JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Arbeitsprotokoll

Rest Kommunikation

Softwareentwicklung
5BHIT 2017/18

Martin Wölfer

Version 0.1
Note:
Betreuer: RAFM
Begonnen am 20. Dezember 2017
Beendet am 20. Dezember 2017

Inhaltsverzeichnis

1 Aufgabenstellung

Es soll ein Programm erstellt werden mit welchem man auf die Google Drive API zugreift. Die Response des Server soll anschließend passend in einer GUI ausgegeben werden.

2 Ergebnisse

2.1 view

Die Klasse `view.py` wurde automatisch von PySide erstellt. In dieser Klasse wurde bereits automatisch auf den close-button das Signal der `close()` Methode gelegt. Generell ist zu sagen, dass diese Klasse nicht weiter verändert werden soll.

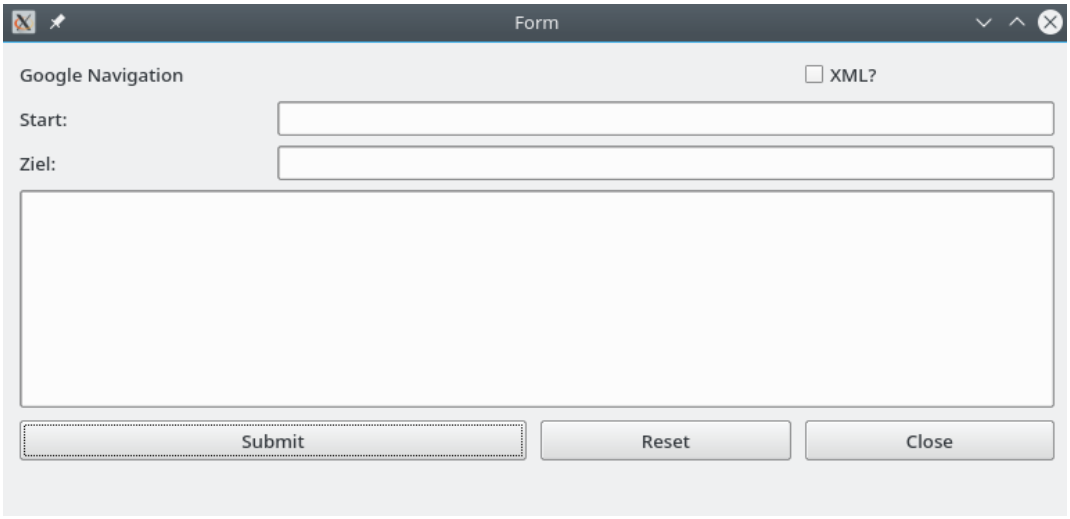

Abbildung 1: Layout der View

```
class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(769, 339)
        self.gridLayout = QtGui.QGridLayout(Form)
        self.gridLayout.setObjectName("gridLayout")
        self.output_ok = QtGui.QLabel(Form)
        self.output_ok.setText("")
        self.output_ok.setObjectName("output_ok")
        self.gridLayout.addWidget(self.output_ok, 5, 0, 1, 1)
        self.button_submit = QtGui.QPushButton(Form)
        self.button_submit.setObjectName("button_submit")
        self.gridLayout.addWidget(self.button_submit, 4, 0, 1, 2)
        self.button_reset = QtGui.QPushButton(Form)
        self.button_reset.setObjectName("button_reset")
        self.gridLayout.addWidget(self.button_reset, 4, 2, 1, 1)
        self.label_3 = QtGui.QLabel(Form)
        self.label_3.setObjectName("label_3")
        self.gridLayout.addWidget(self.label_3, 2, 0, 1, 1)
        self.label_2 = QtGui.QLabel(Form)
        self.label_2.setObjectName("label_2")
        self.gridLayout.addWidget(self.label_2, 1, 0, 1, 1)
        self.input_ziel = QtGui.QLineEdit(Form)
        self.input_ziel.setObjectName("input_ziel")
        self.gridLayout.addWidget(self.input_ziel, 2, 1, 1, 3)
        self.label = QtGui.QLabel(Form)
        self.label.setObjectName("label")
        self.gridLayout.addWidget(self.label, 0, 0, 1, 2)
        self.button_close = QtGui.QPushButton(Form)
```

```
        self.button_close.setObjectName("button_close")
        self.gridLayout.addWidget(self.button_close, 4, 3, 1, 1)
        self.input_start = QtGui.QLineEdit(Form)
        self.input_start.setObjectName("input_start")
        self.gridLayout.addWidget(self.input_start, 1, 1, 1, 3)
        self.output = QtGui.QTextBrowser(Form)
        self.output.setObjectName("output")
        self.gridLayout.addWidget(self.output, 3, 0, 1, 4)
        self.mode = QtGui.QCheckBox(Form)
        self.mode.setObjectName("mode")
        self.gridLayout.addWidget(self.mode, 0, 3, 1, 1)

        self.retranslateUi(Form)
        QtCore.QObject.connect(self.button_close, QtCore.SIGNAL("clicked()"), Form.close)
        QtCore.QMetaObject.connectSlotsByName(Form)

    def retranslateUi(self, Form):
        Form.setWindowTitle(QtGui.QApplication.translate("Form", "Form", None, QtGui.QApplication.
            UnicodeUTF8))
        self.button_submit.setText(QtGui.QApplication.translate("Form", "Submit", None, QtGui.
            QApplication.UnicodeUTF8))
        self.button_reset.setText(QtGui.QApplication.translate("Form", "Reset", None, QtGui.QApplication
            .UnicodeUTF8))
        self.label_3.setText(QtGui.QApplication.translate("Form", "Ziel:", None, QtGui.QApplication.
            UnicodeUTF8))
        self.label_2.setText(QtGui.QApplication.translate("Form", "Start:", None, QtGui.QApplication.
