

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования «Московский государственный технический
университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту на тему:
Домашняя библиотека

Студент

_____ Мхитарян В. К.
(Подпись, дата)

Руководитель курсового проекта

_____ Исаев А.Л.
(Подпись, дата)

Содержание

Введение	4
1 Аналитическая часть	5
1.1 Формализация задачи	5
1.2 Общие сведения о БД СУБД	5
1.3 Типы БД	5
1.3.1 Иерархическая модель БД	5
1.3.2 Сетевая модель	6
1.3.3 Реляционная модель	6
1.3.4 Сравнение моделей	7
1.4 СУБД	8
1.5 Framework	8
1.6 Вывод	8
2 Конструкторская часть	9
2.1 Проектирование таблиц базы данных	9
2.2 Проектирование системы изменения данных	10
2.3 Проектирование регистрации и аутентификации пользователя	11
2.4 Вывод	11
3 Технологическая часть	12
3.1 Выбор инструментов разработки	12
3.2 Реализация хранения данных	12
3.3 Реализация доступа к данным	16
3.4 Frontend-разработка	17
3.5 Интерфейс приложения	19
Заключение	22
Список использованных источников	23

Введение

Информационные технологии стали неотъемлемой частью жизни современного человека. Сегодняшние читатели могут получить знания в разных направлениях, в том числе за счет нетрадиционных форм доступа к информационным ресурсам. Но как быть с уже существующими книгами? У большинства людей дома есть домашняя библиотека, в которой находится разное количество книг, и не всегда можно с легкостью запомнить их расположение.

Целью данной курсовой работы является создание клиент–серверного приложения «Домашняя библиотека», которое предоставляет доступ к книгам домашней библиотеки и возможность просмотра ее местоположения.

Актуальность разработки состоит в том, что с помощью такого приложения значительно уменьшается трудоемкость ведения учета информации о книгах и их поиск.

Для выполнения цели необходимо выполнить следующие задачи:

- формализовать задачу в виде определения необходимого функционала;
- провести анализ существующих СУБД;
- спроектировать базу данных, необходимую для хранения и структурирования данных;
- реализовать спроектированную базу данных с использованием выбранной СУБД;
- реализовать приложение для взаимодействия с реализованной БД.

1 Аналитическая часть

В данном разделе будут рассмотрены общие сведения о БД и СУБД, типы БД и используемые framework.

1.1 Формализация задачи

В соответствии с техническим заданием на курсовой проект пользователям необходима система авторизации и регистрации для предоставления индивидуальной информации.

Также должна быть предусмотрена возможность поиска и просмотра местоположения книг, находящихся в домашней библиотеке.

1.2 Общие сведения о БД СУБД

База данных представляет собой совокупность определенным образом организованных данных, которые хранятся в памяти вычислительной системы и отображают состояние объектов и их взаимосвязи в рассматриваемой предметной области.

Под системой управления базами данных понимается совокупность программных и языковых средств, предназначенных для создания и обработки БД.

1.3 Типы БД

Модель данных определяет логическую структуру БД и то, каким образом данные будут храниться, организовываться и обрабатываться.

Существует 3 типа моделей организации данных:

- иерархическая модель БД;
- сетевая модель БД;
- реляционная модель.

1.3.1 Иерархическая модель БД

Иерархическая модель БД представляет собой древовидную (иерархическую) структуру, состоящую из объектов (данных) различных уровней. Каждый объект может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении предка (объект более близкий к корню) к потомку (объект более низкого уровня), при этом возможна ситуация, когда объект-предок имеет несколько потомков, тогда как у объекта-потомка обязателен только один предок.

Иерархической базой данных является файловая система, состоящая из корневого каталога, в котором имеется иерархия подкаталогов и файлов.

Связи записей реализуются в виде физических указателей с одной записи на другую. Основной недостаток иерархической структуры – невозможность реализовать отношения "многие-ко-многим" а также ситуации, в которых запись имеет несколько предков.

(1)

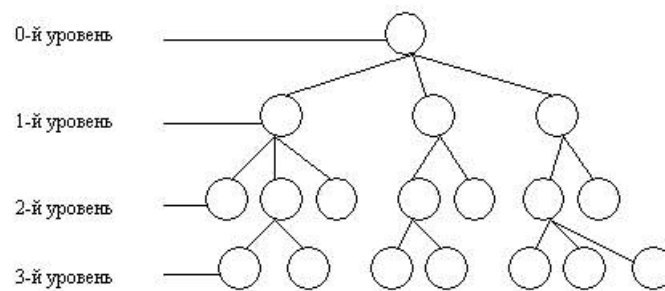


Рис. 1: Иерархическая модель БД

1.3.2 Сетевая модель

Сетевая модель данных является расширением иерархического подхода. Разница между иерархической моделью данных и сетевой заключается в том, что в иерархических структурах запись-потомок должна иметь в точности одного предка, а в сетевой структуре у потомка может быть любое число предков. Записи в такой модели связаны списками с указателями.

Примером сетевой СУБД является IDMS (интегрированная система управления данными) от компании Computer Associates international Inc.

Популярность сетевой модели совпала с популярностью иерархической модели. Некоторые данные намного естественнее моделировать с несколькими предками для одного дочернего элемента. Сетевая модель позволяла моделировать отношения «многие-ко-многим».

И хотя эта модель широко применялась на практике, она так и не стала доминантной по двум основным причинам. Во-первых, компания IBM решила не отказываться от иерархической модели в расширениях для своих продуктов, таких как IMS и DL/I. Во-вторых, через некоторое время ее сменила реляционная модель, предлагавшая более высокоуровневый, декларативный интерфейс. (1)

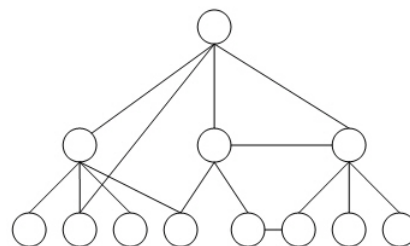


Рис. 2: Сетевая модель БД

1.3.3 Реляционная модель

В реляционной модели, в отличие от иерархической или сетевой, не существует физических отношений. Вся информация хранится в виде таблиц (отношений), состоящих из рядов и столбцов. А данные двух таблиц связаны общими столбцами, а не физическими ссылками или указателями. Объекты и их отношения представлены таблицами.

