UTDesign GettIt

Release 0.1

Marcus Deng, Monica Neivandt, Jack Tackett, Xander Wong

May 06, 2019

CONTENTS

CHAPTER

ONE

UTDESIGN_PROCUREMENT

1.1 utdesign_procurement package

1.1.1 Submodules

1.1.2 utdesign_procurement.apigateway module

class utdesign_procurement.apigateway.**ApiGateway**(*email_handler*)

Bases: object

addCost()

Adds a cost (refund, reimbursement, or new project) to a project. Can only be done by the admin.

Expected Input:

```
{
    projectNumber: (int),
    type: (string: refund, reimbursement, new budget),
    amount: (string, dollar amount),
    comment: (string),
    actor: (string, email of admin)
}
```

calculateBudget(*projectNumber*)

Return the available and pending budget for the given project number.

Parameters **projectNumber** – int.

Returns list containing available and pending budget

findProject()

Find all projects with the given project numbers and recalculate their budget. If none given, then return all authorized projects.

Expected Input:

```
{
    projectNumbers: (list of ints, optional)
}
```

Returns:

```
[
    {
```

(continues on next page)

(continued from previous page)

```
        "_id": ObjectId,
        "projectNumber": (int),
        "sponsorName": (string),
        "projectName": (string),
        "membersEmails": (list of strings),
        "defaultBudget": (int),
        "availableBudget": (int),
        "pendingBudget": (int),
        "status": (string)
    }
]
```

Returns list of projects

getAdminEmails()
Return a list of emails of all admins.

Returns a list of emails of all admins

getAdminList()
Return the emails of all admins in the system.

Returns:

```
[
    (string)
]
```

Returns list of emails for all admins

getCosts()
Return all the costs associated with a list of project numbers.

Expected input:

```
{
    projectNumbers: (list of ints, optional)
}
```

Returns:

```
[
    {
        "_id": ObjectId,
        "type": (string),
        "amount": (int),
        "comment": (string),
        "actor": (string) //email,
        "projectNumber": (int),
        "timestamp": (string)
    }
]
```

Returns list of costs (refund, reimbursement, etc.)

getTeamEmails(*projectNumber*)
Take the projectNumber of a project and return a list of emails of users associated with that project.

Parameters **projectNumber** – the projectNumber of a project

Returns list of emails of members of the specified project

managerList()
Return a list of technical managers who are associated to the given project number

Expected Input:

```
{
    "projectNumber": (int)
}
```

Returns:

```
[
    (string)
]
```

Returns list of technical manager's emails

procurementApproveManager()
Change the status of a procurement request to manager approved. This in effect sends the request to the admin for review.

Expected Input:

```
{
    "_id": (string)
}
```

procurementCancel()
Change the status of a procurement request to cancelled. The request is then unable to be considered by any user.

Expected Input:

```
{
    "_id": (string)
}
```

procurementComplete()
Change the status of a procurement request to complete. This indicates that items have been picked up by the student and no further actions need to be taken.

Expected Input:

```
{
    "_id": (string)
}
```

procurementOrder()
Change the status of a procurement request to ordered. This indicates that items have been purchased by the admin.

The shipping cost is set during this step.

Expected Input:

```
{
    "_id": (string),
    "amount": (string)
}
```

procurementReady()
Change the status of a procurement request to ready for pickup. This indicates that the purchased items are in the office and can be picked up by the student.

Expected Input:

```
{
    "_id": (string)
}
```

procurementRejectAdmin()
Change the status of a procurement request to rejected, initiated by the admin. This permanently rejects a request such that a student can not edit for resubmission. It can no longer be considered by any user.

Expected Input:

```
{
    "_id": (string),
    "comment": (string)
}
```

procurementRejectManager()
Change the status of a procurement request to rejected, initiated by the technical manager. This permanently rejects a request such that a student can not edit for resubmission. It can no longer be considered by any user.

Expected Input:

```
{
    "_id": (string),
    "comment": (string)
}
```

procurementResubmitToAdmin()
Change the status of a procurement request to manager approved. This happens after a request has been sent back to the students for updates and then submitted directly back to the admin for reconsideration.

Expected Input:

```
{
    "_id": (string),
    "requestNumber": (int) optional,
    "manager": (string), //email of manager who can approve this
    "vendor": (string),
    "projectNumber": (int),
    "URL": (string),
    "justification": (string) optional,
    "additionalInfo": (string) optional,
    "items": [
        {
        "description": (string),
```

(continues on next page)

(continued from previous page)

```
        "partNo": (string),
        "itemURL": (string),
        "quantity": (integer),
        "unitCost": (string),
        "totalCost": (string)
        }
    ]
}
```

procurementResubmitToManager()

Change the status of a procurement request to pending. This happens after a request was sent back to the student and has been submitted to the manager for reconsideration.

