Лабораторная работа с теорией
MVC шаблон

Шаблон MVC описывает способ построения структуры приложения, целью которого является отделение бизнес-логики от пользовательского интерфейса. В результате, приложение легче масштабируется, тестируется, сопровождается и реализуется. Концептуальная схема шаблона MVC приведена на рисунке 1.

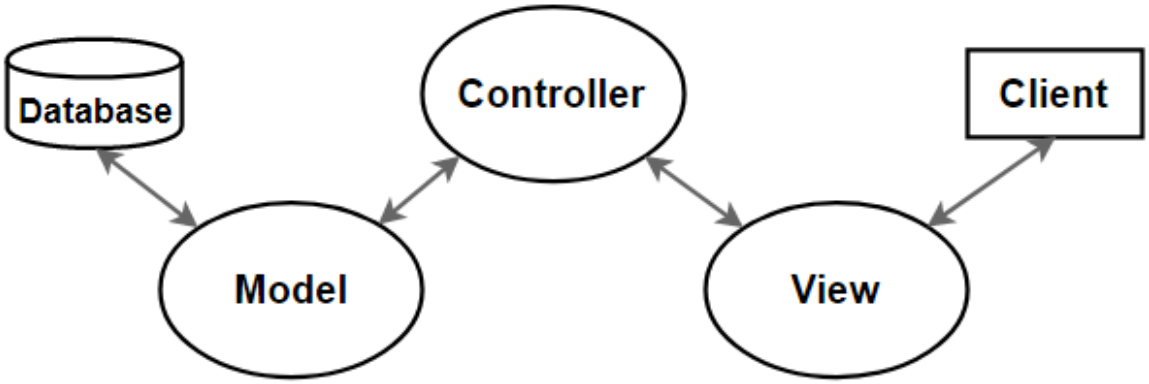

Рисунок 1. Концептуальная схема шаблона MVC

Модель (`Model`) содержит бизнес-логику приложения, описывает данные и методы работы с ними: запросы к базе данных, правила валидации. Модель реагирует на запросы контроллера, изменяя свое состояние.

Модель не должна напрямую взаимодействовать с пользователем. Все переменные, относящиеся к запросу пользователя должны обрабатываться в контроллере.

Модель не должна генерировать HTML или другой код отображения, это должно быть реализовано в виде.

Вид (`View`) используется для создания внешнего отображения данных, полученных из контроллера. Виды содержат HTML-разметку и конструкции шаблонизатора для формирования динамических частей страницы.

Виды обычно разделяют на общий шаблон, содержащий разметку, общую для всех страниц и части шаблона, которые используют для отображения данных, выводимых из модели, или динамических частей страницы.

Виды не должны напрямую обращаться к базе данных (это делают модели).

Виды не должны работать с данными, полученными из запросов пользователя (это реализует контроллер).

Контроллер (`Controller`**)** связующее звено, соединяющее модели, виды и другие компоненты в рабочее приложение. Контроллер отвечает за обработку запросов пользователя.

Контроллер не должен содержать SQL-запросов (их размещают в моделях).

Контроллер не должен содержать HTML и другой разметки (их выносят в виды).

Типичную последовательность работы MVC-приложения (на примере выполнения главной страницы `index`) показана на рисунке 2.

