INŻYNIERIA OPROGRAMOWANIA

Projekt: Edytor wzorów matematycznych

Dokumentacja Projektu

Politechnika Śląska

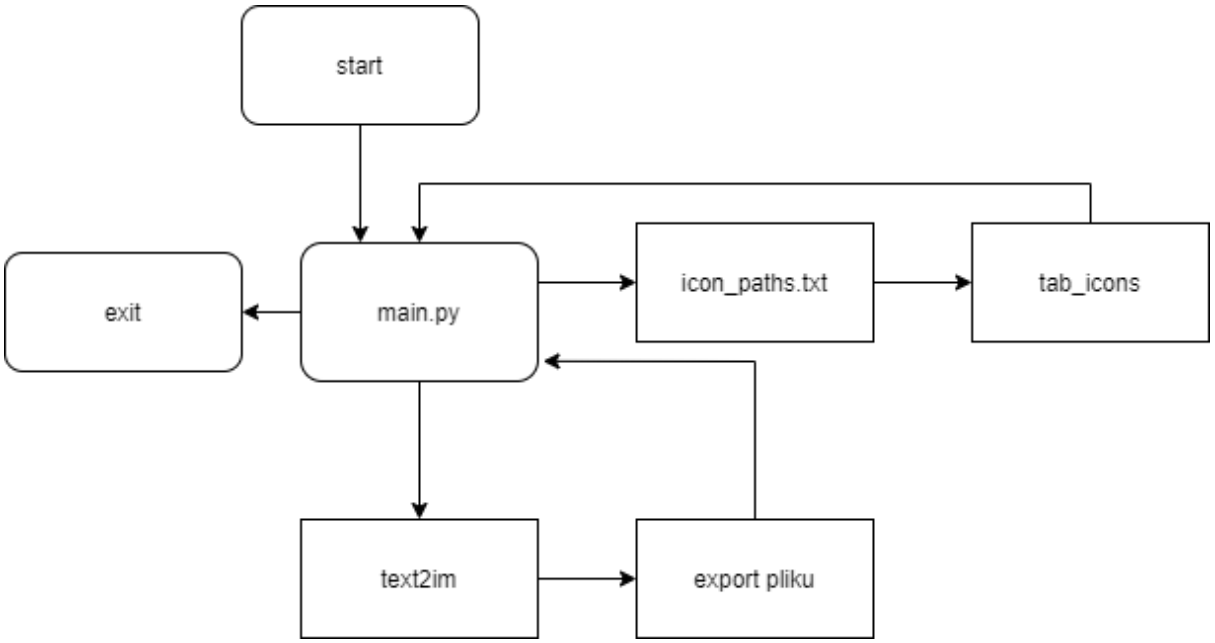
Skład zespołu: OSIECKI Adam
KIERAT Adam
GŁADYSZ Paweł

Opiekun zespołu:
dr inż. Zdzisław Sroczyński

Cel projektu
Celem projektu było stworzenie narzędzią, które pozwoli edytować wzory matematyczne w języku LaTeX za pomocą prostego interfejsu użytkownika oraz eksportować wygenerowane formuły do pliki graficznego.

Realizacja

Schemat graficzny struktury systemu



Język programowania
Początkowo planowane było użycie frameworka ReactJS, ale po zgłębieniu tematu przez naszą grupę projektową postawiliśmy na język Python w wersji 3.9. Wybraliśmy ten język ze względu na prostotę składni, oraz mnogość dostępnych pluginów, które ułatwiły nam pracę

Wykorzystane Pakiety
PyQt5 - to zbiór bibliotek Pythona tworzonych przez Riverbank Computing umożliwiających szybkie projektowanie interfejsów aplikacji okienkowych opartych o międzyplatformowy framework Qt. Dzięki niemu można w bardzo prosty sposób stworzyć interfejs użytkownika.

