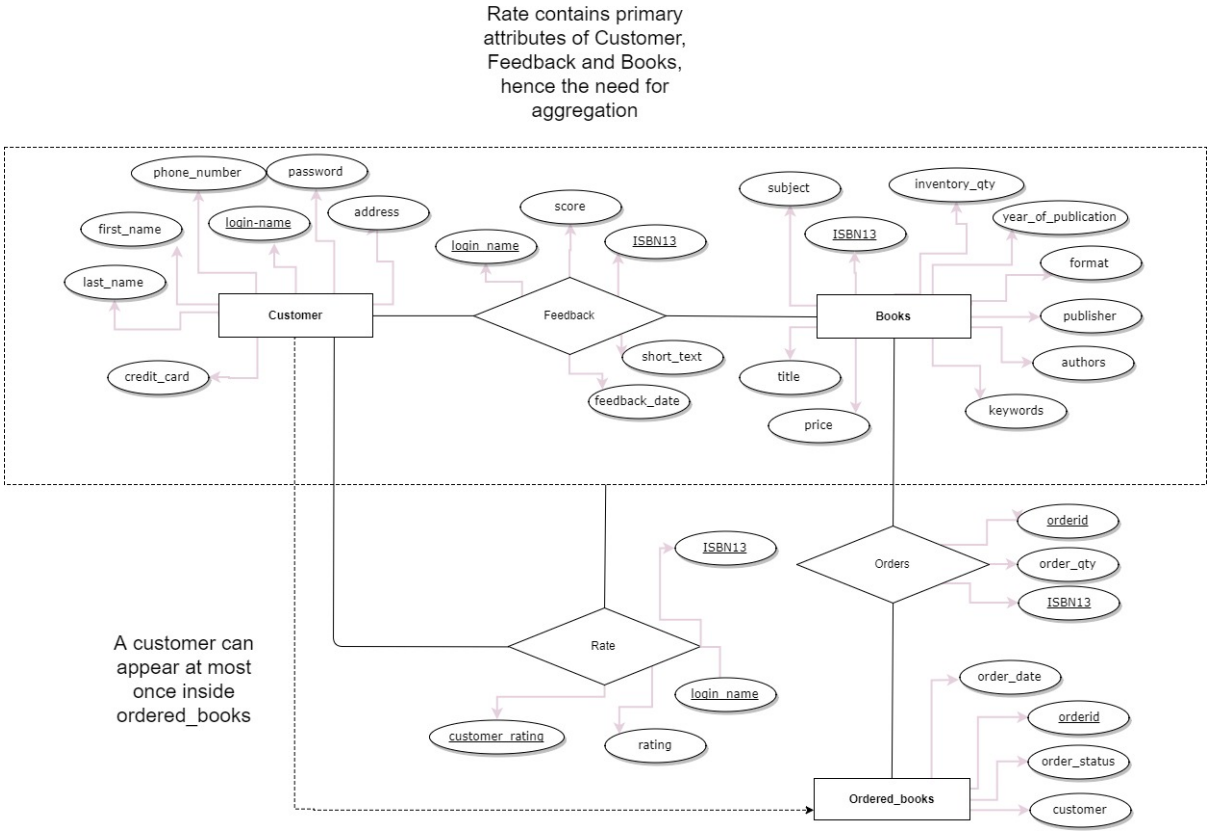
Database Design Project Final Report

Sidney Suen (1001525)
Archit Atul Date (1001695)
Amish Bhandari (1001614)
Arshi Dalvi (1001768)
Nickson Guay (1000998)

ER Diagram



Notes:

- Deleting customers was not specified in the project specs; most websites would archive all customer data nonetheless.
- Ordered_books is a table mapping a user to an ordered, which uniquely identifies an order. Hence, a username corresponds to at most 1 orderid.
- For more details in running the code, please refer to the README file.

