Package 'elearnr'

May 16, 2023

Title R-Based Classroom Test

Version 0.0.1

Description Deliver R tests via a cloud folder, collect results and grade them.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Imports pcloudr (>= 0.0.1),
    stats,
    curl

Suggests spelling

Language en-GB

Remotes local::../pcloudr

R topics documented:

`authorise` *Authorise elearnr*

Description

Interactive procedure to authorise elearnr on 'pCloud.com'.

Usage

```
authoriseme(client.id, client.secret, ports, ...)
```

Arguments

`client.id`	the app client identification issued at registration.
`client.secret`	the app client secret issued at registration.
`ports`	a numeric vector of the redirect ports set during the app registration. A port is the `xxxx` component in the `http://localhost:xxxx` redirect URLs.
`...`	more arguments required by `pcloudr::pcloud.auth()`.

Details

Authorisation is a one-off procedure, that allows elearnr to have a limited access (based on OAuth2) to the free cloud storage service `https://pcloud.com`.

The first part of the authorisation happens on the pCloud website.

1. Sign up if necessary and sign in to `https://docs.pcloud.com/my_apps/` (trailing / is necessary).
2. Create a new app, perhaps named `elearnr`, setting `Folder access` to `Private` and `Write access` to `Yes`.
3. After creating the app, click on its link, to edit settings, and:
   (a) disallow 'implicit grant';
   (b) add one or more redirect URLs, such as `http://localhost:12345`, where the number should be in the range 1,024-65,535(*);
   (c) take note of the *Client ID*, *Client secret* and the used number(s);
   (d) Save, and check that your settings are actually stored (recommended).

(*) Adding more URLs reduces the risk that the used numbers (aka port numbers) are taken.

To authorise elearnr in R, use:

```
authoriseme(client.id, client.secret, ports)
```

where the arguments are those in Step 3.c above. elearnr will open the pCloud website for confirmation and ask for a password in order to save your pCloud access data to the encrypted file `~/pcloudr`.

If you want to set specific security requirements for your encryption password or get more (customisation) information about the procedure, use `vignette("pcloudr")`, to consult the vignette of the package `pcloudr` and its function `pcloudr::pcloud.auth()`, which is essentially equivalent to `authoriseme()`.

Note. As result of this procedure, elearnr remote root path is mapped to the real pCloud folder `/Applications/elearnr`. This restriction is a security feature.

Value

logical value for success.

See Also

`browseVignettes("pcloudr")` `secretR::pwpolicy()`,`pcloudr::pcloud.config()`

`configure` *Configuration*

Description

`read.conf()` and `write.conf()` resp. reads and writes the package configuration file `~/elearnr.conf`, unless an alternative argument is passed. You are not supposed to write the file and the function can be removed in the future. If you really want to, run `read.conf()` immediately after, to keep memory consistence. `is.setup()` can be useful in scripts to know if a course has been setup and its configuration loaded in memory.

Usage

```
read.conf(confile = "~/elearnr.conf")

is.setup()

write.conf(conflist, confile = "~/elearnr.conf")
```

Arguments

`confile`	the package onfiguration file path.
`conflist`	list with the variables to add to the configuration file. See details.

Details

Configuration lines, should be R assignments such as `key = val`. Full or inline comments are possible, If a key is assigned twice, the last value is used. Using a non-default path for `confile` has effect only for the current R session.

`conflist` is a list whose elements are variables symbols used to populate the configuration file. If a list element is unnamed, the variable symbol and value are added as a key and value, that is `list(var)` is added as `var = 123`, assuming that `123` is `var` value. For a named list element, the name is considered as the variable to store, while the element value is used as a comment, thus `list(var = "some comment")` is added as `var = 123 # some comment`. Comments are added inline, unless they start with one or more newlines, in which case they are written above the configuration line.

NOTE The package configuration file, `confile`, is intended to store instructor related data. For students the file returned by `.exam.enrol.file()` is used, which can be read with `exam.show.enrol()`.

