Easydian Common APIs

目录

[1. schema: 2](#_Toc353046120)

[1.1 Shops: 2](#_Toc353046121)

[1.2 Userinfo: 2](#_Toc353046122)

[1.3 Comment 3](#_Toc353046123)

[1.4 News 3](#_Toc353046124)

[2. Common APIs 3](#_Toc353046125)

[3. API Details 4](#_Toc353046126)

[3.1 shops API 4](#_Toc353046127)

[3.1.1 GET /shops 5](#_Toc353046128)

[3.1.2 GET /shop/:id 5](#_Toc353046129)

[3.1.3 PUT /visit/:id 6](#_Toc353046130)

[3.1.5 POST /shop 6](#_Toc353046131)

[3.1.6 DELETE /shop/:id 6](#_Toc353046132)

[3.1.7 GET /validate\_ field 7](#_Toc353046133)

[3.1.8 PUT /goodbad/:id 7](#_Toc353046134)

[3.2 User APIs 7](#_Toc353046135)

[3.2.1 /new\_user 7](#_Toc353046136)

[3.2.2 GET /user/:id 8](#_Toc353046137)

[3.2.3 PUT /login/:id 8](#_Toc353046138)

[3.2.4 PUT /logout/:id 8](#_Toc353046139)

[3.2.5 POST /register 8](#_Toc353046140)

[3.3 Comment APIs 9](#_Toc353046141)

[3.3.1 GET /comment/:id 9](#_Toc353046142)

[3.3.2 POST /comment/:id 9](#_Toc353046143)

[3.3.3 DELETE /comment/:id 9](#_Toc353046144)

[3.3.4 DELETE /comments/:id 10](#_Toc353046145)

[3.4 News APIs 10](#_Toc353046146)

[3.4.1 GET /news/:id 10](#_Toc353046147)

[3.4.2 DELETE /news/:id 10](#_Toc353046148)

[4. Design 12](#_Toc353046149)

# 1. schema:

## 1.1 Shops:

{

shopname : {type:String, default:'fullname'},

shopalias : {type:Array, default: ['short name']},

shoptype : {type:String, default:'Dining'},

shopvisit : {type:Number, default:1000},

shoppriority : {type: Number, default:1000},

shopwebsite: {type:String, default: 'fullurl'},

shopphone: {type:Array, default: ['010-22222222']},

shoponbusiness: {type:Boolean, default: true},

shoponadv: {type:Boolean, default: true},

shopweekstats:[],

weekday: {type:Array, default:[0,0,0,0,0,0,0]},#visit

weekdaygood: {type:Array, default:[0,0,0,0,0,0,0]},

weekdaybad: {type:Array, default:[0,0,0,0,0,0,0]},

shopgoodt: {type:Number, default:0},

shopbadt: {type:Number, default:0},

shopdaystats:{type:Array,default:[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]},

shopmonthstats: {type:Array,default:[0,0,0,0,0,0,0,0,0,0,0,0]}, #visit

shoplogo: {type:String, default: './images/default.jpg'},

shopcover:{type:Array, default: ['beijing','shanghai']},

shopaccount: {type:Number, default:0},

shopcreatetime: {type:Date, default: Date.now},

shopcommentsnum: {type:Number, default:0}

}

## 1.2 Userinfo:

**Use mongoose format directly.**

@addr\_info = new Schema({

address: {type:String,required:true,unique:true}, # add ",unique:true,trim:true" in

gps:{x:0.00,y:0.00},

usetime: {type:Date, default:Date.now}

})

@phone\_info = new Schema({

phone: {type:String,unique:true}

usetime:{type:Date, default: Date.now}

});

@user\_schema = new Schema({

username: {type:String, required: true}, # add ",unique:true,trim:true" in

useralias: {type:String, default: "Mr Lazy"},

useraddress: [@addr\_info],

userpassword: {type:String, default:"md5"}, #havn't decided how to auth

usertargetaddress: [@addr\_info],

userphone: [@phone\_info], #phone num to be unique

logintime: [{login:Date, logduration:Number}],

belogin: {type:Boolean, default: true}

})

## 1.3 Comment

{

shopid : {type:String, default:'shopid'},

comment: {type:String, default:'context'},

createtime: {type:Date, default: Date.now}

}

## 1.4 News

**Node.js will not take responsibility of news, crawler based on python will insert news to mongodb directly.**

{

shopid : {type:String, default:'shopid'},

news: {type:String, default:'abstract'},

newsurl: {type:String, default:'full\_url'},

createtime:{type:Date, default: Date.now}

}

# Common APIs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| For shops | | | 1 | |
| **URL** | **Method** | **parameters** | | **note** |
| /shops | get | Start,Limit, type | | Return the basic info of a group of shops |
| /shop/:id | get | Start,limit,type  News, comments  field | | Get the detail info  News = 1, return news  Comments=1, return comment  Filed =0, don’t return any fields of shops  Start/limit used to return a number of comments or news. |
| /visit/:id | PUT |  | | Update the visit num of a shop |
| / shop | post | Type, value of instance | | Insert a shop to the db |
| /shop/:id | delete |  | | Delete a shop by id |
| /validate\_ field | get | table, field, value | | Validate the given field |
| / goodbad/:id | put | type | | Update the good bad comment |

|  |  |  |  |
| --- | --- | --- | --- |
| For User | | 2 | |
| **URL** | **Method** | **parameters** | **note** |
| / register | post | userinfo | Insert a user info to database |
| /validate\_field | get | Fieldpath, value | The same in shops |
| /user/:id | get | fields | Return the user info |
| /login/:id | put |  |  |
| /logout/:id | put |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| For Comments | | 2 | |
| **URL** | **Method** | **parameters** | **note** |
| / comment/:id | post | comment | Insert a comment to db |
| /comments/:id | get | Start,limit,level | Get a number of comments |
| /comments/:id | delete | Start,limit,level | Delete a rank of comment |
| /comment/:id | delete |  | Delete a give comment |

|  |  |  |  |
| --- | --- | --- | --- |
| For News | | 2 | |
| **URL** | **Method** | **parameters** | **note** |
| /news/:id | get | Start,limit,level | Get a number of news |
| /news/:id | delete | Start,limit,level | a rank of news |