Relational Schema (SQL DDL Code)

```
#create database bookstore;
use bookstore;

create table Books (
        ISBN13 char(17),
   title char(50),
   authors char(50),
   publisher char(50),
   year_of_publication integer,
   inventory_qty integer,
   price numeric(6,2),
   format char(9) check (format = 'hardcover' or format = 'softcover'),
   keywords char(20),
   subject char(20),
   primary key (ISBN13));

create table Customers (
        login_name char(20),
   password char(20),
   first_name char(20),
   last_name char(20),
   credit_card char(20),
   address char(50),
   phone_number char(20),
   primary key (login_name));

create table Feedback (
                ISBN13 char(17),
                login_name char(20),
   score integer,
   short_text char (100),
   feedback_date date,
   primary key (ISBN13, login_name),
   foreign key (ISBN13) references Books(ISBN13),
   foreign key (login_name) references Customers(login_name));

create table Rate (
                login_name char(20),
   ISBN13 char(17),
        customer_rating char(20),
   rating integer check (rating = 0 or rating = 1 or rating = 2),
                CONSTRAINT chk_same CHECK (login_name <> customer_rating),
   primary key (ISBN13,login_name,customer_rating),
   foreign key (customer_rating) references Customers(login_name),
   foreign key (login_name,ISBN13) references Feedback(login_name, ISBN13));
```

```
create table Ordered_books (
        orderid integer,
   customer char(20) not null,
   order_date date,
   order_status char(10),
   primary key (orderid),
   foreign key (customer) references Customers(login_name));

create table Orders (
                orderid integer,
                ISBN13 char(17),
                order_qty integer,
                primary key (orderid,ISBN13),
                foreign key (orderid) references Ordered_books(orderid),
                foreign key (ISBN13) references Books(ISBN13));
```

Implementation

1. Website visitor can register and login. Based on credentials they can either login as a user or a manager. The login name is checked for uniqueness.

```
@app.route('/registration/', methods=['POST'])
def registration_post():
        if request.form['my-form'] == 'register':
                first_name = request.form['first_name'].strip()
                last_name = request.form['last_name'].strip()
                login_name = request.form['username'].strip()
                password = request.form['password'].strip()
                address = request.form['address'].strip()
                credit_card = request.form['ccno'].strip()
                phone_number = request.form['phone'].strip()
                newcustomer = customers(login_name, password, first_name, last_name, credit_card, address, phone_number)
                try:
                        db.session.add(newcustomer)
                        db.session.commit()
                        session['logged_in'] = True
                        session['login_name'] = login_name
                        return redirect('/')
                except IntegrityError:
                        return render_template('registration.html', regerror='Error: Duplicate username found, please try ano
        elif request.form['my-form'] == 'back':
                return redirect('/')
```

```
@app.route('/login/', methods=['POST'])
def do_admin_login():
        POST_USERNAME = str(request.form['login_name'].strip())
        POST_PASSWORD = str(request.form['password'].strip())

        Session = sessionmaker(bind=engine)
        s = Session()
        query = s.query(customers).filter(customers.login_name.in_([POST_USERNAME]), customers.password.in_([POST_PASSWORD]) )
        result = query.first()
        if result:
                flash("successfully logged in")
                session['logged_in'] = True
                session['login_name'] = POST_USERNAME
        else:
                return render_template('login.html', loginerror='''<font color = "white">Error: Wrong username or password.</font>''')
        return redirect('/')
```

2. After registration, a user can order one or more books. A user may order multiple copies of a book, one or more times.

```
def order_post():
        Session = sessionmaker(bind=engine)
        s = Session()
        recolist = []
        index = 0
        manager = ''
        try:
                orderid = 1
                for a in db.engine.execute("select orderid+1 from orders order by orderid desc limit 1;"):
                        orderid = a[0]
        except:
                orderid = 1
        status = 'arrived'
        date = time.strftime("%Y-%m-%d")
        customer = ''
        isbn13str = request.form['isbn13']
        isbn13list = isbn13str.split(',')
        copiesstr = request.form['copies']
        copieslist = copiesstr.split(',')

        for k in copieslist:
                if int(k) <= 0 or k == '':
                        return render_template('bookpage.html', booktable='Invalid quantities for order, please try again.', manager=ma

        for j in isbn13list:
                if j == '':
                        return render_template('bookpage.html', booktable='Wrong format of entries for order, please try again.', manag
                toorder = db.engine.execute("select * from books where isbn13 = '{}';".format(j))
                if toorder == None:
                        return render_template('bookpage.html', booktable='One or more ISBN13 you entered is/are not valid, please try
```

```
    if len(isbn13list) != len(copieslist):
            return render_template('bookpage.html', booktable='Wrong format of entries for order, please try again.', manager=manager

    if 'login_name' in session:
            Login_name=session['login_name']
            customer = Login_name
            if customer == 'manager':
                    manager = '<a class="nav-item nav-link" href="/manager">Manager</a>'
```

