The task of this project was to design a web-app to be used by librarians at some library. This app needs to interface with an underlying SQL database to perform the tasks required of it. We have made the following set of choices in this submission:

| | | |
|---|---|---|
| Programming language | : | Python 3.0 |
| Web development framework | : | Django 2.0.2 |
| Underlying database | : | MySQL 5.7.21 |
| Operating System | : | Apple MacOS High Sierra |

**Database Schema**:
The database schema as provided in the problem description is used as is, except that CARD_ID is internally maintained as an INTEGER in our database. This change is however completely abstracted from the user in the sense that as far as the UI is concerned, all the CARD_ID instances are treated as character-arrays of desired fixed length 8 and the specific format as suggested.

**Django web app:**
The web-app is maintained as a set of 5 core different HTML web-pages where each view serves as a major functional unit, with rough correspondence to the tasks required of this project, and are listed to be:

> `/app/`: Assists users to search books within the database based on a text search field and returns the results in a decreasing order of relevance. Functionality to directly select a book from the list for checking out is also provided.
>
> `/app/newuser/`: Provides a web-interface to allow its user to add a new borrower to the library database and automatically generates and assigns a new user ID to the user. Checks are provided against invalid input and user repetitions.
>
> `/app/checkout/`: A web-interface to allow searching for a book and a user, and upon valid selections to allow checking out of the selected book to the selected user.
>
> `/app/checkin/`: A web-interface to search for existing records of checked out books, and to allow check-ins of these books while interfacing with the database.
>
> `/app/fines/`: An interface which allows modification of the date within the system and to update the fines as of that date.

Django uses a Model-View-Template paradigm to efficiently and cleanly handle different levels of abstractions. The model-layer handles the database modeling, handling and integration. The

view-layer processes the HTML requests and responses. The template-layer is interfaced with the user directly and handles the changes visible to the user. We primarily make use of the view-layer with the help of Django while keeping the template-layer loosely coupled with it. The database modeling and handling is all done in MySQL using raw SQL queries and an SQL connector for Python.

**SQL connector:**
MySQL database server provides a Python interface which is accessible via the MySQLdb interface. Implementation has been abstracted inside a custom implementation of sql_connector.py

**Code directory structure**:
The code resides in the root directory <HOME>/code/. This directory contains:

- Directory /data/: This directory contains the data-files, books.csv and borrowers.csv, required to initialize the database state.
- Directory /django/: This directory contains the code for the developed web-app. The web-app provides an appropriate HTML response depending on the URL (and its parameters) contained within the HTML request. Django does this using a view-paradigm wherein the code written for a view is responsible for manipulating and returning an HTML response to a request.
- Installation file install.py
- Installation config file install_config.py. This file contains parameters which dictates whether or not to use the books or the borrowers data to re-install the system.

**Features and considerations**:

1. Search:
   a. The key words from the search strings should be highlighted in results.
   b. The results should be listed in decreasing order of number of matched keywords.
   c. Repeated keywords in the query should be only used once.
   d. HTML regex matching can be easily broken if not implemented carefully. Employ intelligent algorithms:
      i. keywords are scanned and matched as they are found from left to right
      ii. one keyword cannot be contained within another for this to work, so keywords which are substrings of another keyword are removed

2. Register user:
   a. A user with an existing SSN should be rejected.
   b. Invalid data should be rejected and appropriate error message be displayed:
      i. Name cannot be numeric

   ii. City name cannot be numeric
   iii. SSN has to be 9-digit numeric value
   iv. All fields (except the phone number) are compulsory
c. If registration is successful, print the card for the new user with their information displayed.

3. Checkout:
   a. Option to choose a book based on ISBN should be provided. Display error message if such a book cannot be found.
   b. Provide link from the search-page for direct checkout.
   c. Option to choose a user based on Card ID (in the correct format) should be provided. If no user can be found, display if the ID format was incorrect or if the user cannot be found with the entered ID.
   d. Don't allow more than 3 books to be checked out. If attempted, provide a link to checkin some book for that user.

4. Checkin:
   a. Display all the books currently checked in and provide a search box to refine the results based on the user's card ID, user's name or the book's ISBN.
   b. Provide radio buttons to choose a book to be checked in. When checked-in, refresh the list.

5. Fines:
   a. Provide a means to choose the date to simulate the fines situation.
   b. Show the fines as on that date, when a date is chosen. Once a date is chosen, the system cannot rollback to an earlier state. Thus, the successively chosen dates should be in a chronological order.
   c. Display the fines grouped by the users' total due amounts.
   d. When attempted to pay a fine for a chosen user, display all the fines corresponding to individual loans.
   e. Fines should be payable only for the returned books. Provide a link to the checkin page, if a book is not returned.
   f. If a fine is paid successfully, refresh the fines list.