RESTful API Design

- Required parameters are usually named and passed in the path
- Optional filters are usually passed through query parameters
- Optional fields such as description are passed through request body
- RESTful endpoints are qualified with prefix `/api` to distinguish from those that respond with page to render
- All the /api path will need authentication. A JWT token will need to be put in the Authorization header, with the value 'JWT {token}', indicating a logged-in user each request.
- We decide to use the name *bundle* for the high-level request, and the name *order* for each sub-request contained in a high-level request.

Login

`POST /login`
Login with `{username}` and `{password}`, or with `{access_token}` retrieved from colab's API.

```
//body
{
    "username": "",
    "password": "",
    //or
    "accessToken": ""
}
```

```
//Response
{
  "status": "success", //either success or error.
  "error": "", //if error, will be present. Same apply to all responses.
  "data": { //if success, will be present. Same apply to all responses.
    "id": 36,
    "username": "jayay",
    "displayName": "Jay Wang",
```

```
    "email": "zhewang828@gmail.com",
    "netId": null,
    "isNetIdUser": 0,
    "accountStatus": "ACTIVE",
    "permission": "ADMIN",
    "createdAt": "2017-02-19T07:56:15.000Z",
    "updatedAt": "2017-02-19T07:56:15.000Z",
    "token": "", //JWT token
  }
}
```

Users

GET /api/users/apikey/
When logged in (valid Authorization header configured), get an unexpired API key (JWT token) that can be used for API calls.

```
//Response
{
  "status": "success",
  "data": "xxxx", //JWT token
}
```

GET /api/users/?userId=123
Get all the users' information.

```
//Response
{
  "status": "success",
  "data": [
    {
      "id": 6,
      "username": "buie",
      "displayName": "buie",
      "email": "andrew.buie@duke.edu",
```

```
      "netId": "ajb93",
      "isNetIdUser": 0,
      "accountStatus": "ACTIVE",
      "permission": "ADMIN",
      "createdAt": "2017-02-18T18:42:06.000Z",
      "updatedAt": "2017-02-18T18:42:06.000Z"
    },
    {
      "id": 7,
      "username": "emre",
      "displayName": "emre",
      "email": "emre.sonmez@duke.edu",
      "netId": "ebs32",
      "isNetIdUser": 0,
      "accountStatus": "ACTIVE",
      "permission": "USER",
      "createdAt": "2017-02-18T18:42:06.000Z",
      "updatedAt": "2017-02-18T18:42:06.000Z"
    }
  ]
}
```

`POST /api/users/:username/:password`
Admin only. Create a user with `:name` and `:password` (password is in plain text, will be hased in server side. Relying on https for safety).

```
//Optional body
{
    "displayName": "", //if not provided, default to be the same as username
    "email": "", //required
    "permission": "" //ADMIN, MANAGER, USER
}
```

```
//Response
{
  "status": "success",
  "data": {
    "accountStatus": "ACTIVE",
    "id": 47,
    "username": "jayman",
    "displayName": "jayman",
    "email": "zw48@duke.edu",
    "netId": null,
    "isNetIdUser": 0,
    "permission": "MANAGER",
    "updatedAt": "2017-02-22T06:38:32.000Z",
    "createdAt": "2017-02-22T06:38:32.000Z"
  }
}
```

`DELETE /api/users/:userId` (NOTE: NOT SUPPORTED YET.)
Admin only. Delete the user with `:userId`

```
//Response
{
    "status": "",
    "error": ""
}
```

`PUT /api/users/:userId`
Update an existing user's information.

```
//Optional body
{
    "displayName": "",
    "email": "",
```

```
    "permission": ""
}
```

```
//Response
{
  "status": "success",
  "data": {
    "id": 47,
    "username": "jayman",
    "displayName": "jayman",
    "email": "zw48@duke.edu",
    "netId": null,
    "isNetIdUser": 0,
    "accountStatus": "ACTIVE",
    "permission": "MANAGER",
    "createdAt": "2017-02-22T06:38:32.000Z",
    "updatedAt": "2017-02-22T06:57:23.000Z"
  }
}
```

