**Руководство разработчика**

**Требования к характеристикам компьютера и операционной системе**

* Операционная система: Windows 10 или выше.
* Процессор: не менее 2 ГГц.
* Оперативная память: не менее 8 ГБ.
* Свободное место на диске: не менее 1 ГБ.

**Указание версий языка и библиотек, использованных при разработке приложения**

Язык программирования: Python 3.12.3

Используемые библиотеки:

* flet==0.22.1
* flet-core==0.22.1
* flet-runtime==0.22.1
* flet-route==0.3.3
* matplotlib==3.9.0
* numpy==1.26.4
* pandas==2.2.2
* openpyxl==3.1.3

**Описание архитектуры приложения**

Приложение состоит из следующих частей:

1. **Интерфейс пользователя**: графический интерфейс для взаимодействия с пользователем.
2. **Модуль конфигурационный**: считывает конфигурационную информацию.
3. **Модуль класса данных и загрузчика**: хранение, считывание, сохранение, загрузка, очистка данных.
4. **Модуль генератора графиков**: отвечает за создание графиков.
5. **Модуль генератора отчетов**: генерирует отчеты на основе результатов анализа.

Эти части взаимодействуют между собой через общие функции и данные, передаваемые в виде объектов.

**Описание структуры каталогов**

* **/Work** – корневая папка приложения, в ней – main.py
* **/Data** - каталог для хранения входных данных
* **/Graphics** - каталог графиков
* **/Notes** - документация по проекту
* **/Output -** отчеты
* **/Scripts –** каталог для хранения вспомогательных скриптов.
* **/Library -**каталог для хранения библиотек и модулей, необходимых для работы приложения.

**Листинг main.py**

import flet as ft

from Scripts import config

from Library import data\_collector, graphics\_generator, report\_generator

from Scripts.views import Home, Loading, Graphics, DataTables, Reports, Info

def main(page: ft.Page):

    def route\_change(e):

        if page.route == "/":

            page.views.clear()

            page.views.append(Loading(page))

        elif page.route == "/home":

            page.views.clear()

            page.views.append(Home(page))

        elif page.route == "/graphics":

            page.views.clear()

            page.views.append(Graphics(page, graphGenerator))

        elif page.route == '/datatables':

            page.views.clear()

            page.views.append(DataTables(page, data, personal\_data, uploader))

        elif page.route == '/reports':

            page.views.clear()

            page.views.append(Reports(page, reportGenerator))

        elif page.route == "/info":

            page.views.clear()

            page.views.append(Info(page))

        page.update()

    page.on\_route\_change = route\_change

    page.go(page.route)

    cfg = config.Load()

    data = data\_collector.Data()

    data.read\_data(cfg.databases)

    data.set\_priority()

    personal\_data = data\_collector.Data()

    personal\_data.read\_data(cfg.personal\_databases)

    uploader=data\_collector.Uploader(cfg.personal\_dir)

    graphGenerator = graphics\_generator.GraphGenerator(data, personal\_data, cfg.graphics)

    reportGenerator = report\_generator.ReportGenerator(data, personal\_data, cfg.report\_dir)

    page.go('/home')

if \_\_name\_\_ == "\_\_main\_\_":

    ft.app(target=main)

**Листинг components.py**

import flet as ft

from flet\_route import Params, Basket

import os

class TitleBar(ft.ResponsiveRow):

    def \_\_init\_\_(self, page: ft.Page):

        super().\_\_init\_\_()

        self.page = page

        self.initialize\_controls()

    def initialize\_controls(self):

        if os.name == 'posix':

            return

        self.controls = [

            ft.WindowDragArea(

                content=ft.Container(

                    width=self.page.window\_width,

                    expand=True,

                    margin=0,

                    content=ft.Row(

                        [

                            self.get\_bar\_button(self.minimize\_window, ft.icons.MINIMIZE\_OUTLINED),

                            self.get\_bar\_button(self.maximize\_window, ft.icons.CHECK\_BOX\_OUTLINE\_BLANK),

