NoSql Project – DjonDB

Dan Jiang and Mengjuan Liu

Department of Computer Science

Georgia State University

Atlanta, GA 30303

Email: djiang1@student.gsu.edu mliu9@student.gsu.edu

Abstract

A document-oriented database is a computer program designed for storing, retrieving, and managing document-oriented information, also known as semi-structured data. DjonDB is one type of document DB. All the documents in djondb are stored in files and organized by namespace in the data folder and stored in JSON format. The project of my shopping receipts is implemented by DjonDB.

Introduction

With massive advent of Internet, storing large amount of documents became a must. Such documents range from images to more or less structured text, including large chunks of information encoded in XML. However, relational technology was not natively prepared to support such kind of data.

What makes document databases really different is the fact that documents are usually retrieved through dynamic and unpredictable queries. Thus document databases can usually associate any number of fields of any length to a document. This way we can store, together with a medical image, patient name and birth data. If you late decide to add also sex and profession, you can do it even if it wasn't originally conceived. Therefore, Document databases are usually schema-less; there is no predefined data model.

A document database is, at its core, a key/value store with one major exception.

The format can be XML, JSON, Binary JSON or just about anything, as long as the database can understand it. DjonDB is one type of document DB. In this project, the goal of our work is to implement a application using DjonDB.

Related Works

A. JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON is built on two structures: A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array. An ordered list of values: In most languages, this is realized as an array, vector, list, or sequence.

JSON object: An object is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma).

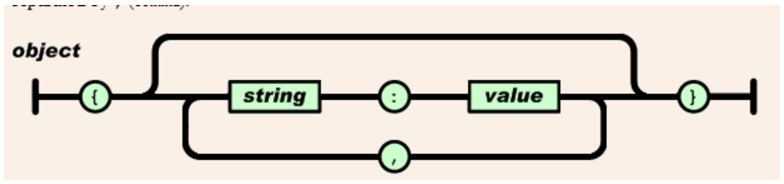


Figure 1: JSON Object

JSON Array: An array is an ordered collection of values. An array begins with [(left bracket) and ends with] (right bracket). Values are separated by , (comma).

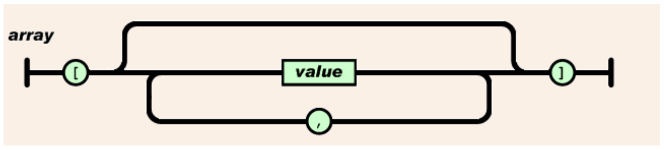


Figure 2: JSON Array

JSON Value: A value can be a string in double quotes, or a number, or true or false or null, or an object or an array. These structures can be nested.

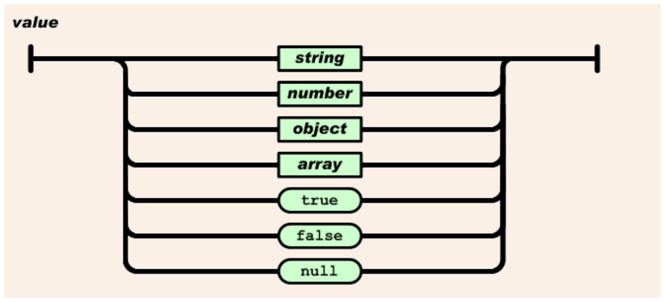


Figure 3: JSON Value

JSON string: A string is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes. A character is represented as a single character string. A string is very much like a C or Java string.

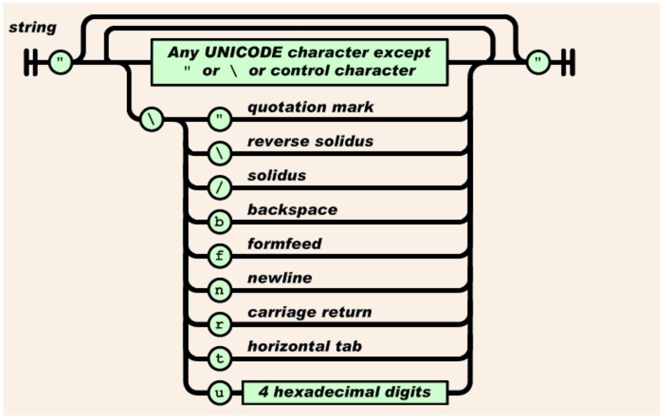


Figure 4: JSON string

JSON number: A number is very much like a C or Java number, except that the octal and hexadecimal formats are not used.

