but not specific implementation details) as we proceed.
- We will freeze the problem statement on or before 12th November, beyond which any modifications to the statement will be communicated via proper announcements.
- The project has to be submitted as a single zip file.
- The last date for submission is 28th November. This is a hard deadline and NO extensions will be possible because the vivas are scheduled to start on 1st Dec 2021.