Learner-inator 3000

By JAMRs (Joel, Anthony, Misha, Richard)

Project Description:

A web-based flashcard app that uses a variation of the SM2 algorithm, based on the Anki memorization program. The app is focused on long-term memorization using a variety of materials (including text, images, and sounds), and allows users to direct their learning through card prioritization.

Key Features:

- Implements Anki's variation of the SM2 algorithm for card ordering & recall
    - An important feature is that cards can be assigned difficulties and priorities, which influences how often they're repeated
    - Algorithm is implemented client-side using Javascript
- Two-faced cards that can contain text, images, and sounds
- Users can create, modify, and study from sets directly on the website
- Users can register/login to save and publically publish sets
- Users can create temporary sets and view public sets as guests (w/o logging in)
- Sets (and users) can be searched for using FTS4 extension of SQLite

Additional Features:

- Cards can be read aloud using Text to Speech (supporting multiple languages)
- Users can fork existing sets (copy content of another set to edit as their own)
- Users have additional descriptors (join-dates, public names, and profile pics)
- Sets have additional metadata (ex. created-dates)

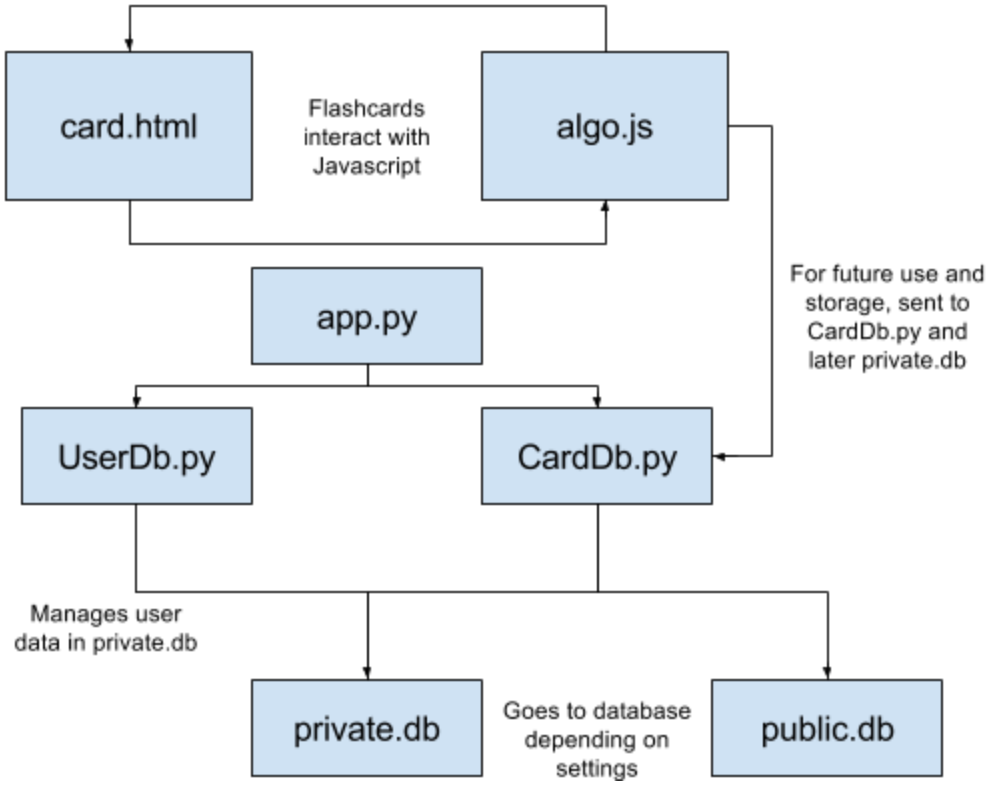
Details:

1. A user makes an account.
2. They can either make, edit, or use a set. They can also browse for sets.
    - Making a new set
        - Initial Settings
        - See Editing a set
    - Editing a set
        - A user can edit card content & add/remove cards
    - Use a set
        - User chooses set and begins studying, storage pulls up relevant priority info from database
        - Flips card and inputs feedback, next card shows
        - Quit at any time; if logged in, JS feedback information sent to server for storage

