Khadev

www.facebook.com/khadev

MEAN STACK 1.0

By Apaichon Punopas

What is MEAN Stack ?

- M - MongoDB
- E - Express
- A - Angular
- N - NodeJS

Course Outline

Day 1

MongoDB

- Introduction of MongoDB
- Installation
- Administrator tool with Robomongo
- Insert a document
- Insert multiple documents
- Document validator
- Update document
- Delete document
- Find document

— Break —

- Finding with condition (>,<,>=,<=)
- Finding with OR
- Finding with AND
- Finding with NOT
- Finding with where
- Finding with Regular Expression
- Finding with complex condition
- Prepare more data for afternoon

— Launch —

- Introduction of Indexing
- Using explain command
- Single field index
- Unique index
- Compound index
- Time to live index
- Using hint

— Break —

NodeJS

- Basic NodeJS
- Install NodeJS and NPM
- JavaScript Basic
- Module
- Asynchronous Callback
- Promise

Course Outline

DAY 2

NodeJS Express

- MongoDB Driver
- Restful
    - Create
    - Read
    - Update
    - Delete
- SOAP
- Routing and Log Center

— Break —

AngularJS

- Installation
- Angular Basic
- Directive
- Expression
- Two way binding
- ng-repeat

— Launch —

- ng-app
- ng-module
- controller
- filter
- angular-material

— Break —

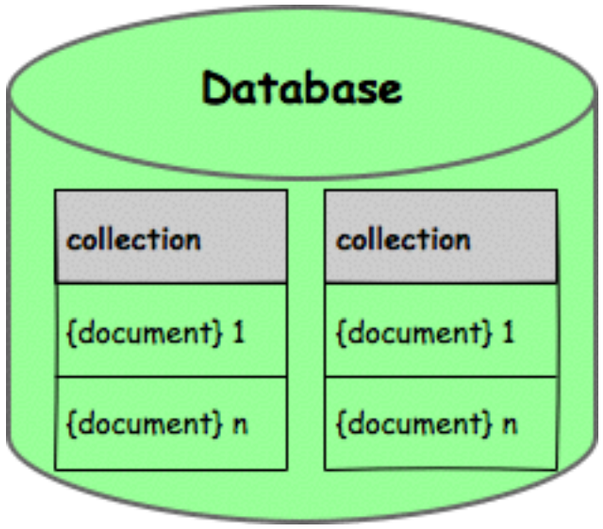
- requirejs
- CRUD
    - Create
    - Read
    - Update
    - Delete
- Unit Test with Karma

MongoDB

- NoSQL
- Document oriented database
- Low resources
- High availability
- Easy scalability

Structure

- Database : Database is a physical container for collections



- Collection : Collection is a group of MongoDB documents
- Document : A document is a set of key-value pairs

Relational Mapping

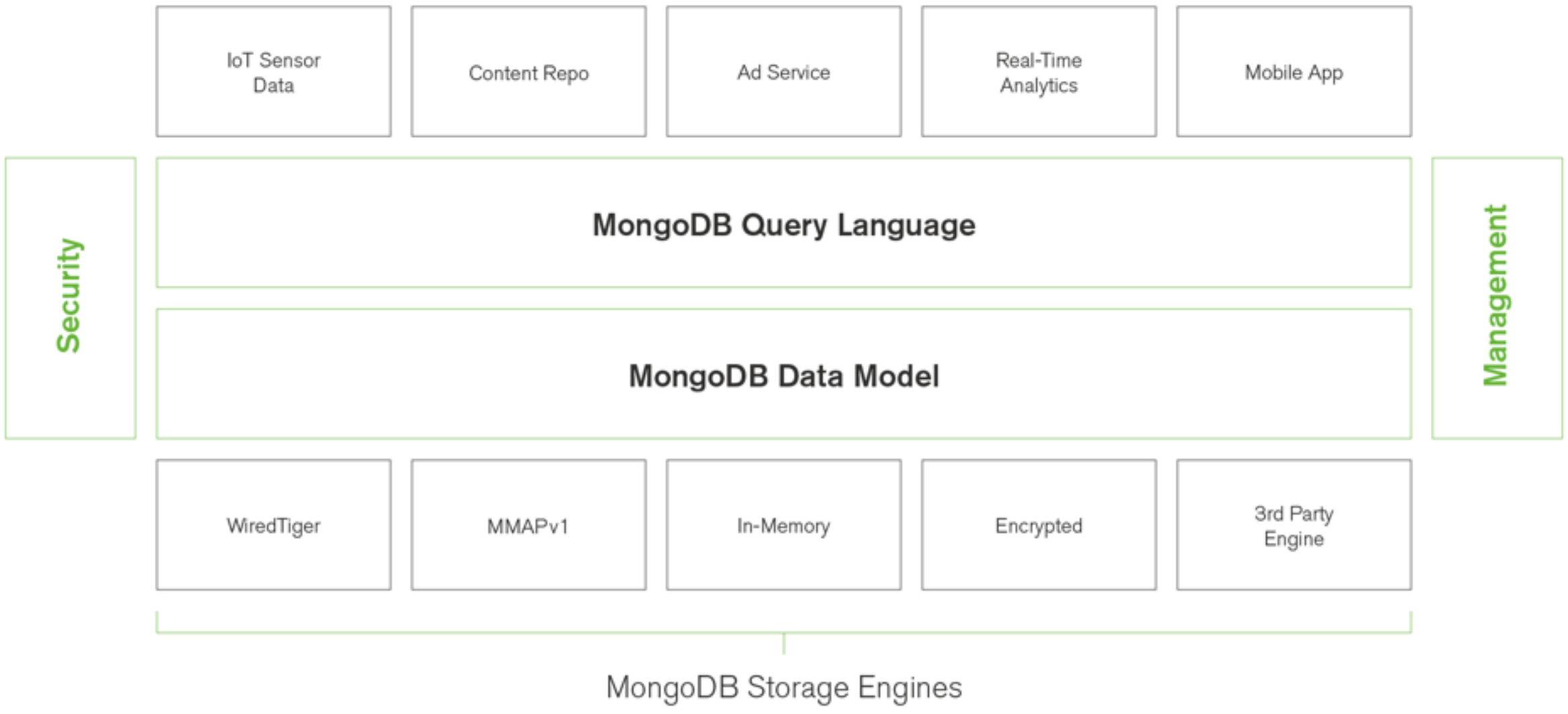
#	Relational Database	MongoDB
1	Database	Database
2	Table	Collection
3	Row	Document
4	Column	Field
5	Table Join	Embedded
6	Primary Key	Primary Key (Default _id)

Installation

1. Download at https://www.mongodb.com/download-center#community
2. Unzip file
3. Open Terminal and go to bin path
4. run command

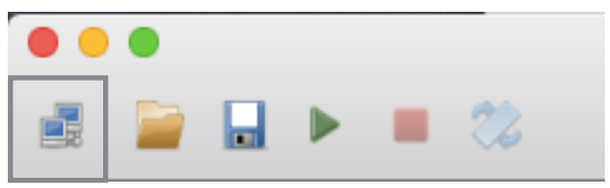
```
bin>mongod - -dbpath “<db directory>”
```

Architecture



Robo Mongo Installation

1. Download at https://robomongo.org/download
2. Follow Installation Wizard
3. Double click on application Icon
4. Setup connection



Insert document

Objective

Insert data into collection.

Format

```
db.collection.insert(
   <document or array of documents>,
   {
     writeConcern: <document>,
     ordered: <boolean>
   }
)
```

Example

```
db.test.insert({name:"Test",createdAt: new Date()})
```

Find document

Objective

Search data from collection.

