Connoisseur Backend API Design

Objectives 3

Design Principal 4

Versioning & Compatibility 4

Name Space 4

Media Type 4

Character Set 4

HTTP verbs 5

Errors 5

Authentication 5

Pagination 6

Rate Limit 6

Request Debugging 6

Other 6

Account Service 7

API spec 7

Create new user 7

list users 7

Retrieve login session 7

Validate login session 8

Invalidate login session 9

Update user 9

Get User Detail 10

Get User Bookmarks 10

Get User Ratings 11

Get User’s Full Information 12

Add Bookmark 12

Delete Bookmark 12

Get rating information for user 13

Other 13

Restaurant Service 13

Rating Service 14

# Objectives

Connoisseur Backend API provides api calls to support all Connoisseur website, iOS or Android client features

# Design Principal

## Versioning & Compatibility

Api version should be part of the api endpoint url

Example: <http://api.exmaple.com/api/v3/users>

Pros and cons:

* + Ability to version specific resource branches.
  + Semantically dev friendly
  + Bookmarking is tightly coupled
  + Enables version navigation/discovery

Cons

* + New versions change resource name and location
  + Complex proliferation of URI aliases
  + Bookmarking is tightly coupled (see pros!)
  + Enables version navigation/discovery (see pros!)
  + Can't use URI easily to compare identity
  + New versions break existing hyperlinks

Always add method and parameters; never remove method or parameter in the same major version.

## Name Space

All URLs listed in the documentation should be preceded with our project’s subdomain  
 https://{yoursubdomain}.dinningconnoisseur.com /api/{apiversion} and API version.

The apiversion is currently v1.

## Media Type

The API currently only supports JSON as an exchange format. Be sure to set both the Content-Type and Accept headers for every request as application/json.

All Date objects are returned in ISO 8601 format:

YYYY-MM-DDTHH:mm:ss.SSSZ

## Character Set

Use UTF-8

## HTTP verbs

Where possible, this API strives to use appropriate HTTP verbs for each action.   
  
**GET**  
use for retrieve server resource(not changing server state)

**POST**

Used for creating resources, or performing custom actions (such as user lifecycle operations). **POST do not follow Idempotency**

**PUT**  
used for replace resource  
  
**DELETE**

Used for delete resource

## Errors

Use HTTP error code to represent the request/response status  
200 - All success response should have this http response code if something returned within the response

204 – success response with no content

404 – resource not found

500 – internal server error

Application Error Codes will used to indicate the exact error message  
all error response should return wth following fields:

errorCode: the internal error code

errorMessage: human readable error message

errorStack: the related error stack if in debug mode

sessionId: the identifier for the user session

## Authentication

All protected api endpoint calls should have an “Authorization:{token}” header or “Authorization:{token}” cookie.

Client need to retrieve a valid token thro

POST /api/v1/users/#id/loginsession call to get a valid token first to call all protected service

## Pagination

Requests that return a list of resources may support paging. Pagination is based on cursor and not on page number. The cursor is opaque to the client and specified in either the ?before or ?after query parameter. For some resources, you can also set a custom page size with the ?limit parameter

The actual page will specified by ?page parameter

|  |  |
| --- | --- |
| Link Relation Type | Description |
| self | Specifies the URL of the current page of results |
| next | Specifies the URL of the immediate next page of results. |
| prev | Specifies the URL of the immediate previous page of results. |

## Rate Limit

TBA

## Request Debugging

TBA

## Logging

TBA

## Other

* + Escape all input/ouputs for prevent cross site request forge

# Account Service

Account service deal all operation with users and authentications

Client need to get a valid token by authentication service before call any other services.

Email will be the unique identifier

## API endpoint alias

Always use /me as an alias to current session user

## User Status

P(Pending): After created, user will in P(Pending) status

A(Active): After email verification

D(Deleted): mark deleted

## Integration with other service

All other service will call “Retrieve login session” to validate the login session they got.

Once user created, queue an email activation message. email service will pickup the message and send the activation email.

