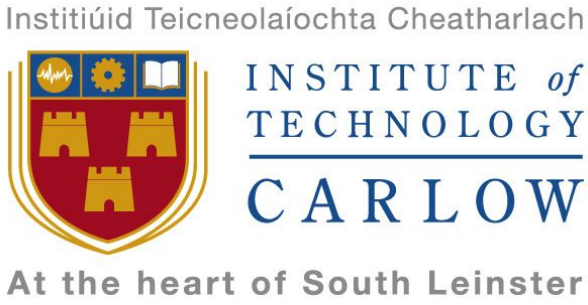
Technical Manual

ExamIT

AUTOMATIC TEST CORRECTION PLATFORM

4th year, Software Development project
Institute Of Technology Carlow



Roger Marciniak

Student Number: C00169733
Supervisor: Paul Barry

4th April, 2016

Table Of Contents

Installation Instructions

In order to install the ExamIT system and make it fully usable you need to follow the steps listed below:

Operating System

1.) Any Linux Desktop or Server distribution will work with ExamIT. Use your chosen distributions guide for installing it. If you had it installed already proceed to the next step.

Development Language Environment

2.) Enter this command into the terminal to check the Python version preinstalled:

```
> python -V
```

If your version is 3.5 or 3.6 you can proceed to the next step, in the case where your version is 2.5/2.7 or any other, you need to install a new version of Python 3.6 using your distributions package manager. When ready, proceed to the next step.

Necessary Packages

3.) Using your distributions package manager, install OpenCV. Check online for the exact package name in your distribution. It will often be ‘opencv’

4.) Using your package manager, install ‘pip’. It is a Python package manager that will be necessary for installing the Python packages required.

5.) Using your package manager, install ‘mongodb’.

PIP (Python Package Installer)

6.) Using the command:

```
> pip install <package>
```

Install the following packages:

- flask
- flask_admin
- flask login
- pymongo
- cv2
- numpy
- imutils
- fpdf
- wand

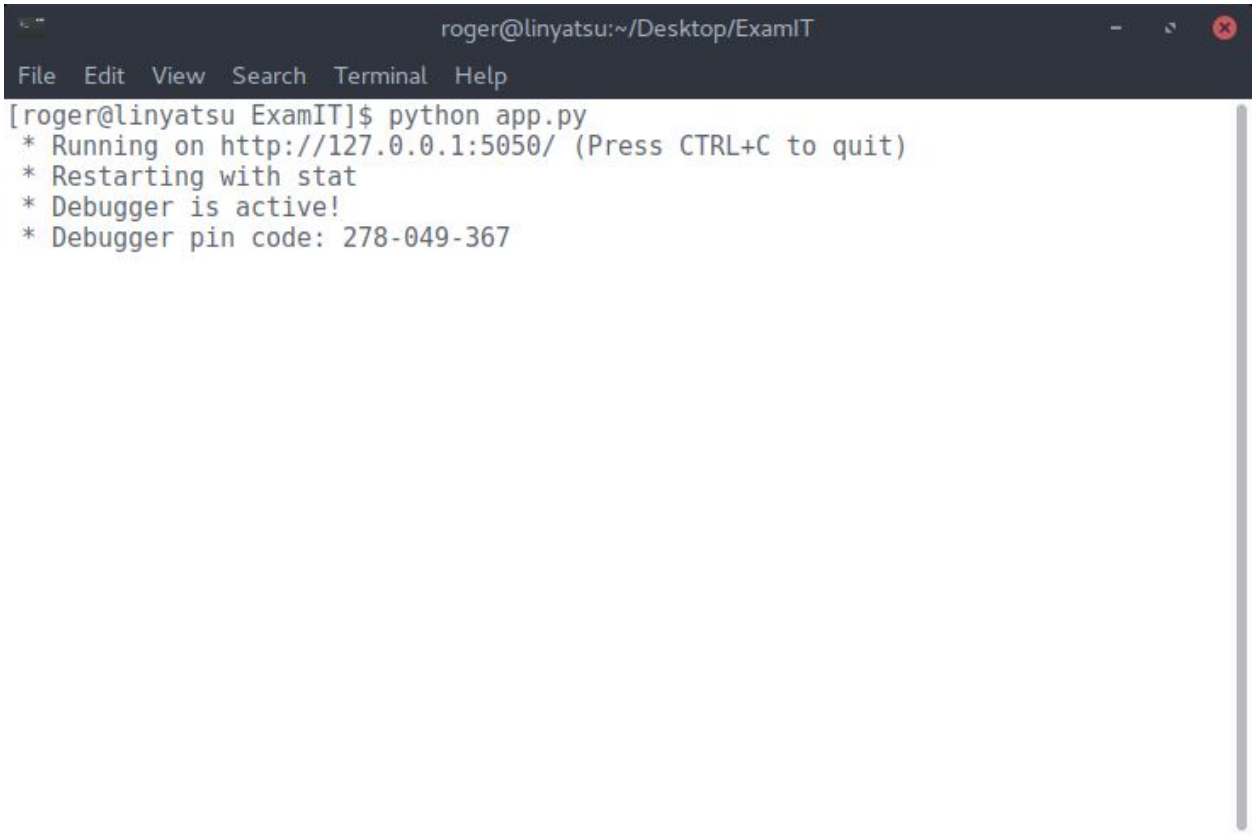
When finished, you are ready to run the server.

