МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

ОТЧЕТ

по лабораторной работе

по дисциплине

Шаблоны проектирования программного обеспечения

РУКОВОДИТЕЛЬ:

_____ Жевнерчук Д.В,
(подпись) (фамилия, и.,о.)

СТУДЕНТ:

_____ Халеев А.А.
_____ Маясов А.О.
(подпись) (фамилия, и.,о.)

21-ВМз
(шифр группы)

Работа защищена «___»_____

С оценкой _____

Нижний Новгород 2023

Задание
Вариант 6.

Реализуйте систему генерации шаблона html страницы по описанию:

1. Состава разделов (header, section1, … , section_n, footer);
2. Структуры каждой секции (количество блоков).

Добавьте опциональную возможность выделять определенный раздел, блок рамкой.

Сгенерированный html код выводите в консоль.

Проектное решение

В работе были реализованы следующие паттерны:

1) Порождающие:
- строитель (Builder);
- фабричный метод (Factory Method).

2) Структурные:
- адаптер (Adapter);
- легковес (Flyweight).

3) Поведенческие:
- стратегия (Strategy);
- состояние (State).

Обоснование выбора паттернов

Порождающие

1) Паттерн Строитель (Builder) используется для создания сложных объектов пошагово. Он разделяет процесс создания объекта от его представления, чтобы один и тот же процесс создания мог создавать разные представления.

В данном проекте класс HtmlBuilder реализует паттерн Строитель. Он используется для построения дерева HTML-элементов. Класс имеет методы add, create_content, to_previous и get_result для добавления элементов, создания содержимого, возврата к предыдущему состоянию и получения результата построения.

Класс HtmlDirector используется для управления процессом построения HTML-дерева. Он содержит экземпляр HtmlBuilder и определяет методы build_tree и get_html для создания и получения готового HTML-документа.

Основная идея паттерна Строитель заключается в том, чтобы разделить сложный процесс создания объекта на отдельные шаги и позволить клиенту контролировать этот процесс. В данном случае, HtmlBuilder является строителем, а HtmlDirector - директором, который управляет строителем и определяет последовательность шагов построения.

2) Паттерн Фабричный метод (Factory Method) используется для создания объектов, но делегирует фактическую логику создания подклассам. Он определяет интерфейс для создания объекта, но позволяет подклассам выбирать класс для создания.

Метод create_content класса HtmlBuilder реализует паттерн Фабричный метод. Он принимает value и specs в качестве аргументов и создает объект типа HtmlTag на основе значения value.

Если value находится в списке HTML_SINGLES, то создается экземпляр SingleTag.
Если value находится в списке HTML_DOUBLES, то создается экземпляр DoubleTag.
Если value находится в списке HTML_UNIQUES, то создается экземпляр UniqueTag.
В противном случае, создается экземпляр TagContent. Возвращается созданный объект.

Непосредственное создание объекта происходит в методе add класса HtmlBuilder, данный метод принимает на вход параметры создаваемого тега и использует фабричный метод для создания объекта на основе предоставленных параметров.

Таким образом, методы add и create_content класса HtmlBuilder предоставляют гибкость в создании и добавлении различных типов элементов в HTML-дерево, основываясь на значениях их аргументов.

Структурные

1) Паттерн Легковес (Flyweight) используется для эффективного разделения объектов на общие и неизменяемые (внутренние) и индивидуальные и изменяемые (внешние) части. Он позволяет экономить память и улучшать производительность, разделяя общие данные между несколькими объектами.

В данном проекте класс UniqueTag реализует паттерн Легковес. Он используется для представления уникальных HTML-тегов, которые могут быть разными объектами, но имеют общие данные и поведение.
Класс UniqueTag имеет поле __instances, которое является словарем, хранящим экземпляры UniqueTag по их именам. Это позволяет обеспечить, что каждый тег имеет только один экземпляр в системе. Когда создается новый тег, проверяется, есть ли уже экземпляр с таким именем в словаре. Если есть, то возвращается существующий экземпляр. Если нет, то создается новый экземпляр и добавляется в словарь.

Суть паттерна Легковес заключается в том, чтобы разделить данные, которые могут быть общими для нескольких объектов, и хранить их в отдельном месте (в данном случае, в словаре __instances). Это позволяет уменьшить потребление памяти и повысить эффективность использования объектов.

Назначение паттерна Легковес состоит в оптимизации работы с большим количеством мелких объектов, снижении потребления памяти и улучшении производительности системы.

2) Паттерн Адаптер используется для преобразования интерфейса одного класса в интерфейс другого класса, с целью обеспечения их совместной работы без изменения исходного кода.

Класс HtmlAdapter представляет реализацию паттерна Адаптер (Adapter). HtmlAdapter обеспечивает адаптацию интерфейса класса HtmlWidget к интерфейсу класса HtmlDirector, чтобы они могли работать вместе.

Конструктор класса HtmlAdapter создает экземпляр класса HtmlDirector, который будет использоваться для построения HTML-страницы.

Метод build_page принимает объект типа HtmlWidget и использует его свойства (цвет, выравнивание, наличие границ) для создания стиля и передачи его в HtmlDirector, чтобы построить дерево HTML-элементов. Метод get_html возвращает HTML-код, сгенерированный HtmlDirector.

Метод create_style является вспомогательным методом, который создает стиль на основе переданных параметров (цвет, выравнивание, наличие границ) и возвращает объект типа Style.
Таким образом, класс HtmlAdapter выполняет функцию адаптера, позволяя классу HtmlWidget работать с классом HtmlDirector, обеспечивая их взаимодействие без изменения исходного кода.

Поведенческие

1) Классы Strategy, Node, и Leaf вместе с HtmlBuilder реализуют паттерн Стратегия (Strategy).

Этот паттерн используется для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости. Он позволяет изменять поведение объекта во время выполнения программы.

В данном проекте:
Strategy является абстрактным базовым классом, который определяет общий интерфейс для всех стратегий. В данном случае он определяет абстрактный метод add.

Node и Leaf являются конкретными стратегиями, реализующими метод add. Node добавляет новый узел DoubleTag в дерево HTML, а Leaf добавляет одинарный тег HTML.

HtmlBuilder используется для создания HTML. Он содержит метод add, который принимает стратегию в качестве аргумента и использует ее для добавления элементов в дерево HTML. Это позволяет изменить поведение метода add во время выполнения, выбирая различные стратегии.

Таким образом, этот паттерн позволяет изменять алгоритм добавления элементов в дерево HTML на лету, без изменения исходного кода класса HtmlBuilder.

