Workflow for Automated Handling of One Submission Email

Sally A.M. Hogenboom

2023-04-26

Setup

```r
library(yOUrEmail)
library(tidyverse)
library(magrittr)

# Authenticate & get Outlook Environment
outlook <- Microsoft365R::get_business_outlook()
```

Retrieve All Submissions

Goal

Retrieve all new (i.e., unread) emails from the submission system.

Requirements

- Outlook Authentication

Execution

- `outlook`: pass down the outlook environment retrieved during successful authentication
- `submissions_folder`: specify the folder where the submissions are received. Some teachers have automated rules set-up which move the submissions to a different *outlook* folder (e.g., 'Nakijken'). If this is the case, specify the exact name of that folder.
- `submissions_email`: built-in option to allow for upcoming changes to the submissions system which will send the submissions from a different emailadres. Defaults to `submit@oupsy.nl`

- `email_status`: unread/read/all; only parse emails which meet the specified `status`.

```r
submissions <- get_submissions(outlook,
  submissions_folder = "Nakijken",
  submissions_email = "submit@oupsy.nl",
  email_status = "unread"
)
```

Expected Behavior

- Retrieve a list of emails with class `ms_outlook_email`

Development To Do's

- *None*

Extract Information

NOTE; This is a workflow for one email. This allows for a better step-by-step testing and understanding of the administrative process. The process can be included in a loop to deal with multiple (new) emails (see below).

Goal

Process the contents of the email to extract the relevant information:

- `course_id` unique identifier with two letters (e.g., PB for Psychology Bachelor) and four numbers
- `course_name` human readable name of the course
- `course_run` courses can have different versions (i.e., runs). The course run determines which grading form should be used for the submission.
- `student_number` unique numeric identifier for each student.
- `student_name` name as parsed from the student's email account.
- `student_email` email address provided by the student which is used to communicate the attained grade with students.
- `submission_date` date on which the email/submission was received.
- `student_submission_note` students have the option to add a note for the teacher. If such a note was provided it is also extracted.

Added additional relevant content: * `grade_before_date` compute the date before which the grading should be finished (15 working days from `submission_date`). * `status` with default value 'To Do'. This signals to the teachers which grading has already been completed. * `assignment`; depending on the course it defaults to the first 'Tentamenkans' (T1), Assignment

A (A), or the Preregistratation (PRE). This signals to teachers whether feedback or a grade is needed. * `graded_on`; optional to keep track of when a grade was sent. * `grading_notes`; optional to keep track of any collaborations/decisions worth remembering.

Requirements

- An email from the submission system with class `ms_outlook_email`
- `install.packages(c("qdapRegex", "bizdays"))`
  - `qdapRegex` is used to extract the students submission note.
  - `bizdays` is used to compute workingdays - not accounting for national holidays.

Execution

```
submission_info <- get_submission_info(
  email = submissions[[1]],
  n_workdays = 15
)
```

Expected Behavior

A tibble with one row and 14 columns: (status, course_id, course_name, course_run, student_number, student_name, student_email, submission_date, student_submission_note, grade_before_date, assignment, graded_on, grade, en grading_notes).

Development To Do's

1. Currently, when computing the `grade_before_date` a shortcut was made to compute the date with the number of working days from the `submission_date`. In doing so, we do not account for national holidays such as Christmas or Koningsdag. Without accounting for these holidays the `grade_before_date` can at times be a couple of days earlier than necessary - but earlier grading does not affect the students negatively.
2. Write tests to account for emails with missing information. Low priority due to the fact that emails are automatically generated with a standardized structure.

Create File Structure

Goal Each submission is stored according to a standardized file structure: `course` > `course_run` > `student`. For new submissions this file structure does not yet exist and is therefore created before saving the submission attachments.

Requirements

- The parsed contents of a new submission email

- Administrative rights allowing the creation of new folders
- This file should be nested within an R project that also (will) contain(s) the student submissions.

Execution

```r
# Example: one email
student_folder <-
  create_folder_structure(submission_info,
    grading_folder = here::here("Ingeleverde Opdrachten")
  )
```

Expected Behavior

- Create a nested file structure with the format: `grading_folder` > `course_id` > `course_run` > `student_folder`.
- Return the path to the student folder for use in subsequent code.

Development To Do's

None

Extract and Save Attachments

Goal

Extract the attachments from the email and place them in the designated student folder.

Requirements

- email with class `ms_outlook_email`
- file path to the student's folder

Execution

```r
# Invisible to prevent list output of 'NULL' results for each downloaded document.
invisible(
  download_attachments(
    email = submissions[[1]],
    student_folder = student_folder
  )
)
```

Expected Behavior

- All attachments are downloaded and stored in the `course` > `course_run` > `student` folder.
- All attachments are renamed to the short and standardized format where possible.