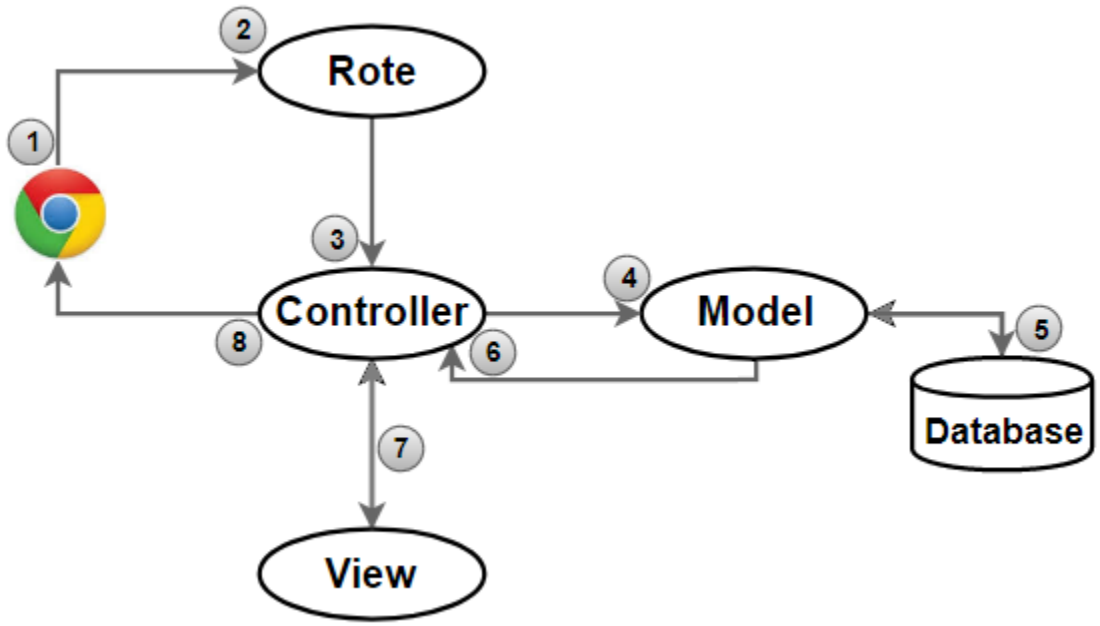

Рисунок 2. Последовательность работы MVC приложения

1. При заходе пользователя на веб-ресурс, скрипт инициализации запускает на выполнение приложение.

2. Приложение получает запрос от пользователя, создает экземпляр приложения и передает управление контроллеру соответствующей страницы (в нашем `index.py`).

3. В контроллере могут содержаться вызовы модели, выбирающие информацию из базы данных. Контроллер импортирует функции из модели (`index_model.py`), отбирающие необходимые данные.

4. В модели (`index_model.py`) осуществляется обращение к базе данных (средствами запросов SQL) для выборки и/или корректировки данных. Запросы SQL размещаются в функциях, которые возвращают отобранные данные.

5. Отобранные запросами данные из базы данных передаются в модель.

6. Полученные из базы данных данные при необходимости обрабатываются и преобразуются. Запросы SQL размещаются в функциях, которые возвращают отобранные данные контроллеру (`index.py`).

7. Контроллер (`index.py`) запускает на генерацию страницу-шаблон (`index.html`), передает ему полученные из модели данные. Также могут передаваться данные, полученные от пользователя. В виде формируется код страницы (средствами HTML-тегов и конструкциями шаблонизатора). Сформированная страница возвращается контроллеру.

8. Контроллер вставляет возвращает сформированную страницу в браузер пользователя.

Разработка приложения «Библиотека» на основе технологии MVC

Данное приложение не претендует на полную реализацию всех функций электронных библиотек и разрабатывается в качестве учебного примера реализации шаблона MVC.

Описание предметной области

В библиотеке хранятся книги. Каждая книга относится к одному жанру, опубликована в одном издательстве, может иметь одного или несколько авторов. Также о книге известна дата ее публикации. Библиотека располагает некоторым количеством экземпляров каждой книги.

Каждый человек может стать читателем в библиотеке. Читатель может взять одну или несколько книг на некоторое время. При этом в библиотеке сохраняется информация о дате выдачи книги и дате ее возврата. Когда читатель берет книгу, количество доступных экземпляров уменьшается, когда возвращает - увеличивается.

В библиотеку могут поступать новые книги в некотором количестве экземпляров.

Пользователи и работники библиотеки могут получить статистическую информацию о работе библиотеки в заданный период.

Логическая схема базы данных

На рисунке 3 приведена логическая схема базы данных «Библиотека»