Format

```
db.collection.find(query, projection)
```

- Query : Filter condition
- Projection : Show fields

Example

```
db.test.find()
db.test.findOne({name:"S6"})
db.test.find({subject:"Physics 2"})
db.test.find().limit(2)
```

Insert multiple documents

Example

```
db.test.insert([
{name:"S1",major:"Computer Science",subject:"Math",score:90},
{name:"S2",major:"Computer Engineer",subject:"Physics",score:75},
{name:"S3",major:"Information Technology",subject:"Physics",score:60},
{name:"S4",major:"Computer Science",subject:"Math",score:77},
{name:"S5",major:"Information Technology",subject:"Physics",score:88},
{name:"S6",major:"Information Technology",subject:"Programming I",score:65},
{name:"S7",major:"Computer Engineer",subject:"Physics",score:55},
{name:"S8",major:"Computer Science",subject:"Programming I",score:66},
{name:"S9",major:"Computer Education",subject:"Math",score:77},
{name:"S10",major:"Information Technology",subject:"Physics",score:59},
{name:"S11",major:"Computer Engineer",subject:"Programming I",score:56},
{name:"S12",major:"Computer Science",subject:"Physics",score:98},
{name:"S13",major:"Information Technology",subject:"Math",score:72},
{name:"S14",major:"Computer Engineer",subject:"Programming I",score:69},
{name:"S15",major:"Computer Engineer",subject:"Physics",score:67},
{name:"S16",major:"Computer Science",subject:"Math",score:73},
{name:"S17",major:"Computer Engineer",subject:"Physics",score:99},
{name:"S18",major:"Computer Engineer",subject:"Math",score:77},
{name:"S19",major:"Information Technology",subject:"Programming I",score:66},
{name:"S20",major:"Information Technology",subject:"Physics",score:66},
])
```

Document validator

Objective

Validate data type before modify

Format

```
db.createCollection( "collection",{
      validator: { <$or,$and>:
         [
            { <field>: { $type: "string" } },
           { <field>: { $exists: true } },
            { <field>: { $regex: /@mongodb\.com$/ } },
            { <field>: { $in: [ "Unknown", "Incomplete" ] } }
         ]
      },
      validationAction: <"warn","error">
   }
)
```

Example

```
db.createCollection( "customers",{
      validator: { $and:
         [
            { idCardNo: { $exists: true,$type:"string" } },
            { firstName: { $exists: true,$type:"string" } },
            { lastName: { $exists: true,$type:"string" } },
            { registeredDate: { $exists: true,$type:"date" } },
            { sex: { $in: [ "M", "F" ] } },
            { email:{ $regex: /@mongodb\.com$/ }},
            { mobileNo:{ $regex: /^\(\d\d\d\)\d\d\d-\d\d\d\d$/ }}
         ]
      },
      validationAction: "error"
   }
)
```

Update Document

Objective

Modify data in collection.

Format

```
db.collection.update(
   <query>,
   <update>,
   {
     upsert: <boolean>,
     multi: <boolean>,
     writeConcern: <document>
   }
)
```

Example

```
db.test.update({"name":"S19"},{"name":"S19","score":75})
```

```
db.test.update({"_id" :ObjectId("573817afbec88ff1af511d54")}
             ,{"$set" : {"name" : "Student 3","score":90,"subject":"Physics 2”}})
```

```
db.test.update({name:"S21"},{"name":"S21","score":75 , "major":"Statistic"
             , "subject":"Physics"},{"upsert":true})
```

Delete Document

Objective

Delete data in collection.

Format

```
db.collection.remove(
   <query>,
   <justOne>
)
```

Example

```
db.test.remove({"subject":"Physics"},{"justOne":true})
```

```
db.test.remove({"subject":"Physics"})
```

— Break —

Finding Condition

- > , >= , < , <=
- OR
- AND
- NOT
- Where
- Special Value
- Complex Condition

Finding by
> , >= , < , <=

Objective

Search data in collection with condition (>,>=,<,<=).

Format

```
db.collection.find({field:{$gt:value}})
db.collection.find({field:{$gte:value}})
db.collection.find({field:{$lt:value}})
db.collection.find({field:{$lte:value}})
```

Example

```
db.test.find({score:{$gt:60}})
db.test.find({score:{$gte:60}})
db.test.find({score:{$lt:60}})
db.test.find({score:{$lte:60}})
```

Finding by OR

Objective

Search data in collection with condition ***OR***.

Format

```
db.collection.find({$or:[{field:value},{field:value}]})
```

Example

```
db.test.find({$or:[{major:"Computer Science"},{subject:"Math II"}]})
```

Finding by AND

Objective

Search data in collection with condition ***AND***.

Format

```
db.collection.find({$and:[{field:value},{field:value}]})
```

Example

```
db.test.find({"$and":[ {"major":"Computer Science"}, {"score":{"$gt":80}}]})
```

Finding by NOT

Objective

Search data in collection with condition ***NOT***.

Format

```
db.collection.find({field:{$not:value,document}})
```

Example

```
db.test.find({major:{$not:{$eq:"Computer Science"}}})
```

Finding by Where

Objective

When need compare in internal fields.

Example

```
db.orderProduct.insert([
{"item":"jacket","qty":5,"inStock":5}
,{"item":"polo","qty":20,"inStock":10}
,{"item":"t-shirt","qty":3,"inStock":99}
])

db.orderProduct.find({$where:function(){return this.qty > this.inStock;} })
```

Finding by Special Value

Objective

Find data of special value such as ***null***,***regular expression*** and ***array***.

Example

```
db.test.insert({"name":"S23",
    "classDates":["Monday",
    "Tuesday",
    "Wednesday",
    "Thursday",
    "Friday"]}
    )

db.test.find({subject:null})
db.test.find({"major" : /tion/})
db.test.find({classDates:{$size:5}})
db.test.find({name:"S23"},{classDates:{"$slice":[2,2]}})
db.test.find({classDates:{$all:["Monday","Friday"]}})
```

Finding by Complex Condition

Condition

- major in computer science and Information technology
- subject is Physics
- score <60 or score >79

Example

```
db.test.find({
    "major": { "$in": ["Computer Engineer", "Inforemation Technology"] }
    , "subject": "Physics"
    , "$or": [{ "score": { "$lt": 60 } },
       { "score": { "$gt": 79 } }
    ]
})
```

Prepare Data for Indexing

- Download data at
  https://github.com/apaichon/courses-mongo/raw/master/mongodata.zip
  https://github.com/apaichon/courses-mongo/blob/master/labimport.sh

- Extract and copy to bin path

- Open file labimport.sh then edit test as <dbname>

```
./mongoimport --db test --collection customers --file customers.json
```

- Open terminal then run command

```
bin>sh labimport.sh
```

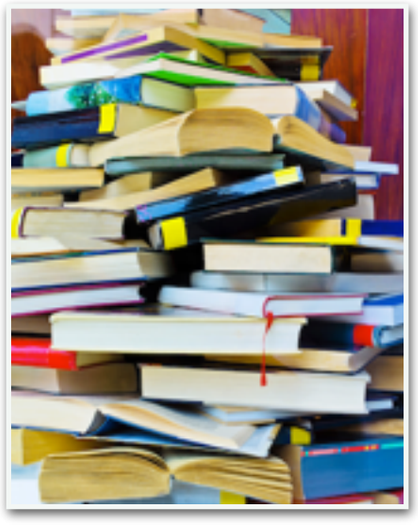
— Launch—

What is Index ?

- The way to increase speed of finding data

#	Pros	Cons
1	Faster finding	Increase Disk space
2		Decrease Insert speed

No Index



Index

Explain

Objective

Help to analysis query performance.

Format

```
db.collection.find(query, projection).explain()
```

Example

```
db.saleBooks.find({customerId:"1690091687999"}).explain()
```

Single field index

Objective

Create main single key that your application always search it.

Format

```
db.collection.createIndex(keys, options)
```

Example

```
db.saleBooks.createIndex({"customerId":1})
```

```
db.saleBooks.find({customerId:"1690091687999"}).explain()
```

Unique index

Objective

Create unique index that not allow has duplicate data.