                            self.get\_bar\_button(self.window\_close, ft.icons.CLOSE, ft.colors.RED),

                        ],

                        alignment=ft.MainAxisAlignment.END

                    ),

                ),

                height=20

            )

        ]

    def maximize\_window(self, e):

        self.page.window\_maximized = True

        self.page.update()

    def minimize\_window(self, e):

        self.page.window\_minimized = True

        self.page.update()

    def window\_close(self, e):

        self.page.window\_close()

    def get\_bar\_button(self, on\_click\_func, icon, hovered\_color=ft.colors.BLUE\_200):

        return ft.FilledButton(

            content=ft.Row(

                [

                    ft.Icon(name=icon, size=9)

                ]

            ),

            style=ft.ButtonStyle(

                color={

                    ft.MaterialState.HOVERED: ft.colors.WHITE,

                    ft.MaterialState.DEFAULT: ft.colors.BLUE\_200,

                },

                bgcolor={

                    ft.MaterialState.HOVERED: hovered\_color,

                    ft.MaterialState.DEFAULT: ft.colors.TRANSPARENT,

                },

                overlay\_color=ft.colors.TRANSPARENT,

                animation\_duration=0,

                shape=ft.BeveledRectangleBorder(),

            ),

            on\_click=on\_click\_func

        )

class Page(ft.View):

    def \_\_init\_\_(self, page, padding=0, scroll=False):

        super().\_\_init\_\_(scroll = scroll, padding=padding)

        self.page = page

        self.setup\_theme()

        self.setup\_sizes()

        self.page.update()

        self.page.window\_maximized = True

        self.page.update()

    def setup\_theme(self):

        theme = ft.Theme()

        theme.page\_transitions.windows = ft.PageTransitionTheme.NONE

        theme.page\_transitions.macos = ft.PageTransitionTheme.NONE

        theme.page\_transitions.linux = ft.PageTransitionTheme.NONE

        self.page.theme = theme

        self.page.theme\_mode = 'dark'

        if os.name == 'nt':

            self.page.window\_title\_bar\_hidden = True

            self.page.window\_title\_bar\_buttons\_hidden = True

    def setup\_sizes(self):

        self.page.window\_min\_height = 800

        self.page.window\_min\_width = 700

        self.page.spacing = 0

        self.page.padding = 0

**Листинг data\_collector.py**

from openpyxl import Workbook

import flet as ft

import numpy as np

import shutil, os

import pandas as pd

class Data:

    def \_\_init\_\_(self):

        self.countries = pd.DataFrame()

        self.dates = pd.DataFrame()

        self.daily\_production = pd.DataFrame()

        self.years = []

        self.countries\_list = []

        self.is\_empty = True

        self.is\_in\_priority = False

        self.paths = {}

    def set\_priority(self):

        self.is\_in\_priority = True

    def remove\_priority(self):

        self.is\_in\_priority = False

    def read\_data(self, databases\_paths):

        for key, path in databases\_paths.items():

            self.paths[key] = path

        if any(not os.path.exists(path) for path in databases\_paths.values()):

            self.destroy()

            return

        if 'countries' in databases\_paths:

            self.countries = pd.read\_excel(databases\_paths['countries'])

            self.set\_countries()

        if 'dates' in databases\_paths:

            self.dates = pd.read\_excel(databases\_paths['dates'])

            self.dates['Дата'] = pd.to\_datetime(self.dates['Дата'], format='%d.%m.%Y')

            self.dates['Дата'] = self.dates['Дата'].dt.strftime('%d.%m.%Y')

            self.set\_years()

        if 'daily\_production' in databases\_paths:

            self.daily\_production = pd.read\_excel(databases\_paths['daily\_production'])

        self.is\_empty = self.countries.empty or self.dates.empty or self.daily\_production.empty

    def set\_years(self):

        df = self.dates[['Дата']].copy()

        df['Дата'] = pd.to\_datetime(df['Дата'], format='mixed', dayfirst=True).dt.year

        self.years = df['Дата'].unique()

    def set\_countries(self):

        self.countries\_list = self.countries['Страна']

    def save\_data(self, name, path):

        if path == None:

            return

        if not path.endswith('.xlsx'):

            path += '.xlsx'

