



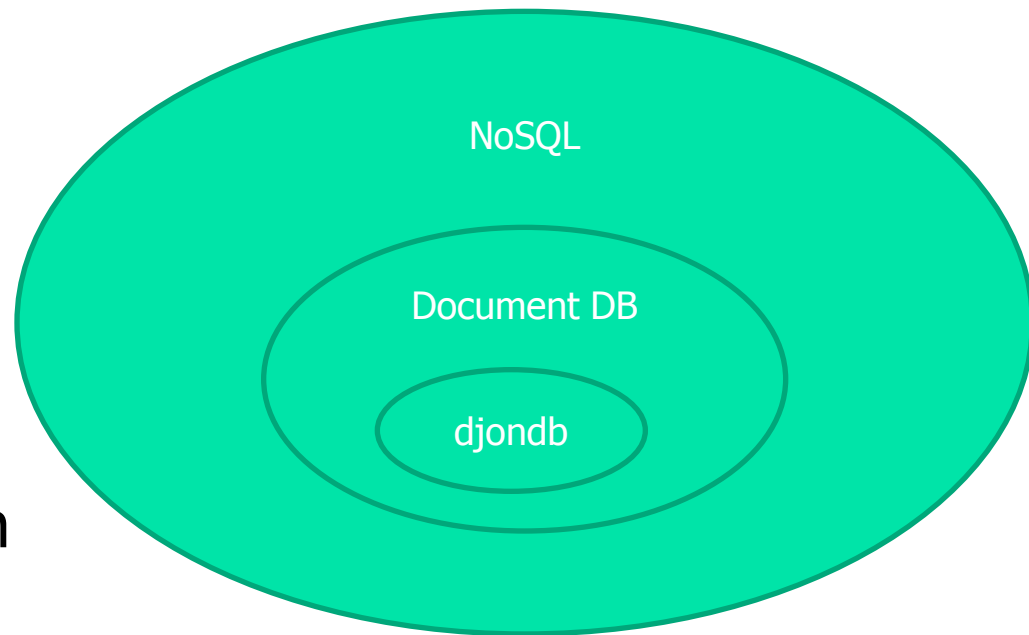
--Application: My Shopping Receipts

Student: Mengjuan Liu, Dan Jiang



Outline

- Document DB
- djonDB
- Application & demonstration





Why document DB?

- 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 later 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.



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 structures

JSON is built on two structures:

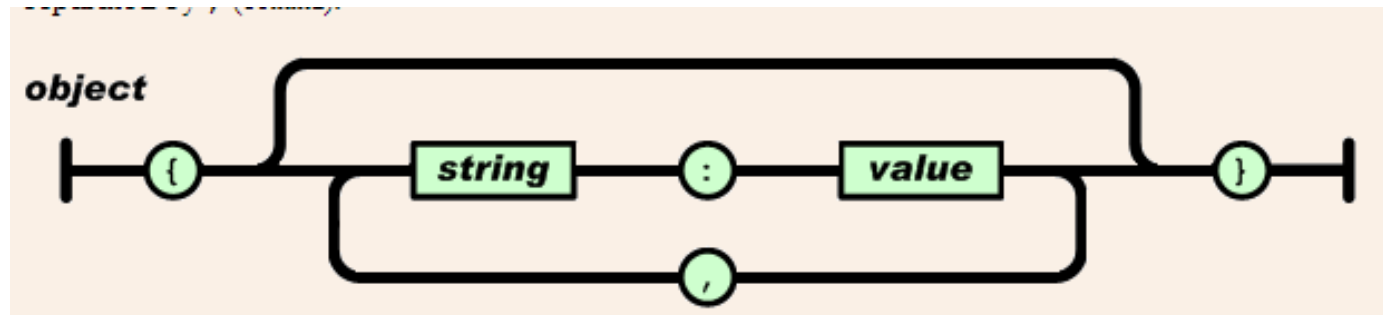
- A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.



JSON data type and format

- JSON-object

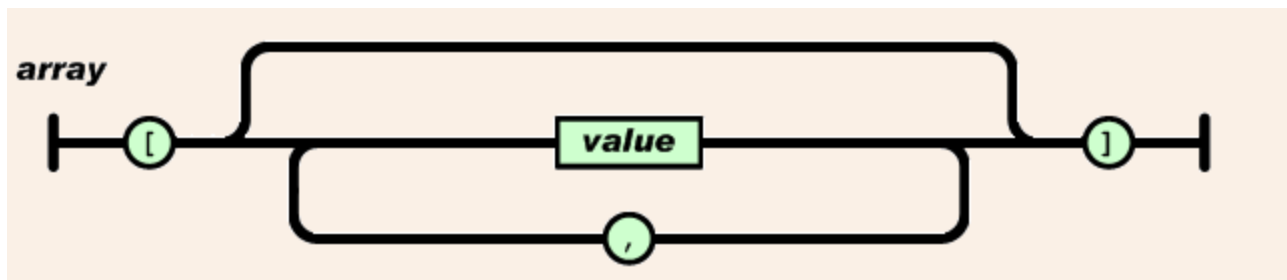
An *object* is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma).