2) Паттерн Состояние (State) позволяет объекту изменять свое поведение в зависимости от своего внутреннего состояния. Это достигается путем делегирования ответственности за определенное поведение состоянию, которое может быть изменено во время выполнения.

В нашем случае, паттерн Состояние реализован внутри класса HtmlBuilder:

Метод to_previous позволяет объекту HtmlBuilder возвращаться к предыдущему состоянию. Состояния хранятся в стеке (node_stack), который используется для восстановления предыдущего состояния. Каждый вызов to_previous приводит к возврату к предыдущему состоянию путем извлечения последнего элемента из стека.

branch_ptr представляет текущее состояние объекта HtmlBuilder. Он изменяется при вызове метода add и восстанавливается при вызове to_previous.

Таким образом, в данном случае паттерн Состояние позволяет объекту HtmlBuilder возвращаться к предыдущему состоянию при построении дерева HTML, что дает большую гибкость при создании сложных структур HTML.

Класс HtmlDirector управляет процессом построения HTML, вызывая методы add и to_previous объекта HtmlBuilder для построения сложного HTML-дерева. Этот класс также демонстрирует, как можно использовать объект HtmlBuilder для создания различных HTML-структур, зависящих от состояния объекта.

Влияние шаблонов проектирования на расширяемость и межмодульную связность системы

1) Строитель (Builder): Паттерн Строитель, реализованный в классе HtmlBuilder, облегчает создание сложных объектов, делая код более понятным и поддерживаемым. Это улучшает расширяемость, поскольку можно легко изменять структуру HTML, меняя только код в классе HtmlBuilder.

2) Фабричный метод (Factory Method): Этот паттерн используется в методе create_content класса HtmlBuilder. Он облегчает создание объектов, делегируя эту задачу специализированным методам. Это уменьшает связанность, так как другие классы не нуждаются в знании о конкретных классах объектов HTML-тегов.

3) Легковес (Flyweight): Класс UniqueTag реализует паттерн Легковес. Этот паттерн сокращает использование памяти путем общего использования объектов для одинаковых данных. Это улучшает производительность, особенно при работе с большим количеством объектов.

4) Адаптер (Adapter): Класс HtmlAdapter реализует паттерн Адаптер. Этот паттерн позволяет классам с несовместимыми интерфейсами работать вместе. Это улучшает расширяемость, позволяя использовать существующий класс в новом контексте без необходимости изменять его код.

5) Стратегия (Strategy): Этот паттерн повышает расширяемость, позволяя легко добавлять новые стратегии без изменения существующего кода. Это также уменьшает связанность, так как классы, использующие стратегии, не нуждаются в знании о конкретных стратегиях.

6) Состояние (State): С помощью паттерна Состояние, состояние объекта управляется внутри самого объекта, что уменьшает зависимость между различными модулями системы.

Все эти шаблоны проектирования вместе улучшают расширяемость и уменьшают межмодульную связанность системы, делая код более модульным, упрощая добавление нового функционала и улучшая производительность. Применение этих шаблонов делает код более чистым, легким для понимания и поддержки, и упрощает процесс тестирования, поскольку каждый модуль или класс можно тестировать независимо.

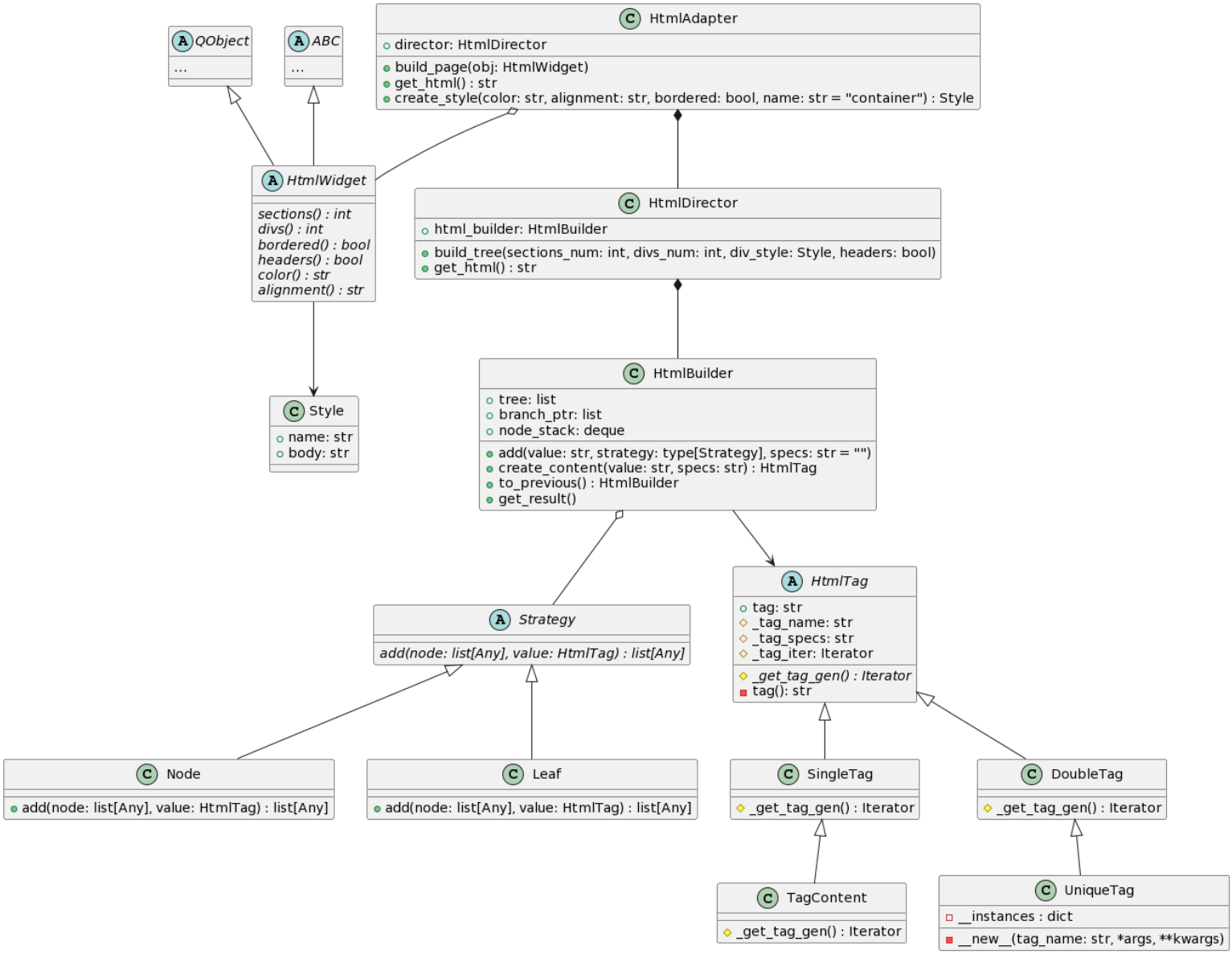


Рис. 1 – Диаграмма классов

Приложение 1

Программный код

main.py:

```
from sys import argv

from PySide6.QtGui import QIcon
from PySide6.QtWidgets import QApplication

from source.app import MainWindow


def main():
    app = QApplication(argv)
    window = MainWindow()
    window.setWindowIcon(QIcon("designed_ui/icons/logo.png"))
    window.show()
    app.exec()


if __name__ == '__main__':
    main()
```