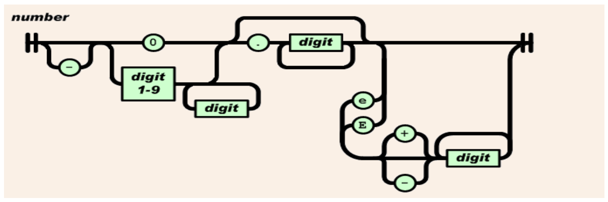


Figure 5: JSON number

JSON example:

```
{
   "firstName": "John",
   "lastName": "Smith",
   "age": 25,
   "address": {
      "streetAddress": "21 2nd Street",
      "city": "New York",
      "state": "NY",
      "postalCode": 10021
   },
   "phoneNumbers": [
      {
         "type": "home",
         "number": "212 555-1234"
      },
      {
         "type": "fax",
         "number": "646 555-4567"
      }
   ]
}
```

This example shows the JSON representation of an record that describes a person. The object has string fields for first name and last name, a number field for age, contains an object representing the person's address, and contains a list (an array) of phone number objects.

B. BSON

BSON, “Binary JSON”, is a binary form for representing simple data structures and associative arrays (often called objects or documents). BSON is a computer date interchange format used mainly as data storage in the database.

BSON Data types: string, integer, double, date, binary data, boolean, null, BSON object and regular expression.

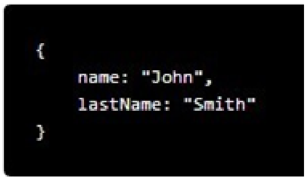
C. DjonDB

All the documents in djondb are stored in files and organized by namespace in the data folder. Each database may contain one or several namespaces, and these namespaces may contain several documents. Usually you would want to organize all the documents of the same type in the same namespace, for example all the documents that represent customers will be stored in a namespace named: "Customers". Database/Namespace/Documents is analog to Databases/Tables/Rows in the RDBMS.

djonDB	**Relational DB**
database	Database
namespace	Table
documents	rows

How to run djondb server? For windows users there's a convenient shortcut to boot up the server, which you will find under the menu "djondb/djondbd". To shutdown the server, just use ctrl+c command.

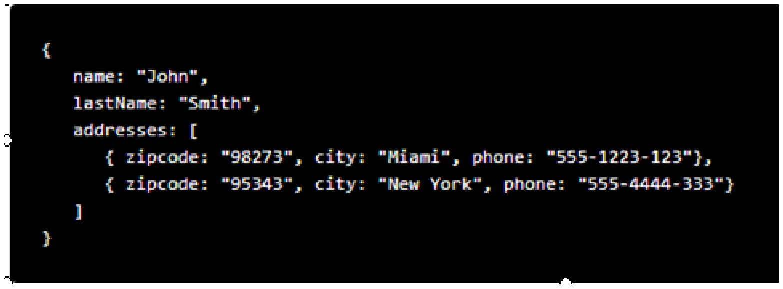
Djondb is a document database, these documents are json documents that could be stored directly to the database, example:



```
{
    name: "John",
    lastName: "Smith"
}
```

Figure 6: DjonDB example

These JSON documents may have several "subdocuments" like this:

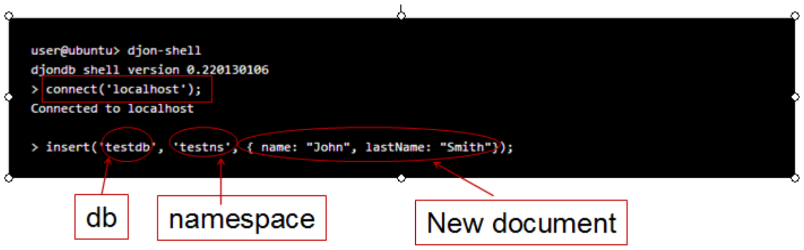


```
{
   name: "John",
   lastName: "Smith",
   addresses: [
      { zipcode: "98273", city: "Miami", phone: "555-1223-123"},
      { zipcode: "95343", city: "New York", phone: "555-4444-333"}
   ]
}
```

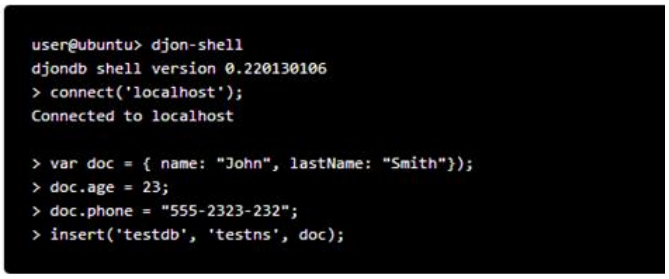
Figure 7: DjonDB example

Djondb drivers supports two different ways to create new documents, using the string representation or using BSONObj objects, these BSONObj classes were created to handle JSON documents in an easier way.

How to create documents using the shell? Djon-shell is a full javascript console that it's very useful to learn how to use Djondb and what is capable of, take a look of the following example:



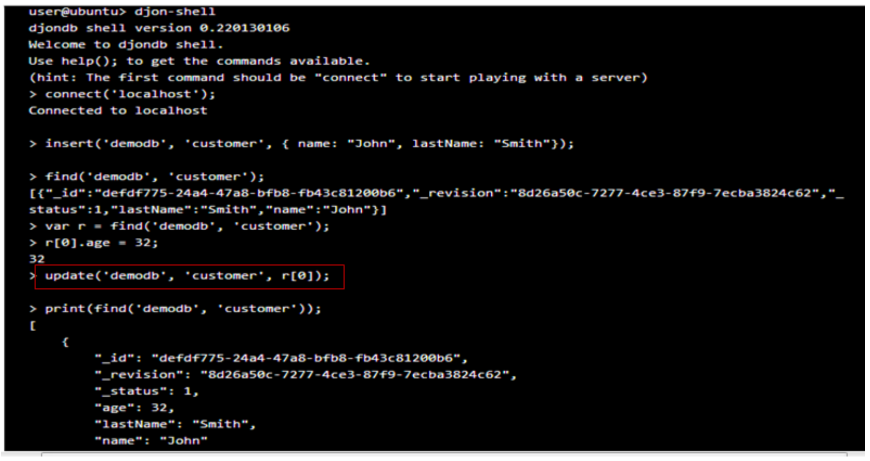
```
user@ubuntu> djon-shell
djondb shell version 0.220130106
> connect('localhost');
Connected to localhost

> insert('testdb', 'testns', { name: "John", lastName: "Smith"});
```



```
user@ubuntu> djon-shell
djondb shell version 0.220130106
> connect('localhost');
Connected to localhost

> var doc = { name: "John", lastName: "Smith"});
> doc.age = 23;
> doc.phone = "555-2323-232";
> insert('testdb', 'testns', doc);
```