Logs

`GET /api/logs/?userId=&itemId=&fromTime=&toTime=&rowPerPage=&pageNumber=`
Get all the logs with specified filters.

```
//Note fromTime/toTime needs to be a URLencoded-JSONstringified js Date object.
//For example, it needs to be something like:
const toTimeParam = encodeURIComponent(JSON.stringify(new Date()));
```

```
//response
{
  "status": "success",
```

```
  "data": [
    //srcUser, destUser, item, bundle, if not undefined, means
    // is a valid entry of this log and can be used to display and redirect.
    {
      "id": 67,
      "content": "item FPGA's quantity is deducted 8 on user Jay Wang's approval
of request #23",
      "createdAt": "2017-02-26T07:54:37.000Z",
      "updatedAt": "2017-02-26T07:54:37.000Z",
      "isValid": true,
      "srcUser": {
        "id": 36,
        "name": "Jay Wang"
      },
      "item": {
        "id": 10091,
        "name": "FPGA"
      },
      "bundle": {
        "id": 23,
        "name": "#23"
      }
    },
    ...
  ]
}
```

Items

An *Item* is assumed to be an instantiated, concrete object with a quantity. There is not distinguishing among the instances. Thus there is no separate *instance* table for now.

`GET /api/items/?id=&rowPerPage=&etc`

Get all the items with proper filtering

```
// Query param
{
    includeTags: ["resistor"],
    excludeTags: ["oscilloscope"],
    id: 1234,
    rowPerPage: 10,
    pageNumber: 1, // assume counting from 1
    name: "50 ohms",
    model: "model text"
}
```

```
//response
{
    "status": "",
    "error": "",
    "data": [
        {
            "id": "",
            "name": "",
            "quantity": 1234,
            "model": "",
            "description": "",
            "itemStatus": "",
            "tags":["resistor", "wand", ..], //array of tag names
            /* vvvvvvvvvvv custom fields come in handy vvvvvvvvvvvvv */
            "location": "husdon",
            "restock_info": null, //if no specific value specified for this
item
        },
        {
            ...
```

```
        }
    ] //array of item objects
}
```

GET /api/items/customFields/

```
//Response
{
  "status": "success",
  "data": [
    {
      "id": 3,
      "name": "location",
      "type": "SHORT",
      "visibility": "PUBLIC",
      "createdAt": "2017-02-21T19:28:36.000Z",
      "updatedAt": "2017-02-21T19:28:36.000Z"
    },
    {
      "id": 4,
      "name": "restock_info",
      "type": "LONG",
      "visibility": "PUBLIC",
      "createdAt": "2017-02-21T19:28:36.000Z",
      "updatedAt": "2017-02-21T19:28:36.000Z"
    },
    ...
  ]
}
```

POST /api/items/customFields/:name/:type/:visibility
Create a new custom field on all items with `:name`, `:type`, and `:visibility`
If field already exist, update it with the `:visibility` given, but the `:type` of the field cannot be

changed after field is created. Doing so to protect legacy data. If such need persists, one shall explicit delete the field, and then create it with a different type.

`:type` is an enum with value set `{'SHORT','LONG','INT','FLOAT'}`
`:visibility` is an enum with value set `{'PRIVATE','PUBLIC'}`
A value of any custom field must **not** exceed 8192 bytes

```
//Response
{
    "status": "",
    "error": "",
    "data": { // the updated or created customfield info
        "id": 8,
        "type": "LONG",
        "visibility": "PUBLIC",
        "name": "name given in request",
        "updatedAt": "2017-02-17T23:21:45.000Z",
        "createdAt": "2017-02-17T23:21:45.000Z"
    }
}
```

`DELETE /api/items/customFields/:name/`
Remove a custom field on items. All values associated with this field will also be deleted.

```
//Response
{
    "status": "",
    "error": "",
}
```

`POST /api/items/:itemName/:quantity/`
Create a new item with `:itemName` and `:quantity`

```
// optional body
{
    "model": "",
    "description": "",
```

```
    "location": "",
    "tags": ["tagId1", "tagId2"], //array of tagIds
    "itemStatus" : "ACTIVE"
}
```

```
//Response
{
    "status": "",
    "error": "",
    "data": {} // return the item just created, with all fields, NO TAGS
}
```

`POST /api/items/import`
Bulk import a list of new items to the system by uploading a **json** file.

```
// request body
{
    "fileHandle": "fjdlksafjidsabklfdsa" // some file handle returned by
filestack
}
```

The entire import is all or nothing. Any syntactic/semantic error that aborts the transaction will prevent any item in file to be imported.
The error message returned by server is extremely helpful. Consider the following example.

```
// file uploaded
[
    {
        "name": "",
        "quantity": 1234,
        "model": "",
        "description": "",
        "itemStatus": "",
        "tags": [ "resistor", "wand" ], //array of tag names
        "location": "husdon", // custom fields
        "restock_info": " ", //custom fields
```

```
    },
    {
        "name" : "test_item2",
        "quantity" : 2,
        "model" : "model 2",
        "nosuchfieldreally" : "nuh"
    }
]
```

```
// Server response
{
  "status": "error",
  "error": {
    "type": "ERROR",
    "message": "Cannot create test_item2 with ill-defined custom fields:
nosuchfieldreally"
  }
}
```

Other errors could be

1. import an item whose name already existed
2. an item referencing a tag that is yet to be defined
3. quantity is negative
4. etc.