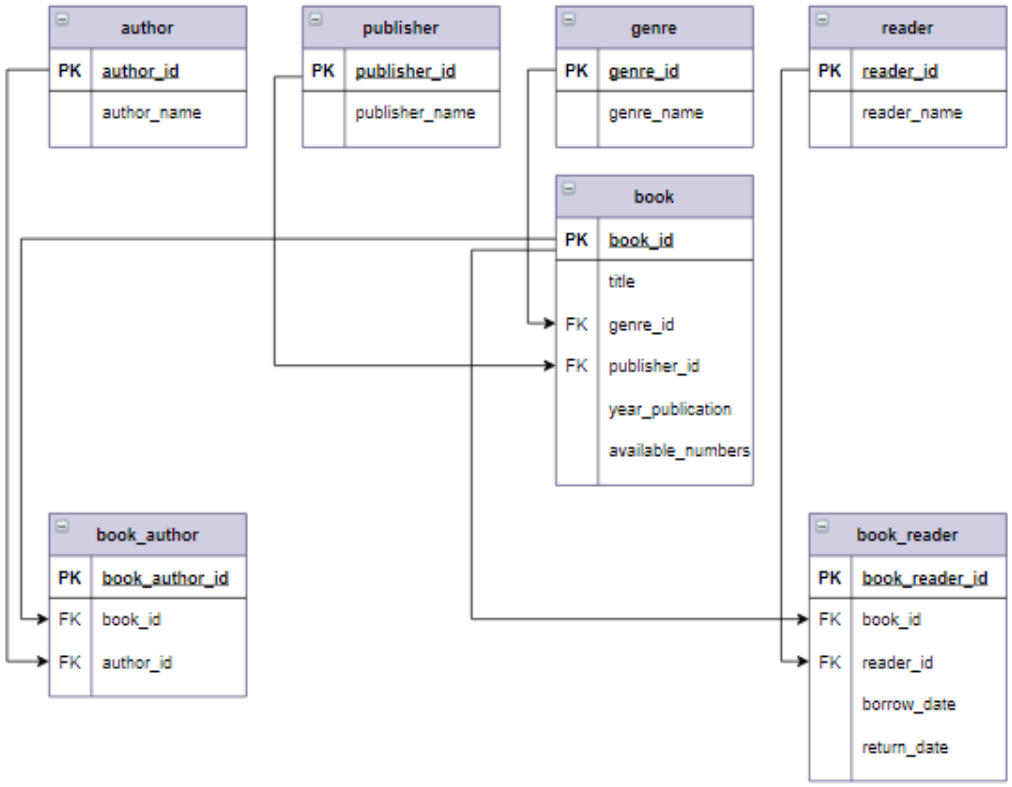

Рисунок 3. Логическая схема базы данных «Библиотека»

Функциональные возможности приложения

Приложение «Библиотека» позволяет:

1. Хранить информацию о пользователях библиотеки.
2. Включать новых пользователей.
3. Вести учет о том, какие книги и когда читатель взял, когда их сдал.
4. Осуществлять поиск книг по заданным характеристикам.
5. Поддерживать в актуальном виде информацию о фонде книг в библиотеке.
6. Добавлять новые книги, указывать их характеристики.
7. Формировать статистическую информацию о работе библиотеки.

Проект приложения

На рисунке 4 показаны страницы разрабатываемого приложения и связи между ними.

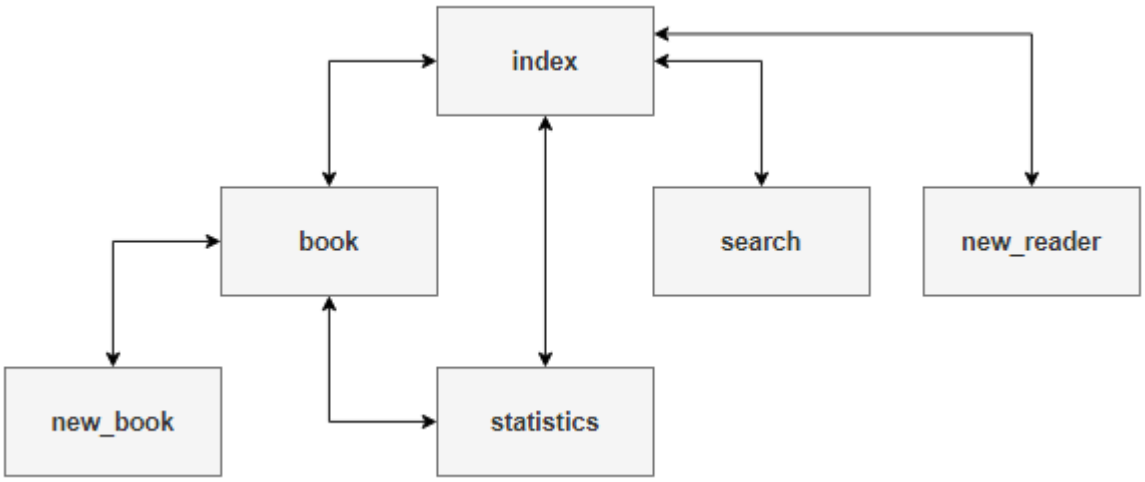

Рисунок 4. Карта сайта

Приложение включает следующие страницы:

- страница с информацией о пользователях и взятых ими книгах (`index`);
- страница для занесения данных о новом пользователе (`new_reader`);
- страница поиска книг по различным характеристикам и выбора одной книги (`search`);
- страница просмотра книг, хранящихся в библиотеке (`book`),
- страница для занесения новой книги (`new_book`);
- страница со статистической информацией о работе библиотеки (`statistics`).

В лабораторной работе будут разработаны три страницы приложения:

- Главная (index);
- Новый читатель (new_reader);
- Поиск (search).

Поэтому рассмотрим структуру только этих страниц.

Страница «Читатель»

Страница предназначена для работы с читателями библиотеки. На этой странице осуществляется:

- поиск информации о читателе;
- переход на страницу Новый читатель;
- вывод информации и книгах, которые брал (взял) читатель;
- переход на страницу с поиском для выбора книги, которую хочет взять читатель;
- выдача книги читателю, установка даты ее выдачи;
- возврат книги, установка даты ее сдачи.