Figure 8: DjonDB command

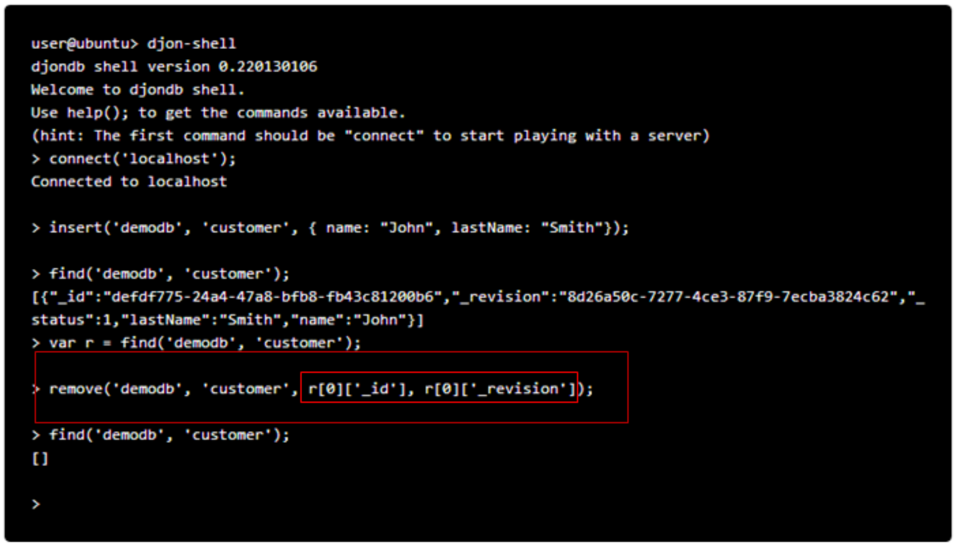
For updating documents, just use the “update” command.



```
user@ubuntu> djon-shell
djondb shell version 0.220130106
Welcome to djondb shell.
Use help(); to get the commands available.
(hint: The first command should be "connect" to start playing with a server)
> connect('localhost');
Connected to localhost

> insert('demodb', 'customer', { name: "John", lastName: "Smith"});

> find('demodb', 'customer');
[{"_id":"defdf775-24a4-47a8-bfb8-fb43c81200b6","_revision":"8d26a50c-7277-4ce3-87f9-7ecba3824c62","_
status":1,"lastName":"Smith","name":"John"}]
> var r = find('demodb', 'customer');
> r[0].age = 32;
32
> update('demodb', 'customer', r[0]);

> print(find('demodb', 'customer'));
[
    {
        "_id": "defdf775-24a4-47a8-bfb8-fb43c81200b6",
        "_revision": "8d26a50c-7277-4ce3-87f9-7ecba3824c62",
        "_status": 1,
        "age": 32,
        "lastName": "Smith",
        "name": "John"
```

Figure 9: DjonDB command

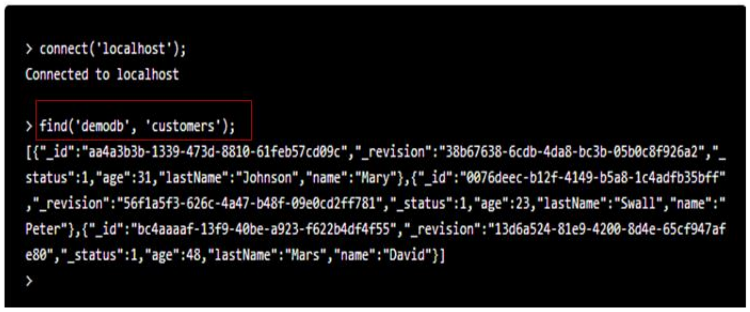
For removing documents, just use “remove” command.



```
user@ubuntu> djon-shell
djondb shell version 0.220130106
Welcome to djondb shell.
Use help(); to get the commands available.
(hint: The first command should be "connect" to start playing with a server)
> connect('localhost');
Connected to localhost

> insert('demodb', 'customer', { name: "John", lastName: "Smith"});

> find('demodb', 'customer');
[{"_id":"defdf775-24a4-47a8-bfb8-fb43c81200b6","_revision":"8d26a50c-7277-4ce3-87f9-7ecba3824c62","_
status":1,"lastName":"Smith","name":"John"}]
> var r = find('demodb', 'customer');

> remove('demodb', 'customer', r[0]['_id'], r[0]['_revision']);

> find('demodb', 'customer');
[]

>
```