Expected Input:

```
{
    "_id": (string),

    "requestNumber": (int) optional,
    "manager": (string), //email of manager who can approve this
    "vendor": (string),
    "projectNumber": (int),
    "URL": (string),
    "justification": (string) optional,
    "additionalInfo": (string) optional,
    "items": [
        {
        "description": (string),
        "partNo": (string),
        "itemURL": (string),
        "quantity": (integer),
        "unitCost": (string),
        "totalCost": (string)
        }
    ]
}
```

procurementSave()

Save a procurement request. Take data as input and use it to create the request and save it to the database. If the submit flag is true, submit the request to the TM.

Expected Input:

```
{
    "submit": (Boolean) optional (not optional if submitted by student),
    "adminEdit": (Boolean) optional (not optional if admin performs an edit),
    "status": (string) optional (not optional if admin performs an edit),
    "requestNumber": (int) optional,
    "manager": (string), //email of manager who can approve this
    "vendor": (string),
    "projectNumber": (int),
    "URL": (string),
    "justification": (string) optional,
    "additionalInfo": (string) optional,
    "items": [
        {
        "description": (string),
```

(continues on next page)

(continued from previous page)

```
        "partNo": (string),
        "itemURL": (string),
        "quantity": (integer),
        "unitCost": (string),
        "totalCost": (string)
        }
    ]
}
```

Returns:

```
{
    "_id": (string),
    "requestNumber" (int):
}
```

Returns a dict containing the MongoDB objectId of the request and the request number

procurementStatuses()

Return a list of procurement requests matching all provided filters. Match with any combination of vendor, projectNumbers, and URL. For non-admin users, restrict projectNumbers to only those projectNumbers which the user is authorized to view. Ignore projectNumbers that the user is not authorized to view.

If the user is a manager, they will not be able to see saved requests

Expected Input:

```
{
    "vendor": (string, optional),
    "projectNumbers": (int or list of ints, optional),
    "URL": (string, optional),
    "statuses": (string or list of strings, optional)
}
```

Returns:

```
[
    {
        "_id": ObjectId
        "status": (string),
        "requestNumber": (int) optional,
        "manager": (string),
        "vendor": (string),
        "projectNumber": (int),
        "URL": (string),
        "justification": (string),
        "additionalInfo": (string),
        "items": [
            {
            "description": (string),
            "partNo": (string),
            "itemURL": (string),
            "quantity": (integer),
            "unitCost": (string),
            "totalCost": (string)
            }
```

(continues on next page)

(continued from previous page)

```
        ],
        "requestSubtotal": (int),
        "requestTotal": (int),
        "history": (list of strings)
    }
]
```

Returns list of requests matching filter

procurementUpdateAdmin()

Change the status of a procurement request to updates for admin. This will send the request to a student without requiring approval from the technical manager upon resubmission.

Expected Input:

```
{
    "_id": (string),
    "comment": (string)
}
```

procurementUpdateManager()

Change the status of a procurement request to updates for manager. This will allow the student who submitted the request to make changes or cancel it.

Expected Input:

```
{
    "_id": (string),
    "comment": (string)
}
```

procurementUpdateManagerAdmin()

Change the status of a procurement request to updates for manager. Initiated by the admin. The request will go back to the student and once submitted, will again go to the technical manager.

Expected Input:

```
{
    "_id": (string),
    "comment": (string)
}
```

projectAdd()

Add a project. Can only be done by an admin. If the projectNumber is already in use, throw an error.

Expected Input:

```
{
    "projectNumber": (int),
    "sponsorName": (string),
    "projectName": (string),
    "membersEmails: [(string), ...], # list of strings
    "defaultBudget": (string),
    "availableBudget": (string),
    "pendingBudget": (string)
}
```

projectData()

Return a list of at most 10 projects from the database. The projects may be sorted by a key and ordered by ascending or descending, and the pageNumber decides which 10 projects are returned. pageNumber must be a non-negative integer.