Format

```
db.collection.createIndex(keys, {"unique":true})
```

Example

```
db.customers.createIndex({"idCardNo":1},{"unique":true})
```

```
db.customers.insert({"idCardNo":"1698042855799","name":"test"})
```

Compound index

Objective

Create multiple keys to be index.

Format

```
db.collection.createIndex( { <field1>: <type>, <field2>: <type2>, ... } )
```

Example

```
db.saleBooks.createIndex({"saleDate":-1,"bookId":1},
  {"name":"idx_saleBooks_saleDate_bookId"})
```

```
db.getCollection('saleBooks').find( {"saleDate":
  {"$gte": new Date("2015-01-01")} }).explain()
```

Time to live index

Objective

Create keys which want to automatically delete data after expired.

Format

```
db.collection.createIndex(keys,{"expireAfterSeconds":true})
```

Example

```
db.session.createIndex({"logon":1},{"expireAfterSeconds":60})
```

```
db.session.insert({name:"User1",logon: new Date()})
```

```
db.session.find()
```

Khadev
www.facebook.com/khadev

Hint

Objective

Force mongodb to use index name that expected.

Format

```
db.collection.find(query).hint("indexName")
```

Example

```
db.saleBooks.createIndex({"saleDate":-1},{"name":"idx_saleBooks_saleDate"})
```

```
db.getCollection('saleBooks').find( {"saleDate": {"$gte": new
Date("2015-01-01")} }).explain()
```

```
db.getCollection('saleBooks').find( {"saleDate": {"$gte": new
Date("2015-01-01")} }).hint("idx_saleBooks_saleDate").count()
```

Aggregation

Objective

Transform multiple data to few record and change result.
* Process in memory and limit to 200 MB.

Format

```
db.collection.aggregate([{$match: {}},{$group: {}},{$project:{}} ]);
```

Example

```
db.saleBooks.aggregate(
    [ { $match:
        {"$or":[
            {"saleDate":{"$gte": new Date ("2015-01-01")}},
            {"saleDate":{"$lte": new Date ("2015-12-31")}}
        ] }
    },
        {$group: { _id: "$bookId",unitPrice:{$sum:"$price"},
                totalQty:{"$sum":"$qty"} }},
        {$project:{"totalPrice":"$unitPrice",
            "totalQty":"$totalQty",
            "totalSales": {"$multiply":["$unitPrice","$totalQty"]}}}
        ]
 );
```

Map Reduce

Objective

Transform multiple data to few record and change result.

* Slower than aggregate but flexible and support multiple node.

Format

```
db.collection.mapReduce(<map>,<reduce>,{
                                out: <collection>,
                                query: <document>,
                                sort: <document>,
                                limit: <number>,
                                finalize: <function>,
                                scope: <document>,
                                jsMode: <boolean>,
                                verbose: <boolean>,
                                bypassDocumentValidation: <boolean>
                            }
                        )
```

Example

```
db.places.mapReduce(
     function() {
          var key = this.province.substring(2);
          emit(key,1);
     },

     function(key, values) {
          return   Array.sum(values);
     },
     {
          out: "province_total"
     }
)
```

Join

Objective

Refer to another field of foreign collection it help to reduce keep some data in multiple collections and reduce store data size.

Format

```
db.collection.aggregate({
   $lookup:
     {
       from: <collection to join>,
       localField: <field from the input documents>,
       foreignField: <field from the documents of the "from" collection>,
       as: <output array field>
     }
})
```

Example

```
db.saleBooks.aggregate([
   {$lookup:{from:"books"
       ,localField:"bookId"
       ,foreignField:"isbn"
       ,as:"book"
       }}
   ])
```

Security

Objective

Create user and Permission for access to database.

Format

```
db.createUser({ user: "<name>",
  pwd: "<cleartext password>",
  customData: { <any information> },
  roles: [
    { role: "<role>", db: "<database>" } | "<role>",
    ...
  ]
})
```

Example

```
db.createUser(
   {
     user: "dbAdmin",
     pwd: "p@ssw0rd",
     roles: [ "readWrite", "dbAdmin" ]
   }
)
```

Security

Objective

Restart mongod for access to authentication mode.

Example

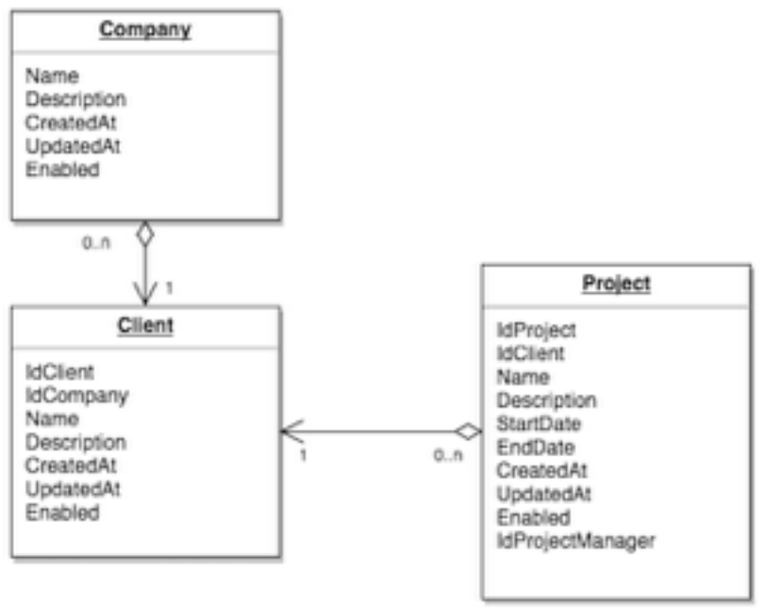
```
bin>mongod - - dbpath “<dbpath>” auth
```

Normalize Design

Objective

If need to reduce duplicate data ,store size and intermediate change data information with relation. ***For mongo possible on 3.2 +***

#	Pros	Cons
1	Reduce duplicate data	Take performance while join
2	Reduce store size	Complicate Relation Design
3	Intermediate change data	Difficult to migrate



Denormalize Design

Objective

Keep imperative data or evidence such as invoice, receipt, etc.
Has no relation and effect from other collection.
MongoDB has ***limitation is 16 MB for store array or embledded***.

Example

```
{"orderId":1 , "orderDate":"2016-07-01",
     "items":
     [
          {
               "productId":1, "productName":"TV 21 inch","qty":1 ,"unitPrice",15000}
          },
          {
               "productId":2, "productName":"Paper A4","qty":12 ,"unitPrice",10}
          }
     ]
}
```

— Break—

Course Outline

Day 1

MongoDB

- Introduction of MongoDB
- Installation
- Administrator tool with Robomongo
- Insert a document
- Insert multiple documents
- Document validator
- Update document
- Delete document
- Find document

— Break —

- Finding with condition (>,<,>=,<=)
- Finding with OR
- Finding with AND
- Finding with NOT
- Finding with where
- Finding with Regular Expression
- Finding with complex condition
- Prepare more data for afternoon

— Launch —

- Introduction of Indexing
- Using explain command
- Single field index
- Unique index
- Compound index
- Time to live index
- Using hint

— Break —

NodeJS

- Basic NodeJS
- Install NodeJS and NPM
- JavaScript Basic
- Module
- Asynchronous Callback
- Promise

Course Outline

DAY 2

NodeJS Express

- MongoDB Driver
- Restful
    - Create
    - Read
    - Update
    - Delete
- SOAP
- Routing and Log Center

— Break —

AngularJS

- Installation
- Angular Basic
- Directive
- Expression
- Two way binding
- ng-repeat