JSON data type and format cont.

- 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).

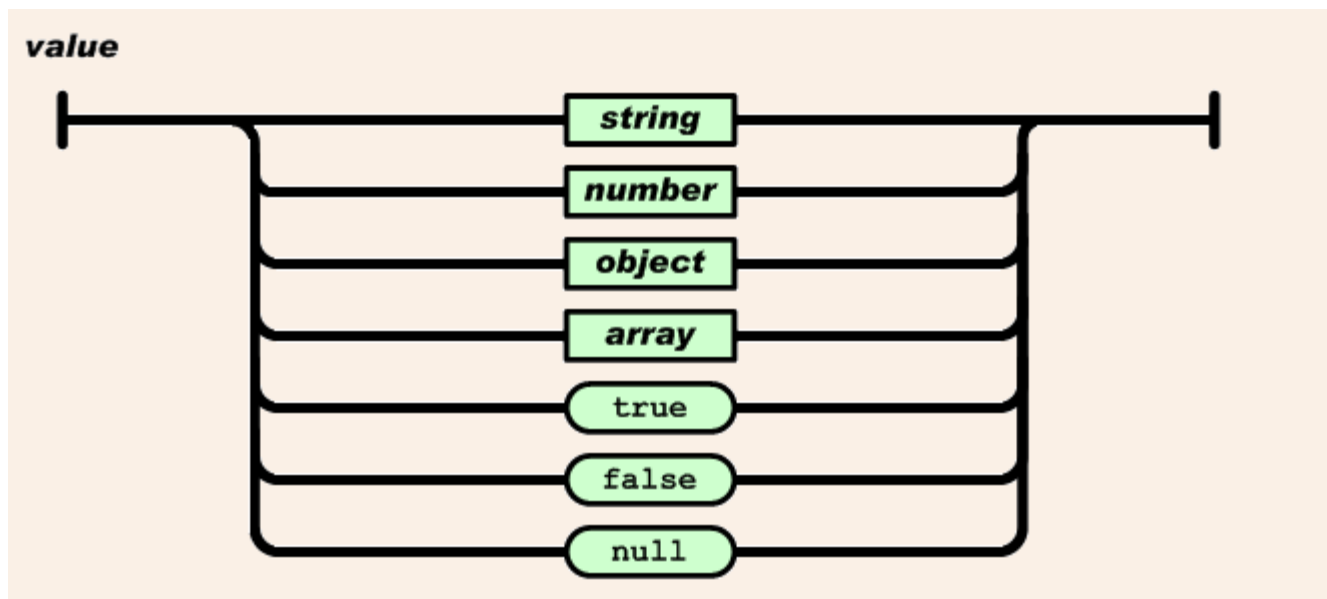




JSON data type and format cont.

- 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.