app.py:

```
import os
from enum import IntEnum
from fnmatch import fnmatch

from PySide6.QtCore import Signal
from PySide6.QtWebEngineWidgets import QWebEngineView
from PySide6.QtWidgets import QInputDialog, QMainWindow, QStackedLayout, QTextEdit

from designed_ui.designed_interface import Ui_MainWindow
from source.html_utils import HtmlAdapter, HtmlWidget


class MainWindow(QMainWindow, Ui_MainWindow, HtmlWidget):
    temp_saved = Signal()
    temp_loaded = Signal()
    temp_generated = Signal()

    class Mode(IntEnum):
        TEXT = 0
        HTML = 1

    def __init__(self):
        super().__init__()
        self.setupUi(self)
        self.current_dir = os.path.dirname(os.path.abspath(__file__))

        self.update_templates()
        self.text_edit = QTextEdit()
        self.html_render = QWebEngineView()
        self.html_render.reload()
        self.text_lay = QStackedLayout()
        self.text_lay.setStackingMode(QStackedLayout.StackOne)
        self.text_lay.addWidget(self.text_edit)
        self.text_lay.addWidget(self.html_render)
        self.text_view.setLayout(self.text_lay)
        # ---------------------------- connections-------------------------------------
        self.text_btn.clicked.connect(self.show_text)
        self.render_btn.clicked.connect(self.show_html)
        self.generate_btn.clicked.connect(self.generate)
        self.clear_btn.clicked.connect(self.text_edit.clear)
        self.clear_btn.clicked.connect(self.show_text)
        self.save_btn.clicked.connect(self.save_modal)
        self.load_btn.clicked.connect(self.load_template)
        self.temp_saved.connect(self.update_templates)
        self.temp_loaded.connect(self.show_text)
        self.temp_generated.connect(self.show_text)
        # ------------------------------------------------------------------------------

    @property
    def sections(self) -> int:
        return self.sections_spin.value()

    @property
    def divs(self) -> int:
        return self.divs_spin.value()

    @property
    def bordered(self) -> bool:
        return self.bordered_check.isChecked()

    @property
    def color(self) -> str:
        return self.text_color_spin.currentText()

    @property
    def alignment(self) -> str:
        return self.alignment_spin.currentText()

    @property
    def headers(self) -> bool:
        return self.headers_check.isChecked()
```

```
def generate(self) -> None:
    html_agent = HtmlAdapter()
    html_agent.build_page(self)
    html_text = html_agent.get_html()
    self.text_edit.setPlainText(html_text)
    self.temp_generated.emit()

def show_text(self) -> None:
    self.text_lay.setCurrentIndex(self.Mode.TEXT)


def show_html(self) -> None:
    self.html_render.setHtml(self.text_edit.toPlainText())
    self.text_lay.setCurrentIndex(self.Mode.HTML)

def save_modal(self) -> None:
    modal = QInputDialog()
    modal.setWindowTitle("HTML template saving")
    modal.setLabelText("Enter filename:")
    modal.exec()
    if modal.accepted:
        self.save_template(modal.textValue())

def save_template(self, filename) -> None:
    with open(f"templates/{filename.rstrip('.html')}.html", "w") as output:
        output.write(self.text_edit.toPlainText())
    self.temp_saved.emit()

def update_templates(self) -> None:
    self.templates.clear()
    self.templates.addItems(self.get_templates())

@staticmethod
def get_templates() -> tuple[str, ...]:
    return tuple(entry for entry in os.listdir("templates")
                 if fnmatch(entry, "*.html"))

def load_template(self) -> None:
    template_name = str(self.templates.currentText())
    with open(f"templates/{template_name}", "r") as temp:
        template_text = temp.read()
        self.text_edit.setPlainText(template_text)
        self.temp_loaded.emit()
```