В реляционных моделях нет необходимости просматривать все указатели, что облегчает выполнение запросов на выборку информации по сравнению с сетевыми и иерархическими БД. Это одна из основных причин, почему реляционная модель оказалась более удобна.

Распространенные реляционные СУБД: MySQL, PostgreSQL, Access, Oracle, DB2, MS-SQL Server, SQLite. Каждая реляционная таблица представляет собой двумерный массив и обладает следующими свойствами:

- каждый элемент таблицы — один элемент данных;
- все элементы в одном столбце имеют одинаковый тип;
- каждый столбец имеет уникальное имя;
- одинаковые строки (записи, кортежи) в таблице отсутствуют;
- порядок следования строк и столбцов может быть произвольным.

Каждое поле содержит одну характеристику объекта предметной области. В записи собраны сведения об одном экземпляре этого объекта.

Некоторые поля могут быть определены как ключевые. Это значит, что для ускорения поиска конкретных значений будет использоваться индексация. Когда поля двух различных таблиц получают данные из одного набора, можно использовать оператор JOIN для выбора связанных записей двух таблиц, сопоставив значения полей. Такие действия можно расширить до объединения нескольких полей в нескольких таблицах. Поскольку отношения здесь определяются только временем поиска, реляционные базы данных классифицируются как динамические системы.

1.3.4 Сравнение моделей

Иерархическая модель данных поддерживает отношения типа «один-к-одному» или «один-ко-многим». Она позволяет быстро получать данные, но не отличается гибкостью. Иногда роль элемента (родителя или потомка) неясна и не подходит для иерархической модели.

Вторая, сетевая модель данных, имеет более гибкую структуру, чем иерархическая, и поддерживает отношение «многие ко многим». Но быстро становится слишком сложной и неудобной для управления.

Третья модель – реляционная – более гибкая, чем иерархическая и проще для управления, чем сетевая. Реляционная модель сегодня используется чаще всего, так как имеет множество преимуществ, таких как:

- простота использования;
- гибкость;
- независимость данных;
- безопасность;
- простота практического применения;
- слияние данных;
- целостность данных.

В связи с этим далее будет рассматриваться реляционная модель. Теперь необходимо рассмотреть систему управления такой моделью.

1.4 СУБД

Самых популярных системы управления реляционными базами данных:

- MySQL;
- PostgreSQL;
- SQLite.

В данном проекте будет рассмотрена СУБД SQLite.

Это компактная встраиваемая СУБД. Слово «встраиваемый» (embedded) означает, что SQLite не использует парадигму клиент-сервер, то есть движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а представляет собой библиотеку, компонуемую с программой, и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций (API) библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу.

Однако SQLite популярна скорее в случаях, когда не требуется выносить базу данных на отдельную машину и данные требуется хранить в рамках одной ОС. Будучи файловой БД, она предоставляет отличный набор инструментов для более простой (в сравнении с серверными БД) обработки любых видов данных. (2)

Преимущества:

- **Файловая.** Вся БД хранится в одном файле, что облегчает перемещение.
- **Стандартизированная.** SQLite использует SQL, некоторые функции не используются (RIGHT OUTER JOIN или FOR EACH STATEMENT);
- **Отсутствие пользовательского.** Продвинутое БД предоставляют пользователям возможность управлять связями в таблицах в соответствии с привилегиями, но у SQLite такой функции нет.
- **Невозможность дополнительной настройки.** SQLite нельзя сделать более производительной, путем изменения настроек.

1.5 Framework

SQLite совместима со множеством фреймворков, которые содержат в себе требуемые методы обращения к БД. Среди возможных вариантов для использования в проекте были выбраны библиотеки sqlite и SQLAlchemy (3).

SQLAlchemy — это программная библиотека на языке Python для работы с реляционными СУБД с применением технологии ORM. Служит для синхронизации объектов Python и записей реляционной базы данных.

В качестве web-framework был выбран Flask, который предоставляет все необходимые инструменты для создания подобного проекта, так как предоставляет возможность для написания как frontend, так и backend для полноценного запуска приложения.

1.6 Вывод

В результате проведенного анализа в качестве модели данных была выбрана реляционная модель, в качестве СУБД — SQLite.

Таким образом, подобрав необходимый набор инструментов для реализации web-приложения, можно приступить к проектированию решения поставленной задачи.

2 Конструкторская часть

В соответствии с техническим заданием и аналитическим разделом должно быть получено полноценное приложение для взаимодействия с БД.

2.1 Проектирование таблиц базы данных

База данных домашней библиотеки состоит из следующих таблиц:

- таблица пользователей сайта **User**;
- таблица книг библиотеки **Book**;
- таблица расположения книг в библиотеке **Location**;
- таблица состояния книги относительно каждого пользователя **Status**.

Таблица User

Позволяет однозначно идентифицировать пользователя сайта, реализовать авторизацию пользователя. Имеет связь «один-ко-многим» с таблицей Status.

Таблица содержит следующие поля:

- id - целочисленное поле, идентификационный номер клиента;
- username - символьное поле, название аккаунта;
- email - символьное поле, адрес электронной почты клиента;
- password hash - символьное поле, хэш пароля пользователя;
- about_me - символьное поле, информация о пользователе;
- last_seen - поле даты, время последнего посещения пользователя;
- image - текстовое поле, хранит аватар пользователя в формате base64.

Таблица Status

Хранит данные о статусе книги относительно пользователя. Связана с таблицами книг Book и пользователей User связью «один-к-одному».

Таблица содержит следующие поля:

- user_id - целочисленное поле, идентификатор пользователя;
- book_id - целочисленное поле, идентификатор книги;
- status - целочисленное поле, состояние книги по отношению к пользователю.

Таблица Book

В данной таблице хранятся данные о книгах. Эта таблица связана с таблицами местоположения книг Location и таблицей статуса Status связью «один-к-одному».