— Launch —

- ng-app
- ng-module
- controller
- filter

— Break —

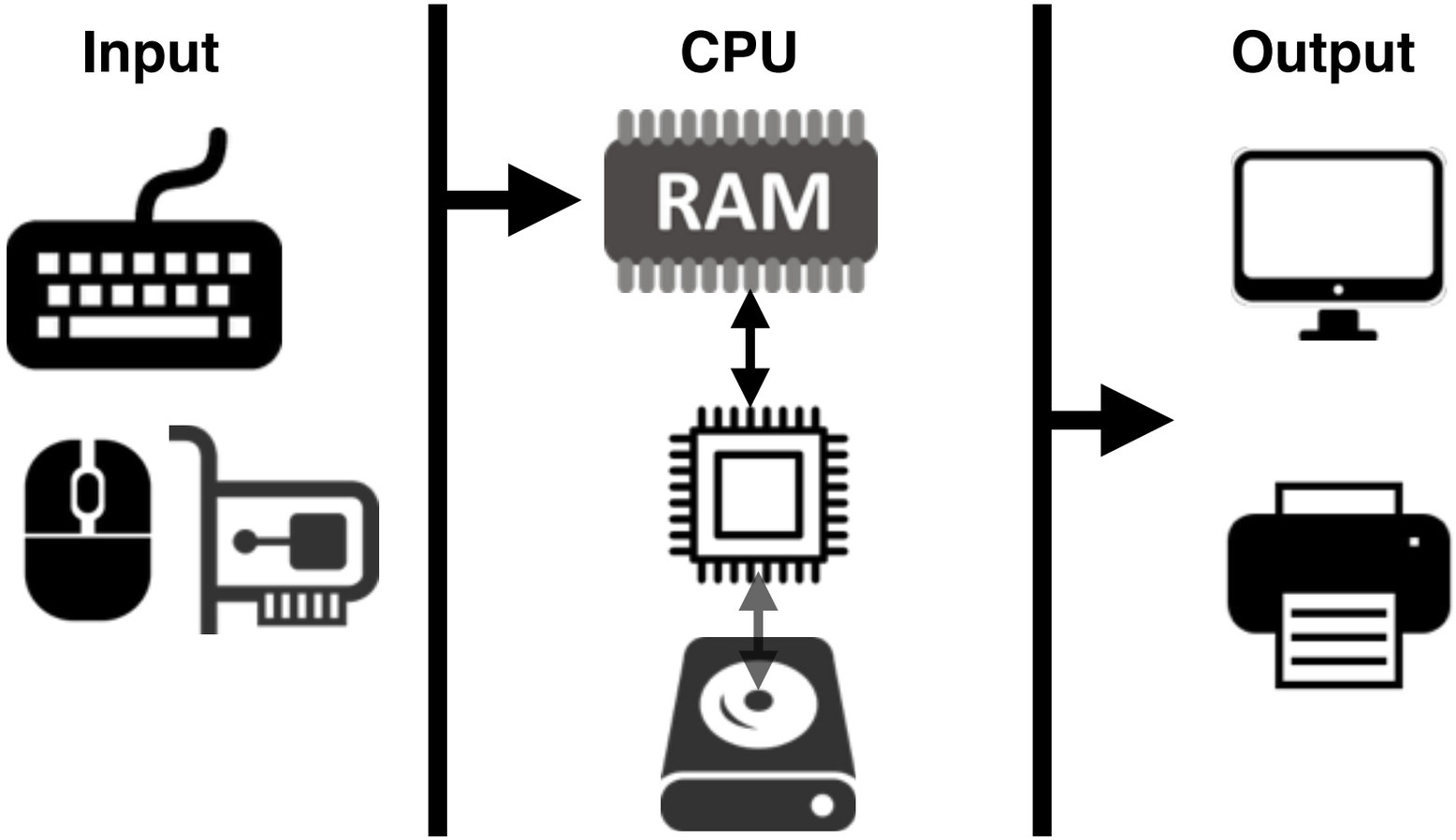
- requirejs
- CRUD
    - Create
    - Read
    - Update
    - Delete
- Unit Test with Chai and Mocha

What is NodeJS ?

- Node.js is a server-side platform.
- Built on Google Chrome's JavaScript Engine (V8 Engine).
- It was developed by Ryan Dahl in 2009.
- It is an open source.
- Cross-platform runtime environment for developing server-side and networking applications.
- Node.js uses an event-driven, **non-blocking I/O** model.

What is I/O ?

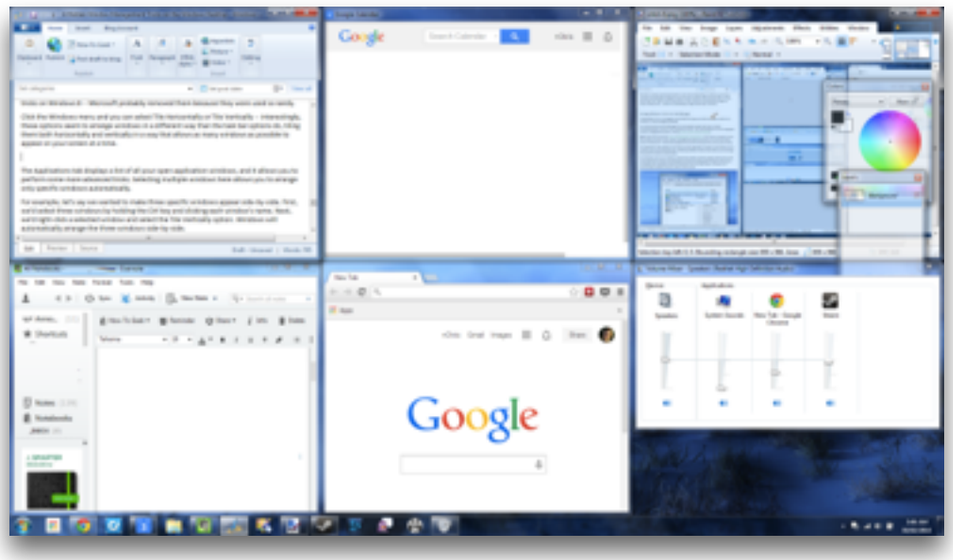
- The term I/O is used to describe any program, operation or device that transfers data to or from a computer and to or from a peripheral device.



Khadev
www.facebook.com/khadev

What is Blocking I/O ?

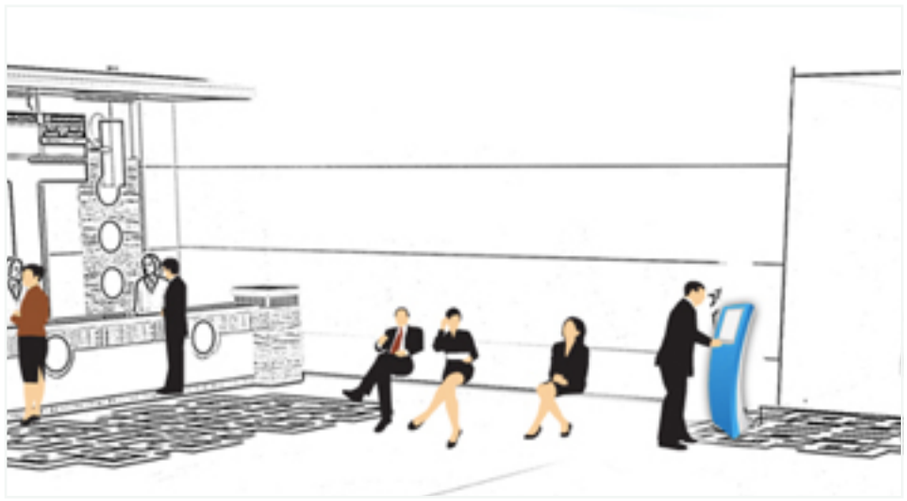
- Most I/O requests are considered blocking requests, meaning that control **does not return to the application until the I/O is complete**. The delayed from systems calls such read() and write() can be quite long. Using systems call that block is sometimes call synchronous programming.