            UnicodeUTF8))
        self.label.setText(QtGui.QApplication.translate("Form", "Google Navigation", None, QtGui.
            QApplication.UnicodeUTF8))
        self.button_close.setText(QtGui.QApplication.translate("Form", "Close", None, QtGui.QApplication
            .UnicodeUTF8))
        self.mode.setText(QtGui.QApplication.translate("Form", "XML?", None, QtGui.QApplication.
            UnicodeUTF8))
```

2.2 controller

In dieser Klasse wird das `QWidget` initialisiert, indem der `controller` selber ein `QWidget` darstellt indem er davon erbt. Es wird die GUI dem `QWidget` hinzugfeügt und gestartet. Im `controller` gibt es 3 Methoden:

2.2.1 init()

Dadurch dass der controller von QWidget erbt, muss natürlich zuerst der Superkonstruktor aufgerufen werden. Anschließend wird ein Objekt von der view klasse erzeugt, welches mit `.setupUi()` ”gestartet” wird. Es wird noch ein Objektattribut mit einem Objekt vom Model gesetzt, und anschließend werden dem `button_submit` und dem `button_reset` 2 Methoden mit `connect()` hinzugefügt, und zwar `self.submit` und `self.reset`. Wichtig: Keine Klammern da es sich um einen sogenannten **callback** handelt.

```
def __init__(self):
    """
    When the controller gets initialized, the GUI gets set up, a model member is set and the buttons are
        connected
    to the respective functions

    Important: The Signal on the close Button which closes the windows already got connected
    to the close() slot in the QT-Designer:
    'QtCore.QObject.connect(self.button_close, QtCore.SIGNAL("clicked()"), Form.close)
    QtCore.QMetaObject.connectSlotsByName(Form)'
    """
    QWidget.__init__(self)
```

```
12      self.view = view.Ui_Form()
        self.view.setupUi(self)
14      self.model = model.Model()
        self.view.button_submit.clicked.connect(self.submit)
16      self.view.button_reset.clicked.connect(self.reset)
```

2.2.2 submit()

Ruft lediglich die `get_route` Methode des Models auf und setzt das GUI Element der View auf den output dieser Methode

```
    def submit(self):
2       """
        This method gets called when the Submit button gets pressed
4
        The get_route function from the model gets called, which returns the route and a status
6       :return: None
        """
8       # get_route gets called with the route-start, route-destionation and a paremeter which
        # determines whether the query should be done via XML or JSON
10      route = self.model.get_route(self.view.input_start.text(), self.view.input_ziel.text(), self.view.
            mode.isChecked())
        self.view.output.setText(route[0])
12      self.view.output_ok.setText(route[1])
```

2.2.3 reset()

Setzt alle Felder auf ihren Ursprungswert zurück.

```
    def reset(self):
2       """
        Gets called when the reset button is pressed
4
        Sets all input and output fields to empty strings and unchecks the box
6       :return:
        """
8       self.view.output.setText("")
        self.view.input_ziel.setText("")
10      self.view.input_start.setText("")
        self.view.output_ok.setText("")
12      self.view.mode.setChecked(False)
```

2.3 model

In dieser Klasse wird nun auf die tatsächliche API zugegriffen. Sie besteht lediglich aus der `get_route()` Methode, da es keine Objektattribute gibt

2.3.1 get_route()

Die Methode hat 3 Parameter: `start`, `destination` und `is_xml`. Der Parameter `is_xml` ist ein boolean, und ist dann true sobald eine xml Anfrage statt einer JSON anfrage gestellt werden soll.