Expected Input:

```
{
    'sortBy': (string in 'projectNumber', 'sponsorName', 'projectName',
↪'membersEmails',
        'defaultBudget')
        (Optional. Default "projectNumber")
    'order': (string in 'ascending', 'descending')
        (Optional. Default: "ascending")
    'pageNumber': (int)
        (Optional. Default: 0)
    'keywordSearch': (dict)
        {
            "projectNumber": (int, optional),
            "sponsorName": (string, optional),
            "projectName": (string, optional),
            "membersEmails": (string, optional),
            "defaultBudget": (string, optional)
        }
}
```

Returns:

```
[
    {
        "projectNumber": (int),
        "sponsorName": (string),
        "projectName": (string),
        "membersEmails": (string),
        "defaultBudget": (string)
    }
]
```

Returns a list of at most 10 projects

projectEdit()

Change a project's attributes with the given project number. It can only be done by an admin. Project number is required, but its value cannot be changed.

Expected Input:

```
{
    projectNumber: (int),
    sponsorName: (string, optional),
    projectName: (string, optional),
    membersEmails: [(string), ..., optional]
}
```

projectInactivate()

Change status of project to inactivate.

Expected Input:

```
{
    "_id": (string)
}
```

projectPages()

Return an int: the number of pages it would take to display all current projects, filtered per the "Expected Input" JSON object described below, if 10 projects are displayed per page. At present time, the page size (number of projects per page, i.e. 10) cannot be configured.

Expected Input:

```
{
    "projectNumber": (int),
    "sponsorName": (string, optional),
    "projectName": (string, optional),
    "membersEmails": (string, optional),
    "defaultBudget": (string, optional)
}
```

Returns the number of pages it would take to display all specified projects

projectSingleData()

Return the data of a project in the database. See "Returns" below for which data is returned. The project is found by its projectNumber.

Expected Input:

```
{
    'projectNumber': (int)
}
```

Returns:

```
{
    'projectNumber': (int),
    'sponsorName': (str),
    'projectName': (str),
    'defaultBudget': (???),
    'membersEmails': (list of str)
}
```

Returns a dict containing a single project's data, as per above

projectSpreadsheetData()

This REST endpoint returns a list of 10 projects from the database. The projects may be sorted by a key and ordered by ascending or descending, and the pageNumber decides which 10 projects are returned. pageNumber must be a non-negative integer.

Expected Input:

```
{
    "bulkStatus": (string, Optional, default "valid".
        Whether these are the "valid", "invalid", "existing", or
        "conflicting" bulk projects)

    'pageNumber': (int)
```

(continues on next page)

(continued from previous page)

```
        (Optional. Default: 0)
}
```

Returns:

```
[
    {
        "projectNumbers": (list of ints),
        "firstName": (string),
        "lastName": (string),
        "netID": (string),
        "email": (string),
        "course": (string),
        "role": "student",
        "status": (string), //"current" or "removed"
    }
]
```

projectSpreadsheetMetadata()

Returns a dict summarizing the current state of the bulk project data.

Returns:

```
{
    'valid': (int, number of valid users),
    'invalid': (int, number of invalid users),
    'conflicting': (int, number of conflicting users),
}
```

Returns a dict summarizing the current state of the bulk project dataa

projectSpreadsheetOverwrite()

Marks all projects with a given project number to be ignored in the bulk project data list, so that they will not be uploaded when projectSpreadsheetUpload is called.

projectSpreadsheetPages()

Return an int: the number of pages it would take to display all current projects if 10 projects are displayed per page. At present time, the page size (number of projects per page) cannot be configured.

Expected input:

```
{
    "bulkStatus": (string, Optional, default "valid".
        Whether these are the "valid", "invalid", or "conflicting")
}
```

Returns number of pages required to display all current projects

projectSpreadsheetRevalidate()

Takes a dict of project information, a status, and an index within the list of projects with that status. The project at that index in that list of projects will be striken out, and the new project information will be re-validated, and that validated information will be appended to the end of whichever list is appropriate given its new validation status.

Expected Input:

```
{
    "bulkStatus": (str. The status of the user being replaced.
        Either "valid", "invalid", or "conflicting")
    "index": (int. The index within the list of users with
        the given status in the set of bulk user data)
    "project" :  {
        “projectNumbers”: (list of ints),
        "firstName": (string),
        "lastName": (string),
        "netID": (string),
        "email": (string),
        "course": (string),
        “role”: “student”,
        “status”: (string), //”current” or “removed”
    }
}
```

Returns:

```
{
    'project': (dict. The validated user, with the same schema as
        in the expected input),
    'status': (str. The new status of the validated project. Either
        "valid", "invalid", or "conflicting")
}
```

Returns

projectSpreadsheetSubmit()

If all of the bulk project data is valid, then all projects will be added to the database.