Таблица содержит следующие поля:

- id – целочисленное поле, идентификатор книги;
- title – символьное поле, название книги;
- about_book – символьное поле, информация о книге;
- author – символьное поле, имя автора;
- image – текстовое поле, хранящее изображение книги.

Таблица Location

Таблица, содержащая информацию о расположении книг в библиотеке. С помощью связи «один-к-одному» связана с таблицей книг Book.

Таблица содержит следующие поля:

- id – целочисленное поле, идентификатор локации;
- shelving – целочисленное поле, номер стеллажа;
- shelf – целочисленное поле, номер полки;
- column – целочисленное поле, номер ряда;
- position – целочисленное поле, номер позиции в ряду;
- book_id – целочисленное поле, идентификатор книги.

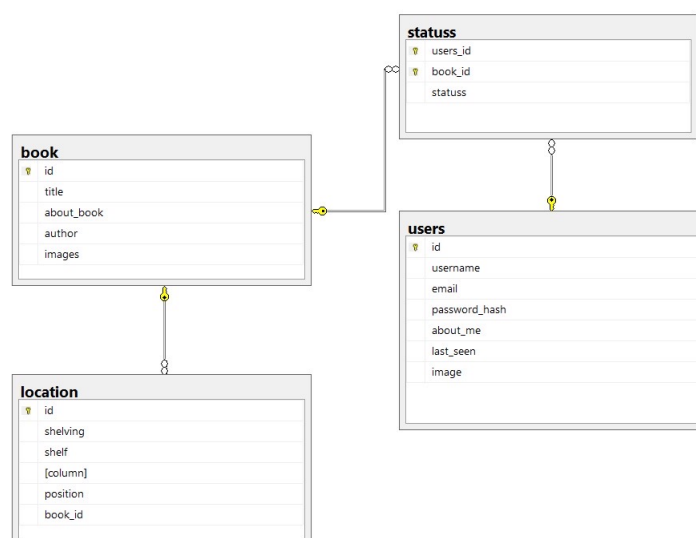


Рис. 3: Диаграмма БД

2.2 Проектирование системы изменения данных

Во Flask каждую таблицу представляет класс. Он отображает информацию о данных сущности. Он содержит поля и поведение данных. С помощью методов класса осуществляются операции добавления, удаления и обновления записей.

2.3 Проектирование регистрации и аутентификации пользователя

Регистрация пользователя в приложении является добавлением в базу данных (в таблицу User) записи, содержащей необходимую информацию для аутентификации. Для этого пользователь вводит соответствующие данные в поля регистрационной формы.

Framework Flask предоставляет собой набор данных базовых инструментов для реализации web-приложения. В этот функционал включена реализация аутентификации пользователя.

2.4 Вывод

Была разработана модель приложения, дающего возможность вести учет книг, реализующего регистрацию, аутентификацию и авторизацию пользователей.

3 Технологическая часть

После проектирования структуры поставленной задачи, требуется реализовать набор функций, необходимый для создания web-приложения, а также конкретизировать полный список инструментов, используемых для запуска приложения.

3.1 Выбор инструментов разработки

В ходе реализации были использованы следующие технологии и средства:

- язык программирования Python;
- СУБД SQLite;
- библиотека Flask.

Такой набор инструментов был выбран, потому что для каждого из элементов предусмотрено взаимодействие с другими. Также данные инструменты полностью выполняют задачи, необходимые для реализации проекта.

Использован компьютер с операционной системой macOS Mojave.

3.2 Реализация хранения данных

Для работы с базой данных требуется объявить классы таблиц БД.

Листинг 1: Класс «Пользователи»

```
1 class User(UserMixin, db.Model):
2     id = db.Column(db.Integer, primary_key = True)
3     username = db.Column(db.String(64), index = True, unique = True)
4     email = db.Column(db.String(128), index = True, unique = True)
5     password_hash = db.Column(db.String(128))
6     about_me = db.Column(db.String(150))
7     last_seen = db.Column(db.DateTime, default = datetime.utcnow)
8     image = db.Column(db.Text)
9
10    book = db.relationship('Book', secondary='status', backref='users',
11                           lazy='dynamic')
12
13    def __repr__(self):
14        return '<User {}>'.format(self.username)
15
16    def set_password(self, password):
17        self.password_hash = generate_password_hash(password)
18
19    def check_password(self, password):
20        return check_password_hash(self.password_hash, password)
21
22    def avatar(self, size):
23        digest = md5(self.email.lower().encode('utf-8')).hexdigest()
24        return 'http://www.gravatar.com/avatar/{}?d=identicon&s={}'.format(digest, size)
25
```

```

26 def get_reset_password_token(self, expires_in=600):
27     return jwt.encode({'reset_password': self.id, 'exp': time() +
28                         expires_in},
29                        app.config['SECRET_KEY'], algorithm='HS256'
30                        ).decode('utf-8')
31
32 def get_book(id):
33     conn = sqlite3.connect("app.db")
34     cursor = conn.cursor()
35     cursor.execute("select title, author, image from book where id
36                     = %d", id)
37     res = cursor.fetchall()
38     print(res)
39     return res
40
41 def get_books(self):
42     conn = sqlite3.connect("app.db")
43     cursor = conn.cursor()
44     cursor.execute("select author, title, username, book.image,
45                     book.id from (user left outer join status on user.id ==
46                     status.user_id) as us \
47                     join book on us.book_id == book.id \
48                     where us.id = (?)", (current_user.id, ))
49     res = cursor.fetchall()
50     return res
51
52 def get_books_count(self):
53     conn = sqlite3.connect("app.db")
54     cursor = conn.cursor()
55     cursor.execute("select count(book.id) from (user left outer
56                     join status on user.id == status.user_id) as us \
57                     join book on us.book_id == book.id \
58                     where us.id = (?)", (current_user.id, ))
59     res = cursor.fetchall()
60     return res[0][0]