`PUT /api/items/:itemId/`
Update by *overriding* an existing item with `:itemId`

```
// optional body. All provided fields will be updated as is
{
     "name": "",
     "quantity": 6666,
     "location": "",
     "model": "",
```

```
     "description": "",
     "tags": ["", "", ""],
     "comment": "I am kip and I override the quantity", // used for quantity log
}
```

```
//Response
{
    "status": "",
    "error": "",
    "data": {} // return the item updated, with all fields, NO TAGS
}
```

PUT /api/items/:itemId/:quantityDelta

```
//Optional body.
{
    "comment": "" //to log acquisition/loss
}
```

Update by *applying the arguments as delta onto existing record* of item with `:itemId`
Other fields cannot be updated using delta and have to use the previous api
quantityDelta could be positive or negative

```
//Response
{
    "status": "",
    "error": "",
    "data": {} // return the item updated, with all fields
}
```

DELETE /api/items/:itemId/
delete an existing item with `:itemId`

```
//Response
{
    "status": "",
```

```
    "error": ""
}
```

Tags

`GET /api/tags/`
get all the tags

```
{//Response
  "status": "success",
  "data": [//array of tag objects
    {
      "id": 1,
      "name": "resistor",
      "createdAt": "2017-02-22T07:49:26.000Z",
      "updatedAt": "2017-02-22T07:49:26.000Z"
    },
    ...
  ]
}
```

`POST /api/tags/`
create new tags from the array provided (will ignore those already existing).

```
//body
{
    "tags": ["tagName1", "tagName2", ...]
}
```

```
//Response
{
  "status": "success"
}
//or
{
```

```
  "status": "error",
  "error": {
    "type": "ERROR",
    "message": "syntax error. body.tags need to be a non-empty array"
  }
}
```

`DELETE /api/tags/:tagId/`
delete an existing tag with `:tagId`. Also removes

```
//Response
{
  "status": "success"
}
```

OrdersV2

`POST /api/ordersv2/:itemId/:quantity/:userId/`
create a new order instance. For users, it is essentially 'add to cart'. For admin and manager, it could be used as a direct dispatch (can either dispatch a loan or dispatch a disburse).

```
//body
{
    "adminComment": "",// optional.
    // When orderStatus is DISPATCHED, this will indicate the optional reason.
    // Will be ignored for CARTED.
    "orderStatus": "",// CARTED or DISPATCHED
    // normal user can only use 'CARTED', performing add-to-cart.
    // admin/manager can use 'DISPATCHED', which will lead to the creation of
    // a new DISPATCHED bundle containing only this order.
    // Item quantity will be deducted accordingly as well.
    "type": "",// LOAN or DISBURSE
    // This field will be ignored if orderStatus is CARTED.
    // But if orderStatus is DISPATCHED, this field will indicate
```

```
    // whether this dispatch is a loan or a disbursement.
}
```

```
//add to cart response
{
  "status": "success",
  "data": {
    "orderType": "DISBURSE",// this field doesn't have meanings when the order
is carted. It's only meaningful when associated with a bundle.
    "id": 399,
    "userId": "36",
    "itemId": "10195",
    "quantity": "8",
    "orderStatus": "CARTED",
    "userComment": null,
    "adminComment": null,
    "updatedAt": "2017-03-19T22:14:53.000Z",
    "createdAt": "2017-03-19T22:14:53.000Z",
    "userName": "Jay Wang",
    "displayName": "Jay Wang",
    "itemName": "Inductor"
  }
}
```

```
//dispatch response
{
  "status": "success",
  "data": {// new bundle instance
    "id": 96,
    "userId": "65",
    "bundleType": "LOAN",
    "bundleStatus": "DISPATCHED",
    "adminComment": "Hi",
```

```
    "updatedAt": "2017-03-19T22:13:55.000Z",
    "createdAt": "2017-03-19T22:13:55.000Z"
  }
}
```