Otherwise, a 400 error is returned.

projectSpreadsheetUpload(*sheet*)

Upload a spreadsheet to the server for bulk project adding. This bulk project data will be stored until it is submitted.

Each project will be evaluated and sorted into four categories:

- valid - The project can be added without complication.
- invalid - The project has some invalid attribute.
- **conflicting - A project with this project number already exists,** and its data conflicts with the data given in the spreadsheet.

The data can be modified by /projectSpreadsheetRevalidate and finally submitted using /projectSpreadsheetSubmit.

Parameters **sheet** – The spreadsheet file

projectValidate()

Return true if the project number(s) exist in the database.

Expected Input:

```
{
}
```

Returns:

```
{
    "valid": boolean
}
```

Returns a boolean, as per above

reportDownload(*uuid*)
Download a report, specified by a UUID.

Parameters **uuid** – specifies which report will be downloaded

Returns an .xlsx file

reportGenerate()
Generate a report and return a UUID for the report.

Expected Input:

```
{
    'sortBy': (string in projectNumbers, firstName, lastName, netID,
        email, course, role, status)
        (Optional. Default "email")
    'order': (string in 'ascending', 'descending')
        (Optional. Default: "ascending")
    primaryFilter: {
    },
    secondaryFilter: {
    }
}
```

Returns a uuid for the generated report

requestData()
Return a list of at most 10 requests from the database. The requests may be sorted by a key and ordered by ascending or descending, and the pageNumber decides which 10 users are returned. pageNumber must be a non-negative integer.

Expected Input:

```
{
    'sortBy': (string in 'projectNumbers', 'firstName', 'lastName', 'netID',
        'email', 'course', 'role', 'status')
        (Optional. Default "email")
    'order': (string in 'ascending', 'descending')
        (Optional. Default: 'ascending')
    'pageNumber': (int)
        (Optional. Default: 0)
    primaryFilter: {
    },
    secondaryFilter: {
    }
}
```

Returns a list of at most 10 requests

requestPages()
Takes as input filters. Determines how many pages, each displaying 10 requests, it would take to show all procurement requests.

Expected Input:

```
{
    primaryFilter: {
    },
    secondaryFilter: {
    }
}
```

Returns the number of pages it would take to display all filtered requests

sequence()
Check the current sequence value of requests, return it, and increment the value.

userAdd()
Add new user to the database with provided data.

Expected Input:

```
{
    "projectNumbers": (int or list of ints),
    "firstName": (string),
    "lastName": (string),
    "netID": (string),
    "email": (string),
    "course": (string),
    "role": (string)
}
```

Returns uuid of invitation for new user

userData()
Return a list of at most 10 user documents from the database. The users may be sorted by a key and ordered ascending or descending, and the pageNumber decides which 10 users are returned. pageNumber must be a non-negative integer.

Expected Input:

```
{
    'sortBy': (string in projectNumbers, firstName, lastName, netID,
        email, course, role, status)
        (Optional. Default "email")
    'order': (string in 'ascending', 'descending')
        (Optional. Default: "ascending")
    'pageNumber': (int)
        (Optional. Default: 0)
    'keywordSearch': (dict)
        {
            "projectNumbers": (int or list of ints, optional),
            "firstName": (string, optional),
            "lastName": (string, optional),
            "netID": (string, optional),
            "email": (string, optional),
```

(continues on next page)

(continued from previous page)

```
            "course": (string, optional),
            "role": (string, optional)
        }
}
```

Returns:

```
[
    {
        "projectNumbers": (list of ints),
        "firstName": (string),
        "lastName": (string),
        "netID": (string),
        "email": (string),
        "course": (string),
        "role": "student",
        "status": (string), //"current" or "removed"
    }
]
```

Returns a list of at most 10 users

userEdit()

Edit a user document in the databse. Takes a MongoDB ObjectID and optional data fields for a user. The ObjectID is used to identify the user whose information will be edited and the optional data is what the existing data will be changed to.

Expected Input:

```
{
    "_id": (string)
    "projectNumbers": (list of ints, optional),
    "firstName": (string, optional),
    "lastName": (string, optional),
    "netID": (string, optional),
    "course": (string, optional)
}
```

userForgotPassword()

Allow a user to change their password. Send recovery link to user's email address. Link allows a user to set a new password through the userVerify endpoint, but will expire at some point in the near future.

Expected Input:

```
{
    "email": (string),
}
```

Returns uuid of invitation for forgot password form

userInfo()

This function is for debugging, and probably shouldn't be here.

Returns a variable called ret

userLogin()
Take an email and password, check if the password's hash is associated with the given email, and if so, log in a user.