```

Листинг 2: Класс «Книги»

```

1 class Book(db.Model):
2     #__searchable__ = ['title']
3     id = db.Column(db.Integer, primary_key = True)
4     title = db.Column(db.String(60), index = True)
5     about_book = db.Column(db.String(150))
6     author = db.Column(db.String(60), index = True)
7     location = db.relationship('Location', uselist=False, backref='
8         books')
9     image = db.Column(db.Text)
10
11     user = db.relationship(
12         'User', secondary='status',
13         backref='books', lazy='dynamic')
14
15 def __repr__(self):
16     return '<Book {}>'.format(self.image)

```

```

16
17 def get_book(self, id):
18     conn = sqlite3.connect("app.db")
19     cursor = conn.cursor()
20     cursor.execute("select title, author, image from book where id
21                     = %d", id)
22     res = cursor.fetchall()
23     print(res)
24     return res
25
26 def search_title(search):
27     conn = sqlite3.connect("app.db")
28     cursor = conn.cursor()
29     cursor.execute("select id, title, author, image from book where
30                     title like (?) ", (search, ))
31     res = cursor.fetchall()
32     conn.close()
33     return res
34
35 def search_author(search):
36     conn = sqlite3.connect("app.db")
37     cursor = conn.cursor()
38     cursor.execute("select id, title, author, image from book where
39                     author like (?) ", (search, ))
40     res = cursor.fetchall()
41     conn.close()
42     return res
43
44 def no_search(search):
45     conn = sqlite3.connect("app.db")
46     cursor = conn.cursor()
47     cursor.execute("select id, title, author, image from book where
48                     author like (?) or title like (?)", (search, search))
49     res = cursor.fetchall()
50     conn.close()
51     return res
52
53 def other_books_author(bookid):
54     conn = sqlite3.connect("app.db")
55     cursor = conn.cursor()
56     cursor.execute("select book.id, book.title, book.author, book.
57                     image, B.title, B.author, B.about_book, B.image, B.shelving,
58                     B.shelf, B.column, B.position, B.id " +
59                     "from ( " +
60                         "select book.id, title, author, about_book,
61                             image, shelving, shelf, column,
62                             position " +
63                         "from book left join location on book.id =
64                             location.book_id where book.id = (?) " +
65                         " " +
66                         ") as B left join book on book.author = B.
67                             author and B.id != book.id;", (bookid, )

```

```

57         res = cursor.fetchall()
58         conn.close()
59         return res
60
61     def get_status_book(bookid):
62         conn = sqlite3.connect("app.db")
63         cursor = conn.cursor()
64         cursor.execute("select status " +
65                        "from ( " +
66                        "select username, book.id, status " +
67                        "from (user join status on user.id ==
68                        status.user_id) as us " +
69                        "join book on us.book_id == book.id" +
70                        ") as res where res.id = (?)", (bookid, ))
71         res = cursor.fetchone()
72         conn.close()
73         return res
74
75     def delete_book(book_id):
76         conn = sqlite3.connect("app.db")
77         cursor = conn.cursor()
78         cursor.execute("delete from status where user_id = (?) and
79                        book_id = (?)", (current_user.id, book_id))
80         conn.commit()
81         cursor.execute("delete from location where book_id = (?)", (
82                        book_id, ))
83         conn.commit()
84         cursor.execute("delete from book where id = (?)", (book_id, ))
85         conn.commit()
86         conn.close()

```

Листинг 3: Класс «Статус»

```

1 class Status(db.Model):
2     __tablename__ = 'status'
3     __table_args__ = (
4         PrimaryKeyConstraint('user_id', 'book_id'),
5     )
6
7     user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
8     book_id = db.Column(db.Integer, db.ForeignKey('book.id'))
9     status = db.Column(db.Integer)
10
11     def set_status(status, book_id, username):
12         conn = sqlite3.connect("app.db")
13         cursor = conn.cursor()
14         cursor.execute("update status set status = (?) " +
15                        "where book_id = (?) and user_id = (?)", (status
16                        , book_id, current_user.id))
17
18         conn.commit()
19         conn.close()
20
21     def join_book(book_id, status):
22         conn = sqlite3.connect("app.db")

```

```

21     cursor = conn.cursor()
22     cursor.execute("INSERT INTO status VALUES ((?), (?), (?))", (
23         current_user.id, book_id, status))
24     conn.commit()
25     conn.close()
26
27 def delete_status(book_id):
28     conn = sqlite3.connect("app.db")
29     cursor = conn.cursor()
30     cursor.execute("delete from status where user_id = (?) and
31         book_id = (?)", (current_user.id, book_id))
32     conn.commit()
33     conn.close()

```

Листинг 4: Класс «Локация»

```

1 class Location(db.Model):
2     id = db.Column(db.Integer, primary_key=True)
3     shelving = db.Column(db.Integer, index=True)
4     shelf = db.Column(db.Integer, index=True)
5     column = db.Column(db.Integer, index=True)
6     position = db.Column(db.Integer, index=True)
7     book_id = db.Column(db.Integer, db.ForeignKey('book.id'))
8
9     def update_location(shelving, shelf, column, position, bookid):
10         conn = sqlite3.connect("app.db")
11         cursor = conn.cursor()
12         cursor.execute("UPDATE location SET shelving = (?), shelf = (?),
13             column = (?), position = (?) WHERE id = (?)", (shelving,
14                 shelf, column, position, bookid))
15         conn.commit()
16         conn.close()

```

3.3 Реализация доступа к данным

Чтобы обеспечить доступ к данным нужно создать форму, позволяющую добавлять и изменять записи в таблицах.

Центр данного механизма – класс `Form`, которая описывает структуру объекта, его поведение и представление. `FlaskForm` отображает поля в виде HTML input. Поля формы являются классами. Они управляют данными формы. Например, `StringField`, `PasswordField`, `IntegerField` работают с разными данными.

Реализация формы для доступа к данным на примере работы с книгами представлена в листинге 5.

Класс `FlaskForm` требует только указания полей этой формы. По эти данным `Flask` сгенерирует поля формы нужного типа.