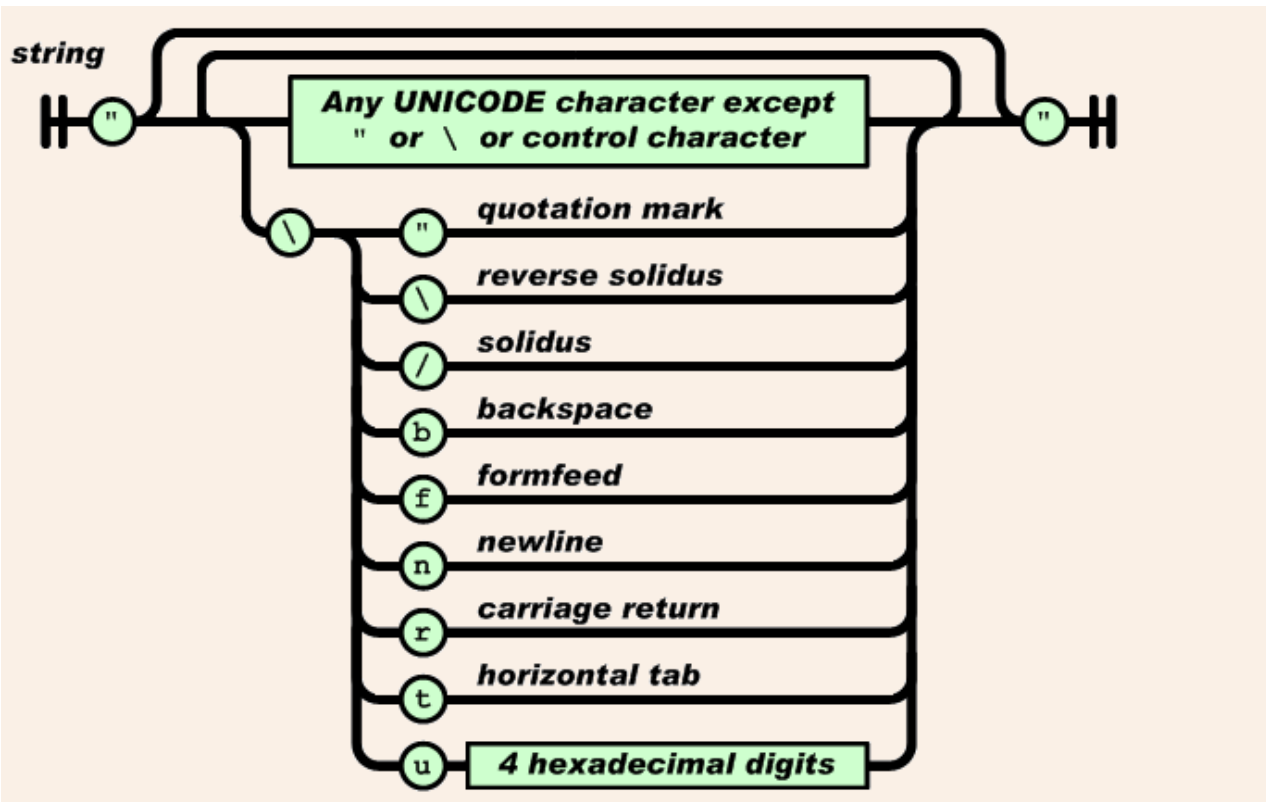




JSON data type and format cont.

- 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.

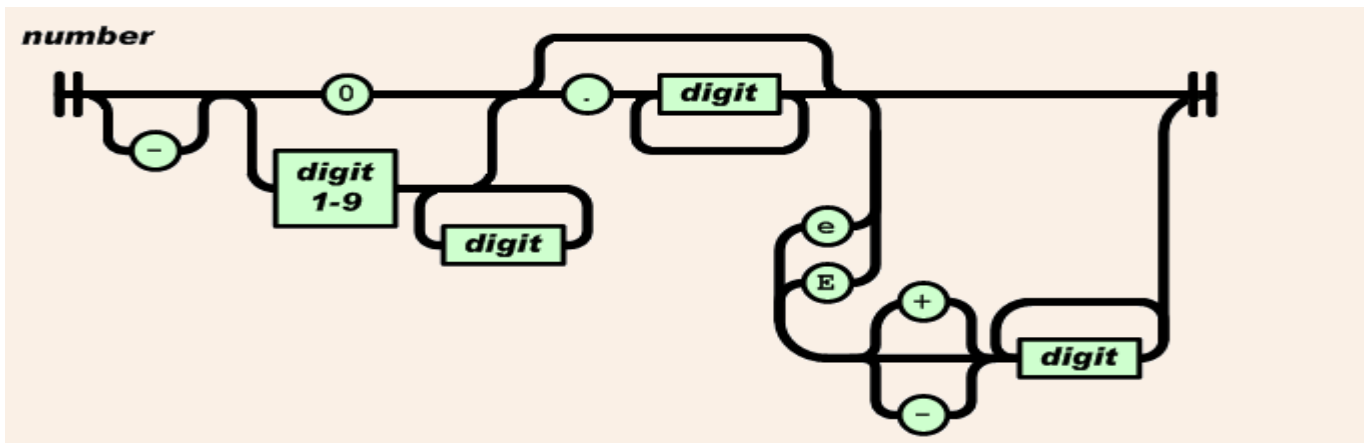




JSON data type and format cont.

- JSON-number

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





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"
    }
  ]
}
```

The example shows the JSON representation of a 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.



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 data interchange format used mainly as a data storage in the database.
- **BSON Data types**
 - String
 - Integer
 - Double
 - Date
 - Binary data
 - Boolean
 - Null
 - BSON object
 - Regular expression



djonDB

-An Open Source NoSQL for Business users



- **Databases and namespaces**

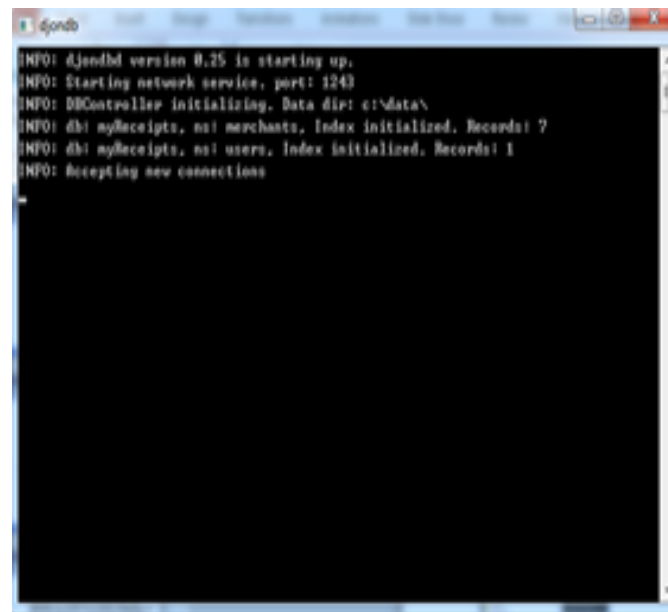
All the documents in djondb are stored in files and organized by namespace in the data folder. Each database may contain one or several namespaces, and these namespaces may contain several documents. Usually you would want to organize all the documents of the same type in the same namespace, for example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

djonDB	Relational DB
database	Database
namespace	Table
documents	rows



Running djondb server

- For windows users there's a convenient shortcut to boot up the server, that you will find under the menu "djondb/djondbd".
- To shutdown the server: ctrl+c





■ Inserting documents

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

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

■ 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"}
  ]
}
```