Expected Input:

```
{
    "email": (string),
    "password": (string)
}
```

Returns A message if login is successful

userLogout()
Logs out a user by expiring their session.

userPages()
Return an int: the number of pages it would take to display all current users, filtered per the incoming JSON object described below, if 10 users are displayed per page. At present time, the page size (number of users per page, i.e. 10) cannot be configured.

Expected Input:

```
{
    "projectNumbers": (int or list of ints, optional),
    "firstName": (string, optional),
    "lastName": (string, optional),
    "netID": (string, optional),
    "email": (string, optional),
    "course": (string, optional),
    "role": (string, optional)
}
```

Returns the total number of pages required to display all specified users

userProjects()
Return a list of projectNumbers associated with the authenticated user.

Will only return the projects associated with the user who calls this function.

Returns:

```
{
    'projectNumbers': (list of ints)
}
```

Returns a list of project numbers

userRemove()
"Remove" a user from the system. Changes the status of a user from "current" to "removed", which has the effect that the user is no longer able to interact with the system. Doesn't delete the user from the database.

Expected Input:

```
{
    "_id": (string)
}
```

userSingleData()

Return the data of a user in the database. See "Returns" below for which data is returned. The user is found by their email.

Expected Input:

```
{
    'email': (int)
}
```

Returns:

```
{
    'projectNumbers':, (list of ints)
    'firstName':, (str)
    'lastName':, (str)
    'netID':, (str)
    'course':, (str)
    'email':, (str)
    'role': (str)
}
```

Returns a dict containing a single project's data, as per above

userSpreadsheetData()

This REST endpoint returns a list of 10 users from the database. The users may be sorted by a key and ordered by ascending or descending, and the pageNumber decides which 10 users are returned. pageNumber must be a non-negative integer.

Expected Input:

```
{
    "bulkStatus": (string, Optional, default "valid".
        Whether these are the "valid", "invalid", "existing", or
        "conflicting" bulk users)

    'pageNumber': (int)
        (Optional. Default: 0)
}
```

Returns:

```
[
    {
        "projectNumbers": (list of ints),
        "firstName": (string),
        "lastName": (string),
        "netID": (string),
        "email": (string),
        "course": (string),
        "role": "student",
        "comment": {
            merge: (boolean, if a user with this email exists),
            invalidRole: (boolean)
            missingProjects: (list of ints)
            missingAttributes: (list of strings)
            conflictingAttributes: (list of strings),
```

(continues on next page)

(continued from previous page)

```
        }
    }
]
```

userSpreadsheetMetadata()

Returns a dict summarizing the current state of the bulk user data.

Returns:

```
{
    'valid': (int, number of valid users),
    'invalid': (int, number of invalid users),
    'existing': (int, number of existing users),
    'conflicting': (int, number of conflicting users),
}
```

Returns a dict summarizing the current state of the bulk user data

userSpreadsheetPages()

Return an int: the number of pages it would take to display all current users if 10 users are displayed per page. At present time, the page size (number of users per page) cannot be configured.

Expected input:

```
{
    "bulkStatus": (string, Optional, default "valid".
        Whether these are the "valid", "invalid", "existing", or
        "conflicting" bulk users)
}
```

Returns:

```
[
    {
        "projectNumbers": (list of ints),
        "firstName": (string),
        "lastName": (string),
        "netID": (string),
        "email": (string),
        "course": (string),
        "role": "student",
        "comment": {
            merge: (boolean, if a user with this email exists),
            invalidRole: (boolean)
            missingProjects: (list of ints)
            missingAttributes: (list of strings)
            conflictingAttributes: (list of strings),
        }
    },
    ...
]
```

userSpreadsheetRevalidate()

Takes a dict of user information, a status, and an index within the list of users with that status. The user at that index in that list of users will be striken out, and the new user information will be re-validated, and that validated information will be appended to the end of whichever list is appropriate given its new validation status.

Expected Input:

```
{
    "bulkStatus": (str. The status of the user being replaced.
        Either "valid", "invalid", "existing", or "conflicting")
    "index": (int. The index within the list of users with
        the given status in the set of bulk user data)
    "user" :  {
        "projectNumbers": (list of ints),
        "firstName": (string),
        "lastName": (string),
        "netID": (string),
        "email": (string),
        "course": (string),
        "role": "student",
        "comment": {
            merge: (boolean, if a user with this email exists),
            invalidRole: (boolean)
            missingProjects: (list of ints)
            missingAttributes: (list of strings)
            conflictingAttributes: (list of strings),
        }
    }
}
```

Returns:

```
{
    'user': (dict. The validated user, with the same schema as
        in the expected input),
    'status': (str. The new status of the validated user. Either
        "valid", "invalid", "existing", or "conflicting")
}
```

Returns

userSpreadsheetSubmit()

If all of the bulk user data is valid, then all users will be added to the database and invited by email.