Figure 10: DjonDB command

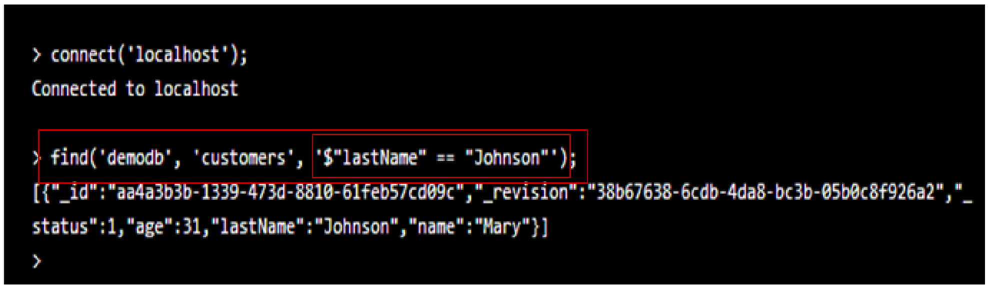
To retrieve all your documents in a given namespace you just specify the database and the required namespace as follows:



```
> connect('localhost');
Connected to localhost

> find('demodb', 'customers');
[{"_id":"aa4a3b3b-1339-473d-8810-61feb57cd09c","_revision":"38b67638-6cdb-4da8-bc3b-05b0c8f926a2","_
status":1,"age":31,"lastName":"Johnson","name":"Mary"},{"_id":"0076deec-b12f-4149-b5a8-1c4adfb35bff"
,"_revision":"56f1a5f3-626c-4a47-b48f-09e0cd2ff781","_status":1,"age":23,"lastName":"Swall","name":"
Peter"},{"_id":"bc4aaaaf-13f9-40be-a923-f622b4df4f55","_revision":"13d6a524-81e9-4200-8d4e-65cf947af
e80","_status":1,"age":48,"lastName":"Mars","name":"David"}]
>
```

Figure 11: DjonDB command

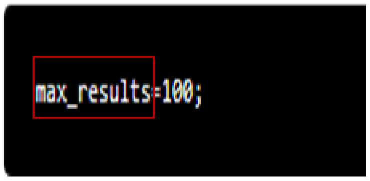
Filtering your results, use “find” command.



```
> connect('localhost');
Connected to localhost

> find('demodb', 'customers', '$"lastName" == "Johnson"');
[{"_id":"aa4a3b3b-1339-473d-8810-61feb57cd09c","_revision":"38b67638-6cdb-4da8-bc3b-05b0c8f926a2","_
status":1,"age":31,"lastName":"Johnson","name":"Mary"}]
>
```

Figure 12: DjonDB command

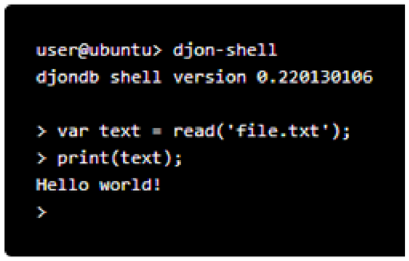
DjonDB can limit the results to avoid retrieving all the database in a single find, the default limit is 30 documents, but you can change this using the parameter max_results in the /etc/djondb.conf like this:



```
max_results=100;
```

Figure 13: DjonDB command

Using “print” command can read files in Djondb shell. This shell command allows you to read a file from this into a variable; it will be readed as text. This command can also show a message into the console, which has a nice feature.