Value

`read.conf()`: a list whose names/values are the configuration file keys/values, which is also stored in memory.

`read.conf()`: `TRUE` when a course configuration data is found in memory else otherwise.

`write.conf()`: is used only for its side effect.

`enrol` *Enrolment*

Description

`exam.enrol()` prompts for student data and classroom-exam code and save it to the binary *student enrol file* provided by `.exam.enrol.file()`.

Usage

```
exam.enrol()

exam.update.code(code = NULL)

exam.show.enrol(nice.format = TRUE)
```

Arguments

`code`	the classroom-exam code to update.
`nice.format`	`TRUE` for a nicely formatted output, or `FALSE` for scripting.

Details

You can at any time obtain your enrol with `exam.show.enrol()` and, if necessary, the classroom-exam code can be later updated with `exam.update.code()`, which is interactive if a code argument is not passed.

`exam-session` *Exam Session*

Description

`exam.download()` download the exam data and #' `exam.send()` send it in cloud.

Usage

```
exam.download()

exam.info()

exam.send(...)

exam.clearmem()
```

Arguments

`...`	names of the arguments are the names of the variables required to send, values are typically he same variables.

Details

Each argument of `exam.send()` should be named: the argument names are the variables to send and the argument values are the related values

After sending, it si suggested to use `exam.clearmem()` to avoid name conflicts.

Examples

```
## Send 'a' with its actual value and 'b' with value taken by ;foo'
## Not run:
exam.send(a = a, b = foo)

## End(Not run)
```

`grading` *Grade exams*

Description

`download.exams()` downloads the exam-upload dir to a time-named directory inside the local course directory. `grade.exams` produces grading files in the same directory.

Usage

```
download.exams()

grade.exams()
```

Details

The exam-upload dir and the local course directory are stored in `elearnrEnv$config$updir` and `elearnrEnv$config$course.locdir`. RDS exam are store in the `sent` sub-directory.

Grading file are `results.txt` and `results.csv`, with summary data in text and CSV format, and `details.txt` with summary detailed text data.

`load-assign` *Load assignment in cloud*

Description

Laod assignment with and without solutions and set its send status as active.

Usage

```
upload.assign(rdata, rds, remind = TRUE)

set.send.active(on = TRUE)

get.send.status(on = TRUE)
```

Arguments

`rdata`	path to RData file
`rds`	path to rds file with solutions
`remind`	a friendly remainder to activate the exam with `set.send.active()`.
`on`	if `TRUE`/`FALSE` the remote `sendactive` flag is updated accordingly.

Details

`set.send.active()` updates the 'expire' file in cloud, setting `sendactive` flag to `TRUE~ or` FALSE`and the time of its update. The`exam.send' function, reads this file, which contains also the exam build, and determines if it could send the data.

You can use `get.send.status()` to obtain the exam random build number, the activity flag and the time of the last status update.

Value

The only function to return non-null data is `get.send.status`, which give a list with the exam random build number, the active-send status and the last time the status was updated.

`remote-paths` *Remote Paths*

Description

If you comply with the procedure detailed in `authoriseme()`, when you are required to enter pCloud remote paths by elearnr functions, for example `setup.course()`, the remote root (/) is relative to a specific pCloud-assigned folder. This is a security feature resulting from setting the private folder option in pCloud (see point 2 in `authoriseme()` 'Details').

The path of the pCloud-assigned folder is `/Applications/elearnr`, unless during the authorisation you used an app name other than 'elearnr'. Put it another way, this is the only cloud folder elearnr can read and write, therefore the actual cloud path `/Applications/elearnr/my-course-upload` is seen by elearnr simply as `/my-course-upload`.

See Also

`authoriseme()`

`structure` *Create the local and cloud folder structure*

Description

Set a local work directory and create the cloud folders `/<course>-download` and `/<course>-upload` folders, where `<course>` is a prefix denoting a short name for the course. Recall, as noted in remote-paths, that to elearnr the cloud root is by default relative to `/Applications/elearnr`, which is the only path where elearnr has access. The content of both cloud folders, can be deleted with `clear.structure()`, or just the content of the upload folder if passing `both = FALSE`. This function does not require any confirmationm and automatically skips non-exiting folders, therefore can be used in scripts before executing `setup.course()`.