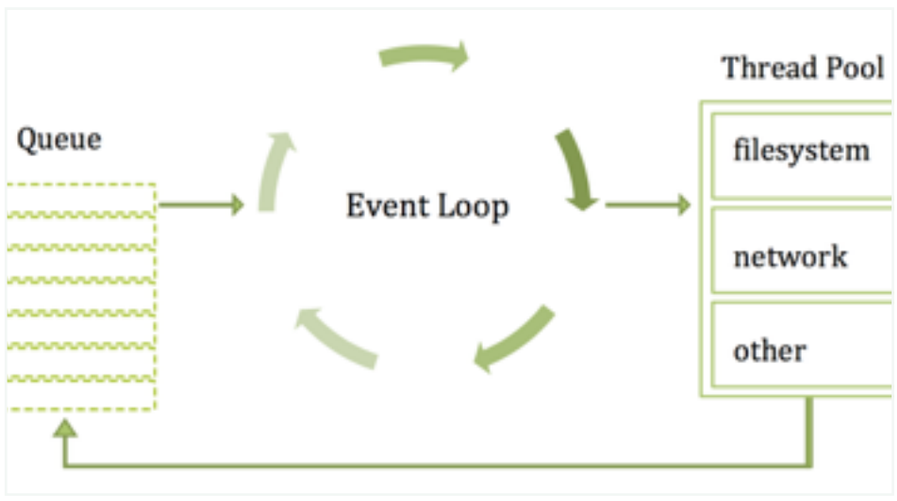


Khadev
www.facebook.com/khadev

What is non blocking I/O ?

- Programs that use non-blocking I/O tend to follow the rule that every function has to return immediately, i.e. all the functions in such programs are nonblocking. Thus control passes very quickly from one routine to the next.





Create NodeJS App

Workshop 1

1. At Terminal type command.

```
>npm init
```

2. Naming your application and version.

3. Fill information step by step.

4. After finish will get file name is **package.json**

Installation NodeJS

1. Go to https://nodejs.org/en/download/
2. Choose NodeJS for your OS.
3. Extract file and Click on Installer package.
4. After finish installed then open terminal and type command.

```
>node -v
```

First NodeJS Application

1. Create file and type code following.

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello NodeJS');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

5. Save file as “workshop1.js”

6. Run command

```
>node workshop1.js
```

JavaScript Basic

- Assign variable

```
var a = 1;
var b = "Hello";
var c = true;
var d = [1,2,3,4,5,"Test","JavaScript"]
var e = {id:1,fname:"John", lname:"Doe"};
var f = [{id:1,fname:'John',lname:"Doe"},{id:2,fname:"Jane",lname:"Doe"}];
var e = function(a,b){
   return a+b;
}
var h = {id:1,fname:"John",lname:"Doe"
      ,say:function(){
         return "Hello JavaScript";
      }
   }
```

JavaScript Basic

- Operator

```
var result = (1*5) + (10-2) % 2 ;
```

- Condition

```
if (totalCar < 100 ) {
     traffic= "normal";
} else if (totalCar < 200) {
    traffic = "jam";
} else {
    traffic = "Heavy jam";
}
```

```
switch(grade){
    case 4:
        console.log('Perfect');
    break;
    case 3:
        console.log('Good');
    break;
    case 2:
        console.log('Standard');
    break;
    case 1 :
        console.log('Need to Improve');
    break;
    case 0 :
        console.log('Fail');
    break;
    default:
        console.log('Not in range')
    break;
}
```

JavaScript Basic

- Loop

```
var cars =['Ferrari','Benz','BMW','Mazda','Toyota','Honda'];
for (i = 0; i < cars.length; i++) {
    console.log(cars[i]);
}

for(var i in cars){
   console.log(cars[i]);
}

var i =0,length = cars.length;
while(i < length){
   console.log(cars[i]);
   i++;
}
```

Module

Objective

- Code Grouping
- Make code to smaller
- Code Scalable
- Reusable

Create a Module

Workshop 2

1. Create file name as **aggregation.js** then input code following.

```
exports.sum = function(){
    this._sum = 0;
    for (var i = 0; i < arguments.length; i++) {
       this._sum+=arguments[i];
    }
}

exports.avg = function(){
    this._sum = 0;
    for (var i = 0; i < arguments.length; i++) {
       this._sum+=arguments[i];
    }
    this._avg = this._sum / arguments.length;
}
```

Khadev
www.facebook.com/khadev

Create a Module

2. Create file name as **workshop2.js** then input code following.

```
var a = require('./aggregation');
    a.sum(1,2,4,5,6,7);
    console.log(a._sum);
    a.avg(4,2,5,6,7,9,18,11);
    console.log(a._avg);
```

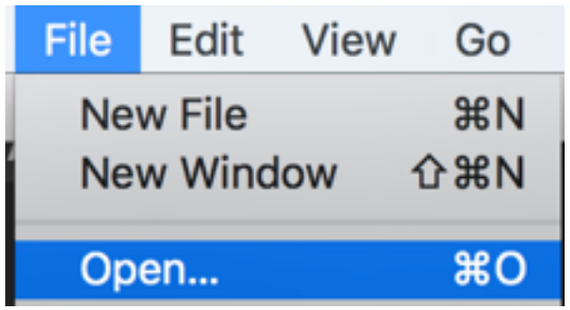
3. Run application.

```
>node workshop2.js
```