Starting the Server

7.) To start the server you need to go into the root folder of the application. From there, you need to open up a terminal window and type in the following command:

```
> python app.py
```

You should see the following in the terminal window now:



8.) To use the app simply open up a browser and paste in the following address:

http://127.0.0.1:5050

Or

http://localhost:5050

Code

App.py

```
import os

import flask_admin as admin
import flask_login as login
from flask import (Flask, redirect, render_template, request,
                   send_from_directory, url_for)

from user import User
from views import AdminIndexView, BlankView

# Create Flask application
app = Flask(__name__)
app.config['SECRET_KEY'] = os.urandom(32)
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024


# bower_components
@app.route('/bower_components/<path:path>')
def send_bower(path):
    return send_from_directory(os.path.join(app.root_path,
                                            'bower_components'), path)


@app.route('/dist/<path:path>')
def send_dist(path):
    return send_from_directory(os.path.join(app.root_path, 'dist'), path)


@app.route('/js/<path:path>')
def send_js(path):
    return send_from_directory(os.path.join(app.root_path, 'js'), path)


# Initialize flask-login
```

```
def init_login():
    login_manager = login.LoginManager()
    login_manager.init_app(app)

    # Create user loader function
    @login_manager.user_loader
    def load_user(user_id):
        return User.get(user_id)


# Flask views
@app.route('/')
def index():
    return render_template("sb-admin/redirect.html")


@app.route('/student/')
def student():
    return render_template("sb-admin/student.html")


@app.errorhandler(404)
def fourOhFour(error):
    return '404: You have wandered too far young adventurer!'


# Initialize flask-login
init_login()


# Create admin
admin = admin.Admin(app,
                    'ExamIT v0.8',
                    index_view=AdminIndexView())
# admin.add_view(BlankView(name='Blank', url='blank', endpoint='blank'))

if __name__ == '__main__':
    app.run(debug=True, port=5050)
```

4
4
5
4
6
4
7
4
8
4
9
5
0
5
1
5
2
5
3
5
4
5
5
5
6
5
7
5
8
5
9
6
0
6
1
6
2
6
3
6
4
6
5
6

