Custom LaTeX Report Generator Using PyLaTeX and PyQt

MANVI CHADHA

April 8, 2025

1 Introduction

In many engineering and research applications, it is often required to generate customized reports in PDF format based on user-selected data or components. This project presents a solution using Python where a graphical user interface (GUI) is developed with PyQt to let users select specific sections of a LaTeX report. The selected components are then dynamically compiled using PyLaTeX, and the resulting TeX file is sent to Overleaf for final PDF compilation. This approach allows for modular report generation without requiring a local LaTeX distribution.

2 Problem Statement

The objective of this project is to develop a GUI application in Python that lists various report components from a pre-defined LaTeX template. The user can check or uncheck the components they wish to include in the final report. Once selections are made, the application generates a custom LaTeX (`.tex`) file with only the chosen components. Since a local LaTeX distribution is not assumed, the generated file is then uploaded to an Overleaf project for final PDF compilation.

2.1 Project Goals

- Create a PyQt-based GUI to list the available report components.
- Allow users to select components to include in the report.
- Generate a custom `.tex` file using PyLaTeX based on the user's selection.
- Integrate the custom `.tex` file with a provided Overleaf template for PDF generation.
- Set up a GitHub repository to manage the source code and documentation.
- Prepare a ZIP file and a video screencast for project submission.

3 Methodology

The project is divided into several key phases:

1. **Environment Setup:** Install Python, PyQt5, and PyLaTeX. Create the project folder with all necessary files.
2. **GUI Development:** Using PyQt5, develop an application that lists report components (e.g., Title, Abstract, Introduction, Methodology, Results, Conclusion, References) as checkable items. A button triggers the report generation.
3. **Dynamic TeX Generation:** Implement functions with PyLaTeX to generate a custom LaTeX file that contains only the selected components.
4. **Overleaf Integration:** Since no local LaTeX installation is available, upload the generated `.tex` file to an Overleaf project (using a provided template) and compile the PDF report online.
5. **Version Control and Submission:** Create a GitHub repository for the project code, add `osdag-admin` as a collaborator, and prepare a ZIP archive. Also, record a screencast video demonstrating the application.

4 Design and Implementation

4.1 Input Parameters and Components

The application presents a list of report components:

- Title
- Abstract
- Introduction
- Methodology
- Results
- Conclusion
- References

The user can select one or more of these components using a PyQt5 list widget. Once selections are made, the Python script uses PyLaTeX to generate a corresponding `custom_report.tex` file.

4.2 TeX File Generation

Using PyLaTeX, the program conditionally adds sections to the document. For example, if the user selects "Title" and "Abstract," then only those parts appear in the generated file. The script appends the appropriate LaTeX commands to include the selected sections.

4.3 Overleaf Integration

The generated `custom_report.tex` file is then manually or automatically uploaded into a previously provided Overleaf project template. Overleaf compiles this file online to produce the final PDF output.

4.4 GitHub and Submission Requirements

The project's source code (including `main.py`, `requirements.txt`, and `README.md`) is maintained in a GitHub repository. The repository also includes:

- A ZIP file containing all project files.
- A link to the Overleaf project (with proper sharing settings).
- A video screencast demonstrating the application.

5 Sample Python Code

Below is an excerpt of the key sections of the Python code that implements the GUI and TeX generation:

```
import sys
import re
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QCheckBox, QPushButton, QLabel, QScrollArea

# Function to extract a section (or subsection) from the template file.
def extract_component(component, full_text):
    # This regex looks for \section or \subsection (possibly with a star) with the exact component title.
    # It then collects all text until the next \section or \subsection or end of document.
    pattern = r"(\\(?:sub)*section\*?\{" + re.escape(component) + r"\})(.*?)(?=\\(?:sub)*section|\\end\{document\})"
    match = re.search(pattern, full_text, re.DOTALL)
    if match:
        return match.group(0)
    else:
        # If section not found, return a comment.
        return f"% {component} not found in template.\n\n"

class ComponentSelector(QWidget):
    def __init__(self, components, template_file):
        super().__init__()
        self.setWindowTitle("LaTeX Report Customizer")
        self.setGeometry(100, 100, 450, 350)
        self.checkboxes = []
        self.template_file = template_file

        layout = QVBoxLayout()

        # Title Label
        label = QLabel("Select components to include in the report:")
        layout.addWidget(label)

        # Create a scrollable area for the list of components
        scroll_area = QScrollArea()
        scroll_area.setWidgetResizable(True)
        scroll_widget = QWidget()
        scroll_layout = QVBoxLayout(scroll_widget)

        # Add a checkbox for each component
```

```
        for comp in components:
            cb = QCheckBox(comp)
            scroll_layout.addWidget(cb)
            self.checkboxes.append(cb)

        scroll_area.setWidget(scroll_widget)
        layout.addWidget(scroll_area)

        # Button to generate the custom LaTeX file
        generate_button = QPushButton("Generate Custom LaTeX File")
        generate_button.clicked.connect(self.generate_latex)
        layout.addWidget(generate_button)

        self.setLayout(layout)

    def generate_latex(self):
        # Gather selected components
        selected = [cb.text() for cb in self.checkboxes if cb.isChecked()]
        if not selected:
            print("No components selected!")
            return

        print("Selected components:", selected)

        # Read the full template content
        try:
            with open(self.template_file, "r") as f:
                template_text = f.read()
        except Exception as e:
            print(f"Error reading {self.template_file}: {e}")
            return

        # Build the new document with selected components
        custom_body = ""
        for comp in selected:
            component_text = extract_component(comp, template_text)
            custom_body += component_text + "\n\n"

        # Construct the custom LaTeX document with header and footer from the template.
        # (For simplicity, we use the header up to \begin{document} and footer starting from \end{document}.)
        header_match = re.search(r"^(.*?\\begin\{document\})", template_text, re.DOTALL)
        footer_match = re.search(r"(\\end\{document\}.*)$", template_text, re.DOTALL)
```

```
80
81          if header_match and footer_match:
82              header = header_match.group(1)
83              footer = footer_match.group(1)
84          else:
85              # Fallback header/footer if regex fails
86              header = r"\documentclass{article}\begin{document}"
87              footer = r"\end{document}"
88
89          final_tex = header + "\n\n" + custom_body + "\n" + footer
90
91          # Write the custom LaTeX file
92          output_filename = "custom_report.tex"
93          with open(output_filename, "w") as f_out:
94              f_out.write(final_tex)
95          print(f"Custom LaTeX file '{output_filename}' created.")
96
97  if __name__ == "__main__":
98      app = QApplication(sys.argv)
99      # List of components to be extracted.
100     # You can update these strings based on the actual section titles in template.tex.
101     components = [
102         "Input Parameters",
103         "List of Input Section",
104         "Design Checks",
105         "Selected Member Data",
106         "Spacing Check",
107         "Member Check",
108         "Design Log"
109     ]
110     # Specify the template filename (make sure template.tex is in your project folder)
111     template_file = "template.tex"
112     window = ComponentSelector(components, template_file)
113     window.show()
114     sys.exit(app.exec_())
```