## API spec

Full api spec for Account Service

### Create new user

POST /user/

Create a new user in Connoisseur

Request parameters:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameter | Description | Param Type | Data Type | Required | Default |
| email |  | Body | String | Y |  |
| Username |  | Body | String | Y |  |
| firstName |  | Body | String | Y |  |
| lastName |  | Body | String | Y |  |
| gender |  | Body | String |  |  |
| password |  | Body | String | Y |  |

Request Example

curl -v -X POST -H "Accept: application/json" -H "Content-Type: application/json" -H "Authorization: ${api\_token}" -d ‘{

"email": "rayxiaonet@gmail.com",

"firstName": "ray",

"gender": "m",

"lastName": "xiao",

"password": "samplepassword",

"userName": "RayXiao"

}’

response example:

### list users

GET /user/

List users in connoisseur. Default limit is 50

### Retrieve login session

POST /user/loginsession

Retrieve the login token for a user

Request parameters:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameter | Description | Param Type | Data Type | Required | Default |
| email |  | Body | String | Y |  |
| password |  | Body | String | Y |  |

Request example

{

"password": "123",

"email": "rayxiaonet@gmail.com"

}

Response example

{

“token”:”AAAAAAAAAAAAAAA”,

“expire”:”2017-04-50T12:34:55.678Z”

}

### Validate login session

GET /user/loginsession/:sessionToken

Check if the session token is valid

This need to be called every single time other service try to setup connection with a new client session.

Request parameters:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameter | Description | Param Type | Data Type | Required | Default |
| sessionToken |  | Path | String | Y |  |

Request example

Response example

{

“token”:”AAAAAAAAAAAAAAA”,

“expire”:”2017-04-50T12:34:55.678Z”,

“userId”:”1”

}

### Invalidate login session

DELETE /user/:id/loginsession/:token

Invalidate the login token for a user. If no token specified, invalidate all existing tokens.

Request parameters:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameter | Description | Param Type | Data Type | Required | Default |
| id |  | Path | String | Y |  |
| Token |  | Body | String |  |  |

Request example

Response example

### Update user

POST /user/:id/

POST /me

Update user information. This call do not expect new value of all property provided, it will only updates the value for provided properties.

Request parameters:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameter | Description | Param Type | Data Type | Required | Default |
| Id | User id | Path | number | Y |  |
| email |  | Body | String |  |  |
| username |  | Body | String |  |  |
| firstName |  | Body | String |  |  |
| lastName |  | Body | String |  |  |
| gender |  | Body | String |  |  |
| password |  | Body | String |  |  |

Request example

curl -v -X POST -H "Accept: application/json" -H "Content-Type: application/json" -H "Authorization: ${api\_token}" -d ‘{

"email": "rayxiaonet@gmail.com",

"firstName": "ray",

"gender": "m",

"lastName": "xiao",

"password": "samplepassword",

"userName": "RayXiao"

}’

Response example

### Get User Detail

GET /user/:id/

GET /me/

Get user basic information

Request parameters:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameter | Description | Param Type | Data Type | Required | Default |
| Id | User id | Path | number | Y |  |

Request example

curl -v -X GET -H "Accept: application/json" -H "Content-Type: application/json" -H "Authorization: ${api\_token}" http://account.connoisseur.com/api/v1/user/1

Response example

{

"email": "rayxiaonet@gmail.com",

"firstName": "ray",

"gender": "m",

"lastName": "xiao",

"password": "samplepassword",

"userName": "RayXiao"

}

### Get User Bookmarks

GET /user/:id/bookmarks/

GET /me/bookmarks/

Get user’s bookmark information. Limit results to 50 by default

Only Admin user can access other peopl’s bookmarks

Request parameters:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameter | Description | Param Type | Data Type | Required | Default |
| Id | User id | Path | Number | Y |  |

Request example

curl -v -X GET -H "Accept: application/json" -H "Content-Type: application/json" -H "Authorization: ${api\_token}" http://account.connoisseur.com/api/v1/user/1/bookmarks/?offset=50