На рисунке 5 приведен макет главной страницы. Когда читатель приходит в библиотеку в первый раз, в базу данных заносится его личная информация (переход на страницу «Новый читатель»). Если же человек уже был в библиотеке – выполняется поиск информации о нем в базе данных. Далее определяется цель посещения библиотеки: пользователь пришел взять какие-то книги или сдать те, которые брал раньше. В первом случае осуществляется подбор книги по критериям, заданных пользователем. Читатель может взять одну или несколько книг, при этом в его карточку заносится взятая книга и дата ее выдачи. Во втором случае, когда человек принес книги, отмечается дата сдачи этой книги.

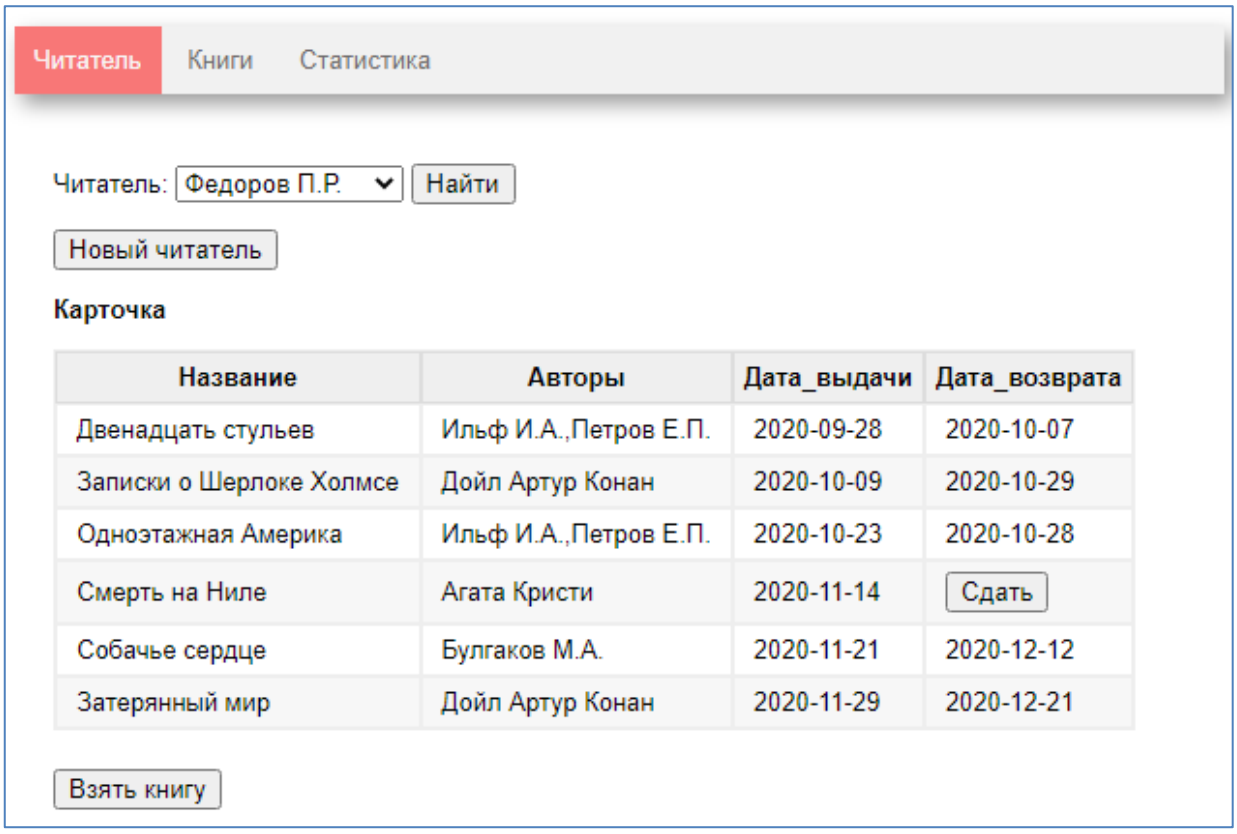

Рисунок 5. Макет страницы «Читатель»

Страница «Новый читатель»

Станица предназначена для ввода информации о новом читателе. На этой странице осуществляется:

- ввод данных о читателе;
- сохранение информации о читателе и возврат на главную страницу;
- информацию о читателе можно не сохранять и также вернуться на главную страницу.

На рисунке 6 приведен макет страницы «Новый читатель». В поле для ввода на странице можно занести информацию о читателе (его фамилию и инициалы). Если все верно, необходимо нажать кнопку «Добавить читателя», тогда информация будет занесена в базу данных. Если нового читателя заносить не нужно – нажать кнопку «Отменить». В обоих случаях будет осуществлен переход на страницу «Читатель».