3. User records: Upon user demand, you should print the full record of a user:
    - his/her account information
    - his/her full history of orders (book name, number of copies, date etc.)
    - his/her full history of feedbacks
    - the list of all the feedbacks he/she ranked with respect to usefulness

```
@app.route('/getrecord/', methods=['GET'])
def getrecord():
        if not session.get('logged_in'):
                return redirect('/login')
        username = ''
        if 'login_name' in session:
                Login_name=session['login_name']
                username = Login_name
        rs = []
        qresult = db.engine.execute("select * from Customers where login_name='%s'"%username)
        for row in qresult:
                rs.append(row)
        info1 = rs[0]

        rs = []
        qresult = db.engine.execute("select b.title, o.isbn13, ob.orderid, ob.order_date, ob.order_status, o.order_qty from ordered_books
        for row in qresult:
                rs.append(row)
        orderlist = rs
        ordertable = OrderTable(orderlist)

        rs = []
        qresult = db.engine.execute("select t1.login_name, t1.title, t1.isbn13, t1.score, t1.short_text, t1.feedback_date, t2.avg_rating
        for row in qresult:
                rs.append(row)
        feedbacklist = rs
        feedbacktable = FeedbackTable(feedbacklist)

        rs = []
        qresult = db.engine.execute("select b.title, r.isbn13, r.login_name, r.rating from Books b, Rate r where (r.customer_rating = '{}
        for row in qresult:
                rs.append(row)
        ratelist = rs
        ratetable = RatingTable(ratelist)

        if username == 'manager':
                return render_template('userrecord.html', username=info1[0], password=info1[1], first_name=info1[2], last_name=info1[3],
        return render_template('userrecord.html', username=info1[0], password=info1[1], first_name=info1[2], last_name=info1[3], credit_c
```

4. New book: The store manager can add new book into the database along with its details.

```
@app.route('/manager/recordnew/', methods=['GET'])
def recordnew():
        username = session['login_name']
        if username != 'manager':
                return redirect(url_for('index'))
        return render_template('recordnew.html', recerror='')

@app.route('/manager/recordnew/', methods=['POST'])
def recordnew_post():
        title = request.form['title']
        isbn13 = request.form['isbn13']
        authors = request.form['authors']
        publisher = request.form['publisher']
        year_of_publication = request.form['year']
        inventory_qty = request.form['copies']
        price = request.form['price']
        book_format = request.form['format']
        keywords = request.form['keywords']
        subject = request.form['subject']
        try:
                db.engine.execute("insert into books values ('{}','{}','{}','{}','{}','{}','{}','{}','{}','{}');".format(isbn13, title, a
                return render_template('manager.html', record='Successfully recorded new book.', add='')
        except IntegrityError:
                return render_template('recordnew.html', recerror='Duplicate ISBN13, please check the book details again.')
```

5. The manager can increment the number of books in the library

```
@app.route('/manager/addcopy/', methods=['GET'])
def addcopy():
        username = session['login_name']
        if username != 'manager':
                return redirect(url_for('index'))
        return render_template('addcopy.html')

@app.route('/manager/addcopy/', methods=['POST'])
def addcopy_post():
        try:
                isbn13 = request.form['isbn13']
                copies = request.form['copies']
                db.engine.execute("update books set inventory_qty = inventory_qty + {} where isbn13 = '{}';".format(copies, isbn13))
                return render_template('manager.html', record='', add='Successfully added copies to book.')
        except:
                return render_template('manager.html', record='', add='The book for the ISBN13 doesn\'t exist, please check your entries
```