Development To Do's

1. Research how to prevent the list of NULL results from within the `download_attachments` function. Tried invisible `attachment$download()` but that did not change anything.

Add Grading Form

Goal * Select the correct grading form (dependent on `course` > `course_run`) and add it to the student's folder. * Update the file name to include the student's details. Keep it in the same format as the files that were received from the submission system.

Requirements

Grading forms differ in their naming conventions. In addition, some courses see a different grading form for each `course_run`, while others keep the same form for years. To prepare this step, a human must once add & update the name of each grading form manually. The `grading_form_folder` must have the following structure:

- A folder for each `course` (e.g., `PB0812`)
- A grading form for each `course_run` (e.g., `PB0812202144.xlsx`). If a course does not have multiple `course_run`s a `course_id` is sufficient (e.g., `PB0412.xlsx`).

Technically, to get started only those grading forms need to be added from courses where new submissions can be expected (1-year after course start).

Execution

```r
invisible(
  move_grading_form(
    grading_form_folder = here::here("Beoordelingsformulieren"),
    student_folder = student_folder,
    submission_info = submission_info
  )
)
```

Expected Behavior

The required grading form is copied from the `grading_form_folder` to the `student_folder`. The name is changed to align with the naming conventions of the other received documents.

Development To Do's

1. Resolve printing `FALSE` after executing the function.

Update Submission Overview

Goal

Update the current submission overview with the latest submission. Reorder the overview such that the 'To Do's' are at the top, with the most urgent grading (i.e., closest `grade_before_date`) at the top.

Requirements

- The `submission_info`
- The name of the `submission_overview`, which if it doesn't yet exist will be created.

Execution

```
update_submission_overview(
  submission_info = submission_info,
  submission_overview = here::here("NakijkOverzicht.xlsx")
)
```

Expected Behavior

- An excel sheet with the name of `submission_overview` is created which contains the most recent and historic `submission_info`.

Development To Do's

1. Conditional formatting
2. Compute a 'grade_by_n_days' or add an excel formula to automatically update.
3. Complete historic information from read/graded outlook emails.
4. Prevent warnings from showing

Draft Confirmation Email

Goal

Draft a confirmation email to the student which contains: 1. A general confirmation that the assignment was received 1. A general statement of being able to expect the grading at the latest before the `grade_before_date`.

Requirements

- The submission_info from a single submission as extracted in the step above.
- The email content to allow a reply to the correct email (for keeping easy inbox structures)
- install.packages("blastula")

Execution

```
draft_confirmation_email(
  outlook = outlook,
  submission_info = get_submission_info(email = submissions[[1]]),
  email = submissions[[1]],
  teacher_name = "Sally"
)
```

Expected Behavior

A standardized email is drafted to the student's email adress to confirm receiving a new submission. This includes a mention of the date prior to which the students can expect their grade/feedback.

Development To Do's

1. Optimize the styling of the email such that it includes the OU house style, a logo, the correct font, a formal email signature, and a footer.
2. Check whether existing footers are added below the created email.

Plagiarism Scan

Goal

Forward the email, containing only the relevant files, to the plagiarism scan software.

Requirements

- The email with class ms_outlook
- A personal email adres linked to Urkund

Execution

```
forward_plagiarism_scan(email = submissions[[1]],
                        student_folder = student_folder,
                        urkund_email = "sally.hogenboom.ounl@analysis.urkund.com")
```

Expected Behavior

- An email is send to the `urkund_email` adres which includes the report (i.e., `verslag`) as attachment.

Development To Do's

1. Normally you would forward the email to the urkund email adress while removing the irrelevant attachments. However, this is not possible in the `Mircosoft365R` package for two reasons. First, creating a 'forward' email takes a crazy amount of time to process. Second, each attachments that is removed requires a manual confirmation in an alert pop-up. This prevents it from implementation as part of an automated workflow.
2. Because of this issue the current workflow does not allow for the plagiarism email to become part of the email 'thread' and is instead sent separately.
3. The new course for Kwalitatief (PB1612) does not follow the standardized name format, which is why the automated forward does not work.

Mark Email As Read

Goal

Mark the email as 'Read' so that it will not be processed again.

Requirements

- The `email` with class `ms_outlook`

Execution

```r
# Mark email as processed
invisible(submissions[[1]]$update("isRead" = TRUE))
```

Expected Behavior

- Within the outlook environment the email is now marked as 'Read'

Development To Do's

- *None*