`POST /api/ordersv2/:userId`
Submit `:userId`'s cart. A new PENDING bundle will be created containing all of this user's carted orders. The orderType of each order row in the database will be updated to be the same as the bundleType. For example, if you submit the cart with the type LOAN, the newly created bundle will have LOAN as its bundleType, and all the orders associated will have LOAN as their orderType.

```
//body
{
    "userComment": "", //optional request reason.
    "type": "", //LOAN or DISBURSE indicating this request's type.
}
```

```
//response
{
  "status": "success",
  "data": {// new bundle instance
    "id": 97,
    "userId": "36",
    "bundleType": "LOAN",
    "bundleStatus": "PENDING",
    "userComment": "aaa",
    "adminComment": null,
    "updatedAt": "2017-03-19T22:17:51.000Z",
    "createdAt": "2017-03-19T22:17:51.000Z"
  }
}
```

`GET /api/ordersv2/?bundleIds=&itemIds=&userIds=&statuses=&types=&rowPerPage=&pageNumber=`
Get all the bundles with specified filters. bundleIds, itemIds, userIds, statuses and types, as implied by their plural form, can be used to pass in an array of ids, e.g. `/?itemIds=1&itemIds=2&itemIds=3` represents `itemIds=[1, 2, 3]` and, for example, `/?types=LOAN` can also be used to query a single type.

```
//response
{
  "status": "success",
  "data": [
    {
      "id": 97,
      "userId": 36,
      "bundleType": "DISBURSE",
      "bundleStatus": "PENDING", // a pending disburse request
      "userComment": "i need it",
      "adminComment": null,
      "createdAt": "2017-03-19T22:17:51.000Z",
      "updatedAt": "2017-03-19T22:17:51.000Z",
      "displayName": "Jay Wang"
    },
    {
      "id": 96,
      "userId": 65,
      "bundleType": "LOAN",
      "bundleStatus": "DISPATCHED", // a dispatched loan request
      "userComment": null,
      "adminComment": "dispatch it to you!",
      "createdAt": "2017-03-19T22:13:55.000Z",
      "updatedAt": "2017-03-19T22:13:55.000Z",
      "displayName": "tracer"
    },
    ...
  ]
}
```

`GET /api/ordersv2/cart/:userId`
get all the carted orders of a user

```
//response
{
  "status": "success",
  "data": [
    {
      "id": 400,
      "userId": 36,
      "itemId": 10035,
      "bundleId": null,
      "quantity": 3,
      "orderType": "DISBURSE",// this field doesn't have meanings when the order
is carted. It's only meaningful when associated with a bundle.
      "orderStatus": "CARTED",
      "userComment": null,
      "adminComment": null,
      "createdAt": "2017-03-19T22:23:50.000Z",
      "updatedAt": "2017-03-19T22:23:50.000Z",
      "userName": "Jay Wang",
      "displayName": "Jay Wang",
      "itemName": "1K resistance"
    },
    ...
  ]
}
```

`GET /api/ordersv2/:bundleId`
get all the orders belonging to **`:bundleId`**

```
//reponse
{
  "status": "success",
  "data": [
    {
      "id": 236,
```

```
      "userId": 36,
      "itemId": 10035,
      "bundleId": 17,
      "quantity": 3,
      "orderType": "LOAN",
      "orderStatus": "PENDING",
      "userComment": null,
      "adminComment": null,
      "createdAt": "2017-02-23T01:29:10.000Z",
      "updatedAt": "2017-02-23T01:41:06.000Z",
      "userName": "jayay",
      "itemName": "1K resistor"
    },
    ...
  ]
}
```

PUT /api/ordersv2/:bundleId
Approve or deny a bundle with the type (loan/disburse) indicated.The `{orderType}` and `{orderStatus}` of each order belonging to this bundle will be updated to be the same as the `{bundleType}` and `{bundleStatus}`.

```
//body
{
    "adminComment": ""
    "bundleStatus": ""// APPROVED/DENIED if status was PENDING
    "type": ""// LOAN or DISBURSE indicating the type of the approval
    // Will be ignored for other statuses.
}
```

```
//response
{
  "status": "success",
  "data": {
```

```
    "id": 97,
    "userId": 36,
    "bundleType": "DISBURSE",
    "bundleStatus": "APPROVED",
    "userComment": "aaa",
    "adminComment": "apppprove",
    "createdAt": "2017-03-19T22:17:51.000Z",
    "updatedAt": "2017-03-19T22:27:33.000Z"
  }
}
```