- 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.



Creating 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"});
```

db

namespace

New document

```
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);
```



■ Updating documents

```
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"
  }
]
>
```



■ Removing documents

```
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');
[]

>
```



■ Retriving all your documents

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":"bc4aaaaaf-13f9-40be-a923-f622b4df4f55","_revision":"13d6a524-81e9-4200-8d4e-65cf947afe80","_status":1,"age":48,"lastName":"Mars","name":"David"}]

>
```



■ Filtering your results

```
> 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"}]

>
```

Here you will noticed that we specified the filter `'$"lastName" == "Johnson"'`, the field you want use as filter is placed between the tag `' '` or `'"` "



- **Selecting fields**

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

> find('demodb', 'customers', '$"name", $"age", ');
[{"age":31,"name":"Mary"}, {"age":23,"name":"Peter"}, {"age":48,"name":"David"}]

>
```



- **Limiting your results**

At this moment djondb will 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;
```




- **Read in djondb shell**

This shell command allows you to read a file from this into a variable, it will be readed as text.

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

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



■ Load

The load command is very similar to read command, however it will allow us to create script files and load them using this command. Example:

Copy the following code into a file named test.js

```
function hello(name) {  
  print('Hello ' + name);  
}
```

Then open a console and load the test.js script like this:

```
user@ubuntu> djon-shell  
djondb shell version 0.220130106  
  
> load('test.js');  
> hello("Peter");  
Hello Peter  
>
```



■ Print

To show a message into the console we can use the print function, this function has a nice feature. take a look of this sample:

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

> connect('localhost');
> find('testdb', 'testns');
[{"_id":"61b28fe-2ec7-468f-b5ba-59b477db29e","_revision":"c25b1a4-4fb0-4bc0-b1ed-2ade5ff94aa","_status":1,"address":[{"phone":"555-"}, {"phone":"555-"}],"age":18,"date":"as","lastName":"Test last name","name":"Test"}]
> print(find('testdb', 'testns'));
```

```
[
  {
    "_id": "61b28fe-2ec7-468f-b5ba-59b477db29e",
    "_revision": "c25b1a4-4fb0-4bc0-b1ed-2ade5ff94aa",
    "_status": 1,
    "address": [
      {
        "phone": "555-"
      },
      {
        "phone": "555-"
      }
    ],
    "age": 18,
    "date": "as",
    "lastName": "Test last name",
    "name": "Test"
  }
]
```

Nice format



Application

Application: my shopping receipt.



Application

- Goal: Save our receipts
- Programming language: jsp, java
- Database: djonDB
- Web server: tomcat
- Framework: mvc
- Implement:
 - Display the receipts
 - Add receipt
 - Add merchant information
 - Add item information



DjonDB Setup

- 1. Download DjonDB from <http://djondb.com/downloads.html> and install it (better use 64bit version and 64bit java version)**
- 2. Start Tomcat**
- 3. Open DjonDB**



Database Design

Find Database and Data (use “find” command)