Otherwise, a 400 error is returned.

userSpreadsheetUpload(*sheet*)

Upload a spreadsheet to the server for bulk user adding. This bulk user data will be stored until it is submitted.

Each user will be evaluated and sorted into four categories:

- valid - The user can be added without complication.
- invalid - The user has some invalid attribute.
- existing - A user with this email already exists, but can be updated without problems.
- conflicting - A user with this email already exists, and its data conflicts with the data given in the spreadsheet.

The data can be modified by /userSpreadsheetRevalidate and finally submitted using /userSpreadsheetSubmit.

Parameters **sheet** – The spreadsheet file

userVerify()

Set the password of a new user. Checks that a provided email is matched in the database to a provided UUID. If so, creates and stores a password hash and salt for the user.

The UUID is a key in an invitation document. An invitation is created when a user is first invited to use the system and when a user forgets their password.

Expected Input:

```
{
    "uuid": (string),
    "email": (string),
    "password": (string)
}
```

validateProject(*data*, *comment=False*)

Validate a project to check for a projectNumber, sponsorName, projectName, memberEmails, and defaultBudget. If types are wrong, things are coerced if possible.

A sanitized version of the project dict is returned.

If comment is True, then the project also has a 'comment' attribute

```
comment: {
    missingAttributes: (list of strings)
    existingNumber: (boolean, true if a project with this projectNumber␣
↪exists),
}
```

Parameters

- **data** – dict. The project to be validated.
- **myComment** – bool. As described above.

Returns sanitized version of the project dict

validateUser(*data*, *comment=False*)

Validate a user to check for a role, firstName, lastName, email, and optional netID. Non-admin users are also checked for valid project numbers and courses.

A sanitized version of the user dict is returned, or None if anything was wrong with the dict.

If comment is True, then the user also has a 'comment' attribute

```
comment: {
    merge: (boolean, if a user with this email exists),
    invalidRole: (boolean)
    missingProjects: (list of ints)
    missingAttributes: (list of strings)
    conflictingAttributes: (list of strings),
}
```

Parameters

- **data** – dict. The user to be validated.
- **myComment** – bool. As described above.

Returns sanitized version of the user dict, or None if anything was wrong with the dict

1.1.3 utdesign_procurement.emailer module

class utdesign_procurement.emailer.**EmailHandler**(*email_user*, *email_password*, *email_inwardly*, *template_dir*, *domain='utdprocure.utdallas.edu'*)

Bases: object

Fills templates and sends emails using an Emailer in a separate thread.

In the various function names "confirm" refers to confirming that an action taken by a user has in fact been completed successfully. On the other hand "notify" is to notify the user that a different user has taken some action relevant to them.

Parameters

- **email_user** – (str). Gmail username of the email account to be used
- **email_password** – (str). Password for the gmail account to be used
- **email_inwardly** – (bool). If True, emails are sent to the email_user and not to their intended recipient. Good for debugging.
- **template_dir** – (str). The directory for the mako templates.
- **domain** – (str). The domain name of the system. This will be used to populate URLs in templates.

confirmRequestManagerAdmin(*email*, *requestNumber*, *projectNumber*, *action*)

Sends an email to some email address confirming that some action has happened to some request.

Parameters

- **email** –
- **requestNumber** –
- **projectNumber** –
- **action** –

Returns

confirmStudent(*teamEmails*, *requestNumber*, *projectNumber*, *action*)

Sends an email to all members of a team confirming that some action has happened to some request.

Param teamEmails: (list of str). The email addresses of the users

Param requestNumber: (dict). The request being notified about.

Param projectNumber: (dict). The project whose members are being notified.

Param action: (dict). The action happening to the request.

Returns

die()

End the email queue listening process gracefully

Returns

notifyCancelled(*email*, *projectNumber*, *requestNumber*)

Notify some user about the cancellation of a request.

Parameters

- **email** –
- **projectNumber** –

- **requestNumber** –

Returns

notifyProjectAdd(*teamEmails*, *projectNumber*, *projectName*)
Notify a user that they have been added to a project.

Parameters

- **teamEmails** –
- **projectNumber** –
- **projectName** –

Returns

notifyProjectEdit(*membersEmails*, *projectNumber*, *projectName*, *sponsorName*)
Notify a user that a project that they were a member of has been edit.