6
6
7
6
8
6
9
7
0
7
1
7
2

Views.py

```
import shutil
import time
from random import sample

import flask_admin as admin
import flask_login as login
from flask import (flash, redirect, render_template, request, send_file,
                   session, url_for)
from flask_admin import expose, helpers
from pymongo import MongoClient

import corrector
import genPDF
import PDF2jpg
from loginform import LoginForm

ALLOWED_EXTENSIONS = set(['pdf'])

# prepares db
client = MongoClient()
db = client.examit
```

```
cats = db.cats
quests = db.quests  # question col
tests = db.tests
results = db.results


# Create customized index view class that handles login & registration
class AdminIndexView(admin.AdminIndexView):

    # converts answer keys from ABCDE form to 01234 needed for correction
    def letter2num(self, l):
        num = ord(l) - 65
        return num

    # takes a test cursor, returns a dict of answer key items
    def getAnswerKey(self, test):
        key = {}
        for i, question in enumerate(test['QUESTIONS']):
            key[i] = self.letter2num(question['KEY'])
        return key

    # ensures the file uploaded is an allowed file type
    def allowed_file(self, filename):
        print(filename)
        return '.' in filename and \
            filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

    def get_quests(self):
        columns = ["Question", "Category", "Key", "Creation Date", "Unique ID"]
        found = quests.find()
        rows = []
        for q in found:
            row = []
            row.append(q['QUESTION'])
            row.append(q['CATEGORY'])
            row.append(q['KEY'])
            row.append(q['CREATED'])
            row.append(q['_id'])
            rows.append(row)
        return (columns, rows)

    def get_cats(self):
        columns = ["Category", "Creation Date", "Unique ID"]
        found = cats.find()
```

```
        rows = []
        for c in found:
            row = []
            row.append(c['CATEGORY'])
            row.append(c['CREATED'])
            row.append(c['_id'])
            rows.append(row)
        return (columns, rows)

    def get_tests(self):
        columns = ["Test", "Lecturer", "Time Allowed", "Module",
                   "Questions", "Category", "Creation Date", "Unique ID"]
        found = tests.find()
        rows = []
        for t in found:
            row = []
            row.append(t['TITLE'])
            row.append(t['LECTURER'])
            row.append(t['TIME_ALLOWED'])
            row.append(t['MODULE'])
            row.append(t['QUESTCNT'])
            row.append(t['CATEGORY'])
            row.append(t['CREATED'])
            row.append(t['_id'])
            rows.append(row)
        return (columns, rows)

    def _tools(self):
        (qcols, qrows) = self.get_quests()
        self.qtable = {"questions": {"columns": qcols, "rows": qrows}}

        (ccols, crows) = self.get_cats()
        self.ctable = {"categories": {"columns": ccols, "rows": crows}}

        (tcols, trows) = self.get_tests()
        self.ttable = {"tests": {"columns": tcols, "rows": trows}}

    @expose('/')
    def index(self):
        if not login.current_user.is_authenticated:
            return redirect(url_for('.login_view'))

        self.header = "Welcome to ExamIT"
        return render_template('sb-admin/pages/start.html', admin_view=self)
```

```
@expose('/categories/', methods=['GET', 'POST'])
def cats(self):
    if not login.current_user.is_authenticated:
        return redirect(url_for('.login_view'))

    if request.method == 'POST':
        cat = request.form.get('category').strip()
        current_time = time.localtime()
        ctime = time.strftime('%a, %d %b %Y %H:%M:%S GMT',
                              current_time)

        cat_to_db = {"CATEGORY": cat,
                     "CREATED": ctime}

        db_checker = {"CATEGORY": cat}

        result = cats.replace_one(db_checker, cat_to_db, upsert=True)
        if result.modified_count == 1:
            flash('Category/already existed!',
                  category='info')
        else:
            flash('Category was successfully added!',
                  category='success')
        return redirect(url_for('.cats'))

    self._tools()
    self.header = "Categories"
    return render_template('sb-admin/pages/cats.html',
                           admin_view=self)

@expose('/questions/display/')
def questions(self):
    if not login.current_user.is_authenticated:
        return redirect(url_for('.login_view'))

    self._tools()
    self.header = "Questions"
    return render_template('sb-admin/pages/questions.html',
                           admin_view=self)

@expose('/questions/add/', methods=['GET', 'POST'])
def add_question(self):
    if not login.current_user.is_authenticated:
```

```
        return redirect(url_for('.login_view'))

    catlist = []
    found = cats.find()
    for c in found:
        catlist.append(c['CATEGORY'])
    catlist.sort()

    if request.method == 'POST':
        cat = request.form.get('category')
        qbody = " ".join(request.form.get('question').split())
        answers = []
        answers.extend([" ".join(request.form.get('A').split()),
                        " ".join(request.form.get('B').split()),
                        " ".join(request.form.get('C').split()),
                        " ".join(request.form.get('D').split()),
                        " ".join(request.form.get('E').split())])
        anskey = request.form.get('anskey')
        current_time = time.localtime()
        ctime = time.strftime('%a, %d %b %Y %H:%M:%S GMT',
                              current_time)