```
djondb shell
Use help(); to get the commands available.
(hint: The first command should be "connect" to playing with a server)
> connect('localhost');
Connected to localhost
> find('myReceipts', 'merchants');
[{"_id": "ac9bc19-558c-4115-a506-a327d7b05c5d", "_revision": "88468266-34e3-48c7-a
e51-0ce2d96cb286", "_status": 1, "address1": "1100", "address2": "Hammond", "category":
"Grocery", "city": "Dunwoody", "id": "1", "name": "Publix", "phone": "404-0000000", "stat
e": "GA", "zipcode": "30346"}, {"_id": "1d039874-d63c-4185-ae9c-6d3a9243891d", "_revis
ion": "76c76125-8445-4da4-83e6-0df764056e01", "_status": 1, "address1": "10", "address
2": "Perimeter Parkway", "category": "Fashion", "city": "Dunwoody", "id": "2", "name": "T
arget", "phone": "404-0000000", "state": "GA", "zipcode": "30346"}, {"_id": "04dcda8f-cb
14-45a4-aaa6-7b3dec6817ff", "_revision": "4d2b2ee4-a577-45c8-9685-6b4b78f7554a", "_
status": 1, "address1": "100", "address2": "PTree Dunwoody", "category": "Home", "city":
"Dunwoody", "id": "3", "name": "HomeDepot", "phone": "404-0000000", "state": "GA", "zipco
de": "30346"}, {"_id": "cfe95bb9-2f32-4eed-b74f-d867315f0445", "_revision": "79a35f30
-d000-417c-93c2-cd9a1149b7c3", "_status": 1, "address1": "200", "address2": "PTree Dun
woody", "category": "Health", "city": "Dunwoody", "id": "4", "name": "NorthSide", "phone
": "404-0000000", "state": "GA", "zipcode": "30346"}, {"_id": "e65d6b55-8caa-41ba-bbc7-b
64acc5ad586", "_revision": "d87cb56f-2825-4d3a-934a-7ef13b9c700a", "_status": 1, "add
ress1": "300", "address2": "Mt Vernon", "category": "Entertainment", "city": "Dunwoody
", "id": "5", "name": "RegalCinema", "phone": "404-0000000", "state": "GA", "zipcode": "303
46"}, {"_id": "76c2e065-853e-4e24-b7f4-e3e508434830", "_revision": "6603e1db-c37b-46
a1-8f90-fc392e1e3081", "_status": 1, "address1": "400", "address2": "Somewhere", "categ
```



Insert data to DjonDB (use “insert” command)

```
insert('myReceipts', 'merchants', {id:"8", name:"Walmart", phone:"404-987-2356", state:"GA", zipcode:"30329"});