6. Feedback recordings: Users can record their feedback for a book. Details like date, numerical score and comments can be recorded. No changes is allowed and only one feedback per user is allowed.

```
elif request.form['my-form'] == 'Feedback':
        isbn13Form = request.form['feedback_isbn13']
        scoreForm = request.form['score']
        commentForm = request.form['comment']
        date = time.strftime("%Y-%m-%d")
        login_name=session['login_name']
        try:
                db.engine.execute("insert into feedback (isbn13, login_name, score, short_text, feedback_date) values ('{}','{}','{}',
                success = "Your feedback for this book has been recorded successfully"
        except IntegrityError:
                return render_template('bookpage.html', booktable='You have already rated this book before or you specified an invalic
        except Exception:
                return render_template('bookpage.html', booktable='Something went wrong, please try again', manager=manager)
        return render_template('bookpage.html', booktable=success, manager=manager)
```

7. Users can rate the usefulness of other's feedback. Users are not allowed to rate their own feedback, and are not allowed to rate a feedback twice.

```
elif request.form['my-form'] == 'Rate':
        login_nameForm = request.form['login_name']
        isbn13Form = str(request.form['rate_isbn13'])
        rateForm = int(request.form['rating'])
        login_name=session['login_name']
        if login_nameForm == login_name:
                return render_template('bookpage.html', booktable='ERROR: You are not allowed to rate your own feedback')
        try:
                db.engine.execute("insert into rate (login_name, isbn13, customer_rating, rating) values ('{}','{}','{}','{}')".format(
                success = '''Your rating for <font color = "red">''' + login_nameForm + "'s feedback for this book</font> has been reco
        except IntegrityError:
                return render_template('bookpage.html', booktable='ERROR: You have already rated this feedback before or you specified a
        except Exception:
                return render_template('bookpage.html', booktable='Something went wrong, please try again')
        return render_template('bookpage.html', booktable=success)

return render_template('bookpage.html', booktable=booktable.__html__(), manager=manager)
```

8. User will be able to browse books, and request for which books using conjunctive queries on the authors, and/or publisher, and/or title, and/or subject. Users can also specify results to be sorted by a) year, or b) by the average score of the feedbacks

```
@app.route('/browse/', methods=['POST'])
def browse_post():
        Login_name=session['login_name']
        username = Login_name
        manager = ''
        if username == 'manager':
                manager = '<a class="nav-item nav-link" href="/manager">Manager</a>'

        if request.form['my-form'] == 'search':
                authorForm = request.form['author']
                publisherForm = request.form['publisher']
                titleForm = request.form['title']
                subjectForm = request.form['subject']
                wherequery = " where"
                if authorForm:
                        wherequery += " bo.authors = '{}' and".format(authorForm)
                if publisherForm:
                        wherequery += " bo.publisher = '{}' and".format(publisherForm)
                if titleForm:
                        wherequery += " bo.title = '{}' and".format(titleForm)
                if subjectForm:
                        wherequery += " bo.subject = '{}'".format(subjectForm)
                if wherequery == " where":
                        wherequery = ""
                if wherequery[-3:] =="and":
                        wherequery = wherequery[:-3]

                optionForm = request.form['options']

    # sort by year, descending order
    if optionForm == 'year':
            sort_order = 'year_of_publication'
    # sort by score, descending order
    elif optionForm == 'score':
            sort_order = 'avgscore'

    sqlquery = "select b.isbn13, b.title, b.authors, b.publisher, b.year_of_publication,
    b.inventory_qty, b.price, b.format as bookformat, b.keywords, b.subject,
    c.avgscore from (select bo.isbn13, bo.title, bo.authors, bo.publisher, bo.year_of_publication,
                    bo.inventory_qty, bo.price, bo.format, bo.keywords, bo.subject from Books bo{})
    as b left outer join (select avg(score) as avgscore, isbn13 from feedback group by isbn13)
    as c on b.isbn13 = c.isbn13 order by {} desc;".format(wherequery, sort_order)

    print(sqlquery)

    booklist = []
    qresult = db.engine.execute(sqlquery)
    for row in qresult:
            booklist.append(row)

    booktable = BrowseTable(booklist)
    return render_template('bookpage.html', booktable='<h2>Browse Results</h2> <br>'+booktable.__html__(),
                           manager=manager)
```