        quest_to_db = {"CATEGORY": cat,
                       "QUESTION": qbody,
                       "ANSWERS": answers,
                       "KEY": anskey,
                       "CREATED": ctime}

        db_checker = {"CATEGORY": cat,
                      "QUESTION": qbody}

        # if a question with the same category/question combo exists,
        # then it is replaced with the new one,
        # otherwise treated as a new question and added to the db
        result = quests.replace_one(db_checker, quest_to_db, upsert=True)
        if result.modified_count == 1:
            flash('Category/Answer combination existed and was updated!',
                  category='info')
        else:
            flash('Question was successfully added!',
                  category='success')
        return render_template('sb-admin/pages/qadd.html',
                               categs=catlist,
                               admin_view=self)
```

```
        self._tools()
        self.header = "Add Question"
        return render_template('sb-admin/pages/qadd.html',
                               categs=catlist,
                               admin_view=self)

    @expose('/tests/display/', methods=['GET', 'POST'])
    def tests(self):
        if not login.current_user.is_authenticated:
            return redirect(url_for('.login_view'))

        self._tools()
        self.header = "Tests"
        return render_template('sb-admin/pages/tests.html',
                               admin_view=self)

    @expose('/tests/generate/', methods=['GET', 'POST'])
    def gentest(self):
        if not login.current_user.is_authenticated:
            return redirect(url_for('.login_view'))

        catlist = []
        found = cats.find()
        for c in found:
            catlist.append(c['CATEGORY'])

        catlist.sort()
        self.header = "Generate Test"
        return render_template('sb-admin/pages/tgen.html',
                               categs=catlist,
                               admin_view=self)

    @expose('/tests/confirm/', methods=['GET', 'POST'])
    def gentest_conf(self):
        if not login.current_user.is_authenticated:
            return redirect(url_for('.login_view'))

        if request.method == 'POST':
            title = request.form.get('title').strip()
            timeal = request.form.get('timeallowed').strip()
            lecturer = request.form.get('lecturer').strip()
            module = request.form.get('module').strip()
            categ = request.form.get('category').strip()
```

```
        qamount = int(request.form.get('amount'))
        current_time = time.localtime()
        ctime = time.strftime('%a, %d %b %Y %H:%M:%S GMT', current_time)
        # draw n questions from category
        query = list(quests.find({"CATEGORY": categ}))
        if len(query) >= qamount:
            samp = sample(query, qamount)
            for doc in samp:
                doc['_id'] = str(doc['_id'])
            test = {"TITLE": title,
                    "TIME_ALLOWED": timeal,
                    "LECTURER": lecturer,
                    "MODULE": module,
                    "CATEGORY": categ,
                    "QUESTCNT": qamount,
                    "QUESTIONS": samp,
                    "CREATED": ctime}

            db_checker = {"TITLE": title,
                          "MODULE": module,
                          "CATEGORY": categ,
                          "QUESTCNT": qamount}

            session['test'] = test
            session['db_checker'] = db_checker
            return render_template('sb-admin/pages/tgenconf.html',
                                   test=test,
                                   admin_view=self)
        else:  # not enough questions in the category
            flash('Not enough questions in the selected category!',
                  category='danger')
            return redirect(url_for('admin.gentest'))

    self._tools()
    self.header = "Confirm Test Creation"
    return render_template('sb-admin/pages/tgenconf.html', admin_view=self)

@expose('/tests/confirmed/', methods=['GET', 'POST'])
def gentest_confd(self):
    if not login.current_user.is_authenticated:
        return redirect(url_for('.login_view'))

    if request.method == 'POST':
        result = tests.replace_one(session['db_checker'],
```

```
                                session['test'],
                                upsert=True)
            if result.modified_count == 1:
                flash('Test existed and was updated!',
                      category='info')
            else:
                flash('Test was successfully generated!',
                      category='success')
            return render_template('sb-admin/pages/tgenconfd.html',
                                   admin_view=self)

        self.header = "Generation Confirmation"
        return render_template('sb-admin/pages/tgenconfd.html',
                               admin_view=self)

    @expose('/tests/print/', methods=['GET', 'POST'])
    def printtest(self):
        if not login.current_user.is_authenticated:
            return redirect(url_for('.login_view'))

        found = tests.find()

        if request.method == 'POST':
            title = request.form.get('title')
            session['title'] = title
            tfound = tests.find_one({"TITLE": title})
            return render_template('sb-admin/pages/printtest.html',
                                   tests=found,
                                   selected=tfound,
                                   admin_view=self)

        self.header = "Print Test"
        return render_template('sb-admin/pages/printtest.html',