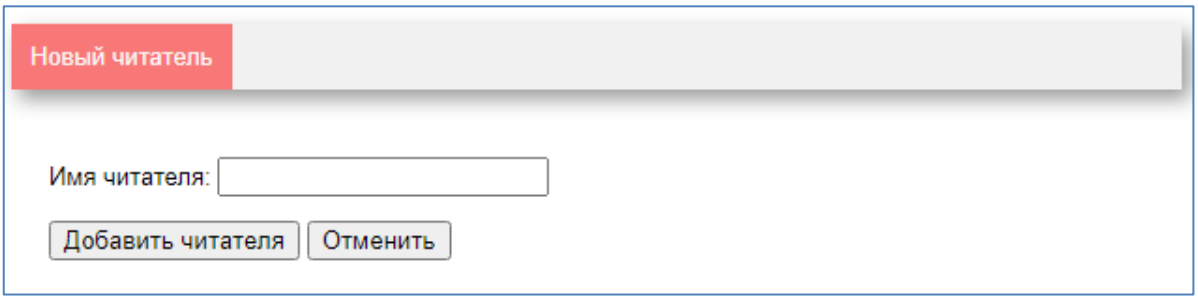

Рисунок 6. Макет страницы «Новый читатель»

Страница «Поиск книги»

Станица предназначена для поиска и выбора одной книги, которую хочет взять читатель. На этой странице осуществляется:

- отбор книг для просмотра по жанрам, авторам и издательствам;
- очистка формы с фильтрами;
- выбор книги, эта книга с текущей датой будет занесена в список книг, которые брал читатель, переход на главную страницу;
- переход на главную страницу без выбора книги.

На рисунке 7 приведен макет страницы «Поиск книги». Пользователь может выбрать один или несколько жанров, авторов или издательств в блоке с поиском. При нажатии кнопки «Найти» в строке с фильтром (красный цвет на рисунке 9) перечисляются выбранные значения фильтров, в блоке с карточками книг отображаются только те книги, которые соответствуют фильтру. При нажатии кнопки «Очистить» в блоке с карточками отображаются все книги библиотеки. Для тех книг, которых нет в наличии кнопка «Выбрать» неактивна. Если кликнуть по кнопке «Выбрать» некоторой книги, эта книга с текущей датой будет занесена в список взятых книг читателя. Если книгу выбирать не нужно, необходимо нажать кнопку «Не выбирать книгу» и вернуться на главную страницу.