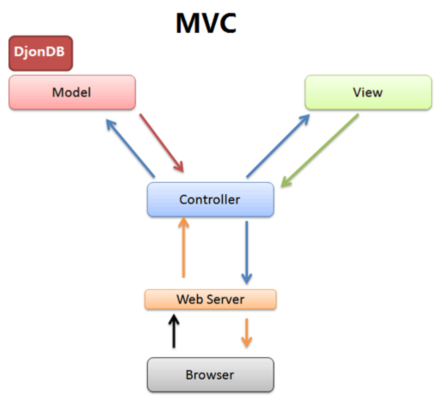
```
user@ubuntu> djon-shell
djondb shell version 0.220130106

> var text = read('file.txt');
> print(text);
Hello world!
>
```

Figure 14: DjonDB command

Overall Design of My Shopping Receipts System

A. System structure and development tool

Based on B/S (Browsers/Servers) structure, which is a technology that no needs to install any program on terminals. User can do inquiring, viewing and other kinds of operations about their business only through a browser. It is easy to maintain and update the system since it only has to do with the server. It also suffices the expansibility of users' needs, the system can communicate with other systems by merging JSP network programming with java technology; especially realize connection with database. The entire project is designed using MVC framework, which includes model, view and controller.



B. Functions

The goal of our project is to save our shopping receipts. The whole system includes the following functions: adding receipts, adding merchant information, adding item information, displaying the receipts and displaying the charts. Each part's function module serves as the business logic layer in MVC, presenting different operations based on different information input. Among them, the access control module is the key of system security, any access and operation from users except the administrator should under its permission.

C. Development Platform

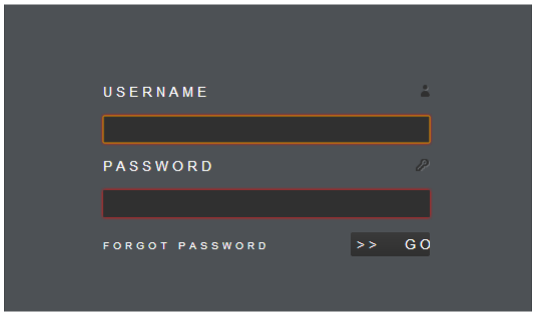
We choose J2EE architecture as system platform, such as Windows, Tomcat, DjonDB, JSP and Java. They are all open-source software and with advantages such as less storage room, cross platforms, high performance, low cost, secure and stable.

Design and Implement

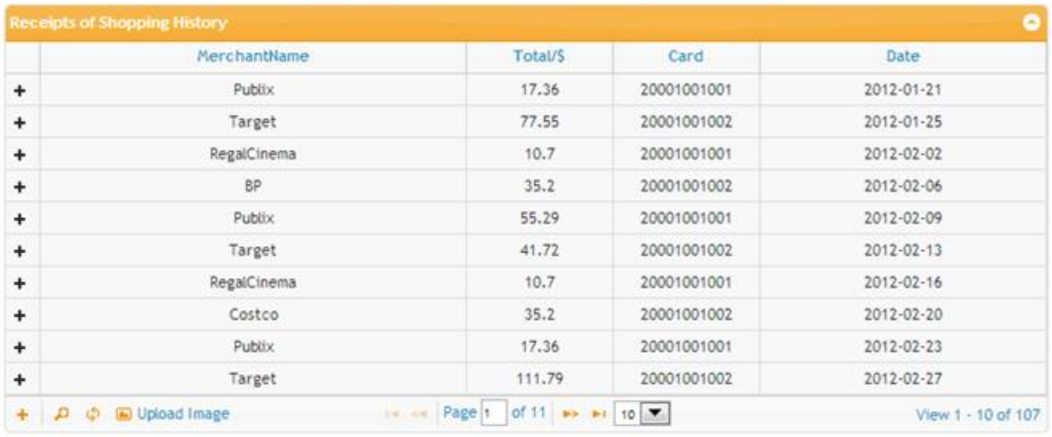
A. Function Implement of Modules

The main function is using JSP to implement the user interface and using Java to implement all the functions interacted to the database. We embedded the Java code into the JSP pages and servlets, which can deal with data from DjonDB. Here, we use the MVC framework to deal with these functions. The JSP files are responsible for the view part, all the data in the DjonDB are the module part in the project.

The following figure is the login menu of this system. After input the username and password, we can access to the main UI of the system.