# 3. API Details

## 3.1 shops API

Response format:

{

info: “success”/ ”failed”,

body: {message body}

}

### 3.1.1 GET /shops

Get the shops registered in easydian

Priority: must

Request Parameters:

Category: specify the category of the shops

Start(optional): start index (sorted by priority), 0 base

Limit(optional): limit the number of items

Request body: N/A

Response Status: 200,400, 500

Response Body: An array of shops, JSON format

{

[{

\_id: xxxx,

shopname: xxxx,

shoplogo: xxxx,

shopwebsite: xxxx,

shopbadt: xxxx,

shopgoodt: xxxx

}],

[]

…

}

### 3.1.2 GET /shop/:id

GET the shop detail

Prority: must

Request Parameters:

Category: specify the category of the shops

Fields: the fields needs to be return

Comments: 1: return comments, 0:don’t return comment, sorted by create date

News: 1: return news, 0: don’t return news, sorted by create date

Start: start index of comments/news

Limit: the num of comments/news

Request body: N/A

Response Status: 200, 400, 500

Response Body:

{

Fields:{

Field1: xxxx,

Field2: xxxx,

Field3: xxxx

},

News: [

{News1},

{News2}

],

Comments: [

{Comment1},

{Comment1},

{Comment1}

]

}

### 3.1.3 PUT /visit/:id

Update the visit num of shop

Priority: should

Request Parameters:

Category: specify the category of the shops

Request body: N/A

Response Status: 200, 400, 500

Response body: {visit:value}

### 3.1.5 POST /shop

Insert a new shop to db

Priority: must

Request Parameters:

Category: specify the category of the shops

Request body:

Instance:A json configuration for a shop

Response Status: 200, 400, 500

Response body: only info failed

### 3.1.6 DELETE /shop/:id

Update value of a field

Priority: should

Request Parameters:

Request body:

Response Status: 200, 400, 500

Response body: only info field

### 3.1.7 GET /validate\_ field

Validates the given field’s value

Priority: should

Request Parameters:

table : shop / user

field: the name of the filed

value: the value of the field

Request body: N/A

Response Status: 200, 400, 500

Response body: only info field

### 3.1.8 PUT /goodbad/:id

Update the value of goodbad comment

Priority: must

Request Parameters:

Category: specify the category of the shops

Type: good/bad

Request body: N/A

Response Status: 200, 404, 500

Response body:

{bgt:num}

## 3.2 User APIs

### 3.2.1 /new\_user

Insert a new user to db

Priority: should

Request Parameters: N/A

Request body: A json file of all the user’s information(I’ll prepare it)

Response Status: 200, 400, 500

Response body:only info field

### 3.2.2 GET /user/:id

Get the user’s detail information

Priority: should

Request Parameters:

Fields: the fields that needs to be returned

Request body: N/A

Response Status: 200, 400, 500

Response body: A json file contains user’s information

### 3.2.3 PUT /login/:id

Send user login info to server

Priority: should

Request Parameters: N/A

Request body: a json file of user login info

Response Status: 200, 404, 500

Response body: A json file contains basic user info, / error info

### 3.2.4 PUT /logout/:id

Send user logout info to server

Priority: should

Request Parameters: N/A

Request body: N/A

Response Status: 200, 404, 500

Response body: A json file contains basic user info, / error info

### 3.2.5 POST /register

Insert a user info to db

Priority: should

Request Parameters: N/A

Request body: A json file of user’ info

Response Status: 200, 404, 500

Response body: A json file contains basic user info, / error info

### 3.3 Comment APIs

:id used here should be the ID of a shop

### 3.3.1 GET /comment/:id

Priority: should

Request Parameters:

Start: start index

Limit: num of comments

Level: 0-5, 5 is the best

Request body: N/A

Response Status: 200, 400, 500

Response body:

[

{

Comment: xxxxxx

},

{… },

…

]

### 3.3.2 POST /comment/:id

Insert a user info to db

Priority: should

Request Parameters: N/A

Request body: A json file of comment

Response Status: 200, 400, 500

Response body: {\_id:xxxx}

### 3.3.3 DELETE /comment/:id

This is id is the id of a comment

Priority: should

Request Parameters: N/A

Request body: N/A

Response Status: 200, 400, 500

Response body: only info field

### 3.3.4 DELETE /comments/:id

Delete a rank of comments

Priority: should

Request Parameters:

Start: start index

Limit: num of comments

Level: 0-5, 5 is the best, -1 means any Level

Request body: N/A

Response Status: 200, 400, 500

Response body: only info field

### 3.4 News APIs

### 3.4.1 GET /news/:id

Priority: should

Request Parameters:

Start: start index

Limit: num of comments

Request body: N/A

Response Status: 200, 400, 500

Response body:

[

{

news: xxxxxx

},

{… },

…

]

### 3.4.2 DELETE /news/:id

Priority: should

Request Parameters:

Start: start index

Limit: num of comments

Request body: N/A

Response Status: 200, 400, 500

Response body: only info field

# 4. Design

Node.js

mongodb

redis

View

ejs, jquery

controller

model

Crawler, java