Ob XML oder JSON, zuerst wird mit `requests.get()` die API aufgerufen und man bekommt eine `response` zurück. Diese response kann nun in die jeweilige Datenstruktur gebracht werden,

und es kann ein String geformt werden welcher zurückgegeben wird. Wichtig ist, dass an ein GUI-Element zurückgegeben wird welches HTML verarbeiten kann, dadurch können Dinge fett geschrieben werden mit <**b**>.

```
def get_route(self, start, destination, is_xml):
    """
    Creates a query to the google drive API, either parses the XML or JSON File
    :param start: the string which holds the information where the route is to be started
    :param destination: the string which holds the information where the route ends
    :param is_xml: boolean which determines whether a XML or JSON file is to be requested
    :return: A html-formatted string where the route is described and a status for the GUI
    """
    output = ""
    if is_xml:
        # GET is used because no information is posted, only received
        # important: the parameter mode=driving and language=de might seem unnecessary, but its
            important
        # that these are at the beginning to prevent injection-strings, for example 'Jaegerstrasse&mode=
            walking'
        response = requests.get(
        url="https://maps.googleapis.com/maps/api/directions/xml?mode=driving&language=de&origin=%s&
            destination=%s" % (
        start, destination))
        # get the root element of the XML structure
        xml_data = ET.fromstring(response.text)
        # get the status, which is the first child element
        status = xml_data[0].text
        # check for certain statuses, and return if certain where returned
        if status == 'NOT_FOUND' or status == 'INVALID_REQUEST' or status == 'ZERO_RESULTS':
            return [output, "Es wurde der Status %s zurueckgegeben, Eingabe ueberpruefen!" % (status)]
        else:
            status_ok = "Berechnung Ok!"

        # get the legend
        leg = xml_data[1][1]
        # from the legend extract the information how long the trip is in duration and distance
        output += "<p>Die Gesamtdauer betraegt <b>%s</b>, die Gesamtentfernung: <b>%s</b> </p>" % (
        leg.find('duration')[1].text, leg.find('distance')[1].text)

        # iterate through all steps in the legend
        for step in leg.findall('step'):
            # append the html_instruction, distance and duration information to the output string
            output += "%s, Entfernung %s, Dauer %s <br>" % (
            step.find('html_instructions').text, step.find('distance')[1].text, step.find('duration')
                [1].text)
        # return the route information and the status
        return [output, status_ok]
    else:
        # request json response
        response = requests.get(
        url="https://maps.googleapis.com/maps/api/directions/json?mode=driving&language=de&origin=%s&
            destination=%s" % (
        start, destination))
        # the requests module is able to convert json into a dictionary which is more easy to work with
        json_data = response.json()
        status = json_data["status"]
        # again check for certain statuses and return if a bad one was returned
        if status == 'NOT_FOUND' or status == 'INVALID_REQUEST' or status == 'ZERO_RESULTS':
            return [output, "Es wurde der Status %s zurueckgegeben, Eingabe ueberpruefen!" % (status)]
        else:
            status_ok = "Berechnung Ok!"

        # get the legend information
        legs = json_data["routes"][0]["legs"][0]
        # get all steps
        steps = legs["steps"]
        # also extract the total duration and distance for the route from the legend
        output += "<p>Die Gesamtdauer betraegt <b>%s</b>, die Gesamtentfernung: <b>%s</b> </p>" % (legs[
            "duration"]["text"], legs["distance"]["text"])

        # iterate through the steps
        for step in steps:
```

```
            # append the html_instructions, distance and duration to the output string
64          output += "%s, Entfernung %s, Dauer %s <br>" % (step["html_instructions"], step["distance"][
                "text"], step["duration"]["text"])
        # return the route string and the status
66      return [output, status_ok]
```

2.4 run

Diese Klasse startet die Application und das Widget

```
if __name__ == '__main__':
2   app = QApplication(sys.argv)
    c = controller.Controller()
4   c.show()

6   sys.exit(app.exec_())
```

Abbildungsverzeichnis