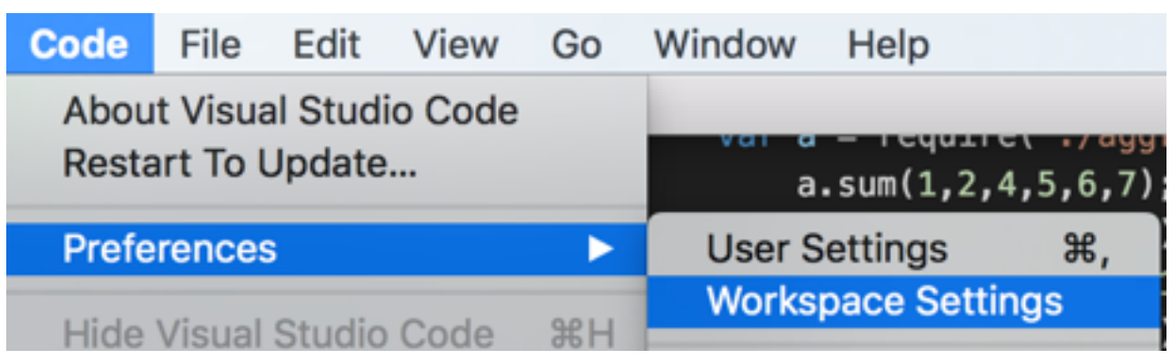
Khadev
www.facebook.com/khadev

Debugging on vscode

1. Open Project Folder.



2. Set workspace.



3. Create file as launch.json on .vscode folder.

Khadev
www.facebook.com/khadev

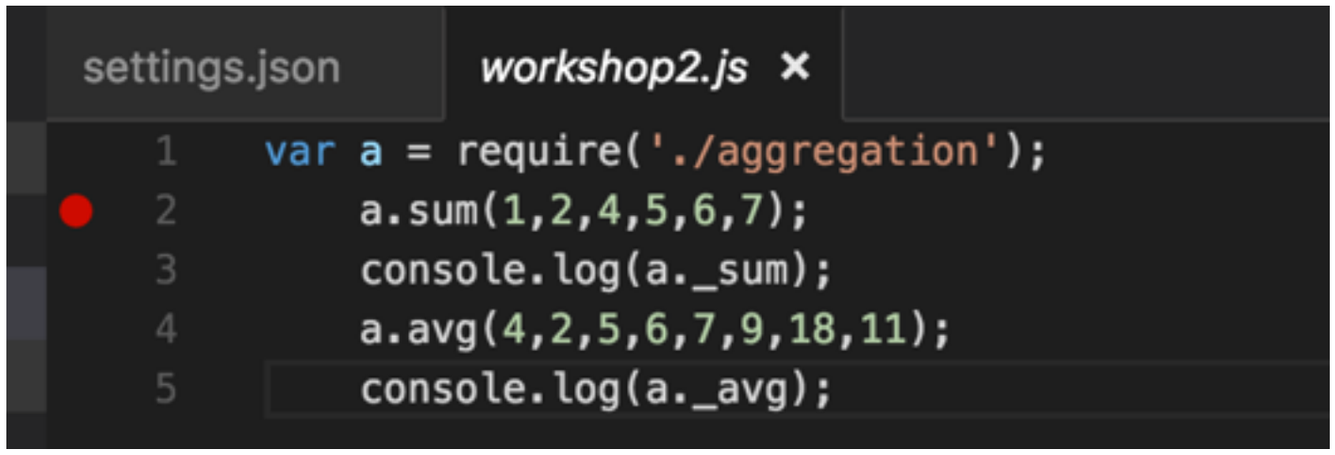
Debugging on vscode

4. Input code in launch.json following.

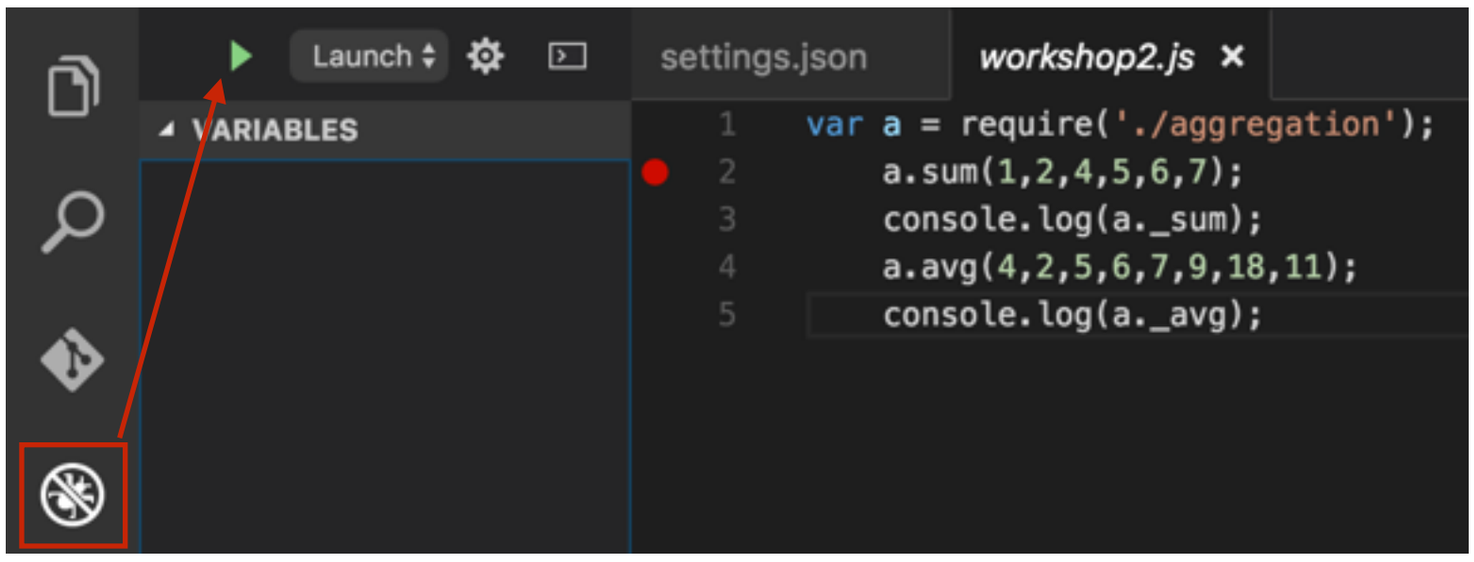
```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Launch",
            "type": "node",
            "request": "launch",
            "program": "${workspaceRoot}/workshop2.js",
            "stopOnEntry": false,
            "args": [],
            "cwd": "${workspaceRoot}",
            "preLaunchTask": null,
            "runtimeExecutable": null,
            "runtimeArgs": [
                "--nolazy"
            ],
            "env": {
                "NODE_ENV": "development"
            },
            "console": "internalConsole",
            "sourceMaps": false,
            "outDir": null
        },
        {
            "name": "Attach",
            "type": "node",
            "request": "attach",
            "port": 5858,
            "address": "localhost",
            "restart": false,
            "sourceMaps": false,
            "outDir": null,
            "localRoot": "${workspaceRoot}",
            "remoteRoot": null
        },
        {
            "name": "Attach to Process",
            "type": "node",
            "request": "attach",
            "processId": "${command.PickProcess}",
            "port": 5858,
            "sourceMaps": false,
            "outDir": null
        }
    ]
}
```

Debugging on vscode

5. Click on the line want to start debugging.



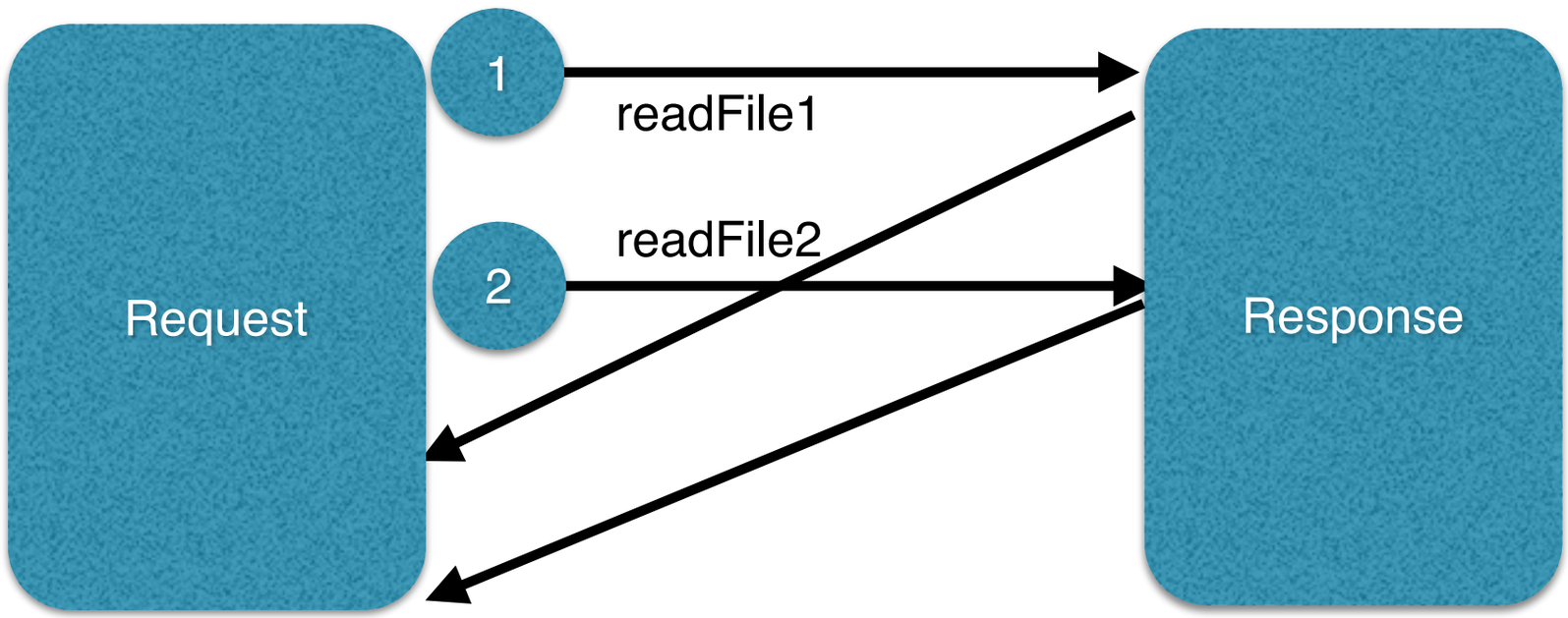
6. Click on debug icon then click start debugging.