Component Breakdown:

- app.py
    - Data transfer to templates
- userDb.py
    - Account management
- cardDb.py
    - Card info and set storage/retrieval
- algo.js
    - Algorithm for flashcards
- public.db (See Schema)
- private.db (See Schema)
- base.html
    - Base template (structure, header, footer) for all pages
- view-set.html
    - Template for “view set” page (~/set/<name>)
- create-set.html
    - Template for creating and editing sets
- search.html
    - Template for search results page
- user.html
    - Template for a user page
- settings.html
    - Template for the user’s settings page
- base-style.css
    - Base styling (non-Bootstrap) for all pages
- Bootstrap CSS Files
    - The various CSS files included with Bootstrap & its components (included bootstrap.css)
- Bootstrap JS Files
    - The various JS files included with Bootstrap & its components (included bootstrap.js)

Component Map:

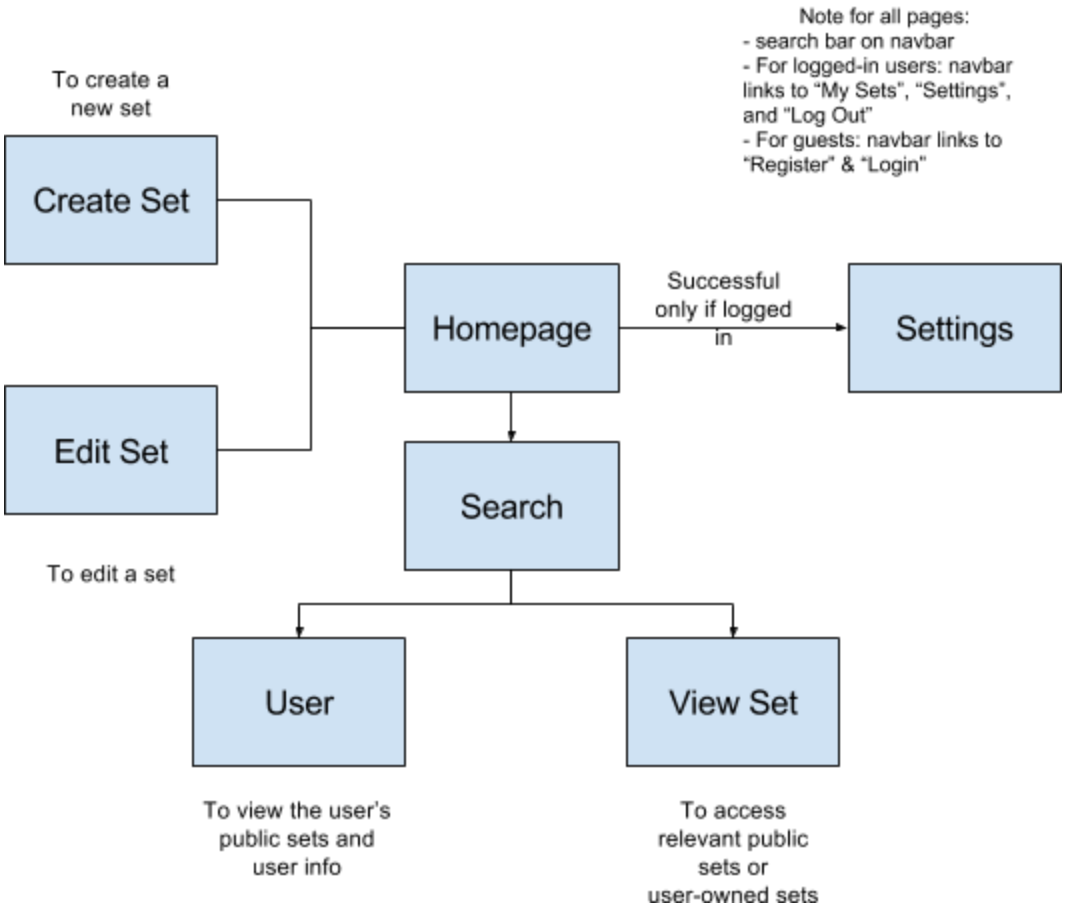


Website Structure:

- Homepage/Index: ~/
    - Describes project, shows sample/existing decks, invites to create, register, login
- View Set: ~/set/<name>
    - Render and present (i.e. use algorithm to order) set for studying
    - Link to edit page (if current user authored the set)
        - Tentatively: allow user to “fork” a set (with “Fork” button), which would give them a personal copy of the set to edit
    - For logged-in users: pull set from database
    - For guests: pull set from cookies; page shows “Save” button (which prompts registration or log in[front end])
- Create Set: ~/new
    - Allows users to build cards (using interface similar to Quizlet)
        - Can include text, images & sounds (from links or files); assign difficulty & priority
    - For logged-in users: “Save” button saves to server

    - For guests: “Save” button allows user to choose to register/login or save locally (cookies)
- Edit Set: ~/edit
    - Allows users to edit sets (using interface similar to Quizlet)
    - Only sets authored by user (based on username for logged-in users, or only locally-stored sets for guests) can be edited, otherwise displays msg & shows link to homepage
    - For logged-in users: “Save” button saves to server
    - For guests: “Save” button allows user to choose to register/login or save locally (cookies)
- Search: ~/search/<query>
    - Displays results page with sets and users (whose names are similar to query)
        - Requires that backend searches both sets and users
    - Results link to user page (~/user/<name>) or set page (~/set/<name>)
- User: ~/user/<name>
    - Lists a user’s publically available sets (and links to them)
    - If current user is same as user, shows all their sets (inventory)
    - Tentative: shows user’s join date, public name, profile pic
- Settings: ~/settings
    - Allows user to password (& possibly public name and profile pic too)
    - If a non-logged-in user tries to access it, auto-redirected to homepage
    - Backend: uses data from POST query to server that contains username
- All Pages:
    - Search box (which directs to ~/search/<query>) present in top navbar
    - For logged-in users: navbar links to “My Sets”, “Settings”, and “Log Out”
    - For guests: navbar links to “Register” & “Login”

Site Map:



Database Schema:

- public.db
    - Stores public card sets for users to add to their library
        - Table Sets: setName, creatorID, cardData (one string)
- private.db
    - Stores user information in Users
        - Table Users: username, hashedPass, userID
        - Table Cards: User, Sets (one string)
- Card data formatting:
    - CardFront:<contentString>||CardBack:<contentString>||CardPriority
    - <contentString>
        - <piece>%%<piece>%%<piece> etc. Each piece has an ID and placing coordinates (presuming canvas card)

Estimated Timeline:

- Temporary Algorithm (random priority) deployed, Databases set up (1/9) [Joel]
- Basic page template (structure, navbar, background) set-up (1/10) [Misha]
- Database Editing Done (1/13) [Joel]
- Set Storage Done (1/13) [Richard]
- Structure & non-card content of view, edit, and create set doe (1/13) [Misha]
- Basic Card Rendering (1/14) [Anthony]
- Proper Algorithm deployed (1/16) [Joel]
- User page & settings page done (1/16) [Misha]
- Flask done with the exception of search routing (1/16) [Richard]
- Implementation of Text to Speech (1/16) [Anthony]
- Public set searching enabled (1/17) [Joel]
- Search page done (1/17) [Misha]
- Flask done (1/17) [Richard]

Task Delegation:

- ❖ Joel: Algorithm implementation (JS), database storage of sets & users, search backend
- ❖ Anthony: Card Rendering/Features and Text to Speech
- ❖ Misha: Front-end - General page design (all minus card creation & rendering)
- ❖ Richard: Flask, Server Backend and Set Storage (User libraries, private/public etc.)

References:

- SM2 Algorithm Specification: https://www.supermemo.com/english/ol/sm2.htm
- Text to Speech API: http://www.responsivevoice.org/api/
- Anki Site: http://ankisrs.net/
- SQLite FTS4 Documentation: https://www.sqlite.org/fts3.html