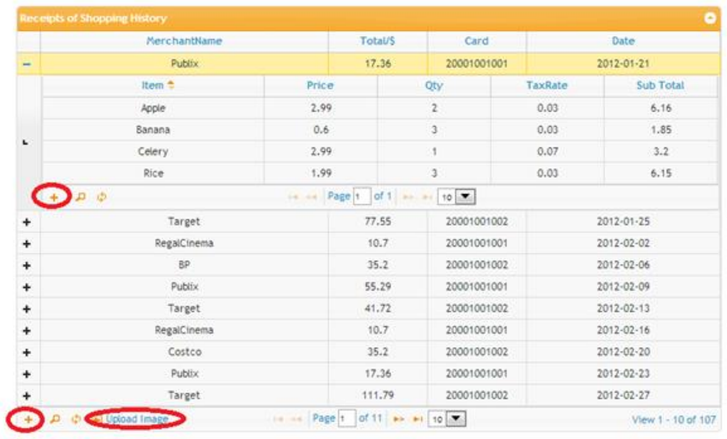
The main UI shows in the following figure, which includes the merchant name, total price for each receipts, card number and date.



Receipts of Shopping History

	MerchantName	Total/$	Card	Date
+	Publix	17.36	20001001001	2012-01-21
+	Target	77.55	20001001002	2012-01-25
+	RegalCinema	10.7	20001001001	2012-02-02
+	BP	35.2	20001001002	2012-02-06
+	Publix	55.29	20001001001	2012-02-09
+	Target	41.72	20001001002	2012-02-13
+	RegalCinema	10.7	20001001001	2012-02-16
+	Costco	35.2	20001001002	2012-02-20
+	Publix	17.36	20001001001	2012-02-23
+	Target	111.79	20001001002	2012-02-27

Expand each merchant name, the item information displayed, which includes the item name, unit price, quantity, tax rate and sub total of each item. We can also add the merchant information and item information into this system.

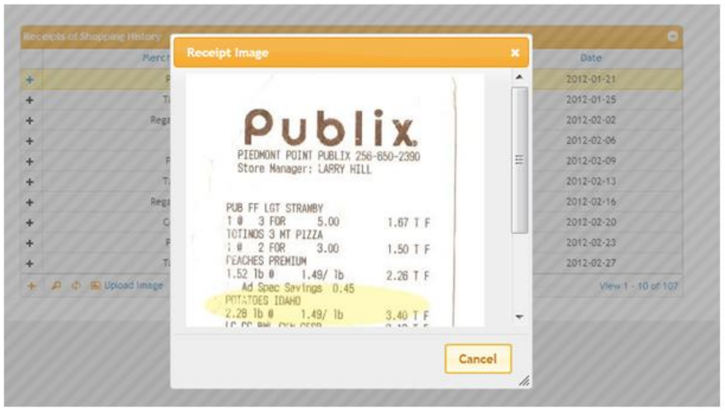


	MerchantName	Total/$	Card	Date
−	Publix	17.36	20001001001	2012-01-21

Item	Price	Qty	TaxRate	Sub Total
Apple	2.99	2	0.03	6.16
Banana	0.6	3	0.03	1.85
Celery	2.99	1	0.07	3.2
Rice	1.99	3	0.03	6.15

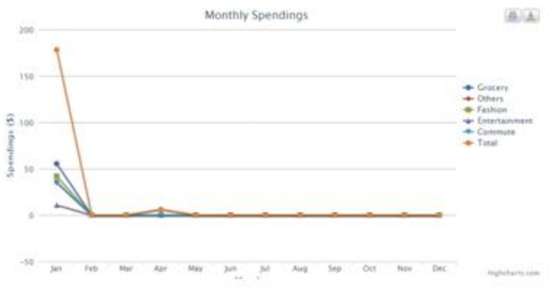
	MerchantName	Total/$	Card	Date
+	Target	77.55	20001001002	2012-01-25
+	RegalCinema	10.7	20001001001	2012-02-02
+	BP	35.2	20001001002	2012-02-06
+	Publix	55.29	20001001001	2012-02-09
+	Target	41.72	20001001002	2012-02-13
+	RegalCinema	10.7	20001001001	2012-02-16
+	Costco	35.2	20001001002	2012-02-20
+	Publix	17.36	20001001001	2012-02-23
+	Target	111.79	20001001002	2012-02-27