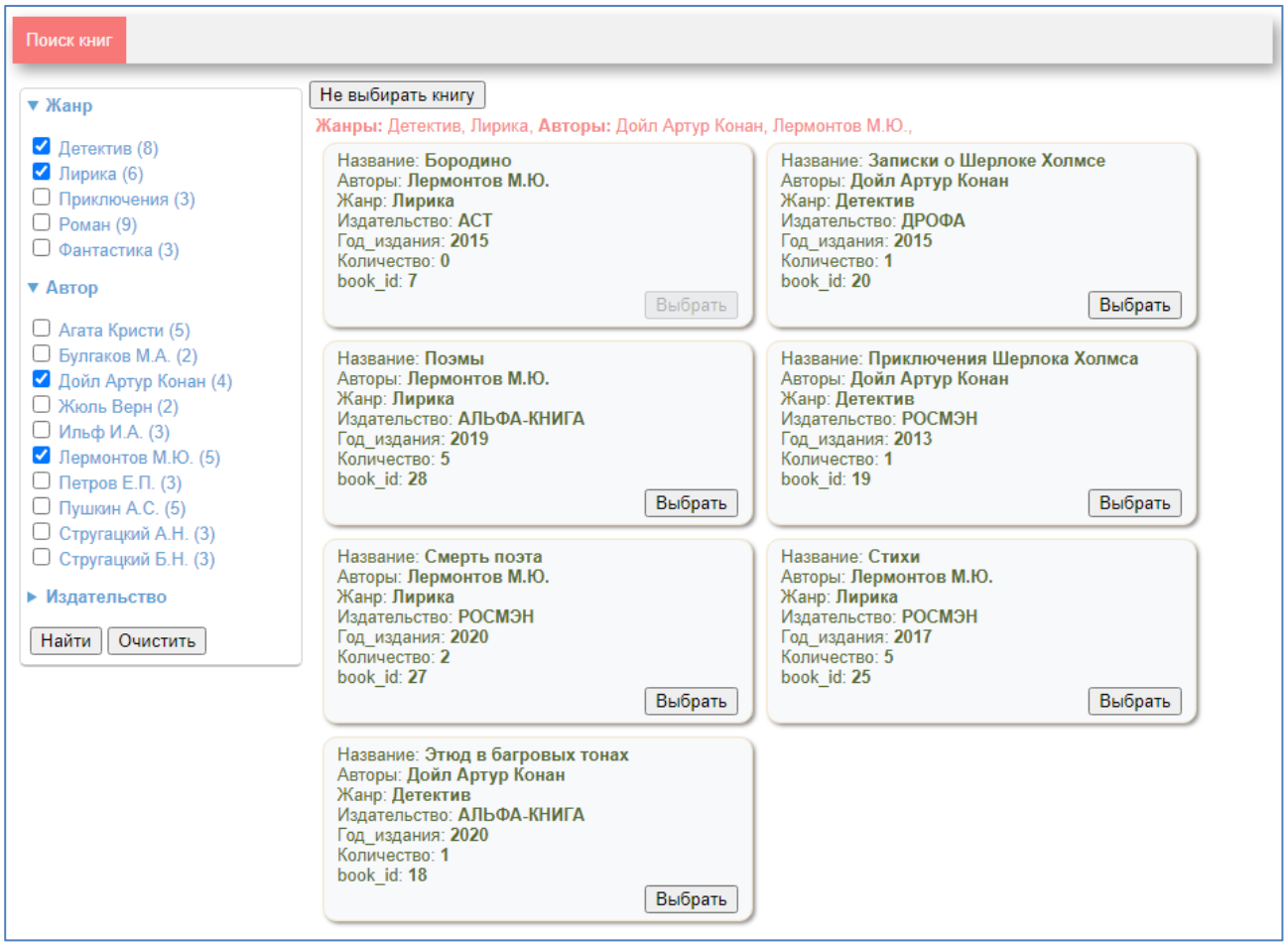

Рисунок 7. Макет страницы «Поиск книги»

РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

Структура каталога приложения

Для начала создадим структуру файлов и папок (таблица 1), необходимых для реализации приложения.

Таблица 1.

Файловая структура приложения

library	
controllers	- *реализация контроллеров страниц*
index.py	- контроллер страницы `index`
book.py	- контроллер страницы `book`
search.py	- контроллер страницы `search`
new_reader.py	- контроллер страницы `new_reader`
templates	- *шаблоны страниц*
index.html	- шаблон страницы `index`
book.html	- шаблон страницы `book`
search.html	- шаблон страницы `search`
new_reader.html	- шаблон страницы new_reader
models	- *реализация моделей для страниц*
index_model.py	- модель страницы `index`
search_model.py	- модель страницы `search`
static	- *содержит статическую информацию*
CSS	- *стили страниц*
style.css	-
images	- *рисунки*
app.py	- главная программа, создает экземпляр приложения
utils.py	- программа с утилитами
library.sqlite	- база данных
flask.py	- модуль `flask`

Отметим, что для реализации одной страницы приложения необходимо создать три файла и разместить их в нужных папках. Для страницы `index`, например, это будут файлы `index.py`, `index_model.py`, `index.html` (в таблице 1 отмечены жирным шрифтом). Если для страницы не требуется информация из базы данных – модель для страницы создавать не нужно.

Структура модулей приложения

Для реализации приложения используется структура модулей (рисунок 8). Также на схеме показаны связи между модулями.