Response example

{

"email": "rayxiaonet@gmail.com",

"userName": "RayXiao"

“bookmarks”:[

{“restaurantId”:”Nana’s bakery”,”createDate”: ”2017-04-50T12:34:55.678Z”},

{“restaurantId”:”Jerry’s awesome steak”,”createDate”: ”2017-04-50T12:34:55.678Z”},

…

],

count:172,

offset:50,

pageSize:50,

next:”http://account.connoisseur.com/api/v1/user/1/bookmarks/?offset=100”

prev: ”http://account.connoisseur.com/api/v1/user/1/bookmarks/?offset=0”

self: ”http://account.connoisseur.com/api/v1/user/1/bookmarks/?offset=50”

}

### Get User Ratings

GET /user/:id/ratings/

GET /me/ratings/

Get user’s ratings information. Limit results to 50 by default, sort by date desc default

Only Admin user can access other peopl’s ratings

Request parameters:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameter | Description | Param Type | Data Type | Required | Default |
| Id | User id | Path | Number | Y |  |

Request example

curl -v -X GET -H "Accept: application/json" -H "Content-Type: application/json" -H "Authorization: ${api\_token}" http://account.connoisseur.com/api/v1/user/1/ratings/?offset=50

Response example

{

"email": "rayxiaonet@gmail.com",

"userName": "RayXiao"

“bookmarks”:[

{“restaurantId”:”Nana’s bakery”,”createDate”: ”2017-04-50T12:34:55.678Z”,”rating”:5, “comment”:”this is really good”},

{“restaurantId”:”Jerry’s awesome steak”,”createDate”: ”2017-04-50T12:34:55.678Z”,”rating”:3},

…

],

count:172,

offset:50,

pageSize:50,

next:”http://account.connoisseur.com/api/v1/user/1/ratings/?offset=100”

prev: ”http://account.connoisseur.com/api/v1/user/1/ratings/?offset=0”

self: ”http://account.connoisseur.com/api/v1/user/1/ratings/?offset=50”

}

### Get User’s Full Information

TBA

### Add Bookmark

POST /user/:id/bookmarks/

POST /me/bookmarks/

Add a restaurant id to the specified user’s bookmark

Only Admin user can access other people’s bookmarks

Request parameters:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameter | Description | Param Type | Data Type | Required | Default |
| Id | User id | Path | Number | Y |  |
| restaurantId |  | body | String | Y |  |

Request example

### Delete Bookmark

DELETE /user/:id/bookmarks/:restaurantId

DELETE /me/bookmarks/:restaurantId

Remove a restaurant id from user’s bookmark list

Only Admin user can access other people’s bookmarks

Request parameters:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameter | Description | Param Type | Data Type | Required | Default |
| Id | User id | Path | Number | Y |  |
| restaurantId |  | Path | String | Y |  |

Request example

### Get rating information for user

GET /user/:id/ratings/

GET /me/ratings/

### Recommend Restaurants for user

GET /user/:id/recommends/?location=xx&keyword=xx

TBD, may need to implement later. checkout the restaurant service first

## Other Thoughts

Add ratings/remove rating should be part of the rating service

Design OAuth(Facebook) integration

# Restaurant Service

Manage all restaurant information

load all restaurant from dynamodb after startup and cache the results

implement retry logic when accessing dynamodb

Search by geo location/neighborhood will be complex and challenging

Add restaurant

POST /restaurant/

Get restaurant information

GET /restaurant/:id

Get restaurant recommendations

GET /restaurant/?filter=

Limit result by 5

Return result randomly

Build the filter by city/neighborhood/zip, name, etc

Modify restaurant information

POST /restaurant/:id

# Rating Service

Add new ratings (with comment)

POST /rating/

Get rating information for restaurant

GET /rating/:restaurantid

Delete an existing rating/comment

# Email Service

Read from email queue and sent out emails.