> find('myReceipts', 'merchants');
[{"_id": "ac7bc7f9-558c-4715-a5b6-a327d7b05c5d", "_revision": "88468266-34e3-48c7-ae51-0ce2d96cb286", "_status": 1, "address1": "1100", "address2": "Hammond", "category": "Grocery", "city": "Dunwoody", "id": "1", "name": "Publix", "phone": "404-0000000", "state": "GA", "zipcode": "30346"}, {"_id": "1d039874-d63c-4185-ae9c-6d3a9243891d", "_revision": "76c76125-8445-4da4-83e6-0df764056e01", "_status": 1, "address1": "10", "address2": "Perimeter Parkway", "category": "Fashion", "city": "Dunwoody", "id": "2", "name": "Target", "phone": "404-0000000", "state": "GA", "zipcode": "30346"}, {"_id": "04dcda8f-cb14-45a4-aaa6-7b3dec6817ff", "_revision": "4d2b2ee4-a577-45c8-9685-6b4b78f7554a", "_status": 1, "address1": "100", "address2": "PTree Dunwoody", "category": "Home", "city": "Dunwoody", "id": "3", "name": "HomeDepot", "phone": "404-0000000", "state": "GA", "zipcode": "30346"}, {"_id": "cfe95bb9-2f32-4eed-b74f-d867315f0445", "_revision": "79a35f30-d000-417c-93c2-cd9a1149b7c3", "_status": 1, "address1": "200", "address2": "PTree Dunwoody", "category": "Health", "city": "Dunwoody", "id": "4", "name": "NorthSide", "phone": "404-0000000", "state": "GA", "zipcode": "30346"}, {"_id": "e65d6b55-8caa-41ba-bbc7-b64acc5ad586", "_revision": "d87cb56f-2825-4d3a-934a-7ef13b9c700a", "_status": 1, "address1": "300", "address2": "Mt Vernon", "category": "Entertainment", "city": "Dunwoody", "id": "5", "name": "RegalCinema", "phone": "404-0000000", "state": "GA", "zipcode": "30346"}, {"_id": "76c2e065-853e-4e24-b7f4-e3e508434830", "_revision": "6603e1db-c37b-46a1-8f90-fc392e1e3081", "_status": 1, "address1": "400", "address2": "Somewhere", "category": "Commute", "city": "Dunwoody", "id": "6", "name": "BP", "phone": "404-0000000", "state": "GA", "zipcode": "30346"}, {"_id": "086d512a-de2b-4ca6-bf41-d734da5dbe5b", "_revision": "5f0828ca-7e6f-440a-b5ec-9547b9a83634", "_status": 1, "address1": "500", "address2": "Peachtree Dunwoody", "category": "Others", "city": "Dunwoody", "id": "7", "name": "Costco", "phone": "404-0000000", "state": "GA", "zipcode": "30346"}, {"_id": "387d3d4-8070-489b-9ec0-976ef08cd45", "_revision": "bb56d44-4fff-4765-92da-028600ad9fb", "_status": 1, "id": "8", "name": "Walmart", "phone": "404-987-2356", "state": "GA", "zipcode": "30329"}]
```




Database Design

Update data to DjonDB (use “update” command)

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

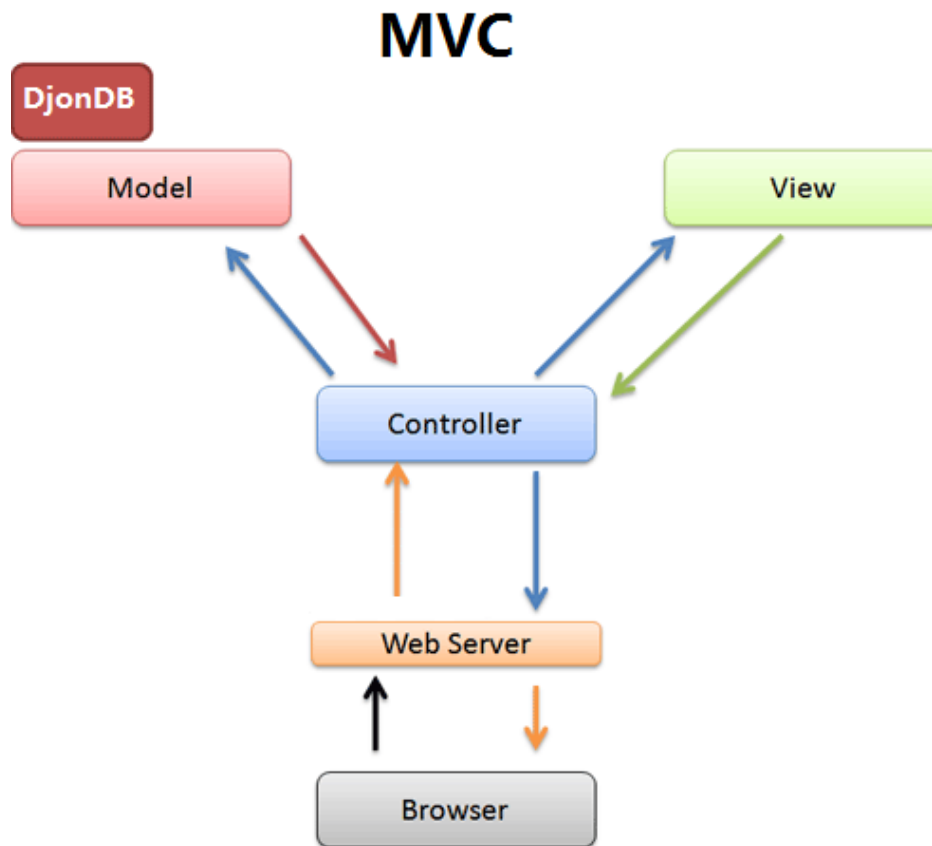
> var r = find('myReceipts','merchants');
> r[7].zipcode="30326";
30326
> update('myReceipts','merchants',r[7]);

> find('myReceipts','merchants');
[{"_id":"ac7bc7f9-558c-4715-a5b6-a327d7b05c5d","_revision":"88468266-34e3-48c7-a
```

```
070-489b-9ec0-976ef08cd45","_revision":"bb56d44-4fff-4765-92da-028600ad9fb","_status":1,"id":"8","name":"Walmart","phone":"404-987-2356","state":"GA","zipcode":"30326"}]
```



Implementation







Implementation

Login:

A screenshot of a login form on a dark grey background. The form contains three main elements: a 'USERNAME' field with a person icon, a 'PASSWORD' field with a key icon, and a 'FORGOT PASSWORD' link. To the right of the 'FORGOT PASSWORD' link is a button with '>>' and 'GO' text. The input fields are highlighted with orange and red borders respectively.

USERNAME 

PASSWORD 

FORGOT PASSWORD >> GO



Implementation

```
<form method="post" action="MainHandler" >
  <fieldset>
    <label for="username">Username <span class="ico">
    <label for="password">Password <span class="ico">
  </fieldset>
  <fieldset>
    <span class="password"><a href="#">Forgot Password</a></span>
```



Implementation

MainHandler:

```
_getSpendingCurve = new getSpendingCurve();
_getRecordYears = new getRecordYears();
_getReceiptImg = new getReceiptImg();

addPostRequestHandler("login", _login);
addPostRequestHandler("logout", _logout);
addPostRequestHandler("register", _register);
addPostRequestHandler("getHisotry", _getHistory);
addPostRequestHandler("getStatPie", _getStatPie);
addPostRequestHandler("getMerchants", _getMerchants);
addPostRequestHandler("getCards", _getCards);
addPostRequestHandler("getReceipts", _getReceipts);
addPostRequestHandler("getReceipts2", _getReceipts2);
addPostRequestHandler("getItems", _getItems);
addPostRequestHandler("addReceipt", _addReceipt);
addPostRequestHandler("addItem", _addItem);
addPostRequestHandler("getSpendingCurve", _getSpendingCurve);
addPostRequestHandler("getRecordYears", _getRecordYears);
addPostRequestHandler("getReceiptImg", _getReceiptImg);

addGetRequestHandler("login", _login);
addGetRequestHandler("logout", _logout);
addGetRequestHandler("register", _register);
addGetRequestHandler("getHisotry", _getHistory);
addGetRequestHandler("getStatPie", _getStatPie);
```