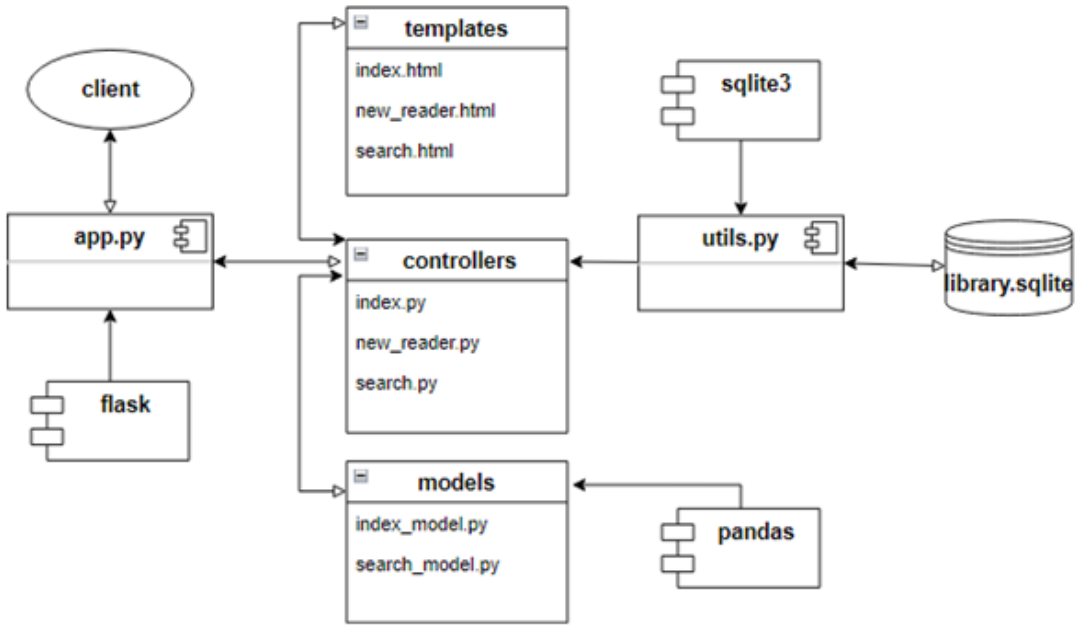

Рисунок 8. Структура модулей приложения

Алгоритм разработки приложения

Рассмотрим процесс разработки в соответствии с технологией использования MVC-шаблона.

1. В `app.py` необходимо:
- импортировать модуль flask;
- создать объект-приложение;
- импортировать все контроллеры, используемые в приложении;
- настроить сессию.
2. В `utils.py` реализовать функцию для создания соединения с базой данных.
3. Для каждой страницы создать:

Котроллер, в нем:
- импортировать все необходимые функции модели
- установить соединение с базой данных (вызвать функцию модуля `units.py`);
- обработать отправленные пользователем данные (если они были переданы);
- вызвать необходимые модели и сохранить их результаты в переменные;
- вызвать генерацию шаблона страницы, указать параметры генерации;

Модель, в ней:
- разместить в отдельной функции запросы sql, выполняющие определенную последовательность действий (это может быть один запрос на выборку или несколько запросов корректировки данных), параметрами функции будут соединение с базой данных и значения для выбора данных, результат – отобранная из базы данных информация, в виде `DataFrame`.

Шаблон, в нем:
- создать структуру html-страницы, для генерации динамических частей использовать конструкции шаблонизатора.

Программный код

app.py:

```
# импорт объекта для создания приложения
from flask import Flask, session

# создание экземпляра объекта приложения
app = Flask(__name__)

# установим секретный ключ для подписи.
app.secret_key = b'_5#y2L"F4Q8z\n\xec]/'

# здесь необходимо указать все контроллеры страниц
# закомментировать еще не реализованные
import controllers.index
import controllers.new_reader
import controllers.search
```

utils.py:

```
import sqlite3

def get_db_connection():
    return sqlite3.connect('library.sqlite')
```

style.css:

```
    /* меню страниц */
.menu {
    background-color: #f1f1f1;
   box-shadow: 4px 5px 10px rgba(0, 0, 0, 0.4);
    width: 100%;
}

/*Теперь уберем маркеры со списка и произведем выравнивание его элементов в одну линию*/

.menu li {
    display: inline-block;
    vertical-align: top;
     padding: 10px;
      color: #6f6d6d;
}

.menu a {
    text-decoration: none; /* Убирает подчеркивание для ссылок */
    color: #6f6d6d;
   }

.menu li:hover {
    background-color: #f87777;
    cursor: pointer;
    color: #f8f8f8;
}
.menu ul {
    padding: 0px;
}

.active{
    background-color: #f87777;
    color: #f8f8f8;
}

.active a:visited{
    color: #f8f8f8;
}
.menu  a:hover{
    color: #f8f8f8;
}

.block {
    padding: 10px 20px;
}
```

Главная страница index

Контроллер:

```
from app import app
from flask import render_template, request, session

#import sqlite3
from utils import get_db_connection
from models.index_model import get_reader, get_book_reader, get_new_reader ,
borrow_book

@app.route('/', methods=['get'])
def index():

    conn = get_db_connection()

    # нажата кнопка Найти
```

```
    if request.values.get('reader'):
        reader_id = int(request.values.get('reader'))
        session['reader_id'] = reader_id


    # нажата кнопка Добавить со страницы Новый читатель
    #(взять в комментарии, пока не реализована страница Новый читатель)
    elif request.values.get('new_reader'):
        new_reader = request.values.get('new_reader')
        session['reader_id'] = get_new_reader(conn, new_reader)

    # нажата кнопка Взять со страницы Поиск
    #(взять в комментарии, пока не реализована страница Поиск)
    elif request.values.get('book'):
        book_id = int(request.values.get('book'))
        borrow_book(conn, book_id,  session['reader_id'])
    # нажата кнопка Не брать книгу со страницы Поиск
    elif request.values.get('noselect'):
        a = 1
    # вошли первый раз
    else:
        session['reader_id']= 1

    df_reader = get_reader(conn)
    df_book_reader = get_book_reader(conn, session['reader_id'])


    # выводим форму
    html = render_template(
        'index.html',
        reader_id = session['reader_id'],
        combo_box = df_reader,
        book_reader = df_book_reader,
        len = len
    )
    return html
```