                               tests=found,
                               admin_view=self)

    @expose('/tests/print/printout/', methods=['GET', 'POST'])
    def printconfd(self):
        if not login.current_user.is_authenticated:
            return redirect(url_for('.login_view'))

        if request.method == 'POST':
            tfound = tests.find_one({"TITLE": session['title']})
            # generates the PDF
```

```
        pdf = genPDF.generate(tfound)  # "pdf/ptest.pdf"
        try:
            return send_file(pdf, attachment_filename='test.pdf')
        except Exception as e:
            return str(e)
    self.header = "Printout"
    return render_template('sb-admin/pages/printconfd.html',
                           admin_view=self)

@expose('/tests/correct/', methods=['GET', 'POST'])
def correcttest(self):
    if not login.current_user.is_authenticated:
        return redirect(url_for('.login_view'))

    found = tests.find()

    if request.method == 'POST':
        title = request.form.get('title')
        tfound = tests.find_one({"TITLE": title})
        # if request does not contain the file part
        if 'file' not in request.files:
            flash('No file was sent', category='danger')
            return redirect(request.url)
        file = request.files['file']
        # if user does not select file, browser will
        # submit an empty part without filename
        if file.filename == '':
            flash('No file was selected', category='danger')
            return redirect(request.url)
        # if file was selected but of the wrong type
        if file and not self.allowed_file(file.filename):
            flash('Please select a .pdf file', category='danger')
            return redirect(request.url)
        # if file was selected & is correct type
        if file and self.allowed_file(file.filename):
            as_jpeg = PDF2jpg.convert(file)
            # fetches the answer key corresponding to the test
            key = self.getAnswerKey(tfound)
            print(key)
            # corrects the test image using the answer key
            # returns (location, score, correct, AMOUNT)
            loc, corr, am, sc, flag = corrector.correct(as_jpeg, key)
            curr_time = time.localtime()
            ctime = time.strftime('%a, %d %b %Y %H:%M:%S GMT', curr_time)
```

```
            corrected = {"TEST": title,
                         "SCORE": sc,
                         "CORRECT": corr,
                         "AMOUNT": am,
                         "FLAG": flag,
                         "CREATED": ctime}
            # insert into the db
            result = results.insert_one(corrected)
            # obtain the MongoDB ObjectID in string form
            id = str(result.inserted_id) + '.png'
            # move and give a unique name to the test image for storage
            # destination = path to file
            destination = shutil.move(loc, 'results/' + id)
            # update the document with test file location
            results.update({'_id': result.inserted_id},
                           {"$set": {"HREF": id}},
                           upsert=False)
            print('NEW_FILE_SAVED={}'.format(destination))
            flash("File was corrected. Visit 'Test Results' to see scores",
                  category='success')
            return render_template('sb-admin/pages/uploadtest.html',
                                   tests=found,
                                   admin_view=self)

        self.header = "Correct Test"
        return render_template('sb-admin/pages/uploadtest.html',
                               tests=found,
                               admin_view=self)

    @expose('/tests/results/')
    def results(self):
        if not login.current_user.is_authenticated:
            return redirect(url_for('.login_view'))

        # returns a list of unique test titles
        tests = results.distinct('TEST')

        self.header = "Corrected Assessments"
        return render_template('sb-admin/pages/corrected.html',
                               tests=tests,
                               admin_view=self)

    @expose('/tests/results/<test>')
    def displayresult(self, test):
```

```
        if not login.current_user.is_authenticated:
            return redirect(url_for('.login_view'))

        res = results.find({"TEST": test})

        self.header = "Results: {}".format(test)
        return render_template('sb-admin/pages/listresults.html',
                               results=res,
                               admin_view=self)

    @expose('/tests/results/one/<img>')
    def showimg(self, img):
        if not login.current_user.is_authenticated:
            return redirect(url_for('.login_view'))

        location = 'results/' + img

        return send_file(location, attachment_filename=img)

    @expose('/login/', methods=('GET', 'POST'))
    def login_view(self):
        # handle user login
        form = LoginForm(request.form)
        if helpers.validate_form_on_submit(form):
            user = form.get_user()
            login.login_user(user)

        if login.current_user.is_authenticated:
            return redirect(url_for('.index'))
        self._template_args['form'] = form
        return render_template('sb-admin/pages/login.html',
                               form=form)

    @expose('/logout/')
    def logout_view(self):
        login.logout_user()
        return redirect(url_for('.index'))


class BlankView(admin.BaseView):

    @expose('/')
    def index(self):
```

```
        return render_template('sb-admin/pages/blank.html',
                               admin_view=self)
```

