SchoolApps

Release Version 1.0.2-dev

Hangzhi Yu, Julian Leucker, Jonathan Weth, Frank Poetzsch-Hefft

20.11.2019

Inhalt:

KAPITEL 1

Development Guide

Diese Anleitung soll für einen einheitliches Development und einheitlichen Code in SchoolApps sorgen.

1.1 Versionsverwaltung – Einsatz von Git

Es wird mit **Git** gearbeitet. Die Verwendung ist Pflicht, die Regelungen im Git-Management Guide sind zu beachten. Alle Veränderungen sollten nach Beendigung der Arbeit an SchoolApps direkt im entsprechenden Featurebranch committed und gepusht werden, auch wenn sie noch nicht fertig oder lauffähig sind, um Datenverlust vorzubeugen (hierbei unbedingt den Bereich Branches im Git-Management Guide beachten).

1.2 Codestyle

Grundsätzlich wird **auf englisch gecodet**: Englische Variablen, englische Kommentare und englische Commits. Die Oberfläche wird natürlich auf deutsch lokalisiert. Lediglich Issues auf GitHub und die Dokumentation werden auf deutsch verfasst.

Weiterhin gilt es, die **PEP-8-Richtlinien** der Python Foundation einzuhalten, sie sorgen für einen sauberen, einheitlichen und übersichtlichen Stil.

Tipp: Viele Editoren können die Pythonskripte auch automatisch nach PEP8 formatieren – meistens klappt das sehr gut (siehe auch „Editoren/IDEs“).

1.3 Editoren/IDEs

Natürlich kann man auch im Texteditor des jeweiligen Betriebssystemes oder in der IDLE von Python programmieren. Allerdings empfiehlt es sich gerade bei so großen (Django-)Projekten wie SchoolApps einen vernünftigen **Editor** oder eine **IDE** (Integrated Development Environment) zu benutzen. Folgende Software ist sehr empfehlenswert:

Atom Ein Opensource-Editor von GitHub. Einfache Bedienung, gute Syntaxhervorhebung, mit Plugins auch Autovervollständigung und Codeformatierung

Visual Code Ein Opensource-Projekt von Microsoft. Sehr ähnlich zu Atom.

PyCharm Community/Professional Edition Kommerzielles Programm, gibt es in einer OpenSource-Version (Community) sowie einer Bezahlversion (Professional). Alle Funktionen von VisualCode und Atom enthalten, zudem wird in der Professional Edition Django von Haus aus unterstützt. TIPP: Die Professional Edition gibt es für den Bildungsbereich kostenlos.

1.4 Dokumentation

„It was hard to write, it should be hard to read."

Alle Programmierer hassen es – dennoch ist **Kommentieren** die beste Dokumentationsmöglichkeit, damit andere Leute den eigenen Quellcode lesen können.

Alle Funktionen (Klassen, etc.) sollten mit Docstrings versehen sein. Docstrings werden nach dem folgenden Schema aufgebaut: https://sphinx-rtd-tutorial.readthedocs.io/en/latest/docstrings.html

Jede App, die sich in ihrer Funktion bzw. Funktionsweise nicht selbst erklärt, sollte eine **README** besitzen, die den Programmierer über Sinn und Zweck aufklärt. Dies gilt insbesondere, wenn diese App auch Funktionen für andere Apps bereitstellt. Diese sollten in der README gesondert erwähnt und sauber dokumentiert werden.

In der **Install.md** (im Wurzelverzeichnis) sollten alle Schritte zur Installation dokumentiert werden. Insbesondere gilt das, wenn man eine neue Bibliothek für eine bestimmte Funktionalität benötigt. Diese muss sofort aufgeführt werden, damit andere Entwickler den Code ohne großes Basteln zum Laufen bekommen.

1.5 Problem, Bug, Idee? – Issues verwenden!

Hier sei nochmal auf den Bereich „Issues" im Git-Management Guide hingewiesen – unbedingt beachten!

KAPITEL 2

Git Management Guide

Diese Anleitung soll bei der Benutzung von Git im Projekt SchoolApps helfen.

2.1 Hinweis

Hilfe und Informationen zu Git allgemein gibt es u. a. bei dieser Anleitung:

https://rogerdudler.github.io/git-guide/index.de.html

2.2 Git-Hosting von SchoolApps

Die aktuelle Version von SchoolApps ist auf GitHub in der Organisation Katharineum gehosted:

https://github.com/Katharineum/school-apps

Das Cloning sowie Pullen und Pushen kann wahlweise mittels SSH oder HTTPS erfolgen (genaueres in der obigen Anleitung). Der Zugriff auf das Repository wird von Herrn Poetzsch-Heffter (@poetzsch, p-h@katharineum.de) geregelt.

2.3 Clonen und Ersteinrichtung

Clonen: `git clone https://github.com/Katharineum/school-apps`

oder `git clone git@github.com:Katharineum/school-apps.git`

Um sich eine Entwicklungsumgebung für SchoolApps einzurichten, wird die Anleitung auf

https://github.com/Katharineum/school-apps/blob/dev/README.md

unter dem Reiter Installation empfohlen.

2.4 Was gibt es zu tun? – Issues

2.4.1 Issues anlegen

Wenn einem in der SchoolApps etwas auffällt was getan werden sollte (Bug, Rechtschreibfehler, Zusätzliche Funktionen, etc.), dann muss ein Issue auf GitHub erstellt werden. Dies gilt auch, wenn man sich sicher ist, das Problem lösen zu können.