        Workbook().save(path)

        dict = {

            'dates' : self.dates,

            'countries' : self.countries,

            'daily\_production' : self.daily\_production,

        }

        dict[name].to\_excel(path, index=False)

    def generate\_datatable(self, df):

        if df.size == 0:

            return ft.Column()

        if df.shape[0] > 12:

            df = df.iloc[0:100]

        headers = [ft.DataColumn(ft.Text(header)) for header in df.columns]

        df\_array = df.to\_numpy()

        def create\_row(row):

            return ft.DataRow(cells=[ft.DataCell(ft.Text(cell)) for cell in row])

        rows = np.apply\_along\_axis(create\_row, axis=1, arr=df\_array)

        if df.shape[0] > 12:

            last\_row = ft.DataRow(

                cells=[ft.DataCell(ft.Text("...")) if i == 0 else ft.DataCell(ft.Text("")) for i in range(len(headers))]

            )

            rows = np.append(rows, [last\_row])

        datatable = ft.DataTable(

            columns=headers,

            rows=rows

        )

        return datatable

    def destroy(self):

        self.countries = pd.DataFrame()

        self.dates = pd.DataFrame()

        self.daily\_production = pd.DataFrame()

        dir = os.path.dirname(list(self.paths.values())[0])

        print(dir)

        shutil.rmtree(dir, ignore\_errors=True)

        self.is\_empty = True

class Uploader():

    def \_\_init\_\_(self, path):

        self.upload\_path = path

    def upload\_data(self, name, path):

        if not os.path.exists(self.upload\_path):

            os.makedirs(self.upload\_path)

        new\_path = os.path.join(self.upload\_path, name + "\_personal.xlsx")

        Workbook().save(new\_path)

        shutil.copy(path, new\_path)

        return new\_path

**Листинг graphics\_generator.py**

import os

import matplotlib

import matplotlib.pyplot as plt

from matplotlib.ticker import MultipleLocator

import numpy as np

import pandas as pd

matplotlib.use('agg')

class GraphGenerator():

    def \_\_init\_\_(self, data, personal\_data, paths):

        self.data = data

        self.personal\_data = personal\_data

        self.paths = paths

        self.current\_data = self.data

        self.check\_dirs()

    def check\_dirs(self):

        for folder in self.paths.values():

            if not os.path.exists(folder):

                os.makedirs(folder)

    def setup\_data(self):

        if self.data.is\_in\_priority:

            self.current\_data = self.data

        else:

            self.current\_data = self.personal\_data

    def plot\_boxwhiskers(self, atribute, start=2006, end=2022, countries=[]):

        path = self.paths['BoxWhiskers']

        margins = {

            "left"   : 0.06,

            "bottom" : 0.03,

            "right"  : 0.99,

            "top"    : 0.97

        }

        fig = plt.figure()

        fig.subplots\_adjust(\*\*margins)

        if atribute == 'Курс':

            df = self.current\_data.dates[['Дата', 'Курс']].copy()

            df['Дата'] = pd.to\_datetime(df['Дата'], format='mixed', dayfirst=True).dt.year

            df = df[(df['Дата'] <= end) & (df['Дата'] >= start)]

            column = df[atribute]

            fig.set\_size\_inches(10, 10)

            plt.ylabel('1$/1P')

            plt.boxplot(column, showmeans=True)

            plt.xticks([])

            plt.yticks(np.linspace(min(column), max(column), endpoint=True))

            path += "/Курс.png"