Листинг 5: Форма добавления книги

```

1 class AddBookForm(FlaskForm):
2     title = StringField('Название книги', validators=[DataRequired()])
3     image = FileField("Обложка книги")
4     about_book = TextAreaField('О книге', validators=[Length(min=0, max=300)])

```

```

5     author = StringField('Автор', validators=[DataRequired()])
6     shelving = IntegerField('Стелаж', validators=[DataRequired()])
7     shelf = IntegerField('Полка', validators=[DataRequired()])
8     column = IntegerField('Ряд', validators=[DataRequired()])
9     position = IntegerField('Позиция', validators=[DataRequired()])
10    submit = SubmitField('Добавить')
11
12    def validate_shelving(self, shelving):
13        res = Location.query.filter_by(shelving=shelving.data, shelf=
            self.shelf.data, column=self.column.data, position=self.
            position.data).first()
14        if res is not None:
15            raise ValidationError('Пожалуйста, используйте другое место .')

```

3.4 Frontend-разработка

Пользовательский интерфейс при разработке web-приложения представляет из себя полноценную верстку проекта. Для этого использовались технологии Bootstrap, JQuery и Ajax.

Bootstrap - это инструмент с открытым исходным кодом для разработки web-приложений с помощью HTML, CSS и JS. Включает в себя HTML- и CSS-шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения. С помощью него настроен дизайн сайта. (4)

jQuery - библиотека JavaScript фокусирующаяся на взаимодействии JavaScript и HTML. Ее функционал позволяет создавать обработчики событий, помогает легко получать доступ к любому элементу, обращаться к атрибутам и содержимому элементов, манипулировать ими. Также библиотека jQuery предоставляет удобный API для работы с AJAX. Разработка jQuery ведется командой добровольцев на пожертвования. (5)

AJAX (Asynchronous JavaScript And Xml) – технология обращения к серверу без перезагрузки страницы. За счет этого уменьшается время отклика и веб-приложение по интерактивности больше напоминает десктоп. Сейчас в порядке вещей, что многие вещи на сайтах осуществляются без перезагрузки страницы. Технически, с помощью AJAX можно обмениваться любыми данными с сервером. Эта технология была использована для динамического обновления таблиц, динамической проверки и отправки форм. (6)

Flask предоставляет инструмент шаблонизатора, который дает возможность вносить динамические данные в html с backend. С помощью шаблонизатора есть возможность проверять данные, изменяя элементы страницы в зависимости от результата проверки.

При рендеринге шаблона переменные в двойных фигурных скобках будут заменяться на вычисленные значения.

Функция `render_template()`, отображающая шаблон, вызывает механизм шаблонов `jinja2`, который поставляется в комплекте с Flask. Jinja2 заменяет `{{ ... }}` блоки соответствующими значениями, заданными аргументами, указанными в `render_template()` вызове.

Расширение Flask-WTF использует классы Python для представления веб-форм. Класс формы просто определяет поля формы как переменные класса.

Продemonстрируем действие шаблонизатора на примере шаблона аутентификации в листинге 6. Шаблон использует значение `form`, полученное из контекста, переданного в представлении.

Листинг 6: Шаблон `login.html`

```
1 {% extends "base.html" %}
2 {% import 'bootstrap/wtf.html' as wtf %}
3
4 {% block app_content %}
5     <div class="container" style="margin: 0 30%">
6         <h1Вход></h1>
7         <div class="row">
8             <div class="col-md-4">
9                 {{ wtf.quick_form(form) }}
10            </div>
11        </div>
12        <p> Новый пользователь ? <a href="{{ url_for('register') }}">Регистрация</a>
13    </p>
14 </div>
15 {% endblock %}
```

Шаблоны также поддерживают операторы управления, заданные внутри `{ % ... % }` блоков. Все шаблоны используют базовый шаблон `{% extends "base.html"%}` для избежания дублирования кода.

Листинг 7: Шаблон `base.html`

```
1 {% extends "base.html" %}
2 {% import 'bootstrap/wtf.html' as wtf %}
3
4 {% block app_content %}
5     <div class="container" style="margin: 0 30%">
6         <h1Вход></h1>
7         <div class="row">
8             <div class="col-md-4">
9                 {{ wtf.quick_form(form) }}
10            </div>
11        </div>
12        <p>Новый< /> пользователь? <a href="{{ url_for('register') }}">Регистрация</a>
13    </p>
14 </div>
15 {% endblock %}
```

Для взаимодействия с backend используются ajax-запросы, которые не требуют обновления страницы для получения данных с сервера.

Листинг 8: Пример части кода с использованием ajax-запроса для изменения статуса книги в БД

```
1 $(".item").click(function() {
2     ...
3     $.ajax({
4         type: "PUT",
5         url: "http://localhost:5000/set_status",
6         data: JSON.stringify({"info": {"status": $(this).val(), "
            book_id": {{ book.book_id }} } }) ,
```

```

7 |         contentType: "application/json; charset=utf-8",
8 |         });
9 |     });

```

В данном фрагменте присутствует работа с библиотекой jQuery. Она начинается с вызова основной функции jQuery() или \$(). ".item" – это селектор, с помощью которого происходит доступ к элементу с class="item".

Ajax-запрос вызывается со следующими параметрами:

- type – определяет тип запроса;
- url – адрес, на который будет отправлен запрос;
- data – данные, которые будут переданы на сервер;
- contentType – формат, в котором передаются данные на сервер.

3.5 Интерфейс приложения

Все незарегистрированные пользователи автоматически перенаправляются на страницу входа с которой можно перейти к регистрации. (рис. 4).

Рис. 4: Страницы авторизации и регистрации

На рисунке 5 представлена страница карточки книги, находящихся в библиотеке. На ней есть возможность выбрать состояние книги относительно пользователя:

- хочу прочитать;
- в процессе;
- прочитано.

Также присутствует возможность сбросить состояние, удалить книгу и изменить местоположение книги в библиотеке.