Parameters

- **teamEmails** –
- **projectNumber** –
- **projectName** –

Returns

notifyProjectInactivate(*teamEmails*, *projectNumber*, *projectName*)
Notify a user that a project that they were a member of has been deactivated.

Parameters

- **teamEmails** –
- **projectNumber** –
- **projectName** –

Returns

notifyRejectedAdmin(*adminEmails*, *projectNumber*, *requestNumber*, *manager*)
Notify some users that the admin has rejected a request.

Parameters

- **adminEmails** –
- **projectNumber** –
- **requestNumber** –
- **manager** –

Returns

notifyRequestAdmin(*adminEmails*, *projectNumber*, *requestNumber*)
Sends a notification about a request to the admins.

Parameters

- **adminEmails** –
- **projectNumber** –
- **requestNumber** –

Returns

notifyRequestManager(*email*, *projectNumber*, *requestNumber*)
Sends a notification about a request to a manager.

Parameters

- **email** –
- **projectNumber** –
- **requestNumber** –

Returns

notifyStudent(*teamEmails*, *requestNumber*, *projectNumber*, *action*, *user*, *role*)
Sends a notification about a request to some students.

Parameters

- **teamEmails** –
- **requestNumber** –
- **projectNumber** –
- **action** –
- **user** –
- **role** –

Returns

notifyStudentRejected(*teamEmails*, *requestNumber*, *projectNumber*, *action*, *user*, *role*, *comment*)
Notify a user that a request has been rejected.

Parameters

- **teamEmails** –
- **projectNumber** –
- **projectName** –

Returns

notifyUpdateManager(*email*, *projectNumber*, *requestNumber*)
Notify some user that the manager has requested updates.

Parameters

- **email** –
- **projectNumber** –
- **requestNumber** –

Returns

notifyUserEdit(*email*, *projectNumbers*, *firstName*, *lastName*, *netID*, *course*)
Notify some users that a user has edited a request.

Parameters

- **email** –
- **projectNumbers** –
- **firstName** –

- **lastName** –
- **netID** –
- **course** –

Returns

notifyUserRemove(*email*, *firstName*, *lastName*)
Notify a user that their account has been deactivated.

Parameters

- **email** –
- **firstName** –
- **lastName** –

Returns

procurementEditAdmin(*teamEmails=None*, *request=None*)
Alerts team members that a procurement request has been submitted.

Param teamEmails: (list of str). The email addresses of the users

Param request: (dict). The request being notified about.

Returns

procurementSave(*teamEmails=None*, *request=None*)
Alerts team members that a procurement request has been submitted.

Param teamEmails: (list of str). The email addresses of the users

Param request: (dict). The request being notified about.

Returns

send(*to*, *subject*, *body*)
Send an email to an address, with a subject, with a given body.

Parameters

- **to** – (str). The recipient email address.
- **subject** – (str). The subject of the email.
- **body** – (str). The HTML content of the message to send.

Returns

start()
Start the email queue listening process

Returns

userAdd(*email=None*, *uuid=None*)
Sends an invitation for a new user to set up their password for the first time.

Parameters

- **email** – The email of the user
- **uuid** – The uuid used for the invitation

Returns

userForgotPassword(*email=None*, *uuid=None*, *expiration=None*)
Sends a recovery link for an existing user to reset their password.

Parameters

- **email** –
- **uuid** –
- **expiration** –

Returns

class utdesign_procurement.emailer.**Emailer**(*email_queue*, *email_user*, *email_password*, *email_inwardly*)
Bases: object

Manages email connections and sends HTML emails in MIMEMultipart messages

emailSend(*to=None*, *subject=None*, *html=None*)
Send an email with some HTML content to some email address with some subject.

Parameters

- **to** – (str). An email address to send to.
- **subject** – (str). A subject for the email.
- **html** – (str). An HTML content to send in the email.

Returns

utdesign_procurement.emailer.**email_listen**(*emailer*, *queue*)
Listens for any messages that come through the queue and calls the emailer's send function appropriately

Parameters

- **emailer** –
- **queue** –

Returns

1.1.4 utdesign_procurement.server module

class utdesign_procurement.server.**Root**(*email_handler*, *show_debugger*)
Bases: *utdesign_procurement.apigateway.ApiGateway*

admin()
Return the admin view.

debug()
Debugging view. Allows random rest endpoints to be tested. TODO Disable this interface in production

index()
This will redirect users to the proper view

Returns

login()
Return the login page / login view.

manager()
Return the manager view.

student()
Return the student view.

verify(*id*)
This is the account setup page. Here, a user can set their password for the first time.