        elif atribute == 'Цена':

            df = self.current\_data.dates[['Дата', 'Цена']].copy()

            df['Дата'] = pd.to\_datetime(df['Дата'], format='mixed', dayfirst=True).dt.year

            df = df[(df['Дата'] <= end) & (df['Дата'] >= start)]

            column = df[atribute]

            fig.set\_size\_inches(10, 10)

            plt.ylabel('Рубли')

            plt.boxplot(column, showmeans=True)

            plt.xticks([])

            plt.yticks(np.linspace(min(column), max(column), endpoint=True))

            path += "/Цена.png"

        elif atribute == 'Добыча':

            df = pd.merge(self.current\_data.daily\_production, self.current\_data.dates, on='date\_id')[['Дата', 'Добыча', 'country\_id']]

            df = pd.merge(df, self.current\_data.countries, on='country\_id')[['Дата', 'Добыча', 'Страна']]

            df['Дата'] = pd.to\_datetime(df['Дата'], format='mixed', dayfirst=True).dt.year

            df = df[(df['Дата'] <= end) & (df['Дата'] >= start)]

            if len(countries) != 0:

                df = df[df['Страна'].isin(countries)]

            df.groupby(['Страна', 'Дата'])['Добыча'].mean()

            grouped\_data = df.groupby(['Страна'])['Добыча'].agg(list)

            fig.set\_size\_inches(15, 10)

            plt.boxplot(grouped\_data, showmeans=True)

            plt.xticks(ticks=range(1, len(grouped\_data) + 1), labels=grouped\_data.index)

            plt.title('Среднедневная добыча 2006-2022 гг.')

            plt.ylabel('Среднедневная добыча (1000 бар/д)')

            plt.grid(True)

            plt.suptitle('')

            path += "/Добыча.png"

        plt.grid(True)

        plt.title(f'{atribute} {start}-{end} гг.')

        self.save\_graph(fig, path)

        return fig

    def plot\_graph(self, atribute, start=2006, end=2022):

        path = self.paths['graphics']

        margins = {

            "left"   : 0.05,

            "bottom" : 0.05,

            "right"  : 0.99,

            "top"    : 0.97

        }

        fig = plt.figure(figsize=(20,15))

        fig.subplots\_adjust(\*\*margins)

        if (atribute == 'Курс'):

            df = self.current\_data.dates[['Дата', 'Курс']].copy()

            df['Дата'] = pd.to\_datetime(self.current\_data.dates['Дата'], format="%d.%m.%Y")

            df = df[(df['Дата'].dt.year <= end) & (df['Дата'].dt.year >= start)]

            plt.plot(df['Дата'], df['Курс'], label='Курс рубля')

            path += "/Курс.png"

        elif (atribute == 'Цена'):

            df = self.current\_data.dates[['Дата', 'Цена']].copy()

            df['Дата'] = pd.to\_datetime(self.current\_data.dates['Дата'], format="%d.%m.%Y")

            df = df[(df['Дата'].dt.year <= end) & (df['Дата'].dt.year >= start)]

            plt.plot(df['Дата'], df['Цена'], label='Цена на нефть')

            path += "/Цена.png"

        plt.title(f'{atribute} {start}-{end} гг.', fontsize=18)

        plt.xlabel('Дата', fontsize=18)

        plt.ylabel('Рубли', fontsize=18)

        plt.xticks(rotation=0)

        plt.tick\_params(axis='both', labelsize=14)

        plt.grid(True)

        self.save\_graph(fig, path)

        return fig

    def hist(self, start=2006, end=2022, countries=[]):

        df = pd.merge(self.current\_data.daily\_production, self.current\_data.dates, on='date\_id')[['Дата', 'Добыча', 'country\_id']]

        df = pd.merge(df, self.current\_data.countries, on='country\_id')[['Дата', 'Добыча', 'Страна']]