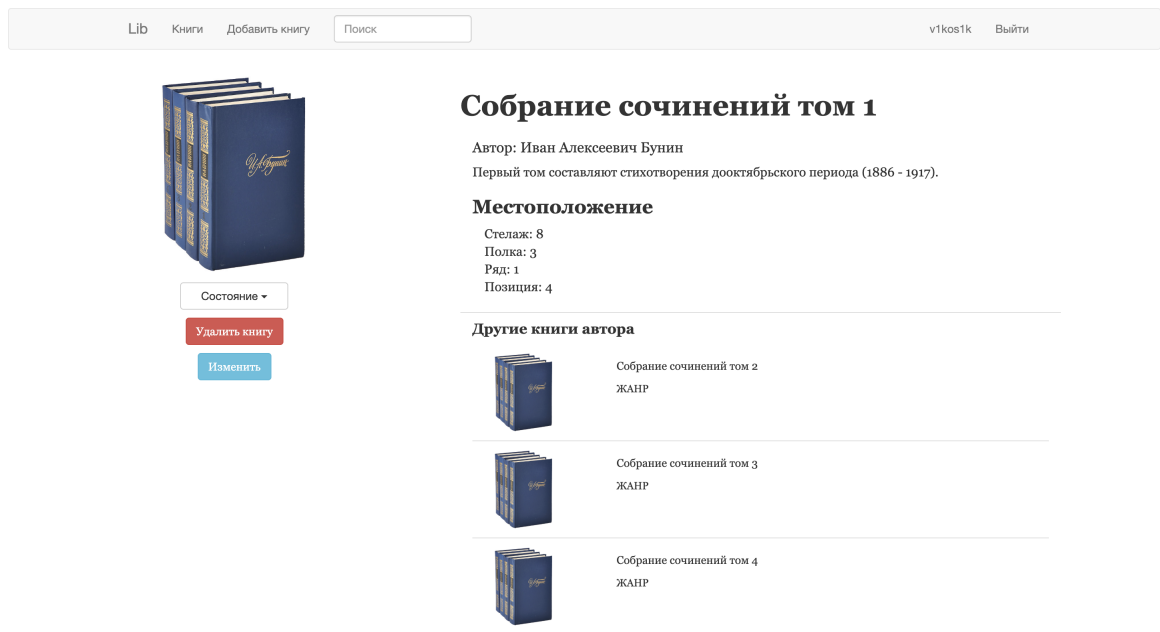


Рис. 5: Страница карточки книги

На рисунке 6 показана страница реализации поиска по автору и/или названию.

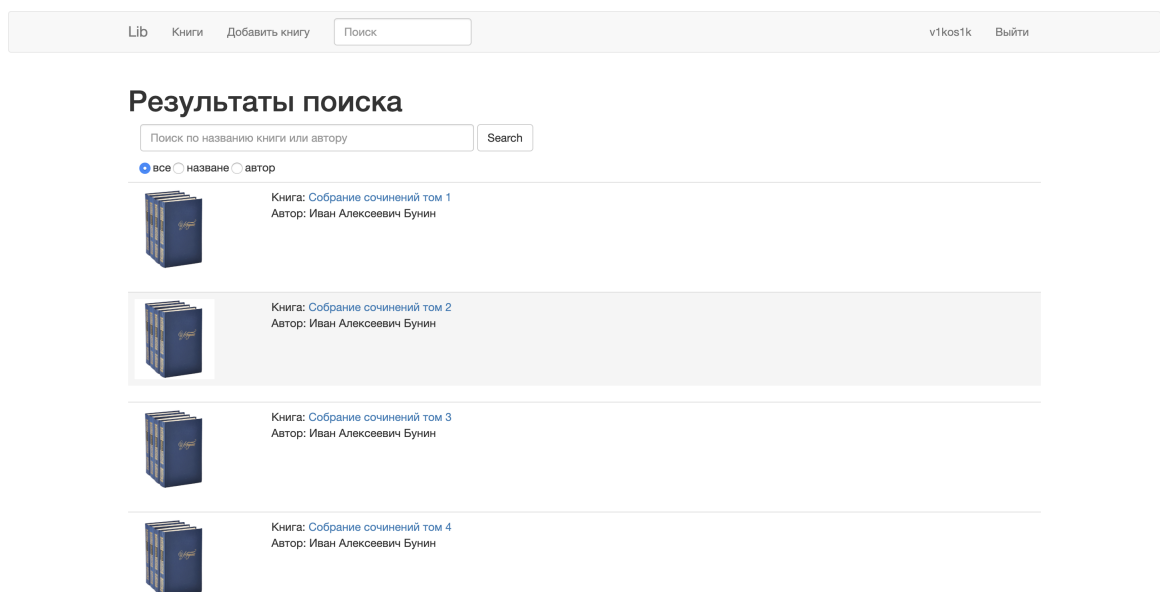


Рис. 6: Страница поиска

Страница с формой добавления книги представлена на рисунке 7.

Lib Книги Добавить книгу Поиск

v1kos1k Выйти

Новая книга

Название книги

Обложка книги

Выберите файл Файл не выбран

О книге

Автор

Стелаж

Полка

Ряд

Позиция


Добавить

Рис. 7: Страница добавления книги


В верхней части личного кабинета (рис. 8) пользователя расположена его личная информация и количество книг, которые с ним связаны. В нижней части находится список этих книг. С этой страницы можно удалить связь с книгой, нажав на крестик в правом верхнем углу блока книги.

Lib Книги Добавить книгу Поиск

v1kos1k Выйти




v1kos1k
sleeeeeeeeeer 000
Last seen on: 2019-09-20 13:38:05.730677
2 books.
[Редактировать профиль](#)



Книга: [Собрание сочинений том 2](#)
Автор: Иван Алексеевич Бунин

✕



Книга: [Долг](#)
Автор: Абдижамил Нурпеисов

✕

Рис. 8: Личный кабинет

Заключение

В результате проделанной работы:

- были продуманы функции, которые должно было решать приложение;
- проведен анализ инструментов, необходимых для проектирования и реализации задачи, в результате которого были выбраны такие инструменты, как SQLite, Flask;
- разработана структура базы данных, состоящая из нескольких сущностей;
- с помощью выбранных инструментов был реализован web-интерфейс, обладающий возможностью регистрировать пользователей, изменять состояние и местоположение книг, добавлять и удалять книги, а также осуществлять поиск по ключевым словам.