Implementation

Main Menu

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

+ Upload Image << Page 1 of 11 >> 10 ▾ View 1 - 10 of 107



Implementation

DjonDB Connection:

```
public static DjondbConnection openConn()
{
    DjondbConnection c = DjondbConnectionManager.getConnection("localhost");
    if(c.open())
    {
        return c;
    }
    else
    {
        return null;
    }
}

public static void closeConn(final DjondbConnection c)
{
    if(c != null)
    {
        c.close();
        DjondbConnectionManager.releaseConnection(c);
    }
}
```



Implementation

Add Receipts

Receipts of Shopping History					
	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
	+ 📄 📄 Page 1 of 1 10				
+	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	
	+ 📄 📄 Upload Image Page 1 of 11 10 View 1 - 10 of 107				



Implementation

Add Receipts:

```
HttpSession session = req.getSession(false);
c = djonDBUtil.openConn();
BSONObj userDoc = djonDBUtil.getCurrentUser(session, c);
if(userDoc != null)
{
    BSONArrayObj receipts = userDoc.getBSONArray(djonDBUtil.USER_RECEIPTS_FIELDNAME);
    BSONObj receipt = new BSONObj();
    int id = receipts.length();
    System.err.println("rid = " + id);
    receipt.add(djonDBUtil.USER_RECEIPT_ID_FIELDNAME, id);
    receipt.add(djonDBUtil.USER_RECEIPT_MERCHANT_FIELDNAME, merchantName);
    receipt.add(djonDBUtil.USER_RECEIPT_PURCHASEDATE_FIELDNAME, purchaseDate);
    receipt.add(djonDBUtil.USER_RECEIPT_CARD_FIELDNAME, cardID);
    receipt.add(djonDBUtil.USER_RECEIPT_ITEMS_FIELDNAME, new BSONArrayObj());
    receipt.add(djonDBUtil.USER_RECEIPT_IMG_FIELDNAME, "./receiptsImgs/"+id+".jpg");
    receipt.add(djonDBUtil.USER_RECEIPT_TOTAL_FIELDNAME, 0.0);
    receipts.add(receipt);
    c.update(djonDBUtil.DBNAME, djonDBUtil.USERS_COLLECTION_NAME, userDoc);
    ret.put("ret", true);
}
}
```