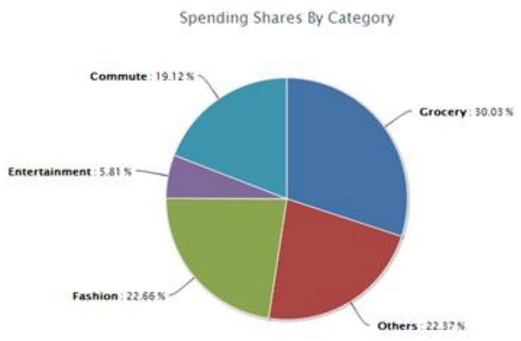
The function "upload image" is used to upload the receipts image, which can help us to save the receipts.



```
Publix
PIEDMONT POINT PUBLIX 256-650-2390
Store Manager: LARRY HILL

PUB FF LGT STRAWBY
1 @   3 FOR      5.00         1.67 T F
TOTINOS 3 MT PIZZA
1 @   2 FOR      3.00         1.50 T F
PEACHES PREMIUM
1.52 lb @      1.49/ lb       2.26 T F
    Ad Spec Savings  0.45
POTATOES IDAHO
2.28 lb @      1.49/ lb       3.40 T F
```

From the main UI, the chart displayed when you click the "spending curve" and "spending pie" tab. You can select the year to change the charts.





The whole project implemented using the Model-View-Controller framework. All the data in the model part are stored in DjonDB. For the view part, all the data are displayed in the web page using JSP. For the controller part, we use some java files to handle them. All functions have the same way to be implemented. For example, for the adding receipts function, we use java bean, java DAO and handler to implement the operations. In java bean file, we give the get() and set() methods for getting and setting value of each field in the tables. In the java handler, all the operations, including add receipts, add item, get cards, get history, get items, get merchants, get receipts image, get record years, get spending curve, get state pie, upload file, are implemented. In the java DAO file, all the methods are used to connect the database tables with the user interfaces (JSP files).

B. Deploy the Modules in the Web Server

In this project, we use the tomcat as the web server, which is used to display all the functions in the browser. Using xml file to configure the index file and set path to some files if needed. The url for the project is http://localhost:8080.

Testing

After the requirements have been defined and the coding process has been completed, the testing process starts. The primary purpose of testing is to detect software failure so that defects may be discovered and corrected. Basically we want to test if the application does what it is supposed to do and does what it needs to do. We used both static testing and dynamic testing including reviews, walkthroughs, inspection and a given set of cases testing. In the static testing process we cross review the code and walkthrough each functional module to inspect the functionality. In the dynamic testing process we tried different users' requirement as input and evaluate the output results. For instance, as an administrator we are supposed to create a new user account. If the user record is already in the database we should see an error message to alert us the user exists. If not we should go back to check the related function module to make some modifications. Also the relationship of data needs to be kept during the manipulation of the database. For instance, if we modify a record under one user account we may not want it affecting other records. By giving different set of inputs and checking the related records we can assess the results as what we expected. In order to check the robotics of the database application the large data input has been tested. The entire performance to process the large input is acceptable. The waterfall testing model was utilized and the logs of testing were kept for future project.

Conclusion

My shopping receipts system is well developed to simulate the database management system (DBMS). We implement all the required function modules such as adding receipts, adding items, displaying receipts and displaying charts. The testing was conducted in multi rounds with both static and dynamic testing methods. Also as a team we cross test the function models to make sure all functions behavior in the right way as what they are supposed to do. The demonstration of the project was shown all the functions of the application. Various design issues were solved during the process of the implementation. With the Array, Array List and Collection data structures we implement the application with optimized algorithm. The demonstration is presented in the class with highly rewarded feedback.