Asynchronous Callback

Objective

- Asynchronous is invoke function by order but no wait result by order last function was called might finish before.



Asynchronous Callback

Workshop 3

```
var fs = require('fs');
function readTotalLines(filePath, callback) {
    fs.readFile(filePath, (err, data) => {
        if (err) throw err;
        var to_string = data.toString();
        var split_lines = to_string.split("\r");
        callback(split_lines.length);
    });

}

var file1 = "./resources/FL_insurance_sample.csv";
var file2 = "./resources/us-500.csv";

readTotalLines(file1, function (totalLines) {
    console.log('file1:', totalLines);
});

readTotalLines(file2, function (totalLines) {
    console.log('file2:', totalLines);
});

readTotalLines(file1, function (totalLines) {
    readTotalLines(file2, function (totalLines2) {
        console.log('total records:', totalLines + totalLines2);
    });
});
```

Promise

Objective

- Promise is asynchronous library to solve callback hell problem and help to make beautiful code.
- There are 3 states as below
  - Pending -> First State pending to process
  - Fulfilled -> Return completed result
  - Rejected -> Return error or fail result

Promise

Workshop 4

```
var fs = require('fs');
function readTotalLines(filePath){
  return new Promise( function(resolve, reject) {
     fs.readFile(filePath, (err, data) => {
        if (err) {reject("Not found " + filePath );return;}

        var to_string = data.toString();
        var split_lines = to_string.split("\r");
        resolve(split_lines.length);
    });
  });
}

var file1 = "./resources/FL_insurance_sample.csv";
var file2 = "./resources/us-500.csv";

Promise.all([readTotalLines(file1)
,readTotalLines(file2)
,readTotalLines('./resources/contacts3.txt')
,readTotalLines('./resources/contacts4.txt')
]
).then(function(values){
  console.log(values);
}).catch(function(errors){
    console.log(errors);
})
```

— End Day 1—

Install Express

Objective

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

1. Open Terminal then type

```
>npm install express - - save
```

First Application with Express

Workshop 5

1. Create file name as ***"app.js"*** then type code following.

```
var express = require("express");
var app = express();

app.get('/', function (req, res) {
  res.send('Hello NodeJS');
})
app.listen(3000);
console.log("My Service is listening to port 3000.");
```

2. Run command.

```
>node app.js
```

Install multiple packages

Workshop 6

1. Open file ***"package.json"*** then edit code at dependencies following.

```
{
  "name": "myservice",
  "description": "Rest Api for my Enterprise.",
  "author": "Apaichon Punopas <apaichon@gmail.com>",
  "dependencies": {
    "body-parser": ">= 1.14.2",
    "crypto-js": ">= 3.1.5",
    "express": ">= 4.13.3",
    "express-session": "^1.13.0",
    "guid":"latest",
    "mongodb": "^2.1.18",
    "rootpath": ">= 0.1.2",
    "winston": ">= 2.1.1",
    "useragent":"latest"
  }
}
```

2. Run command.

```
>npm install
```

Khadev
www.facebook.com/khadev

REST API

Objective

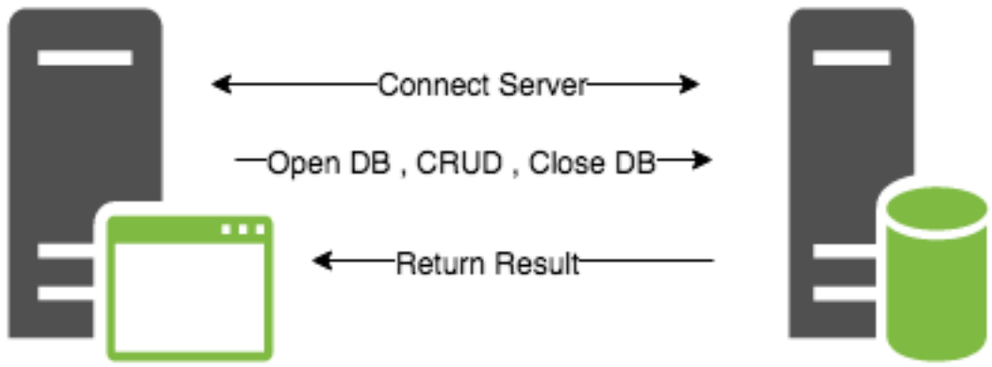
The uniform interface constraint defines the interface between clients and servers. It simplifies and decouples the architecture, which enables each part to evolve independently.

MongoDB Driver

Objective

Create restful api for working with mongodb.

Basic concept working with database driver.



Khadev
www.facebook.com/khadev

REST API with CRUD

Objective

Create restful api for CRUD with mongodb.

Workshop 7 Create ***Insert*** function

- Open file ***"app.js"*** then add code inside following.

```
var MongoClient = require('mongodb').MongoClient;
var dbUrl ="mongodb://localhost:27017/social";
var ObjectID = require('mongodb').ObjectID;

app.post('/contacts/add',function(req,res){
  MongoClient.connect(dbUrl,function(err,db){
      if(err) res.send(err);

      var id = new ObjectID();
      var data = req.body;
      data["_id"] = id.toHexString();
      db.collection('contacts')
      .insertOne(data,function(err,result){
        db.close();
        if(err) res.send(err);
        res.send(result);
      });
  });
});
```

REST API with CRUD

Objective

Create restful api for CRUD with mongodb.

Workshop 8 Create ***get*** function

- Open file ***"app.js"*** then add code inside following.

```
app.get('/contacts/getMany',function(req,res){
  MongoClient.connect(dbUrl,function(err,db){
      if(err) res.send(err);
      db.collection('contacts')
      .find(req.query).toArray(function(err,result){
        db.close();
        if(err) res.send(err);
        res.send(result);
      });
  });
});
```

REST API with CRUD

Objective

Create restful api for CRUD with mongodb.

Workshop 9 Create ***update*** function

- Open file ***"app.js"*** then add code inside following.

```
app.put('/contacts/update',function(req,res){
  MongoClient.connect(dbUrl,function(err,db){
      if(err) res.send(err);
      var opts  = (req.body.opts?req.body.opts:{});
      db.collection('contacts')
      .update(req.body.criteria,req.body.data,opts,function(err,result){
        db.close();
        if(err) res.send(err);
        res.send(result);
      });
  });
})
```

REST API with CRUD

Objective

Create restful api for CRUD with mongodb.

Workshop 10 Create ***delete*** function

- Open file ***"app.js"*** then add code inside following.

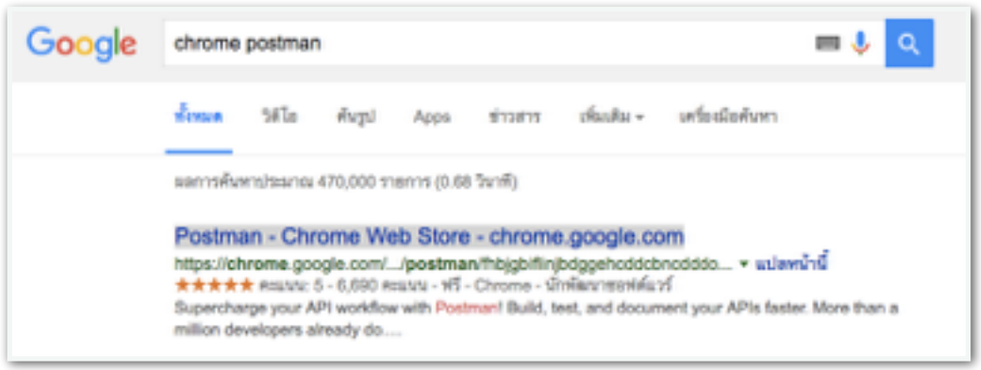
```
app.delete('/contacts/delete',function(req,res){
  MongoClient.connect(dbUrl,function(err,db){
      if(err) res.send(err);
      db.collection('contacts')
      .remove(req.body,function(err,result){
        db.close();
        if(err) res.send(err);
        res.send(result);
      });
  });
});
```