Список использованных источников

- [1] Интернет технологии [Электронный ресурс]
URL: <https://www.internet-technologies.ru/articles/modeli-baz-dannyh-sistemy-upravleniya-bazami-dannyh.html> (Дата обращения: 19.09.2019)
- [2] Сайт разработчика базы данных Sqlite [Электронный ресурс] URL: <https://www.sqlite.org> (Дата обращения: 19.09.2019)
- [3] Сайт разработчика SQLAlchemy [Электронный ресурс] URL: <https://www.sqlalchemy.org> (Дата обращения: 19.09.2019)
- [4] Bootstrap [Электронный ресурс] URL: <https://bootstrap-4.ru> (Дата обращения: 19.09.2019)
- [5] jQuery [Электронный ресурс] URL: <https://jquery.com> (Дата обращения: 19.09.2019)
- [6] AJAX [Электронный ресурс] URL: <https://learn.javascript.ru/ajax-intro> (Дата обращения: 19.09.2019)

Приложение

А. Дополнительный функционал

Добавлен новый функционал приложения:

- при нажатии "в процессе" у занятой книги появляется соответствующее уведомление с именем пользователя, у которого находится книга;
- на главной и поисковой страницах отображается то, что книга занята и кем;
- незарегистрированный пользователь может просматривать содержимое библиотеки и пользоваться поиском.

Листинг 9: Новые функции класса Status

```
1 def check_status_equal_two(book_id):
2     conn = sqlite3.connect("app.db")
3     cursor = conn.cursor()
4     cursor.execute("select S.status, username \
5                     from ( \
6                         select status, user_id \
7                         from status \
8                         where book_id = (?) and status = 2 \
9                     ) as S join user on S.user_id = user.id \
10                    where id != (?);", (book_id, current_user.
11                                     id))
12     res = cursor.fetchone()
13     conn.close()
14     return res
15
16 def check_all_status_equal_two():
17     conn = sqlite3.connect("app.db")
18     cursor = conn.cursor()
19     cursor.execute("select S.status, username, S.book_id \
20                    from ( \
21                        select status, user_id, book_id \
22                        from status \
23                        where status = 2 \
24                    ) as S join user on S.user_id = user.id;")
25     res = cursor.fetchall()
26     conn.close()
27     return res
```