6 Flowchart

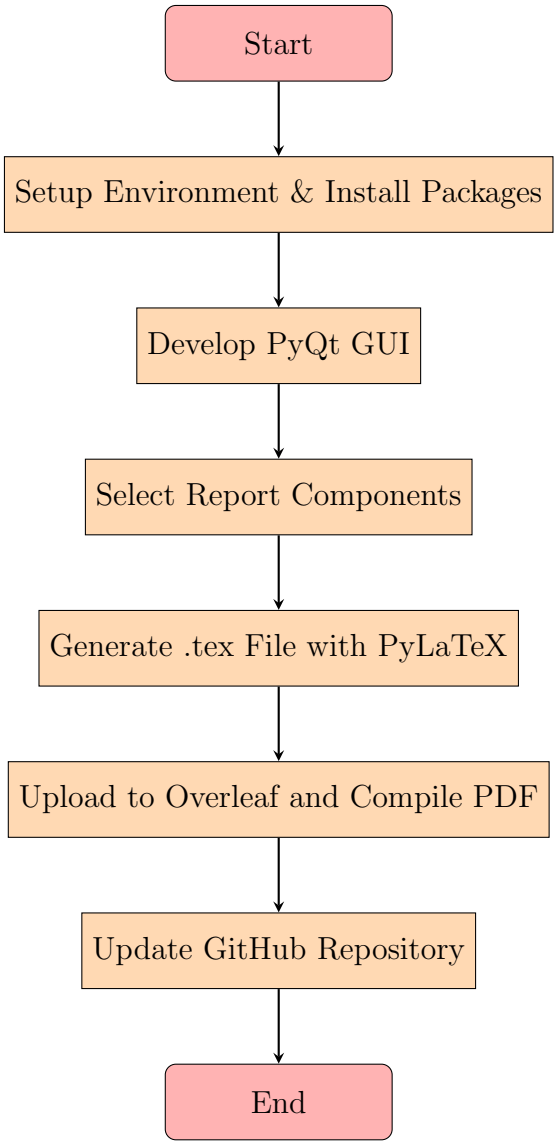

Figure 1: Flowchart of the Custom LaTeX Report Generator Process

7 Expected Outcomes

The final deliverables of the project include:

- A working Python GUI to select report components.
- A `custom_report.tex` file generated based on user selections.
- A final PDF report compiled via Overleaf.
- A GitHub repository containing all source code and documentation.
- A ZIP file and a video screencast that demonstrate the complete workflow.

8 Conclusion

This project demonstrates the design and implementation of a Custom LaTeX Report Generator that utilizes a Python GUI to allow users to dynamically select components of a report. The use of PyLaTeX for document generation and Overleaf for compilation provides an efficient solution for environments without a local LaTeX distribution. The complete integration with version control and submission guidelines ensures that the project meets both technical and academic requirements.

9 References

1. PyQt5 Documentation: *https://riverbankcomputing.com/software/pyqt/*
2. PyLaTeX Documentation: *https://jeltef.github.io/PyLaTeX/current/*
3. Overleaf LaTeX Templates: *https://www.overleaf.com/project/667fbd6d7512a3f904e4e9da*
4. GitHub: *https://github.com/OsdagScreeningTasks/BoltedLapJointAlgorithm/blob/main/*