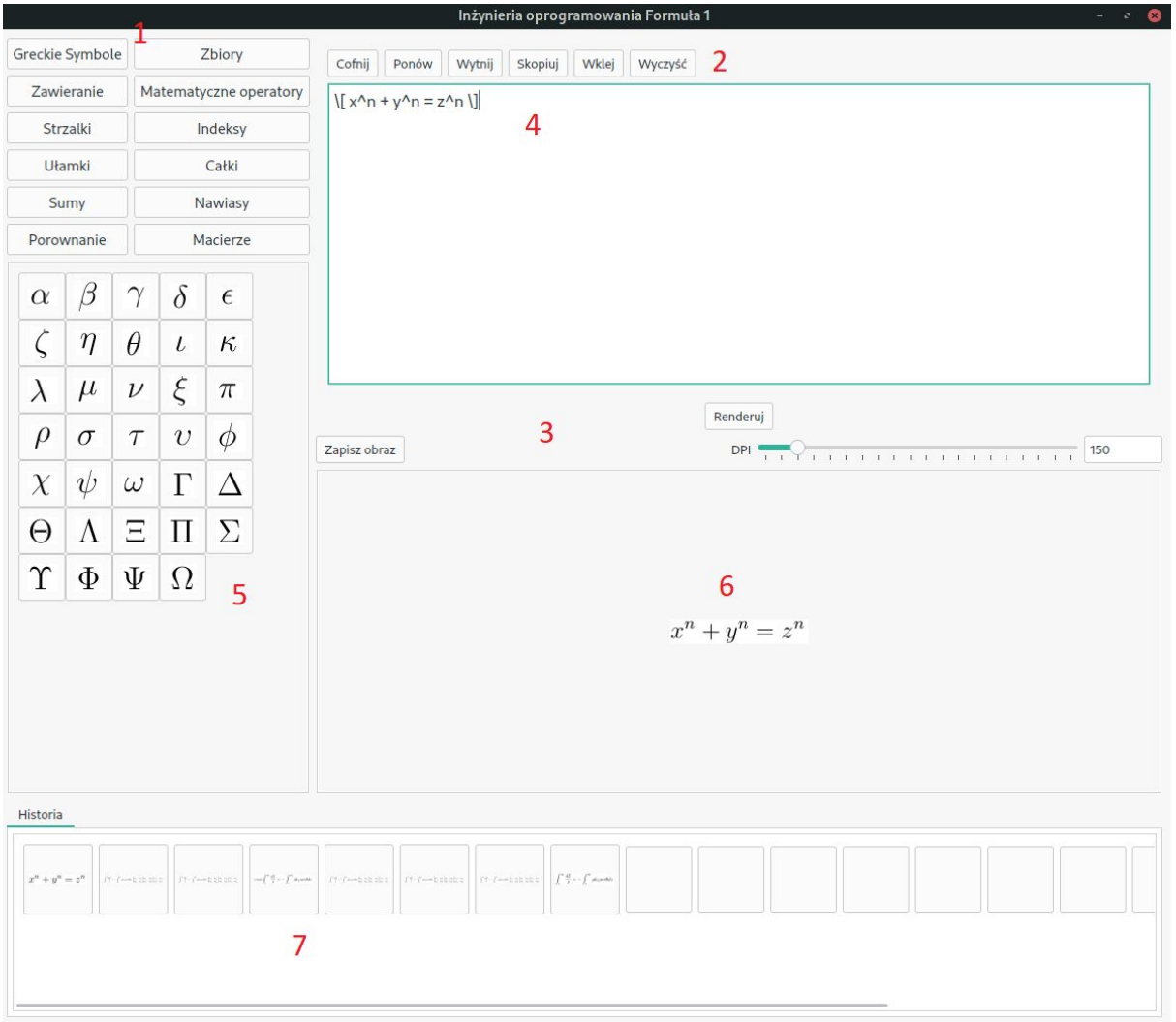
Tex2im - Jest to narzędzie, które konwertuje formuły LaTeX na grafikę o wysokiej rozdzielczości. Używana przez nas wersja jest wersją napisaną w pythonie Wersja. Przykładowy plik png wygenerowany za pomocą tex2im:

$$\psi_{tot}(x, -t_0, r) = \frac{1}{(2\pi)^2} \iint \tilde{\Psi}_{tot}\left(k_x, \frac{c}{2}\sqrt{k_x^2 + k_r^2}, r = 0\right)$$

Dokumentacja wewnętrzna

Instrukcja obsługi programu Formuła 1:

1- Przyciski dzięki którym zmieniamy kategorie dostępnych symboli na panelu numer 5
2- Przyciski do edycji panelu numer 4 np cofnij cofa zmiany.
3- Panel przycisków:
   a) Zapisz - zapisuje wygenerowaną formułę do pliku PDF
   b) Renderuj - Generuje z podanego kodu w panelu 4 graficzny wzór, który wyświetla się w panelu 6.
   c) Suwak DPI - Ustawia wielkość wygenerowanego wzoru
4- Panel w którym wpisujemy formułę
5- Panel dostępnych znaków, po kliknięciu kod generowany jest automatycznie w panelu numeru 4
6- Panel w którym generują się grafiki ze wzorami
7- Historia - panel w którym tworzy się historia wpisanych wzorów do których można wrócić.