`PUT /api/ordersv2/:id/:type`
`:type` must be either `ORDER` or `BUNDLE`.
1) Convert a LOAN order or bundle with `:id` (orderId or bundleId) to DISBURSE. If it's a conversion of a bundle, all the LOAN orders belonging to that bundle will be converted to DISBURSE, and those orders of other types will be ignored.
2) RETURN a LOAN order or bundle with `:id` (orderId or bundleId). The bundleStatus or orderStatus must be APPROVED or DISPATCHED. If it's a RETURN of a bundle, all the LOAN orders belonging to that bundle will be marked as returned and corresponding quantity will be added back to the items.

```
// body
{
    "isConvert": "true",// to convert a LOAN order/bundle to DISBURSE.
    "isReturn": "true",// to RETURN a LOAN order/bundle. If present, will ignore isConvert field.
}
```

```
// return individual loan order
// response of /api/ordersv2/402/ORDER
{
  "status": "success",
  "data": {// updated order instance
    "id": 402,
    "userId": 36,
    "itemId": 10035,
    "bundleId": 98,
```

```
    "quantity": 10,
    "orderType": "LOAN",
    "orderStatus": "RETURNED",
    "userComment": null,
    "adminComment": null,
    "createdAt": "2017-03-20T18:54:56.000Z",
    "updatedAt": "2017-03-20T20:13:17.000Z"
  }
}
```

`DELETE /api/ordersv2/:id/:type/`
delete an order or a bundle with `:id` (orderId or bundleId). `:type` must be either `ORDER` or `BUNDLE`. Only CARTED/PENDING order and PENDING bundle can be deleted.

Loans

`GET /api/loans/?itemIds=&userIds=&orderStatuses=&rowPerPage=&pageNumber=&`
Get all the loans with filter. itemIds, userIds and orderStatuses, as implied by their plural form, can be used to pass in an array of ids, e.g. `/?itemIds=1&itemIds=2&itemIds=3` represents `itemIds = [1, 2, 3]`

```
//response
{
  "status": "success",
  "data": [
    {
      "id": 400,
      "userId": 36,
      "itemId": 10035,
      "bundleId": 98,
      "quantity": 10,
      "orderType": "LOAN",
      "orderStatus": "RETURNED",
      "userComment": null,
      "adminComment": null,
```

```
      "createdAt": "2017-03-19T22:23:50.000Z",
      "updatedAt": "2017-03-20T19:51:54.000Z",
      "userName": "Jay Wang",
      "displayName": "Jay Wang",
      "itemName": "1K resistance"
    },
    ...
  ]
}
```

Emails

`GET /api/emails/subscribe`
Return true if the requesting user is a manager/admin and is subscribed to the system notification

```
//response
{
  "status": "success",
  "data": true
}
```

`PUT /api/emails/subscribe`
Subscribe to the system notification. No op if the requesting user is already subscribed.

```
//response
{
  "status": "success",
  "data": "You are now subscribed to system notifications."
}
```

`DELETE /api/emails/subscribe`
Unsubscribe from the system notification. No op if the requesting user is already unsubscribed.

```
//response
{
  "status": "success",
```

```
  "data": "Successfully unsubscribed"
}
```

`GET /api/emails/template/:templateName`
Retrieve the preamble for the email template

```
//response
{
  "status": "success",
  "data": "Some preample defined by last write"
}
```

`POST /api/emails/template/:templateName`
Update the preamble for the email template

```
//response
{
  "status": "success",
  "data": {
    "id": 1,
    "createdAt": "2017-03-22T15:53:26.000Z",
    "updatedAt": "2017-03-22T15:56:26.000Z",
    "preamble": "some prefix",
    "templateName": "loanReminder"
  }
}
```

`GET /api/emails/reminderDate`
Get all dates to send out reminders on all outstanding loans.

```
//response
{
  "status": "success",
  "data":
  [
      {
```

```
        "date": "cron date string"
      },
      {
        "date": "cron date string 2 "
      },
      ...
  ]
}
```

`POST /api/emails/reminderDate`
Set a date to send out reminders on all outstanding loans. No op if date already exists

```
// request body
{
  "date": "cron date string"
}
```

```
//response
{
  "status": "success",
}
```

`DELETE /api/emails/reminderDate`
Remove a date to send out reminders on all outstanding loans. No op if date does not exists.

```
// request body
{
  "date": "cron date string"
}
```

```
//response
{
  "status": "success",
}
```

---

document