9. For a given book, user can ask for top n most useful feedback. N is specified by the user.

```
# Question 9

elif request.form['my-form'] == 'Get Top Feedback':
        isbn13Form = request.form['topfeedback_isbn13']
        limitForm = request.form['topfeedback']
        login_name=session['login_name']
        feedbackList = []
        qresult = db.engine.execute("select t1.login_name, t1.title, t1.isbn13, t1.score, t1.short_text as short_text,
                                    t1.feedback_date, t2.avg_rating
                                    from (select fb.login_name, b.title, fb.isbn13, fb.score, fb.short_text,
                                          fb.feedback_date from Feedback fb, Books b where fb.isbn13 = '{}'
                                          and b.isbn13 = fb.isbn13)
                                    as t1 left outer join (select login_name, isbn13, avg(rating)
                                                           as avg_rating from Rate where isbn13 = '978-1501138003'
                                                           group by login_name, isbn13) as t2 on
                                    t1.login_name = t2.login_name
                                    order by t2.avg_rating desc limit {};".format(isbn13Form, limitForm))
        for row in qresult:
                feedbackList.append(row)
        feedbacktable = FeedbackTable(feedbackList)
        return render_template('bookpage.html', booktable='<h3>Top '+limitForm+' Feedback for the book</h3> <br>'
                               +feedbacktable.__html__(), manager=manager)
```

10. Book recommendation – when user orders copy of a book say book A, the website recommends a list of other suggested books. A book B is suggested if there exist a user that have purchased both book A and book B. The suggested books are sorted based on decreasing sales count.

```
newob = []
newo = []

while index < len(isbn13list):
        try:
                isbn13 = isbn13list[index]
                copies = int(copieslist[index])
                for rs in db.engine.execute("select inventory_qty from Books where isbn13 = '{}'".format(isbn13)):
                        book_curr_qty = rs[0]
                tempqty = int(book_curr_qty) - copies
                if tempqty < 0:
                        return render_template('bookpage.html', booktable='Sorry, one or more books you ordered is/are
                                               out of stock or you have ordered more than the available quantity.',
                                               manager=manager)
                db.engine.execute("update books set inventory_qty = {} where isbn13 = '{}'".format(tempqty, isbn13))
                db.engine.execute("insert into Ordered_books (orderid, customer, order_date, order_status)
                                  values ('{}','{}',DATE '{}','{}');".format(orderid, customer, date, status))
                db.engine.execute("insert into Orders values ('{}','{}','{}');".format(orderid, isbn13, copies))
                recom = db.engine.execute("select title, isbn13 from books where isbn13
                                          in (select isbn13 from orders where isbn13 <> '{}'
                                              AND orderid in (select orderid from ordered_books where customer
                                                              in (select customer from ordered_books where orderid
                                                                  in (select orderid from orders where isbn13 = '{}')))
                                              group by isbn13 order by sum(order_qty) desc);".format(isbn13,isbn13))
                for rc in recom:
                        if rc not in recolist:
                                recolist.append(rc)
                index += 1
                orderid += 1
        except Exception:
                return render_template('bookpage.html', booktable='Something went wrong, please check your order again.',
                                       manager=manager)
reco = RecTable(recolist)
return render_template('recommendation.html', recommendation=reco.__html__(), manager=manager)
```