html_utils.py:

```python
from abc import ABC, ABCMeta, abstractmethod
from collections import deque
from dataclasses import dataclass
from typing import Any, Union

from PySide6.QtCore import QObject

from source.html_tags import HtmlTag, DoubleTag, SingleTag, UniqueTag, TagContent, HTML_SINGLES, \
HTML_DOUBLES, HTML_UNIQUES


class _ABCQObjectMeta(type(QObject), ABCMeta): ...


class HtmlWidget(QObject, ABC, metaclass=_ABCQObjectMeta):
    @property
    @abstractmethod
    def sections(self) -> int: ...

    @property
    @abstractmethod
    def divs(self) -> int: ...

    @property
    @abstractmethod
    def bordered(self) -> bool: ...

    @property
    @abstractmethod
    def headers(self) -> bool: ...

    @property
    @abstractmethod
    def color(self) -> str: ...

    @property
    @abstractmethod
    def alignment(self) -> str: ...


@dataclass
class Style:
    name: str
    body: str


class HtmlAdapter:
    """ This class is used to implement the "adapter" pattern"""

    def __init__(self) -> None:
        self.director = HtmlDirector()

    def build_page(self, obj: HtmlWidget) -> None:
        style = self.create_style(color=obj.color, alignment=obj.alignment, bordered=obj.bordered)
        self.director.build_tree(sections_num=obj.sections, divs_num=obj.divs, div_style=style,
                                 headers=obj.headers)

    def get_html(self) -> str:
        return self.director.get_html()

    @staticmethod
    def create_style(*, color: str, alignment: str, bordered: bool, name: str = "container") -> "Style":
        style = f".{name} {{"
        if color:
            style += f"color: {color}; "
        if alignment:
            style += f"text-align: {alignment}; "
        if bordered:
            style += "border: 1px solid black; "
        style += "}"
        return Style(name, style)
```

```python
class Strategy(ABC):
    """ This class is a head of hierarchy, that provide to implement a part of the strategy pattern"""

    @staticmethod
    @abstractmethod
    def add(node: list[Any], value: HtmlTag) -> list[Any]:
        pass


class Node(Strategy):
    @staticmethod
    def add(node: list[Any], value: HtmlTag) -> list[Any]:
        if isinstance(value, DoubleTag):
            new_node = [value]
            node.append(new_node)
            return new_node
        else:
            return Leaf.add(node, value)


class Leaf(Strategy):
    @staticmethod
    def add(node: list[Any], value: HtmlTag) -> list[Any]:
        node.append(value)
        return node


class HtmlBuilder:
    """ This class is used to implement those patterns:
        - factory method
        - builder
        - strategy (a part of)
        - state
    """

    def __init__(self) -> None:
        self.tree: list[Any] = list()
        self.branch_ptr: list[Any] = self.tree
        self.node_stack: deque = deque()

    def add(self, value: str, *, strategy: type[Strategy], specs: str = "") -> None:
        """ Implements a part of "strategy" pattern"""
        self.node_stack.append(self.branch_ptr)
        content = self.create_content(value, specs)  # a part of "factory method" pattern
        self.branch_ptr = strategy.add(self.branch_ptr, content)

    @staticmethod
    def create_content(value: str, specs: str) -> HtmlTag:
        """ Implements "factory method" pattern"""
        if value in HTML_SINGLES:
            return SingleTag(value, tag_specs=specs)
        elif value in HTML_DOUBLES:
            return DoubleTag(value, tag_specs=specs)
        elif value in HTML_UNIQUES:
            return UniqueTag(value, tag_specs=specs)
        else:
            return TagContent(value)

    def to_previous(self) -> "HtmlBuilder":
        """ Implements "state" pattern - we can return to previous state using node callstack"""
        if self.node_stack:
            last_branch = self.node_stack.pop()
            if last_branch is self.branch_ptr:
                self.to_previous()  # recursive call
            else:
                self.branch_ptr = last_branch
            if not self.node_stack:
                self.node_stack.append(self.branch_ptr)
        return self

    def get_result(self):
        return self.tree
```

```python
class HtmlDirector:
    def __init__(self) -> None:
        self.html_builder = HtmlBuilder()

    def build_tree(self, *, sections_num: int, divs_num: int, div_style: Style, headers: bool) -> None:
        self.html_builder.add("!DOCTYPE", strategy=Leaf, specs="html")
        self.html_builder.add("html", strategy=Node)
        self.html_builder.add("head", strategy=Node)
        self.html_builder.add("style", strategy=Node, specs='type="text/css"')
        self.html_builder.add(div_style.body, strategy=Leaf)
        self.html_builder.to_previous().to_previous()
        self.html_builder.add("body", strategy=Node)
        self.html_builder.add("header", strategy=Leaf)
        self.html_builder.add("main", strategy=Node)
        for s_num in range(1, sections_num + 1):
            self.html_builder.add("section", strategy=Node)
            for d_num in range(1, divs_num + 1):
                self.html_builder.add("div", strategy=Node, specs=f"class={div_style.name}")
                if headers:
                    h_level = d_num if d_num <= 6 else 6
                    self.html_builder.add(f"h{h_level}", strategy=Node)
                    self.html_builder.add(f"section-{s_num} div-{d_num} message", strategy=Leaf)
                    self.html_builder.to_previous()
                else:
                    self.html_builder.add(f"section-{s_num} div-{d_num} message", strategy=Leaf)
                self.html_builder.to_previous()
            self.html_builder.to_previous()
        self.html_builder.to_previous()
        self.html_builder.add("footer", strategy=Leaf)

    def get_html(self) -> str:
        res = ""
        space_tab = "    "

        def tree_traversal(node: list[Union[HtmlTag, Any]], level: int = -1) -> None:
            """ It's a function that traverses the html tree recursively to write result to nonlocal
"res" variable """
            nonlocal res
            first_tag = node[0]
            res += space_tab * level + first_tag.tag + '\n'
            for child in node[1:]:
                match child:
                    case SingleTag():
                        res += space_tab * (level + 1) + child.tag + '\n'
                    case DoubleTag():
                        res += space_tab * (level + 1) + child.tag + child.tag + '\n'
                    case list():
                        tree_traversal(child, level + 1)  # recursive case
            if isinstance(first_tag, DoubleTag):
                res += space_tab * level + first_tag.tag + '\n'

        tree_traversal(self.html_builder.tree)
        return res
```

html_tags.py:

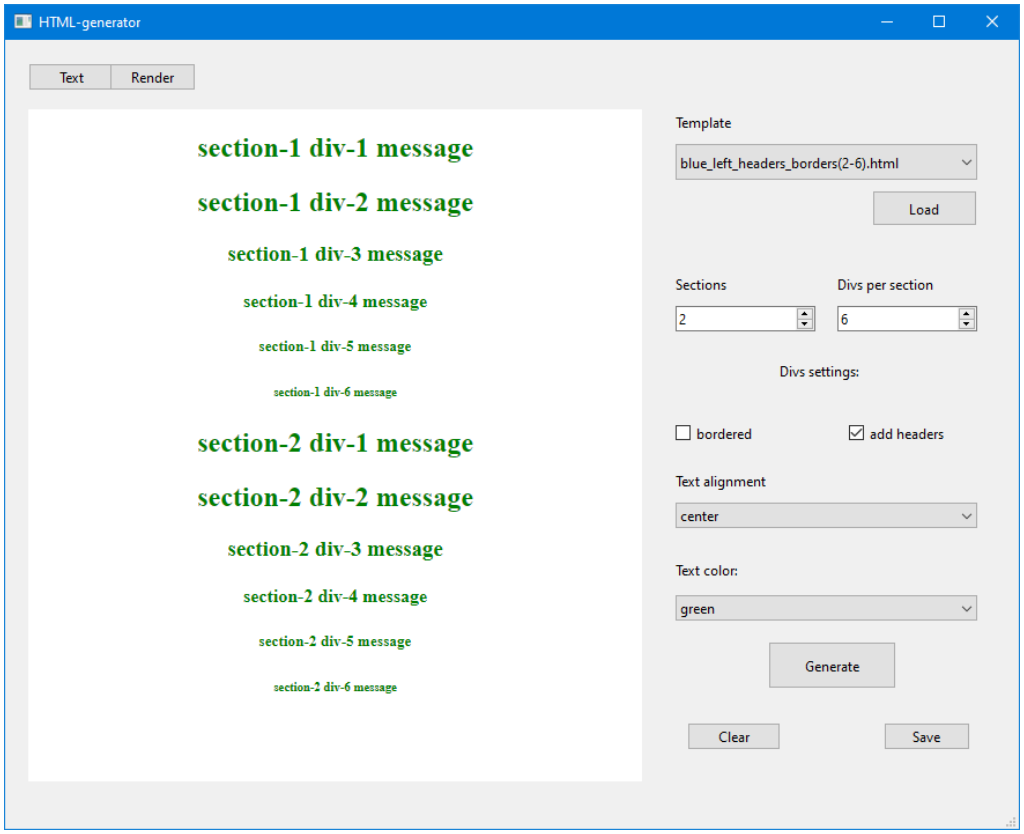
```
from abc import ABC, abstractmethod
from itertools import cycle, repeat
from typing import Iterator

HTML_SINGLES = {'!DOCTYPE', 'area', 'base', 'br', 'col', 'command', 'embed', 'hr', 'img', 'input',
                'keygen', 'link', 'meta', 'param', 'source', 'track', 'wbr'}

HTML_DOUBLES = {"a", "abbr", "address", "article", "aside", "audio", "b", "bdi", "bdo",
                "blockquote", "button", "canvas", "caption", "cite", "code", "data", "datalist",
                "dd", "del", "details", "dfn", "div", "dl", "dt", "em", "fieldset", "figcaption",
                "figure", "footer", "form", "h1", "h2", "h3", "h4", "h5", "h6", "hgroup", "i",
                "iframe", "ins", "kbd", "label", "legend", "li", "map", "mark", "menu",
                "menuitem", "meter", "nav", "noscript", "object", "ol", "optgroup", "option",
                "output", "p", "pre", "progress", "q", "rp", "rt", "ruby", "s", "samp", "script",
                "section", "select", "small", "span", "strong", "style", "sub", "summary", "sup",
                "table", "tbody", "td", "textarea", "tfoot", "th", "thead", "time", "title", "tr",
                "u", "ul", "var", "video"}

HTML_UNIQUES = {"html", "head", "body", "header", "main", "footer"}


class HtmlTag(ABC):
    def __init__(self, tag_name: str, *, tag_specs: str = ""):
        self._tag_name = tag_name
        self._tag_specs = tag_specs
        self._tag_iter = self._get_tag_gen()

    def __repr__(self):
        return f"{type(self).__name__}({self._tag_name})"

    @abstractmethod
    def _get_tag_gen(self) -> Iterator: ...

    @property
    def tag(self) -> str:
        return next(self._tag_iter)


class SingleTag(HtmlTag):
    def _get_tag_gen(self) -> Iterator:
        return repeat(f"<{self._tag_name}{' ' * bool(self._tag_specs)}{self._tag_specs}>")


class DoubleTag(HtmlTag):
    def _get_tag_gen(self) -> Iterator:
        return cycle((f"<{self._tag_name}{' ' * bool(self._tag_specs)}{self._tag_specs}>",
                      f"</{self._tag_name}>"))


# flyweight
class UniqueTag(DoubleTag):
    """ This class is used to implement "flyweight" pattern """
    __instances: dict[str, HtmlTag] = dict()

    def __new__(cls, tag_name: str, *args, **kwargs):
        return cls.__instances.setdefault(tag_name, super().__new__(cls))


class TagContent(SingleTag):

    def _get_tag_gen(self) -> Iterator:
        return repeat(self._tag_name)
```

Приложение 2

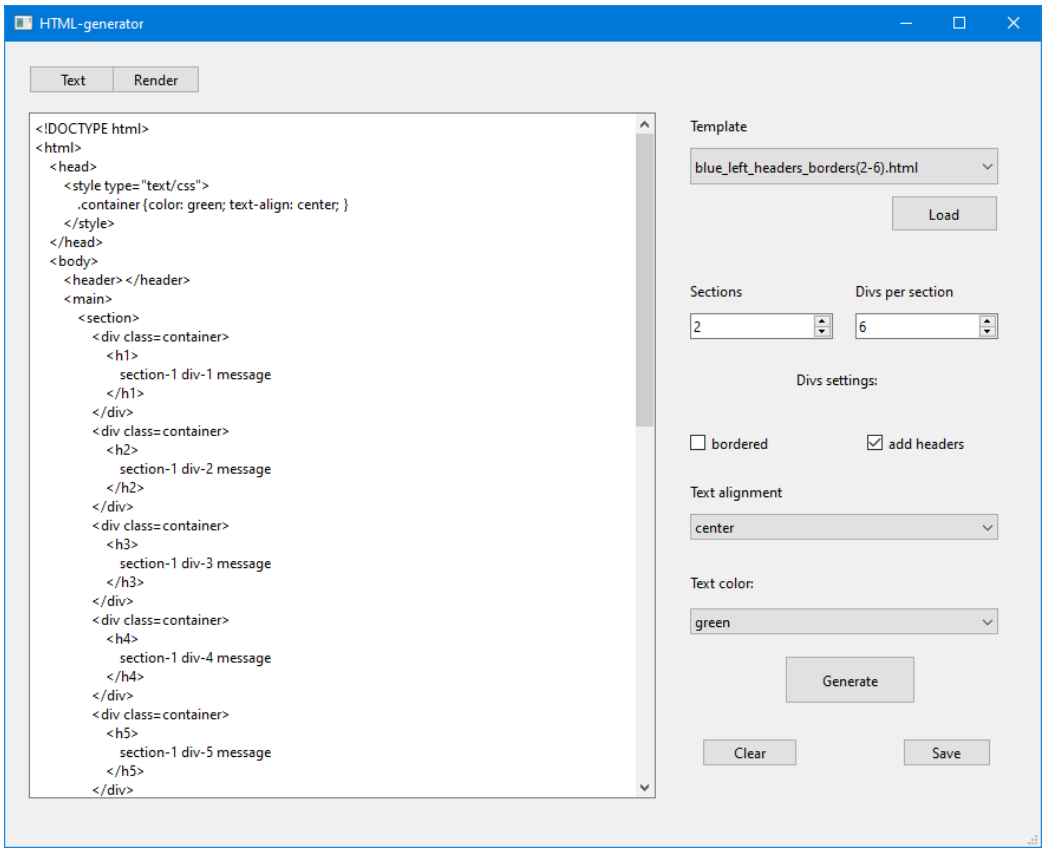
Результаты тестирования

Генерация шаблона без рамки, с разноуровневыми заголовками зеленого цвета, по центру

Окно “Render”:

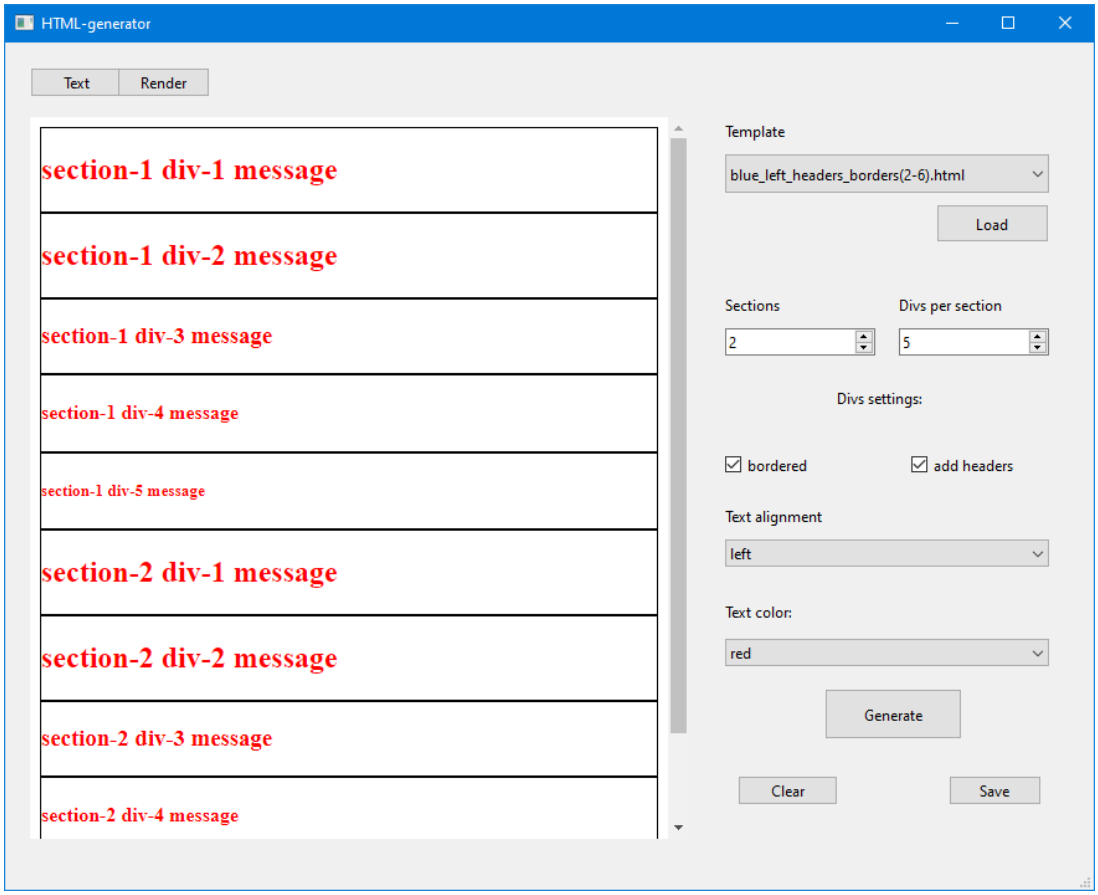


Окно “Text”:

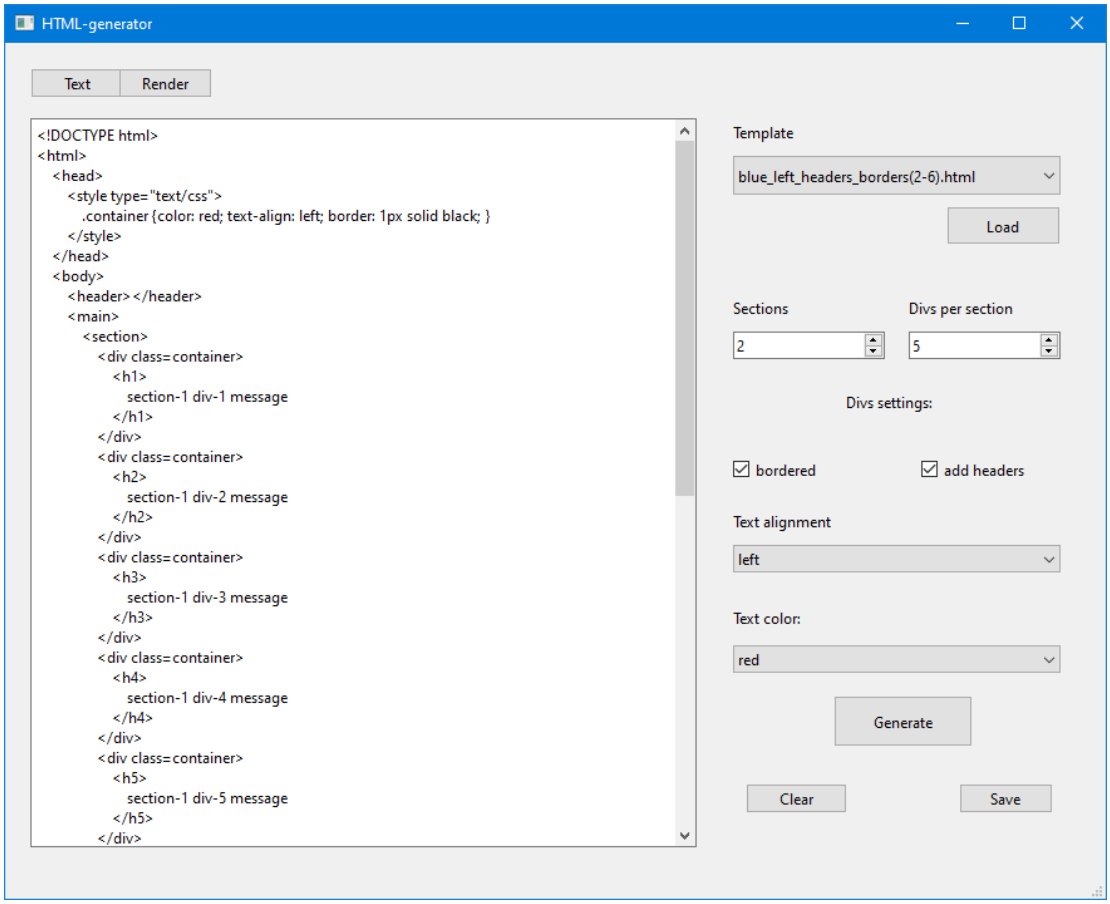


Генерация шаблона с рамками, с разноуровневыми заголовками красного цвета, слева

Окно “Render”:

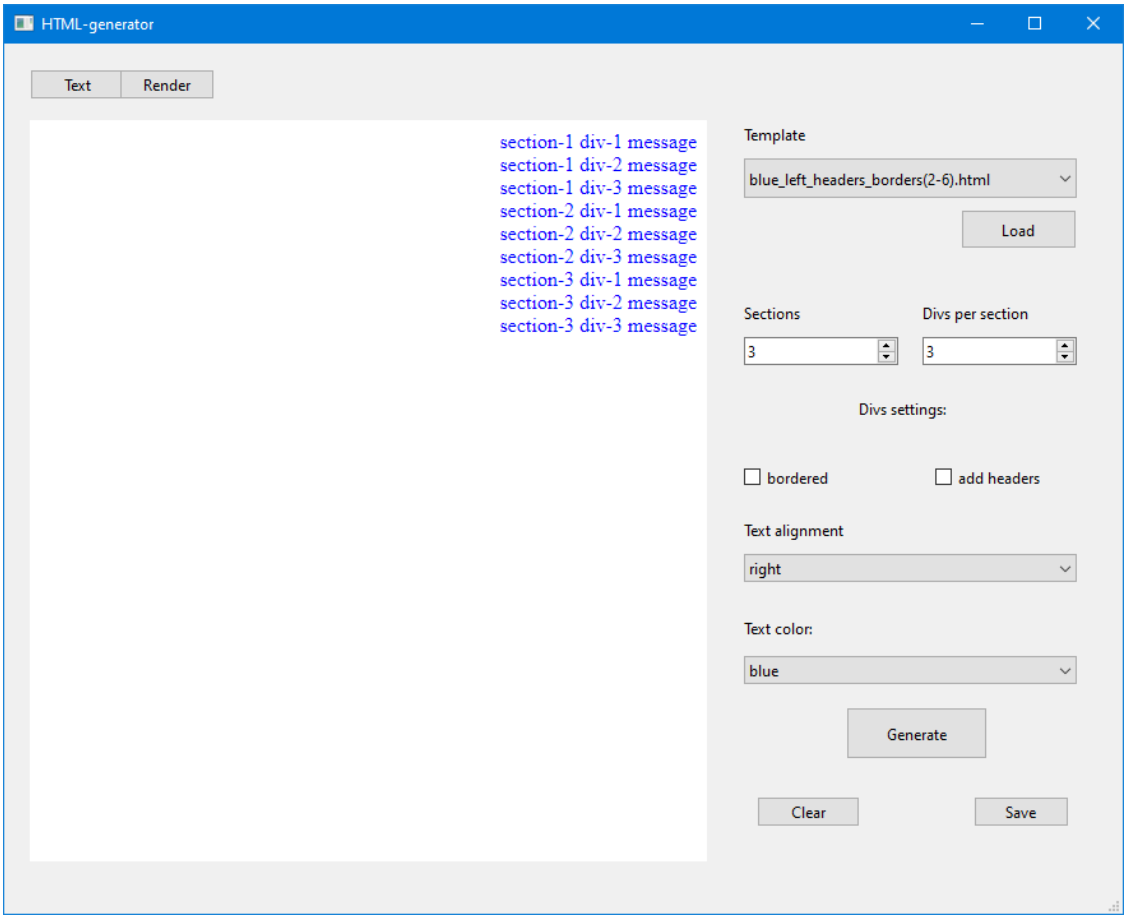


Окно “Text”:

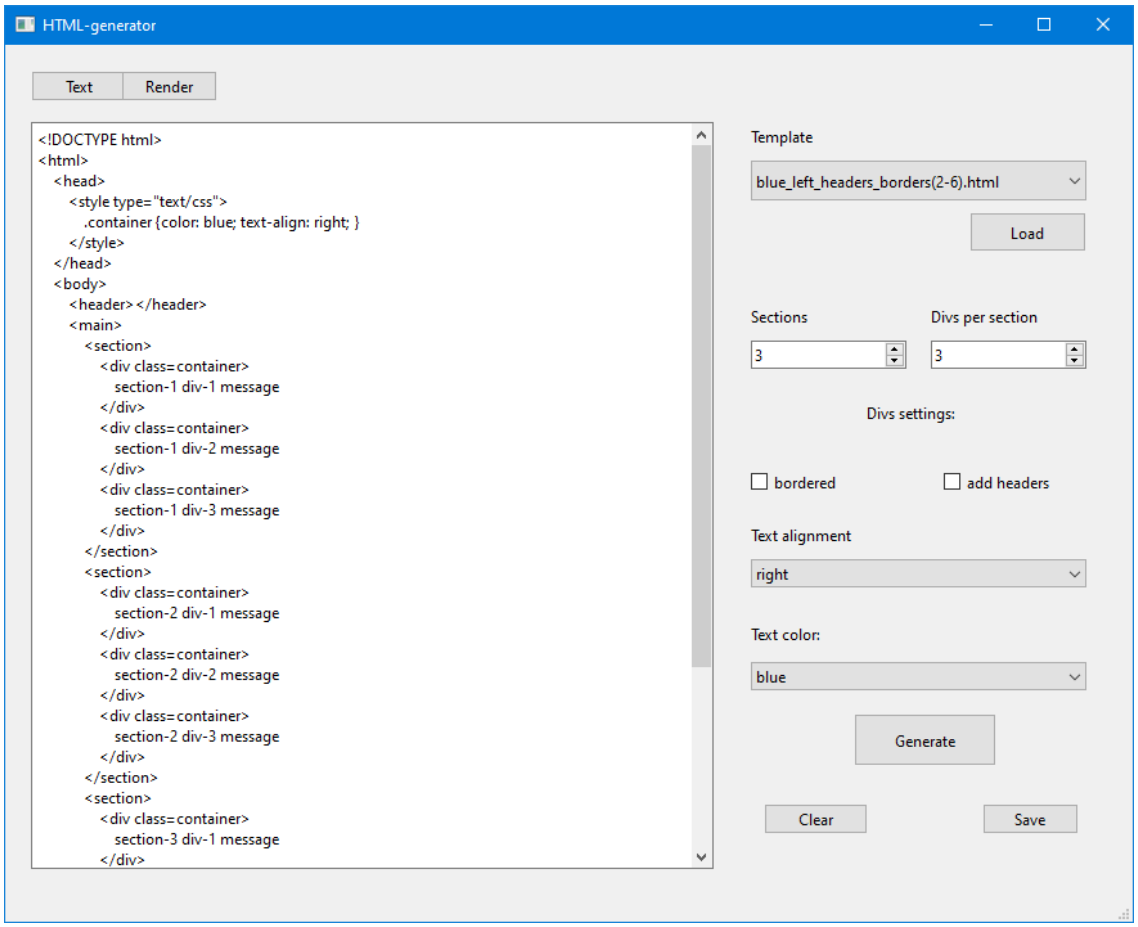


Генерация шаблона без рамок, без заголовков, с текстом синего цвета, справа

Окно “Render”:

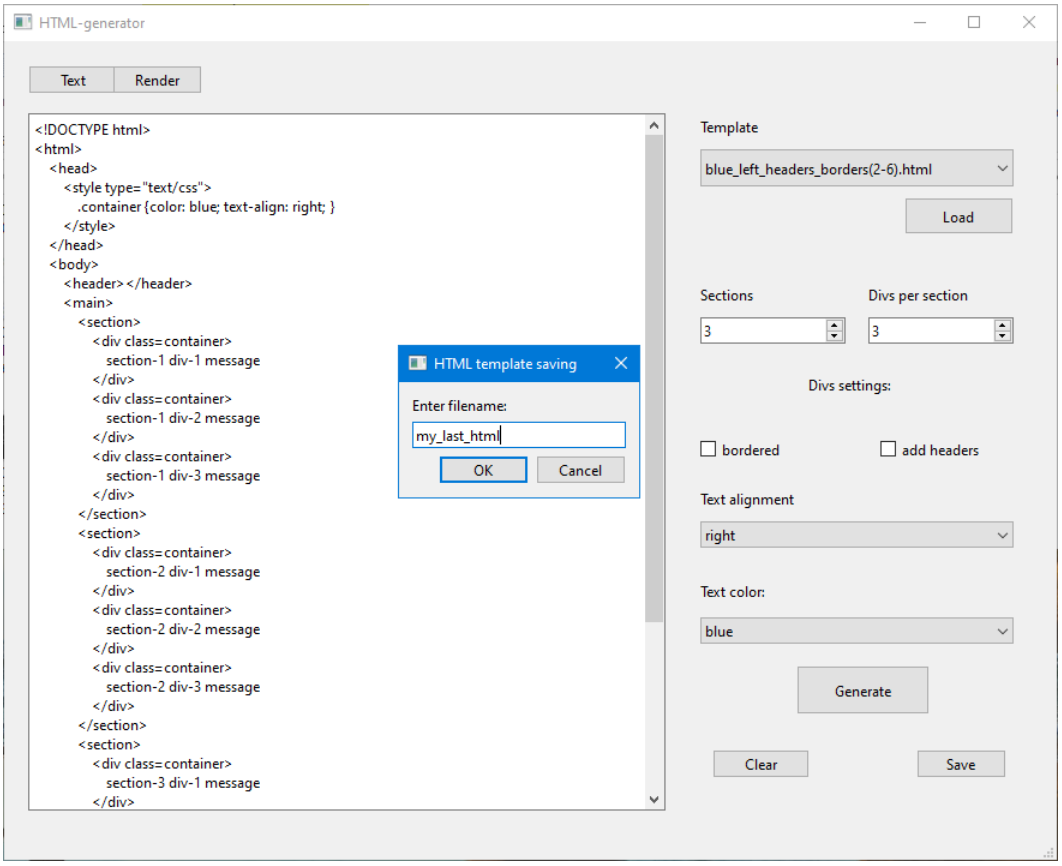


Окно “Text”:

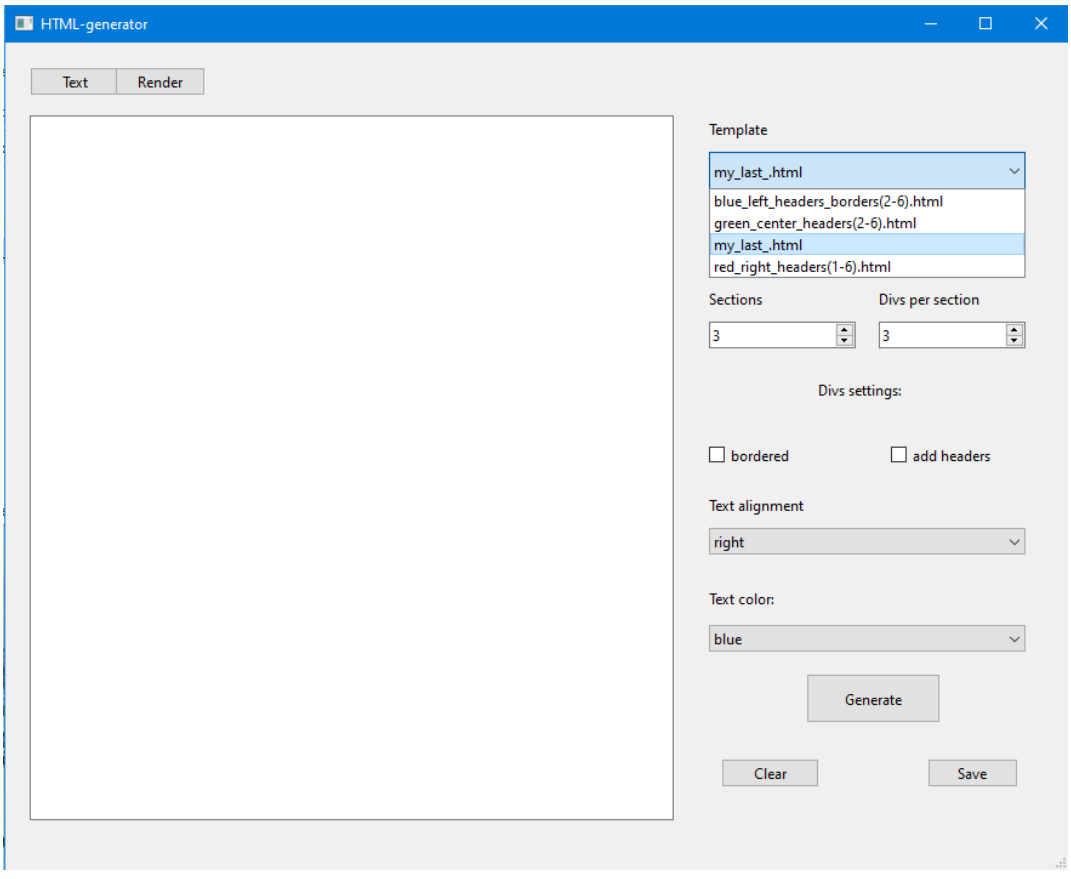


Проверка работоспособности элементов сохранения и загрузки шаблонов

Результат нажатия кнопки “Save”:



Результат нажатия кнопки “Load”:



Вывод:

Разработанное приложение работает без ошибок, полностью выполняет задуманную функциональность. С помощью программы можно генерировать шаблоны разного уровня вложенности, с возможностью добавления заголовков убывающего уровня, выбором цвета текста а также позиции на странице. Также есть возможность очистки окна ввода html-кода, загрузки и сохранения шаблонов.

Использование шаблонов проектирования в проекте позволило улучшить читаемость кода, повысить производительность, гибкость системы, а так же сделало ее расширяемой, позволило снизить межмодульную связность, что положительно сказалось на возможности тестирования системы, ее использования и доработки.