Kod źródłowy:

main.py

Na początku pliki inicjujemy zmienne globalne, którymi będziemy operować w całym kodzie.

```
BUTTON_ICON_SIZE      = 16
THUMBNAIL_SIZE        = 36
MATRIX_THUMBNAIL_SIZE = 48
HIST_THUMBNAIL_SIZE   = 60
BUTTON_FONT_SIZE      = 11
TEXT_EDIT_FONT        = 13
ICON_NCOL             = 5
MATRIX_ICON_NCOL      = 4
TAB_NCOL              = 2
DEFAULT_RESO          = 150
CURRENT_DIR           = os.path.dirname(os.path.abspath(__file__))
ICON_META_FILE        = os.path.join(CURRENT_DIR, 'icon_paths.txt')
TEX2IM_CMD            = os.path.join(CURRENT_DIR,'tex2im/tex2im')
DEMO_IMG              = os.path.join(CURRENT_DIR,'tab_icons/demo.png')
DEMO_FORMULA=\
r'''\int_z^{\infty} \frac{dI}{I} = - \int_z^{\infty} \rho_k_{\lambda}sec \theta dz
'''
```

Ustawienie parametrów wyświetlania okna

```
def getHSpacer():
    h_spacer = QtWidgets.QSpacerItem(0,0,QtWidgets.QSizePolicy.Expanding,
            QtWidgets.QSizePolicy.Minimum)
    return h_spacer

def getVSpacer():
    v_spacer = QtWidgets.QSpacerItem(0,0,QtWidgets.QSizePolicy.Minimum,
            QtWidgets.QSizePolicy.Expanding)
    return v_spacer

def getVLine(parent):
    v_line = QtWidgets.QFrame(parent)
    v_line.setFrameShape(QtWidgets.QFrame.VLine)
    v_line.setFrameShadow(QtWidgets.QFrame.Sunken)
    return v_line

def getHLine(parent):
    h_line = QtWidgets.QFrame(parent)
    h_line.setFrameShape(QtWidgets.QFrame.HLine)
    h_line.setFrameShadow(QtWidgets.QFrame.Sunken)
    return h_line


def getMinSizePolicy():
    sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum,
            QtWidgets.QSizePolicy.Minimum)
    return sizePolicy

def getXMinYExpandSizePolicy():
    sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum,
            QtWidgets.QSizePolicy.Expanding)
    return sizePolicy
```

Wywołanie programu tex2im z odpowiednimi parametrami.

```
#-----Renderowanie poszczególnej formuły i zapis do img-----
def renderFormula(text,reso,outfile=None):
    if len(text)==0:
        return 2, None

    reso_str='%dx%d' %(reso,reso)

    #------------randomowy tmp nazwa pliku-----------
    tmp_tex_fd,tmp_tex_file=tempfile.mkstemp(suffix='.tex',prefix='tmp_latex_',
            dir='/tmp')
    if outfile is None:
        tmp_img_fd,tmp_img_file=tempfile.mkstemp(suffix='.png',prefix='tmp_latex_',
                dir='/tmp')
    else:
        tmp_img_file=outfile

    #------------Wywołanie tex2im aby renderować do img-------------
    try:
        tfile=os.fdopen(tmp_tex_fd,'w')
        tfile.write(text)
        tfile.close()
        cmd='%s -r %s -o %s %s' %(TEX2IM_CMD,reso_str,tmp_img_file,tmp_tex_file)
        proc=subprocess.Popen(cmd,stdout=subprocess.PIPE,stderr=subprocess.PIPE,shell=True)
        rec=proc.wait()
    except:
        rec=1
    finally:
        os.remove(tmp_tex_file)

    return rec,tmp_img_file
```