Corrector.py

```
import imutils
import numpy as np
from imutils import contours

import cv2


"""
This tool corrects the image test using the answer key supplied.
"""


def correct(test, key):
    # ANSWER_KEY maps question number to correct answer
    ANSWER_KEY = key

    # AMOUNT is the amount of questions
    AMOUNT = len(key)  # should be 5|10|15
    print("ANSWER_KEY_LENGTH={}".format(AMOUNT))
```

```
# gets sheet
image = cv2.imread(test)

# converts to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# blurrs the image slightly
blurred = cv2.GaussianBlur(gray, (5, 5), 0)

# finds edges ->
edged = cv2.Canny(blurred, 75, 200)

# applies Otsu's thresholding to binarize the grayscale
thresh = cv2.threshold(gray, 0, 255,
                       cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]

# finds and initialises a list of question circles
cnts = cv2.findContours(thresh.copy(),
                        cv2.RETR_EXTERNAL,
                        cv2.CHAIN_APPROX_NONE)
cnts = cnts[0] if imutils.is_cv2() else cnts[1]
questionCnts = []
print("CNT_AMOUNT={}".format(len(cnts)))

# for every question circle
for c in cnts:
    # create a box around a circle,
    # x, y being top-left coord, w, h its width & height
    (x, y, w, h) = cv2.boundingRect(c)

    # use it to create the aspect ratio
    ar = w / float(h)

    # populates a list with question circles
    # to become a question circle it needs to be:
    # wide & tall enough and have an apect ratio of +-1
    if w >= 50 and h >= 50 and ar >= 0.9 and ar <= 1.1:
        questionCnts.append(c)

# top -> bottom sort question circles
questionCnts = contours.sort_contours(questionCnts,
                                      method="top-to-bottom")[0]
print("QCNT_AMOUNT={}".format(len(questionCnts)))
```

```
    # after filtering, the question counts should be either of the three
    # otherwise, the algorithm has failed to locate the right contours
    if AMOUNT * 5 not in [25, 50, 75]:
        # if the algorithm cannot correct the test it will void it (True)
        # the lecturer can manually correct it later

        # sheet print score
        cv2.putText(image, "{}/{} {:.2f}% VOID".format(0, AMOUNT, 0),
                (50, 150), cv2.FONT_HERSHEY_SIMPLEX, 4.0, (0, 0, 255), 2)

        # the result image with annotations drawn on it
        location = 'results/temp/corrected.png'
        cv2.imwrite(location, image)
        # return format (location, correct, AMOUNT, score, FLAG)
        return (location, 0, AMOUNT, 0, True)

    # if expected number of contours, correcting begins
    # correct q's count
    correct = 0

    # each q has 5 (ABCDE) answers, loop over answers in groups of 5
    for (q, i) in enumerate(np.arange(0, len(questionCnts), 5)):
        # sort question circles group from left to right
        cnts = contours.sort_contours(questionCnts[i:i + 5])[0]
        # bubbled answer index
        bubbled = None

        # for circle in sorted group of 5, create a mask (only show one circle)
        for (j, c) in enumerate(cnts):
            # aka. return a new array of given shape, filled with zeros
            # in this case blank copy of our test
            mask = np.zeros(thresh.shape, dtype="uint8")
            # draw contours of the circle [c] onto our mask
            cv2.drawContours(mask, [c], -1, 255, -1)

            # place the mask on the sheet
            mask = cv2.bitwise_and(thresh, thresh, mask=mask)
            # count non-zero pixels inside the circle/mask
            total = cv2.countNonZero(mask)

            # if the current total contains a more non-zero pixels
            # then it is the answer (currently)
            if bubbled is None or total > bubbled[0]:
                # bubbled = (amount of nonzero pixels, which circle)
```

```
            bubbled = (total, j)

        # colour for the wrong answer contour (red)
        color = (0, 0, 255)

        # answer key at question circle q
        k = ANSWER_KEY[q]

        # is the answer correct according to the answer key? draw (green)
        if k == bubbled[1]:
            color = (0, 255, 0)
            correct += 1

        # draw the contour on the correct answer
        cv2.drawContours(image, [cnts[k]], -1, color, 3)

    # calculate score (OUT OF 5|10|15 Q's)
    score = (correct / AMOUNT) * 100
    print("SCORE={}/{} {:.2f}%".format(correct, AMOUNT, score))

    # sheet print score
    cv2.putText(image, "{}/{} {:.2f}%".format(correct, AMOUNT, score),
            (50, 150), cv2.FONT_HERSHEY_SIMPLEX, 4.0, (0, 0, 255), 2)

    # the result image with annotations drawn on it
    location = 'results/temp/corrected.png'
    cv2.imwrite(location, image)
    # the following do not need to be saved
    # saved for thr purpose of presenting the steps of the algorithm
    cv2.imwrite('results/temp/grayscale.png', gray)
    cv2.imwrite('results/temp/blurred.png', blurred)
    cv2.imwrite('results/temp/edged.png', edged)
    cv2.imwrite('results/temp/thresholded.png', thresh)
    return (location, correct, AMOUNT, score, False)
```