11. Statistics – Every month the store manager wants
    - The list of the m most popular books (copies sold this month)
    - The list of m most popular authors
    - The list of m most popular publishers

```
473  @app.route('/manager/statistics', methods=['POST'])
474  def statistics():
475      date = time.strftime("%Y-%m-%d")
476
477      m = int(request.form['top'])
478      if m>5000:
479          return render_template('manager.html', record='', add='m value is too large, please try a smaller value!')
480      month = request.form['month']
481      year = request.form['year']
482      titlelist = []
483      authorlist = []
484      publisherlist = []
485      statslist = []
486
487      db.engine.execute("create table temp_table select ISBN13 , sum(order_qty) as total_qty from orders where orderid in (select orderid from ord
488
489      titlestat = db.engine.execute("select title from books join temp_table on books.ISBN13 = temp_table.ISBN13;")
490      for ts in titlestat:
491          titlelist.append(ts.title)
492
493      authorstat = db.engine.execute("select authors from books join temp_table on books.ISBN13 = temp_table.ISBN13;")
494      for ast in authorstat:
495          authorlist.append(ast.authors)
496
497      publisherstat = db.engine.execute("select publisher from books join temp_table on books.ISBN13 = temp_table.ISBN13;")
498      for ps in publisherstat:
499          publisherlist.append(ps.publisher)
500
501      for i in range(0,m):
502          try:
503              arg1 = titlelist[i]
504          except IndexError:
505              arg1 = ''
506          try:
507              arg2 = authorlist[i]
508          except IndexError:
509              arg2 = ''
510          try:
511              arg3 = publisherlist[i]
512          except IndexError:
513              arg3 = ''
514          statslist.append(StatTableEntry(arg1, arg2, arg3))
515
516      stats = StatTable(statslist)
517      db.engine.execute("drop table temp_table")
518
519      return render_template('statistics.html', stats=stats.__html__())
520
```

Executing the SQL query (line 487):
db.engine.execute("create table temp_table select ISBN13 , sum(order_qty) as total_qty from orders where orderid in (select orderid from ordered_books where year(order_date) = '%s' and month(order_date) = '%s') group by ISBN13 order by total_qty desc limit %s" % (year, month, m))

Screen Dumps:

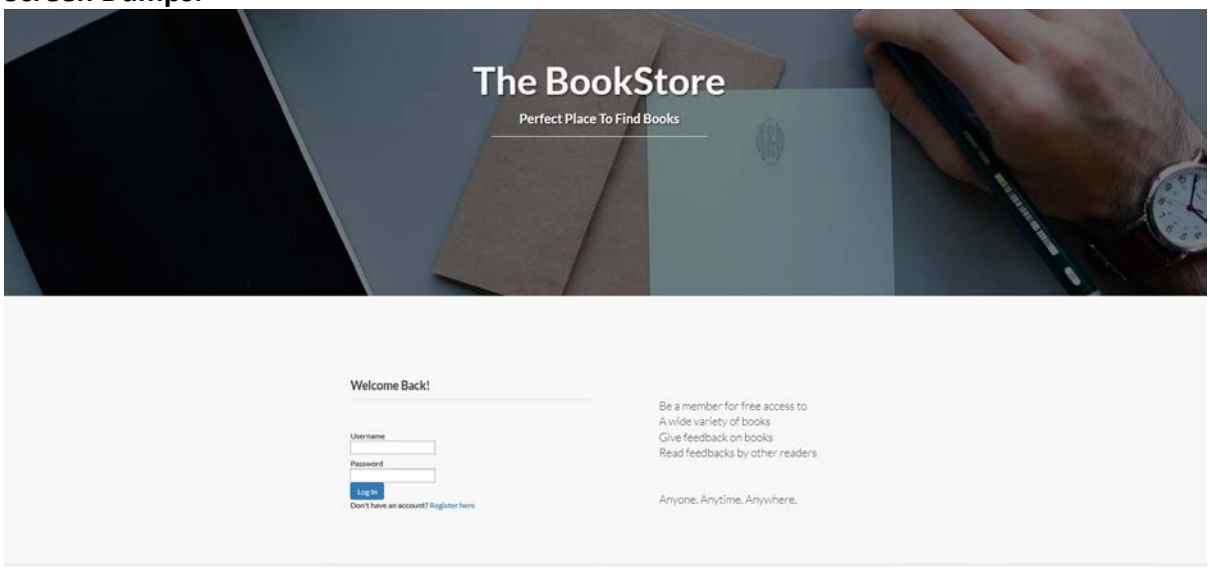


Figure 1: Login page

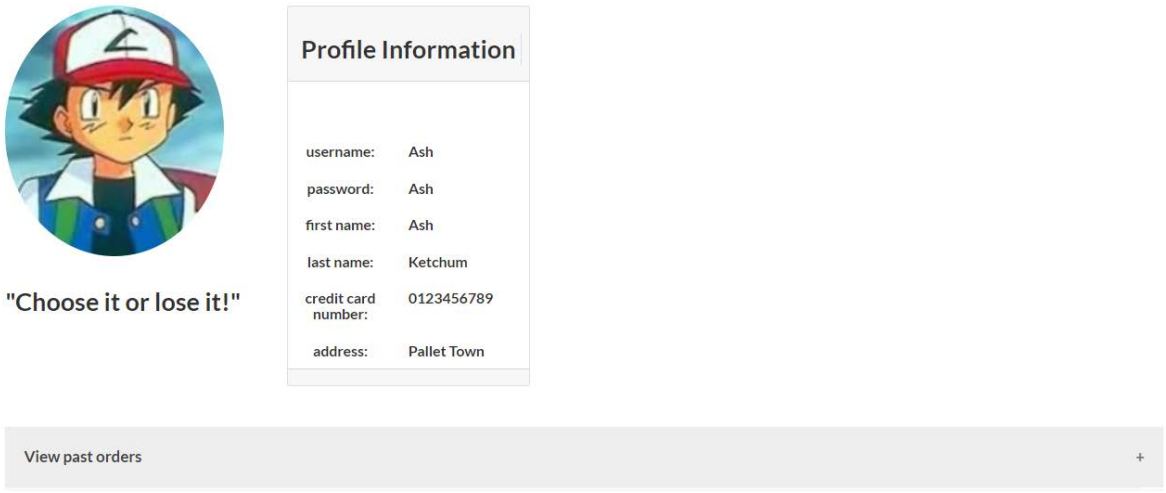


Figure 2: User profile

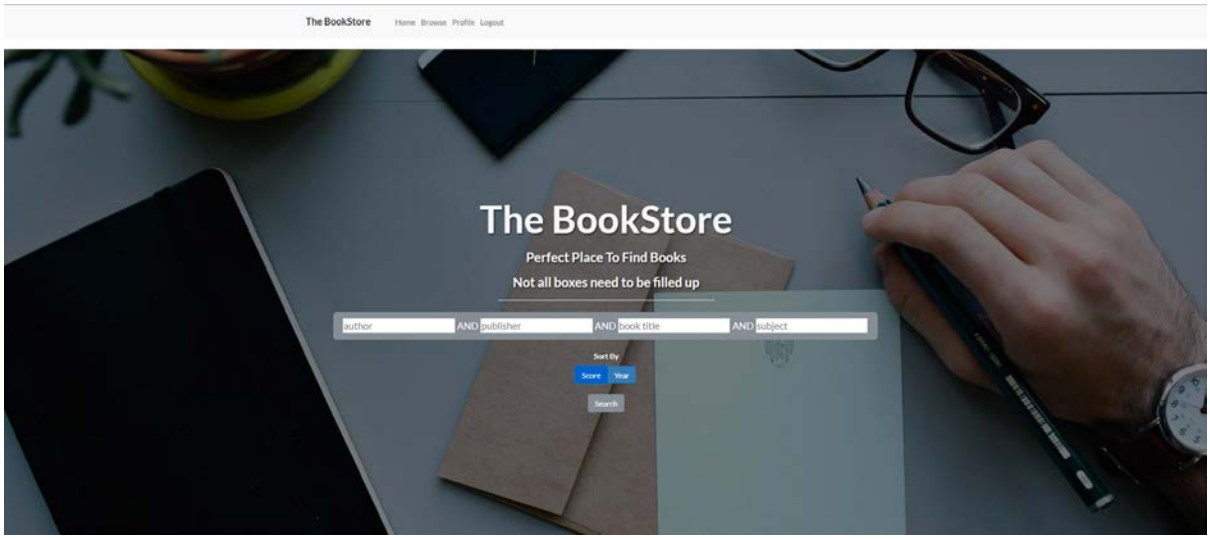