Klasa MainFrame jest odpowiedzialna za wyświetlanie wszystkich obiektów aplikacji.

```
class MainFrame(QtWidgets.QWidget):

    def __init__(self,thumbnail_meta_list):
        super(MainFrame,self).__init__()

        self.thumbnail_meta_list=thumbnail_meta_list
        self.thumbnail_btn_dict={}    # buttons for preset icons


        #-------------------Przyciski-------------------
        self.tab_btn_dict={}
        self.initUI()


    def getThumbnailFrame(self,icon_size,thumbnail_list,nrow=None,ncol=None):
        '''Get thumbnail frame for preset icons
        '''

        frame=QtWidgets.QWidget()
        grid=QtWidgets.QGridLayout()
        grid.setSpacing(0)

        #-------------------Wielkość grida-------------------
        nlist=len(thumbnail_list)
        if nrow is None and ncol is not None:
            nrow=max(1,int(math.ceil(nlist/float(ncol))))
        elif nrow is not None and ncol is None:
            ncol=max(1,int(math.ceil(nlist/float(nrow))))
        else:
            raise Exception("Error")

        #---------------Dodanie przycisków do grida---------------
        positions=[(ii,jj) for ii in range(nrow) for jj in range(ncol)]

        for posii,thumbnailii in zip(positions,thumbnail_list):
```

Funkcja ThumbnailFrame dodaje miniatury formuł.

```
def getThumbnailFrame(self,icon_size,thumbnail_list,nrow=None,ncol=None):
    frame=QtWidgets.QWidget()
    grid=QtWidgets.QGridLayout()
    grid.setSpacing(0)

    #------------------Wielkość grida------------------
    nlist=len(thumbnail_list)
    if nrow is None and ncol is not None:
        nrow=max(1,int(math.ceil(nlist/float(ncol))))
    elif nrow is not None and ncol is None:
        ncol=max(1,int(math.ceil(nlist/float(nrow))))
    else:
        raise Exception("Error")

    #----------------Dodanie przycisków do grida----------------
    positions=[(ii,jj) for ii in range(nrow) for jj in range(ncol)]

    for posii,thumbnailii in zip(positions,thumbnail_list):
        icon_textii,icon_img_pathii=thumbnailii
        icon_img_pathii=os.path.join(CURRENT_DIR, icon_img_pathii)
        buttonii=QtWidgets.QToolButton(self)
        buttonii.setIcon(QIcon(icon_img_pathii))
        buttonii.setIconSize(QtCore.QSize(icon_size,icon_size))
        buttonii.setStyleSheet('background-color:rgb(255,255,255)')
        grid.addWidget(buttonii,*posii)
        self.thumbnail_btn_dict[buttonii]=thumbnailii
        buttonii.clicked.connect(self.thumbnail_btn_click)

        grid.setRowStretch(posii[0],0)
        grid.setColumnStretch(posii[1],0)

    grid.addItem(getVSpacer(),nrow,0)
    grid.addItem(getHSpacer(),0,ncol)
```

getTexFrame tworzy pole testowe dla formuł matematycznych oraz dodaje przyciski edycji.

```
def getTextFrame(self):
    v_layout=QtWidgets.QVBoxLayout()
    h_layout=QtWidgets.QHBoxLayout()

    self.clip_board=QtWidgets.QApplication.clipboard()

    #------------Dodanie przycisków------------
    self.undo_button=QtWidgets.QToolButton()
    self.redo_button=QtWidgets.QToolButton()
    self.cut_button=QtWidgets.QToolButton()
    self.txt_copy_button=QtWidgets.QToolButton()
    self.paste_button=QtWidgets.QToolButton()

    buttons=[self.undo_button, self.redo_button, self.cut_button,
            self.txt_copy_button, self.paste_button]

    icon_names=['Cofnij','Ponów','Wytnij','Kopiuj','Wklej']

    for ii,nameii in enumerate(icon_names):
        buttonii=buttons[ii]
        buttonii.setToolButtonStyle(QtCore.Qt.ToolButtonTextUnderIcon)
        buttonii.setText(nameii)

    for bii in buttons:
        h_layout.addWidget(bii)

    h_layout.addItem(getHSpacer())
    v_layout.addLayout(h_layout)

    #------------------Dodanie text edit------------------
    self.text_box=QtWidgets.QTextEdit()
    font=QFont()
    font.setPointSize(TEXT_EDIT_FONT)
    self.text_box.setFont(font)
```

```
    #-----------------Connect przyciskow-----------------
    self.txt_copy_button.clicked.connect(self.text_box.copy)
    self.paste_button.clicked.connect(self.text_box.paste)
    self.cut_button.clicked.connect(self.text_box.cut)
    self.undo_button.clicked.connect(self.text_box.undo)
    self.redo_button.clicked.connect(self.text_box.redo)

    frame=QtWidgets.QFrame(self)
    frame.setFrameShape(QtWidgets.QFrame.StyledPanel)
    frame.setLayout(v_layout)

    return frame
```