genPDF.py

```
from fpdf import FPDF


"""
This tool generates the test in a PDF format.
"""


class PDF(FPDF):

    def header(self):  # automatically called, do not call
        # Arial bold 8
        self.set_font('Arial', 'B', 8)
        # Title
        self.cell(190, 5, 'ExamIT - IT Carlow', 0, 0, align='C')
        # Line break
        self.ln(10)
```

```
    # Page footer
    def footer(self):  # automatically called, do not call
        # Position at 2cm from bottom
        self.set_y(-20)
        # Arial italic 8
        self.set_font('Arial', 'B', 12)
        # content
        # pdf.image(name='carlow.png', x = 170, y = 12, w = 30, h = 29)


# when called, generates a PDF document of the test
# the library doesn't allow for a cleaner way of doing this
def generate(test):
    pdf = PDF()
    pdf.set_margins(left=10, top=5)
    pdf.add_page()
    pdf.set_font('Arial', 'B', 14)
    pdf.cell(w=47, h=7, txt='C001', border='B', ln=0)
    pdf.cell(w=13)
    pdf.cell(w=70, h=7, txt=' ANSWERBOOK ', border=0, ln=1, align='C')
    pdf.set_font('Arial', 'B', 12)
    pdf.set_text_color(100)
    pdf.cell(w=47, h=7, txt='STUDENT NUMBER', border=0, ln=1, align='L')
    pdf.set_text_color(0)
    pdf.ln(2)
    pdf.cell(w=85, h=5, txt='Title: ', border=0, ln=0, align='R')
    pdf.cell(w=105, h=5, txt=test['TITLE'], border=0, ln=1, align='L')
    pdf.cell(w=85, h=5, txt='Time: ', border=0, ln=0, align='R')
    pdf.cell(w=105, h=5, txt=test['TIME_ALLOWED'], border=0, ln=1, align='L')
    pdf.cell(w=85, h=5, txt='Lecturer: ', border=0, ln=0, align='R')
    pdf.cell(w=105, h=5, txt=test['LECTURER'], border=0, ln=1, align='L')
    pdf.cell(w=85, h=5, txt='Module: ', border=0, ln=0, align='R')
    pdf.cell(w=105, h=5, txt=test['MODULE'], border=0, ln=1, align='L')
    pdf.cell(w=85, h=5, txt='Questions: ', border=0, ln=0, align='R')
    pdf.cell(w=105, h=5, txt=str(test['QUESTCNT']), border=0, ln=1, align='L')
    pdf.ln(5)
    pdf.cell(w=50, h=198)  # pushes question grid to the right

    if test['QUESTCNT'] == 5:
        type = 'pdf/shortwhite.png'
        height = 66
    elif test['QUESTCNT'] == 10:
        type = 'pdf/regularwhite.png'
        height = 132
```

```
else:  # questionAm == 15
    type = 'pdf/longwhite.png'
    height = 198

# question grid
pdf.image(name=type, x=None, y=None, w=80, h=height)

# adds a question sheet to the test
pdf.add_page()
pdf.set_font('Arial', 'B', 14)
pdf.cell(w=190, h=7, txt=' QUESTION SHEET ', border=0, ln=1, align='C')
pdf.set_font('Arial', 'B', 12)
pdf.ln(10)
letter = 'abcde'
for n, q in enumerate(test['QUESTIONS']):
    pdf.cell(w=10, h=5, txt="{}. ".format(n + 1), ln=0, align='R')
    pdf.cell(w=160, h=5, txt=str(q['QUESTION']), border=0, ln=1, align='L')
    pdf.ln(1)
    for l, a in zip(letter, q['ANSWERS']):
        pdf.cell(w=20, h=5, txt="{}. ".format(l), ln=0, align='C')
        pdf.cell(w=170, h=5, txt=a, border=0, ln=1, align='L')
    pdf.ln(3)
pdf.ln(20)
pdf.set_font('Arial', 'B', 13)
pdf.cell(w=190, h=7, txt='Test completion guidelines', ln=1, align='C')
pdf.set_font('Arial', 'B', 12)
pdf.ln(2)
guidelines = ['Only use pen inside the answer circles and for student no.',
              'Try not to go outside the circles when marking them',
              'Try to mark only one circle in each row',
              'The circle filled in more, will be considered your answer',
              'Only return the answerbook']
for n, guide in enumerate(guidelines):
    pdf.cell(w=20, h=5, txt='{}. '.format(n + 1), ln=0, align='R')
    pdf.cell(w=160, h=5, txt=guide, border=0, ln=1, align='L')

# save file, same name, will always be overwritten for space conservation
fname = "pdf/ptest.pdf"
pdf.output(fname)
return fname
```