        df['Дата'] = pd.to\_datetime(df['Дата'], format='mixed', dayfirst=True).dt.year

        df = df[(df['Дата'] <= end) & (df['Дата'] >= start)]

        if len(countries) != 0:

                df = df[df['Страна'].isin(countries)]

        grouped\_data = df.groupby(['Страна', 'Дата'], as\_index=False)['Добыча'].mean()

        years = sorted(grouped\_data['Дата'].unique())

        grouped\_data = grouped\_data.groupby(['Страна'])['Добыча'].agg(list)

        num\_years = len(years)

        n\_cols = min(num\_years, 3)

        n\_rows = (num\_years + n\_cols - 1) // n\_cols

        fig, axes = plt.subplots(n\_rows, n\_cols, figsize=(6 \* n\_cols, 5 \* n\_rows))

        if num\_years == 1:

            axes = [axes]

        else:

            axes = axes.flatten()

        for idx, year in enumerate(years):

            ax = axes[idx]

            ax.hist([prod[idx] for prod in grouped\_data if len(prod) > idx], alpha=0.7, color='skyblue', edgecolor='black', bins=10)

            ax.set\_title(year)

            ax.yaxis.set\_major\_locator(MultipleLocator(base=1))

            ax.xaxis.set\_major\_locator(MultipleLocator(base=500))

            ax.grid(True, axis='y')

            ax.set\_xlabel('Среднедневная добыча')

            ax.set\_ylabel('Частота')

        for idx in range(len(years), len(axes)):

            fig.delaxes(axes[idx])

        plt.suptitle('Гистограммы среднедневной добычи по годам', fontsize=16)

        plt.tight\_layout(rect=[0, 0, 1, 0.98])

        path = self.paths['hist'] + "/Добыча.png"

        self.save\_graph(fig, path)

        return fig

    def diag(self, start=2006, end=2022, countries=[]):

        df = pd.merge(self.current\_data.daily\_production, self.current\_data.dates, on='date\_id')[['Дата', 'Добыча', 'country\_id']]

        df = pd.merge(df, self.current\_data.countries, on='country\_id')[['Дата', 'Добыча', 'Страна']]

        df['Дата'] = pd.to\_datetime(df['Дата'], format='mixed', dayfirst=True).dt.year

        df = df[(df['Дата'] <= end) & (df['Дата'] >= start)]

        if len(countries) != 0:

                df = df[df['Страна'].isin(countries)]

        grouped\_data = df.groupby(['Страна', 'Дата'], as\_index=False)['Добыча'].mean()

        countries = grouped\_data['Страна'].unique()

        years = grouped\_data['Дата'].unique()

        grouped\_data = grouped\_data.groupby(['Страна'])['Добыча'].agg(list)

        margins = {

            "left"   : 0.06,

            "bottom" : 0.06,

            "right"  : 0.89,

            "top"    : 0.97

        }

        fig = plt.figure(figsize=(19, 14))

        fig.subplots\_adjust(\*\*margins)

        plt.title("Кластеризованная столбчатая диаграмма по среднедневной добыче", fontsize=18)

        bar\_width = 0.05

        k = -6

        for c in countries:

            plt.bar([i + k\*bar\_width for i in years], grouped\_data[c], width=bar\_width, align='center', label=c)

            k += 1

        plt.ylabel('Среднедневная добыча (1000 бар/д)', fontsize=18)

        plt.yticks(np.arange(min(df['Добыча']), max(df['Добыча']), 100))

        plt.xticks([y for y in years], labels=years)

        plt.tick\_params(axis='both', labelsize=14)

        plt.grid(True, axis='y')

        plt.legend(loc='upper right', bbox\_to\_anchor=(1.13, 1))

        path = self.paths['diag'] + "/Добыча.png"

        self.save\_graph(fig, path)

        return fig

    def plot\_scatter(self, atribute, start=2006, end=2022, countries=[]):