Implementation

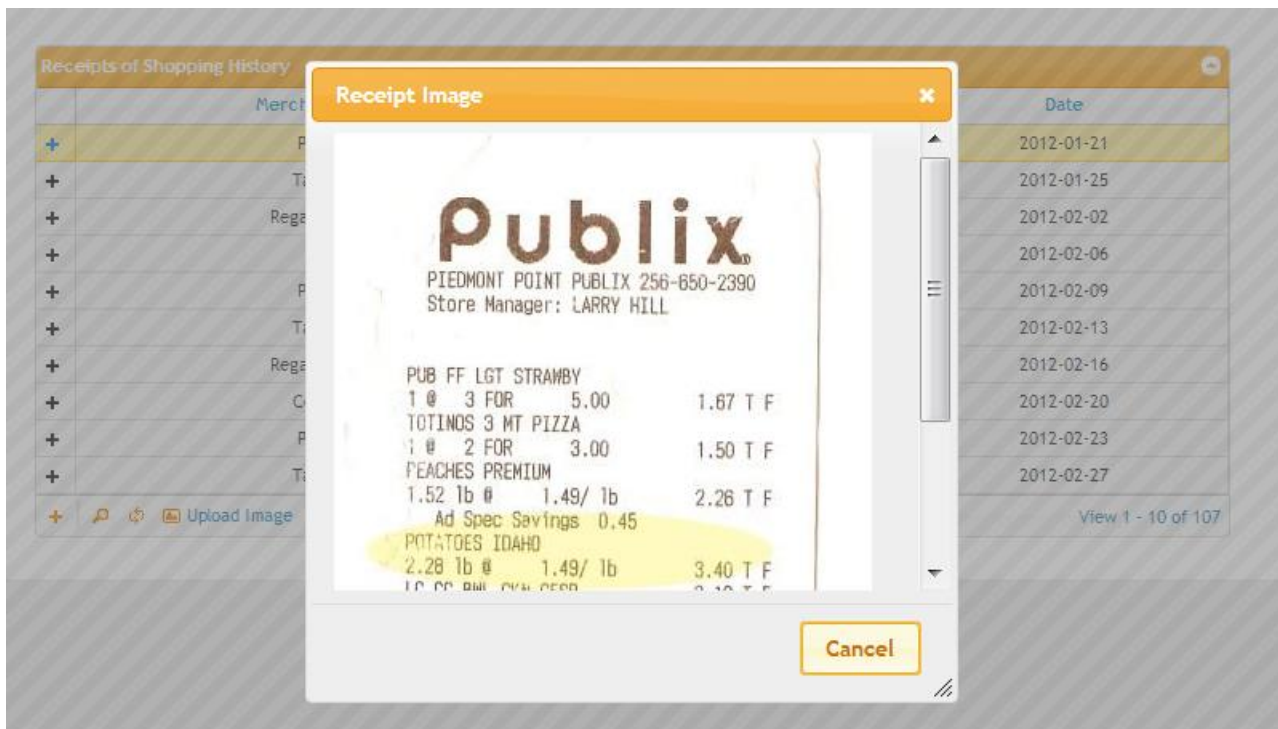
Add Items:

```
BSONArrayObj receipts = userDoc.getBSONArray(djonDBUtil.USER_RECEIPTS_FIELDNAME);
for(int i=0; i<receipts.length(); ++i)
{
    BSONObj receipt = receipts.get(i);
    if(receipt.getInt(djonDBUtil.USER_RECEIPT_ID_FIELDNAME) == rid)
    {
        BSONArrayObj items = receipt.getBSONArray(djonDBUtil.USER_RECEIPT_ITEMS_FIELDNAME);
        BSONObj item = new BSONObj();
        item.add(djonDBUtil.USER_ITEM_NAME_FIELDNAME, itemName);
        item.add(djonDBUtil.USER_ITEM_PRICE_FIELDNAME, itemPrice);
        item.add(djonDBUtil.USER_ITEM_QTY_FIELDNAME, qty);
        item.add(djonDBUtil.USER_ITEM_TAXRATE_FIELDNAME, taxRate);
        items.add(item);
        double total = subtotal + receipt.getDouble(djonDBUtil.USER_RECEIPT_TOTAL_FIELDNAME);
        System.out.println("total = " + total);
        receipt.add(djonDBUtil.USER_RECEIPT_TOTAL_FIELDNAME, total);
        c.update(djonDBUtil.DBNAME, djonDBUtil.USERS_COLLECTION_NAME, userDoc);
        ret.put("ret", true);
        break;
    }
}
```



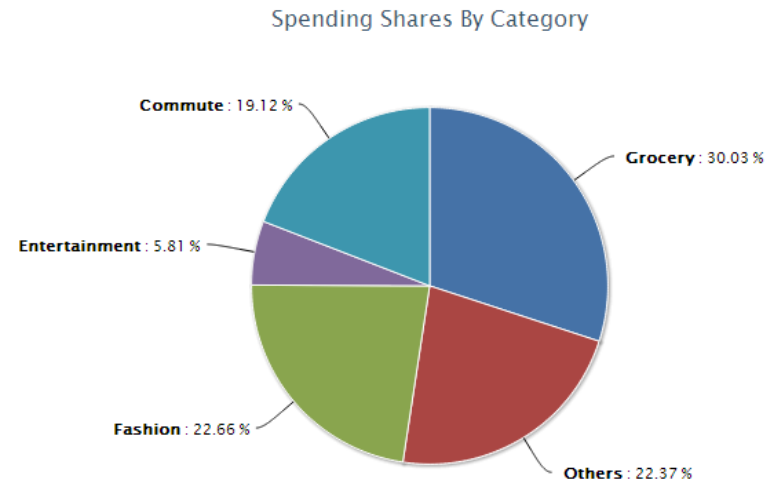
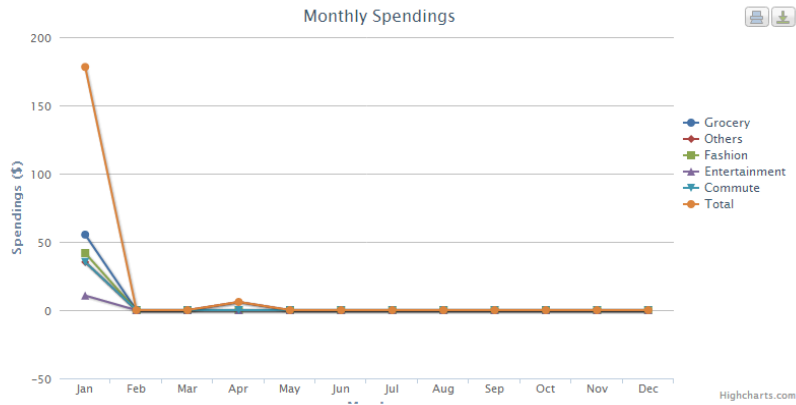
Implementation

Display Receipt





Display Charts





Demo