Figure 3: Browse books. Default selection will return all books

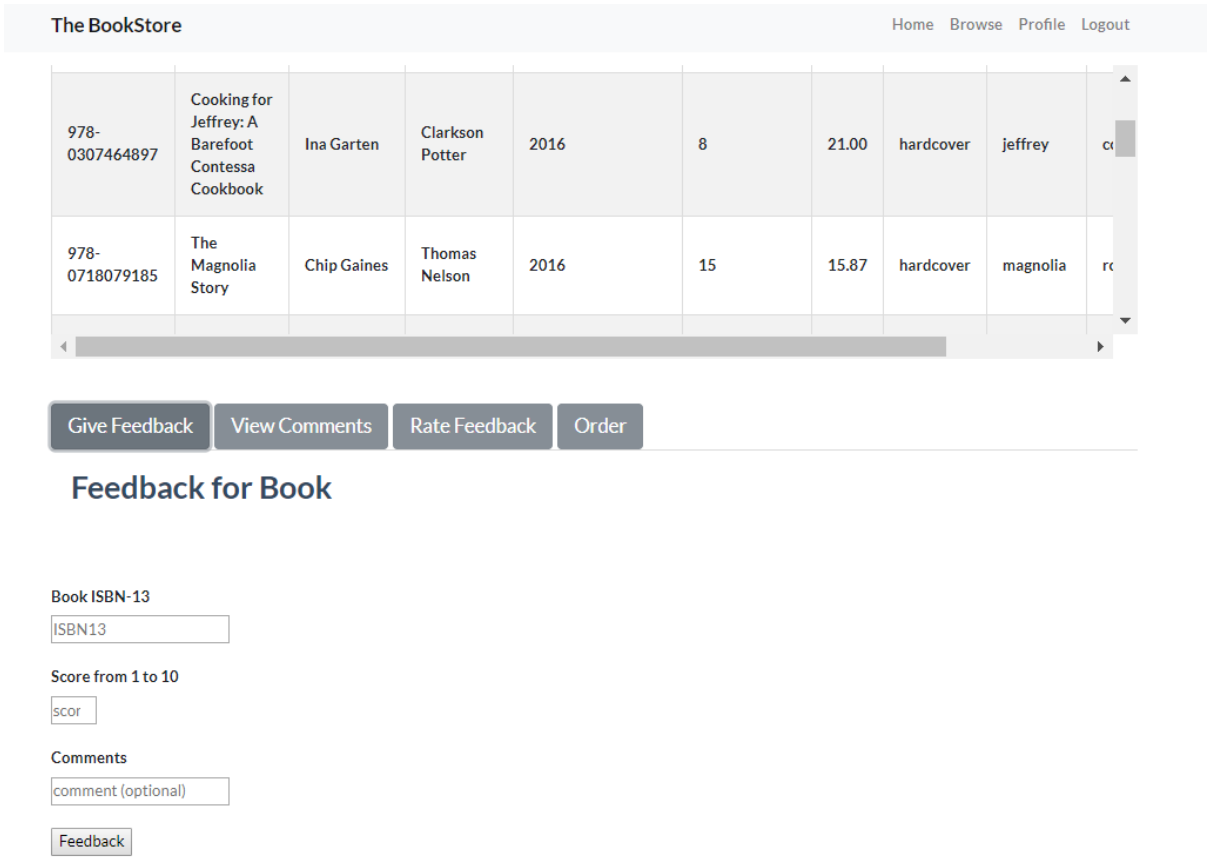


Figure 4: Users may comment/ rate feedback/View comments / order books