PDF2Jpg.py

```
from wand.color import Color
from wand.image import Image


"""
This tool converts the .pdf to .jpg for use with OpenCV.
"""


def convert(test):
    # opens the pdf file stream with the correct resolution
    # so that the elements are qood quality
    with Image(file=test, format='pdf', resolution=300) as img:
        img.compression_quality = 100
        # sets background colour
        img.background_color = Color("white")
        # because the pdf contains transparency that becomes black
        # we need to remove it, the space will assume background colour
```

```
        img.alpha_channel = 'remove'
        # saved as a .jpg so OpenCV can use it
        img.format = 'jpeg'
        fn = 'test.jpg'
        img.save(filename=fn)
        # jpeg_bin = img.make_blob('jpeg')
        return fn
```

Loginform.py

```
from wtforms import fields, form, validators

from user import User


# Define login and registration forms (for flask-login)
class LoginForm(form.Form):
    username = fields.TextField(validators=[validators.required()])
    password = fields.PasswordField(validators=[validators.required()])

    def validate_login(self, field):
        user = self.get_user()

        if user is None:
            raise validators.ValidationError('Invalid user')

        if user.password != self.password.data:
            raise validators.ValidationError('Invalid password')
```

```
    def get_user(self):
        return User.get(self.username.data)
```

User.py

```
from flask_login import UserMixin


class UserNotFoundError(Exception):
    pass


class User(UserMixin):
    # TODO: needs to be dehardcoded if time allows for a proper user solution
    USERS = {
        # username: password
        'roger': 'troll0',
        'mary': 'jane'
    }

    id = None
    password = None

    def __init__(self, id):
        if id not in self.USERS:
```

```
            raise UserNotFoundError()
        self.id = id
        self.password = self.USERS[id]

    @classmethod
    def get(self_class, id):
        try:
            return self_class(id)
        except UserNotFoundError:
            return None
```

There are also HTML files with Jinja2 logic which might be worth having a look at at:

templates/sb-admin/pages