1.1.5 utdesign_procurement.utils module

utdesign_procurement.utils.**authorizedRoles**(**acceptableRoles*, *redirect=False*)
This is a decorator factory which checks the role of a user by their session information. If their role is not in the given list of authorized roles, then they are denied access.

If redirect=True, and the user isn't logged in at all, then they are redirected to the login page. If redirect=False (default), then they are denied access with a 403 error.

utdesign_procurement.utils.**checkProjectNumbers**(*data*, *default=None*)
Return a list of project numbers from the given dict if possible. If not found, return a default value if given, else raise a 400 error.

Parameters

- **data** – The dict from which to obtain project numbers
- **default** – A default value to return if no project numbers are found.

Returns

utdesign_procurement.utils.**checkValidData**(*key*, *data*, *dataType*, *optional=False*, *default=''*, *coerce=False*)
This function takes a data dict, determines whether a key value exists and is the right data type. Returns the data if it is, raises an HTTP error if it isn't.

Parameters

- **key** – the key of the data dict
- **data** – a dict of data
- **dataType** – a data type
- **optional** – True if the data did not need to be provided
- **default** – default string is ""

Returns data, if conditions are met

utdesign_procurement.utils.**checkValidID**(*data*)
This function takes a data dict, determines whether it has a MongoDB ObjectId and that the ID is valid.

Parameters **data** – data dict

Returns data, if conditions are met

utdesign_procurement.utils.**checkValidNumber**(*key*, *data*, *optional=False*, *default=''*)
This function takes a data dict, determines whether a key value exists and is a number. Returns the data if it is, raises an HTTP error if it isn't.

Parameters

- **key** –
- **data** –
- **optional** –

- **default** –

Returns

utdesign_procurement.utils.**convertToCents**(*dollarAmt*)
Converts a string dollar amount to an int cents amount. Strings must have two digits after the .

Parameters **dollarAmt** – the string dollar amount, does not have $ sign

Returns cents in int

utdesign_procurement.utils.**convertToDollarStr**(*cents*)
Concerts a number of cents to a string with a dollar sign.

Parameters **cents** – (int). A number of cents

Returns

utdesign_procurement.utils.**generateSalt**()
This function generates a random salt.

Returns random salt

utdesign_procurement.utils.**getKeywords**(*keywords*)
Parse and sanitize a dict of keywords to be used in filtering the various users in the users collection in mongo.

Parameters **keywords** – (dict).

Returns

utdesign_procurement.utils.**getProjectKeywords**(*keywords*)
Parse and sanitize a dict of keywords to be used in filtering the various projects in the projects collection in mongo.

Parameters **keywords** – (dict).

Returns

utdesign_procurement.utils.**getRequestKeywords**(*data*)
Parse and sanitize a dict of keywords to be used in filtering the various requests in the requests collection in mongo.

Parameters **keywords** – (dict).

Returns

utdesign_procurement.utils.**hashPassword**(*password*, *salt*)
This function hashes a password with a given salt.

Parameters

- **password** – a plaintext password
- **salt** – a random salt

Returns the hashed password

utdesign_procurement.utils.**lenientConvertToCents**(*dollarAmt*)
Converts a string dollar amount to an int cents amount. Strings can have 1 or 2 digits after the . or just be an integer.

Parameters **dollarAmt** – the string dollar amount, does not have $ sign

Returns cents in int

utdesign_procurement.utils.**requestCreate**(*data*, *status*, *optional=False*)

Takes data as an input as uses the data to create a procurement request (dict).

Expected input:

```
{
    "manager": (string), //email of manager who can approve this
    "vendor": (string),
    "projectNumber": (int or list of int),
    "URL": (string),
    "justification": (string) optional,
    "additionalInfo": (string) optional,
    "items": [
        {
        "description": (string),
        "partNo": (string),
        "itemURL": (string),
        "quantity": (int),
        "unitCost": (string),
        "totalCost": (string)
        }
    ]
}
```

Parameters

- **data** – a dict containing data to be stored in the request
- **status** – what will be the initial status of the request?
- **optional** – if True, all fields of data except procjectNumber will be optional

Returns procurement request, stored as a dict

utdesign_procurement.utils.**verifyPassword**(*user*, *password*)

This function hashes a password and checks if it matches the hash of a given user.

Parameters

- **user** – a user document from database
- **password** – a hashed password

1.1.6 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

u

INDEX

M

N

P

R

S

U

V