getImageFrame - tworzy pole do wyświetlania wyrenderowanej formuły

```
def getImageFrame(self):
    v_layout=QtWidgets.QVBoxLayout()
    h_layout=QtWidgets.QHBoxLayout()
    #--------------Dodanie przyciskow--------------
    self.img_save_button=QtWidgets.QToolButton()
    self.img_save_button.setText('Zapisz')
    self.img_save_button.clicked.connect(self.img_save_btn_click)
    h_layout.addWidget(self.img_save_button)
    self.img_slider=QtWidgets.QSlider(QtCore.Qt.Horizontal,self)
    self.img_slider.setMinimum(50)
    self.img_slider.setMaximum(1000)
    self.img_slider.setTickInterval(50)
    self.img_slider.setTickPosition(QtWidgets.QSlider.TicksBelow)
    self.img_slider.setSingleStep(50)
    self.img_slider.setValue(DEFAULT_RESO)
    self.img_slider.valueChanged[int].connect(self.slider_change_value)
    h_layout.addStretch()
    self.slider_text=QtWidgets.QLineEdit(self)
    self.slider_text.setText(str(DEFAULT_RESO))
    self.slider_text.setFixedWidth(80)
    self.slider_text.setSizePolicy(getMinSizePolicy())
    self.slider_text.returnPressed.connect(self.dpi_box_change_value)
    scroll=QtWidgets.QScrollArea(self)
    scroll.setWidgetResizable(True)
    self.img_label=QtWidgets.QLabel()
    self.img_pixmap=QPixmap(DEMO_IMG)
    self.img_file_path=DEMO_IMG
    self.img_label.setPixmap(self.img_pixmap)
    scroll.setWidget(self.img_label)
    self.img_label.setAlignment(QtCore.Qt.AlignCenter)
    h_layout.addWidget(self.img_slider)
    h_layout.addWidget(self.slider_text)
    h_layout.setAlignment(QtCore.Qt.AlignTop)
    v_layout.addLayout(h_layout)
```

Oprogramowanie przycisku renderuj.

```
def render_btn_click(self):

    text=self.text_box.toPlainText()
    reso=self.img_slider.value()
    rec,tmp_img_file=renderFormula(text,reso,None)

    if rec==0:
        self.img_pixmap=QPixmap(tmp_img_file)
        self.img_label.setPixmap(self.img_pixmap)
        self.img_file_path=tmp_img_file

        return 0
```

Efekt wow!: Zapis pliku do pdf oraz slider

```
#Funkcjie img oraz slidera
def slider_change_value(self):
    v=self.img_slider.value()
    v2=v//50*50
    self.img_slider.setValue(v2)
    self.slider_text.setText(str(v2))

def dpi_box_change_value(self):
    v=int(self.slider_text.text())
    self.img_slider.setValue(v)



def img_save_btn_click(self):
    if self.img_label.pixmap() is not None:
        filename = QtWidgets.QFileDialog.getSaveFileName(self, 'Zapisz obraz .JPG!',os.getenv('HOME'), '*.jpg')
        filename_pdf = QtWidgets.QFileDialog.getSaveFileName(self, 'Zapisz plik .PDF!', os.getenv('HOME'), '*.pdf')
        if len(filename[0])>0:
            self.img_pixmap.save(filename[0])

            img_path = str(filename[0])
            pdf_path = str(filename_pdf[0])

            image = Image.open(img_path)

            pdf_bytes = img2pdf.convert(image.filename)
            file = open(pdf_path, "wb")
            file.write(pdf_bytes)
            image.close()
            file.close()
```

Stworzenie folderu history oraz zapis plików do tego folderu

```
#Stworzenie folderu history
if not os.path.exists(history_folder):
    os.makedirs(history_folder)
data_dict={}

#Zapis plików do folderu history
if len(self.main_frame.history_data_list)>0:
    hist_data=[]

    for textii,imgpathii,_ in self.main_frame.history_data_list:
        _,imgfileii=os.path.split(imgpathii)
        targetpathii=os.path.join(history_folder,imgfileii)
        if not os.path.exists(targetpathii):
            shutil.move(imgpathii,targetpathii)
        hist_data.append([textii,targetpathii])
    data_dict['history_data']=hist_data
#Zrzut danych do history.txt
history_file=os.path.join(history_folder,'history.txt')
if len(data_dict)>0:
    with open(history_file,'w') as fout:
        json.dump(data_dict,fout)
```