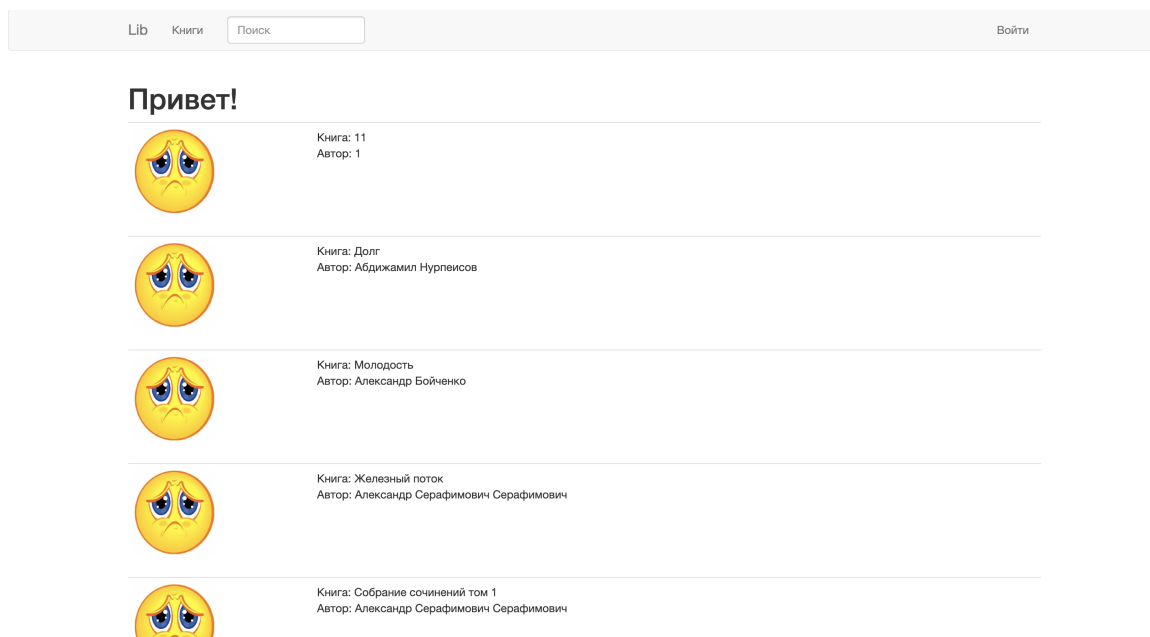


Рис. 9: Вид главной страницы для неавторизованного пользователя

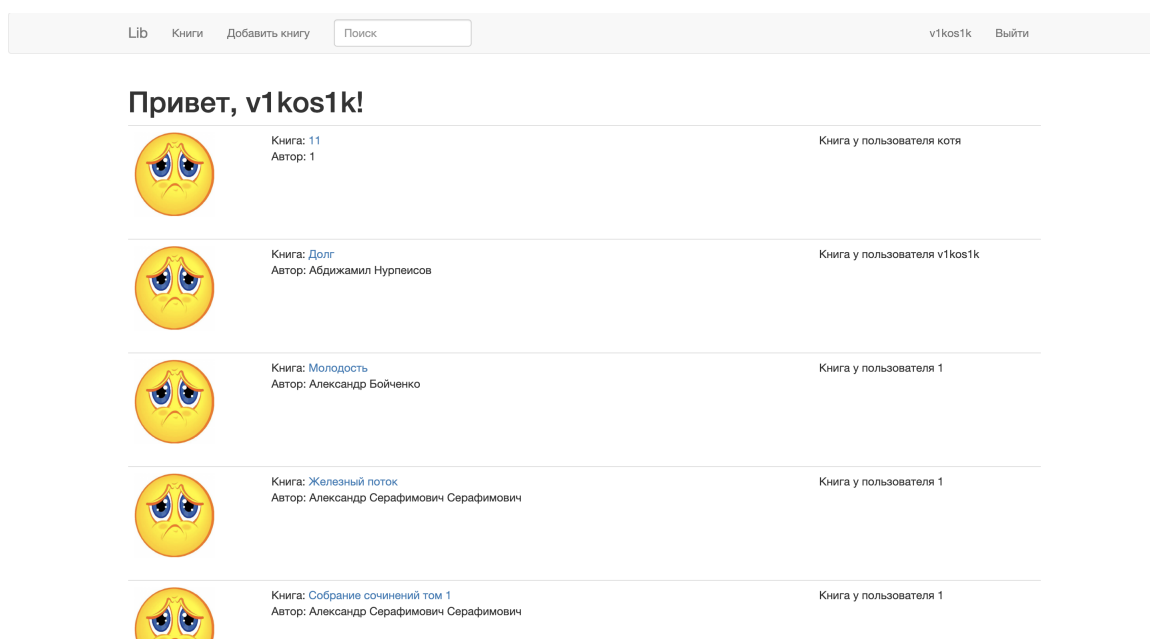


Рис. 10: Вид главной страницы для авторизованного пользователя

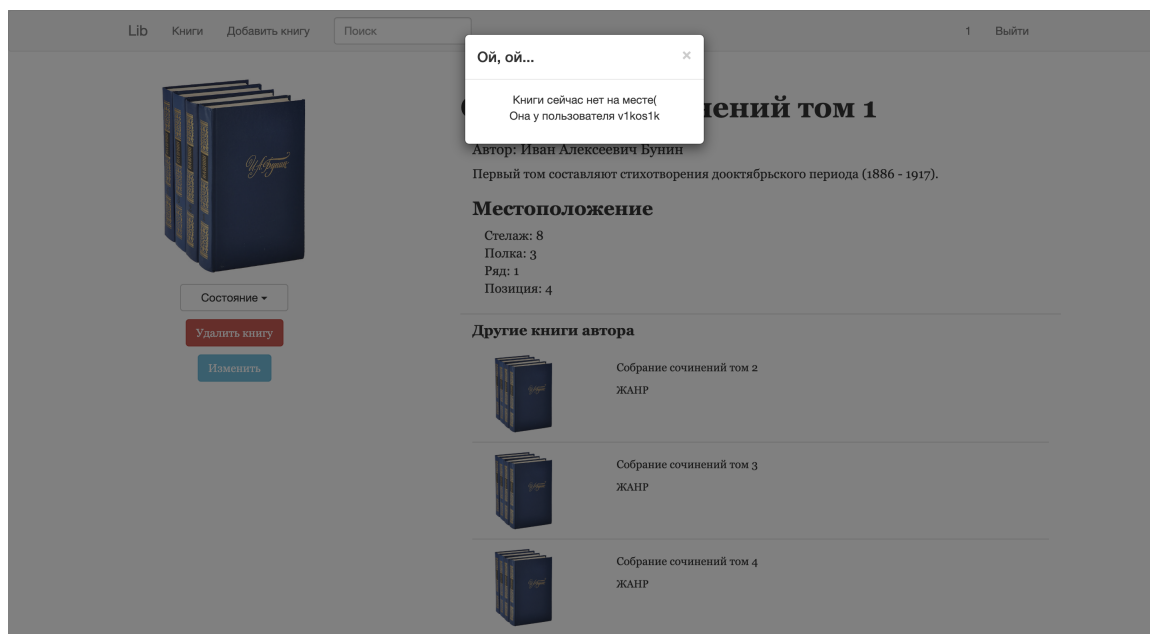


Рис. 11: Пример попытки добавления занятой книги

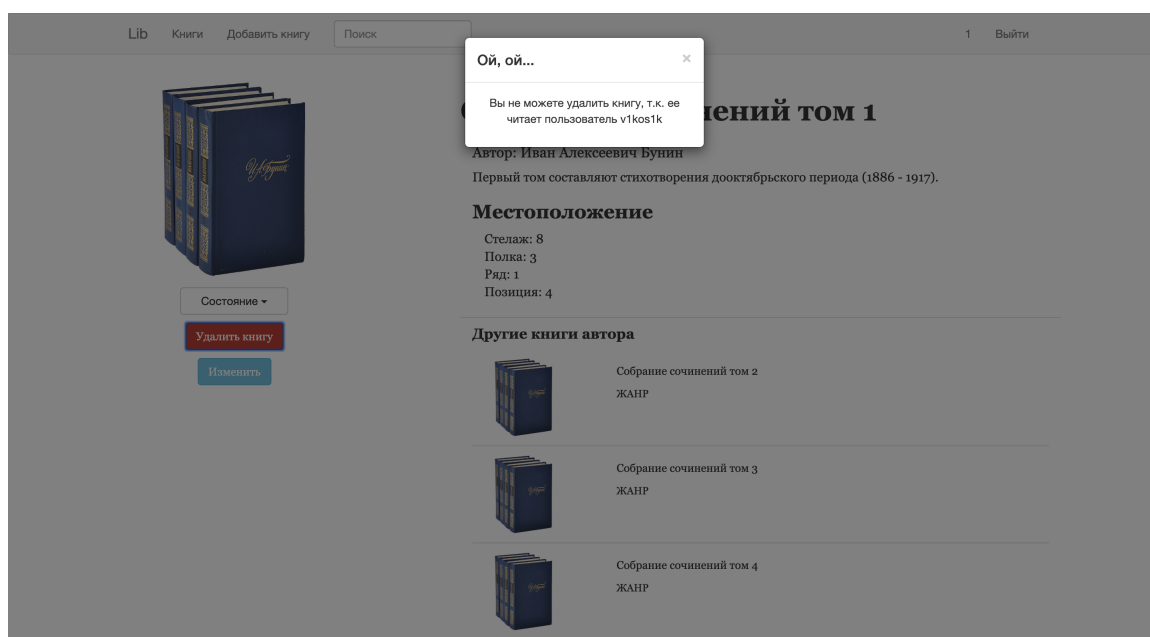


Рис. 12: Пример попытки удаления занятой книги