Usage

```
setup.course(
  course = NULL,
  course.locdir = NULL,
  uplink = NULL,
  confile = "~/elearnr.conf",
  quiet = FALSE
)

clear.structure(course, both = FALSE)

give.exam.code(interactive = TRUE)
```

Arguments

`course`	a prefix without spaces representing a short course name.
`course.locdir,`	path of local course directory.
`uplink`	the upload link to `/<course>-upload` cloud folder as full URL or just the URL embedded code. This argument requires `course` too.
`confile`	the package onfiguration file path. See Configuration
`quiet`	do not give informative messages.
`both`	if as per default `FALSE` delete only content of `/<course>-upload`, else `/<course>-download` too.
`interactive`	if TRUE, use a message to show the value.

Details

`/<course>-upload` is a public upload link ("Request for files") intended to receive student exercise submissions. Submissions will be automatically dowloaded in a directory inside `course.locdir` named after current time with the name format `yyyymmdd-HHMMSS`.
`/<course>-download` contains `stud.boot.txt`, with exam data intented for students, and the data files, `<course>-wsol.rds` and `<course>-nosolRData`, with the solved and unsolved class exercise.

The function saves configuration data to `~/elearnr.conf`, unless an alternative value is given to `confile` argument.
`/<course>-upload`, `/<course>-download`, and `course.locdir` are created if nonexistant; also the two cloud folders are deleted upon-confirmation if not empty. Parameters defaultng to `NULL` can be inserted interactively.

Upload link. For technical reasons, noted in `pcloudr::pcloud.pupload.data()` help, you are prompted to manually create and input the public link to `/<course>-upload`. To do this in the 'pCloud.com' webapp, create and open the folder `/<course>-upload`, then select 'Request files' under the ellipsis folder menu and get link. The upload link can be submitted interactively or passed through `uplink`. Bear in mind that, when you delete and then recreate the folder `/<course>-upload`, the link has to be regenerated.

Classroom-Exam Code. This is a nickname (perhaps too long) for a public link to `stud.boot.txt`, returned by `setup.course()` and intended for distributions to students. The link is necessary to bootstrap student-side functions, e.g. it links to the `<course>-upload` folder and `<course>-nosolRData` data file. If you want to distribute the classroom-exam code at a later tine, use `give.exam.code()`, and with `interactive = FALSE` to return it in a script.

Value

`setup.course()`: While run primarily for its side effects, it also returns *classroom-exam code*, that is the encoded public download link to `stud.boot.txt`, intended for distributions to students.

`clear.structure()`: While run primarily for its side effects, it also returns a list of removed objects inside `/<course>-download` and `/<course>-upload`.

`give.exam.code()`: if `interactive = FALSE`, the classroom-exam code, else `NULL`. In both cases, throw an error when a configuration is not available.

`student.dot` *Students' Dot Functions*

Description

Dot functions are functions starting with a `.exam` prefix not intended for normal student usage, but occasionally necessary for troubleshooting.

Usage

```
.exam.build()

.exam.enrol.file()
```

Details

Student Enrol File. `.exam.enrol.file()` returns the path of the student enrol file. It is also used internally by functions reading and writing from and to it.

Test assignment freshness. When a student takes too much time, their downloaded assignment may be overwritten by a new one uploaded by the instructor. To detect conflicts, when created, assignments are associated with a random build number distinguishing them, and functions, such as `exam.info` and `exam.send`, can use it to determine if the assignment is current. Beyond that, the instructor can ask a student to run `.exam.build()` to compare the build of the student's downloaded assignment with the build of the assignment stored in cloud, which, for the instructor, is returned by `get.send.status()`.

Value

`.exam.build`: the random exam build generated when the assignment was created.

`.exam.enrol.file()`: the student enrol file path.

Index