Модель:

```
import pandas

def get_reader(conn):
    return pandas.read_sql(
    '''
        SELECT * FROM reader
    ''', conn)

def get_book_reader(conn, reader_id):
    # выбираем и выводим записи  о том, какие книги брал читатель
    return pandas.read_sql('''
           WITH get_authors( book_id, authors_name)
           AS(
              SELECT book_id, GROUP_CONCAT(author_name)
              FROM author JOIN book_author USING(author_id)
              GROUP BY book_id
           )
           SELECT title AS Название, authors_name AS Авторы,
                  borrow_date AS Дата_выдачи, return_date AS Дата_возврата,
                  book_reader_id
           FROM
              reader
              JOIN book_reader USING(reader_id)
              JOIN book USING(book_id)
              JOIN get_authors USING(book_id)
```

```
            WHERE reader.reader_id = :id
            ORDER BY 3
            ''', conn, params={"id": reader_id})

# для обработки данных о новом читателе
def get_new_reader(conn, new_reader):
    cur = conn.cursor()
    # добавить нового читателя в базу
    …
    return cur.lastrowid

# для обработки данных о взятой книге
def borrow_book(conn, book_id, reader_id):
    cur = conn.cursor()
    # добавить взятую книгу (book_id) читателю (reader_id) в таблицу book_reader
    # указать текущую дату как дату выдачи книги

    # уменьшить количество экземпляров взятой книги

    return True
```

Шаблон:

```
<!DOCTYPE HTML>

<html>
<head>
    <link rel="stylesheet" type="text/css" href="/static/css/main.css" />
    <title> Карточка читателя </title>
        <!-- Макрос для вывода таблицы без ключевого столбца,
          переданного параметром -->
    {% macro render_table(table_name, relation, id) %}
        <p>
           <b>{{table_name}} </b>
        </p>
        {% set n = len(relation) %}
        <table>
            <tr>
            {% for name in relation.dtypes.index %}
              {% if name != id %}
                <th>{{name}}</th>
              {% endif %}
            {% endfor %}
            </tr>
            <tbody>

            {% for i in range(n) %}
            <tr>
               {% for name in relation.dtypes.index %}
                 {% if name != id %}
                  <td>
                  {% if relation.loc[i, name] == None %}
                     <form action = '' method ="get">
                       <input type="hidden" name = return
value={{relation.loc[i, "book_reader_id"]}}>
                       <input type="submit" value="Сдать">
                     </form>
                  {% else %}
                    {{ relation.loc[i, name] }}
                  {% endif %}
                  </td>
                  {% endif %}
               {% endfor %}
            </tr>
```

```
            {% endfor %}
        </tbody>
        </table>
    {% endmacro %}

<!—Вставить макрос для формирования поля со списком из ЛР 5 -->

</head>
<body>
  <div class=menu>
    <ul>
        <li class = active><a href={{ url_for("index")}}>Читатель</a></li>
        <li><a href={{ url_for("book")}}>Книги</a></li>
        <li><a href={{ url_for("statistics")}}>Статистика</a></li>
    </ul>
  </div>
  <div class = block>

    <p>
     <form action='' method ="get">
       Читатель:
        <!-- Создаем поле со списком с именем reader
          (использовать макрос, созданный в ЛР 5 -->

         <input type="submit" value="Найти">
        </form>
        </p>
        </p>
        <!-- Пока страница не создана, не указывать ее в action -->
        <form action ="{{url_for('new_reader')}}" method="get">
           <input type="submit" value="Новый читатель">
        </form>
      </p>
      {{ render_table("Карточка",book_reader, "book_reader_id") }}
      </p>
        <!-- Пока страница не создана, не указывать ее в action -->
        <form action ={{url_for('search')}} method="get">
           <input type="submit" value="Взять книгу">
        </form>
      </p>
    </div>

</body>
</html>
```

Самостоятельное задание.

1. На Главной странице реализовать возвращение книги читателем по клику по кнопке «Сдать»: увеличить количество экземпляров книг в таблице `book` и указать текущую дату как дату сдачи книги в таблице `book_reader`.

2. Реализовать страницу «Новый читатель».

3. При клике на кнопку «Новый читатель» на Главной странице должна выводиться страницы «Новый читатель».

4. При занесении нового читателя на странице «Новый читатель» и клике на кнопку «Добавить читателя», этот читатель должен отобразиться как выбранный в поле со списком на Главной странице. Если же нажата кнопка «Отменить», на главной странице не должно быть никаких изменений.

5. Реализовать страницу «Поиск» на основе шаблона из ЛР 5.

6. По клику на кнопку «Взять книгу» на главной странице должна выводиться страница с Поиском.

7. При выборе книги на странице Поиск, эта книга должна отобразиться в списке книг читателя с текущей датой выдачи.