Wichtig beim Erstellen eines Issues:

- Aussagekräftigen Titel verwenden (deutsch)
- Problem detailliert beschreiben, falls vorhanden, Lösungsmöglichkeit(en) nennen (deutsch)
- **Assignees (Zuständigkeit)** → Wenn man das Problem selbst lösen will bzw. sich darum kümmern will, empfiehlt es sich, sich selbst einzutragen. Damit ist man auch für die Lösung verantwortlich. Wenn man weiß, dass dieses Problem genau von einer bestimmten Person gelöst werden kann bzw. diese dafür zuständig ist, wird diese eingetragen. Es können auch mehrere Personen eingetragen werden.
- **Labels (Kategorisierung)** → Es gibt zwei Arten von Labels:
  - **Bereichslabel** → Hier wird der Teil der Software genannt, der von dem Issue betroffen ist (im Regelfall eine Django-App, manchmal auch *whole project*)
  - **Kategorisierungslabel** → Hier wird die Art des Issue festgelegt.
    - *bug* → Ein eindeutiger Fehler. (Sicherheits-/ Funktionsbeinträchtigend)
    - *beauty mistake* → Schönheitsfehler. (Meist Designtechnisch)
    - *refactoring* → Code(-style) verbessern, umprogrammieren.
    - *it works but it is wrong* → Unsauber gelöste Probleme, fehlende Komponenten.
    - *new feature* → Neue Funktionen.
    - *discussion* → Zu diskussieren. Diese Issues *sollen nicht* umgesetzt werden, bis dieses Label entfernt wurde.

  Es können auch mehrere Bereichs- bzw. Kategorisierungslabel gleichzeitig verwendet werden.
- **Projects** → Die passende Zuordnung eines GitHub-Projekts für das Kanbansystem.
- **Milestones** → Die Version der Software, zu der dieses Problem gelöst werden muss.

2.4.2 Issues bearbeiten/lösen

In der Issue-Liste auf GitHub kann jeder sehen, was es aktuell zu tun gibt. Vorzugsweise kümmert man sich um Issues auf dem eigenen Spezial- bzw. Aufgabengebiet. Wenn einem Issue bereits jemand zugeteilt ist, ist unbedingt Rücksprache zur betreffenden Person zu halten (Assignees). Das eigentliche Lösen (im Regelfall Programmieren) erfolgt als feature-Branch (siehe unten).

2.5 Am Code arbeiten – Die Branches

Der Hauptbranch ist der **dev**. In ihm ist immer die aktuelle Entwicklung von SchoolApps gespeichert. Er wird bei jeder stabilen Version im **server-Branch** veröffentlicht. Dieser stellt den aktuellen Stand auf dem Server da.

In keinem dieser Branches wird aktiv commited (Ausnahmen bestätigen die Regel), sie werden nur über **Pull Requests** „gefüttert“.

Alle Arbeiten am Code werden in vom dev-Branch abgezweigten Arbeits-Branches durchgeführt. Ein Branch entspricht dabei einer Funktion bzw. einem Issue. Wenn jemand gleichzeitig an mehreren, nicht zusammenhängend Issues arbeitet, wechselt dieser auch die Branches.

Die **Namenskonvention** für diesen Arbeits-Branch ist wie folgt:

- **Ist die Änderung, die programmiert wird, ein neues Feature, so wird der Branch so benannt:** `feature/<der-name-des-feature-auf-englisch>` (Bindestriche, kleingeschrieben)
- **Ist die Änderung, die programmiert wird, ein Bugfix, so wird der Branch so benannt:** `bugfix/<beschreibung-des-bugs-auf-englisch>` (Bindestriche, kleingeschrieben)
- **Ist die Änderung, die programmiert wird, ein Code-Refactoring, so wird der Branch so benannt:** `refactor/<der-name-des-feature-auf-englisch>` (Bindestriche, kleingeschrieben)

2.6 Am Code arbeiten – Committen

Nochmal zur Erinnerung: Es wird nur in feature-Branches direkt committed. Die Commits erfolgen im Gegensatz zu Issues auf **englisch**, bei Bezug zu einem Issue wird das **Issue mit Hashtag** genannt.

Beispiel: `Solve LDAP connection problems (issue #10)`

2.7 Am Code arbeiten – Mergen/Pull Requests

Ist man der Meinung, dass die Arbeiten am feature-Branch abgeschlossen sind, stellt man auf GitHub ein **Pull Request** und lässt die Arbeiten vom Rest des Entwicklerteams absegnen. Dies ist unbedingt zu befolgen.

KAPITEL 3

Dokumentation

Zur Dokumentation wird das Tool sphinx eingesetzt. Die gesamte Dokumentation liegt im Unterordner `docs`.

- Formartierung in restructedText
- **Erstellen der HTML-Ausgabe** make html

untisconnect.api.**format_classes**(*classes*)
Formats a list of Class objects to a combined string

example return: „9abcd“ for classes 9a, 9b, 9c and 9d

Parameter **classes** – Class list

Rückgabe combined string

untisconnect.sub.**generate_sub_table**(*subs*, *events=[]*)
Parse substitutions and prepare than for displaying in plan

Parameter

- **subs** – Substitutions to parse
- **events** – Events to include in table

Rückgabe A list of SubRow objects

untisconnect.sub.**get_header_information**(*subs*, *date*, *events=[]*)
Get header information like affected teachers/classes and missing teachers/classes for a given date :param date: The date as datetime object :param subs: All subs for the given date :return: HeaderInformation object with all kind of information

untisconnect.sub.**parse_type_of_untis_flags**(*flags*)
Get type of substitution by parsing UNTIS flags :param flags: UNTIS flags (string) :return: type (int, constants are provided)

untisconnect.sub.**substitutions_sorter**(*sub*)
Sorting helper (sort function) for substitutions :param sub: Substitution to sort :return: A string for sorting by

KAPITEL 4

Index

- genindex
- modindex
- search

Python-Modulindex

u

Stichwortverzeichnis