Install Postman

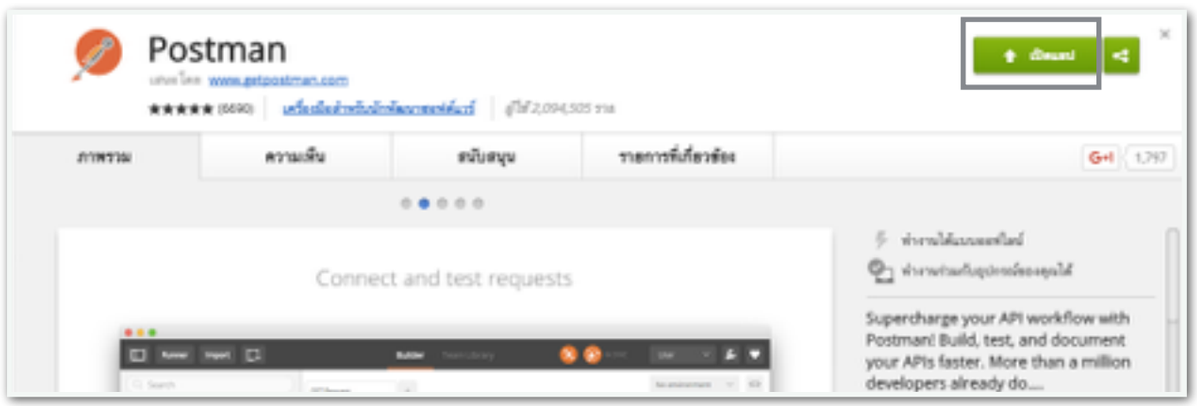
Objective

Install Chrome Postman for test Restful API.

1. Find in Google by wording is ***“Chrome Postman”***



2. Click add chrome plugin



Test CRUD by Postman

Objective

Test CRUD Rest API by Postman to make sure that all function are work correctly .

1. Open ***"Chrome Postman"*** then set parameter following.

Test CRUD by Postman

2. Prepare test parameters following.

#	method	url	parameters

1	post	http://localhost:3000/contacts/add	{
			"firstName" : "<Your first Name>",
			"lastName" : "<Your last name>",
			"middleName" : "",
			"gender" : "M",
			"email" : "<Your email>",
			"mobile" : "<Your Mobile"
			}

2	get	http://localhost:3000/contacts/getMany	

3	put	http://localhost:3000/contacts/update	{
			"criteria":{ "firstName":"<Your first Name>" },
			"data":{ "$set":{ "firstName":"<new name>" } },
			"opts":{ "multi":true }
			}

4	delete	http://localhost:3000/contacts/delete	{
			"firstName" : "<Your first Name>"
			}

Test CRUD by Postman

3. Make sure that in ***“app.js”*** has ***body parser*** .

```
var bodyParser = require('body-parser');
app.use(bodyParser.json());
```

4. Rerun app.js then start to test on postman and check data in mongodb.

— Break —

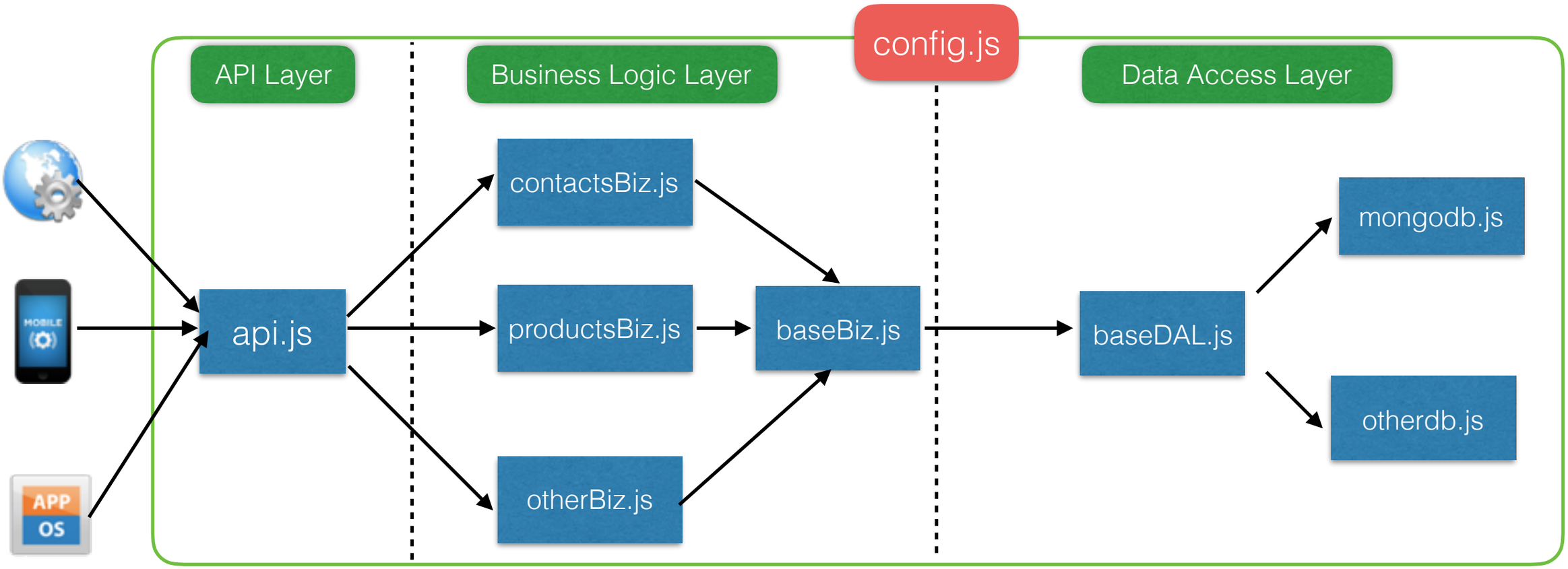
Refactor Code

Objective

- Reduce spaghetti code
- Standardise
- Reusable
- Good Communication
- Easy Readable code

Khadev
www.facebook.com/khadev

Refactor Code



Khadev
www.facebook.com/khadev

Refactor Code

Step by step

1. Create folder ***“dal”*** then create file name as ***“mongodb.js”*** and input code.

2. Create file name as ***“baseDAL.js”*** for use to be centralise of data access.

3. Create file name as ***“config.js”*** for setup database connection parameters.

Khadev
www.facebook.com/khadev

Routing Center

Objective

Manage api router only one route made it easy to control application flow and standardise.

Step by step

1. Create folder ***“api”*** then create file name as ***“centralApi.js”*** and add code.
2. On ***“app.js”*** add code to handle routing centre.
3. Create folder ***“api/crm”*** then create file name as ***“contacts.js”*** and add code.

Khadev
www.facebook.com/khadev

Routing Center

Key of Routing centre

- Use app.all of express framework.
- Separate url /module/object/function.
- Find object and execute by javascript eval.

Khadev
www.facebook.com/khadev

Log Center

Objective

- Keep log any functions.
- Write function only one time.
- Necessary log Information and write technique.

Log Information

#	Field	Description
1	guid	Log id for reference input and output
2	ip	IP address of client call.
3	userName	User name that call service
4	url	Url of resource
5	method	http request method such as get,post ,put ,delete
6	os	Client's Opening System
7	device	Client's Device
8	browser	Client's Browser
9	status	completed,error
10	timestamp	Request time
11	results	Result of function

Log Center

apiLogger Structure

- getGuid
- writeReqLog
  - writeLog
    - writeLogFile
    - writeLogDB
- writeResLog
  - writeLog
    - writeLogFile
    - writeLogDB

Log Center

Common Practice

- Short Term : Keep log in file and prompt to read .
- Long Term : Keep log in Database.
- Keep log both ways if possible

— Launch —