        atr = self.current\_data.dates[['Дата', atribute]].copy()

        atr['Дата'] = pd.to\_datetime(atr['Дата'], format='mixed', dayfirst=True).dt.year

        atr = atr[(atr['Дата'] <= end) & (atr['Дата'] >= start)]

        atr = atr[atribute]

        df = pd.merge(self.current\_data.daily\_production, self.current\_data.dates, on='date\_id')[['Дата', 'Добыча', 'country\_id']]

        df = pd.merge(df, self.current\_data.countries, on='country\_id')[['Дата', 'Добыча', 'Страна']]

        df['Дата'] = pd.to\_datetime(df['Дата'], format='mixed', dayfirst=True).dt.year

        df = df[(df['Дата'] <= end) & (df['Дата'] >= start)]

        if len(countries) != 0:

                df = df[df['Страна'].isin(countries)]

        grouped\_data = df.groupby('Страна')['Добыча'].agg(list)

        margins = {

            "left"   : 0.06,

            "bottom" : 0.06,

            "right"  : 0.988,

            "top"    : 0.986

        }

        fig = plt.figure(figsize=(20, 20))

        fig.subplots\_adjust(\*\*margins)

        for c, prod in grouped\_data.items():

            plt.scatter(atr, prod, label=c, s=10)

        plt.title('Категоризированная диаграмма рассеивания:', fontsize=18)

        plt.xlabel(f'{atribute}', fontsize=18)

        plt.ylabel('Добыча дневная (1000 баррелей/день)', fontsize=18)

        plt.legend(title='Название страны')

        plt.tick\_params(axis='both', labelsize=14)

        plt.grid(True)

        path = self.paths['scatter'] + "/Рассеивание.png"

        self.save\_graph(fig, path)

        return fig

    def save\_graph(self, fig, path):

        print(path)

        fig.savefig(path)

    def clear(self):

        plt.close('all')

**Листинг report\_generator.py**

import os

import pandas as pd

from datetime import datetime

class ReportGenerator():

    def \_\_init\_\_(self, data, personal\_data, path):

        self.data = data

        self.personal\_data = personal\_data

        self.path = path

        self.current\_data = self.data

        self.check\_dir()

    def check\_dir(self):

        if not os.path.exists(self.path):

            os.makedirs(self.path)

    def setup\_data(self):

        if self.data.is\_in\_priority:

            self.current\_data = self.data

        else:

            self.current\_data = self.personal\_data

    def save\_reports(self, save\_dir):

        if not os.path.exists(save\_dir):

            os.makedirs(save\_dir)

        for filename in os.listdir(self.path):

            if filename.endswith(".txt"):

                report\_path = os.path.join(self.path, filename)

                with open(report\_path, 'r') as file:

                    content = file.read()

                    with open(os.path.join(save\_dir, filename), 'w') as save\_file:

                        save\_file.write(content)

    def run\_generator(self, country\_name):

        reports = [

            self.generate\_annual\_average\_report(),

            self.generate\_annual\_minmax\_report(),

            self.generate\_pivot\_table(),

            self.generate\_pivot\_table\_for\_country(country\_name),

        ]

        return reports

    def generate\_annual\_average\_report(self):

        df = self.current\_data.dates.copy()

        df['Дата'] = pd.to\_datetime(df['Дата'], format='%d.%m.%Y').dt.year

        pivot\_table = pd.pivot\_table(df, index=['Дата'], values=['Цена', 'Курс'], aggfunc='mean')

        report\_path = os.path.join(self.path, 'annual\_average\_report.txt')

        with open(report\_path, 'w') as f:

            header = f"{'Год':<5} | {'Цена за баррель (средняя)':<30} | {'Курс доллара (средний)':<20}\n"

            f.write(header)

            f.write('-' \* len(header) + '\n')

            for year, row in pivot\_table.iterrows():

                avg\_price = f"{row['Цена']:.2f}"

                avg\_exchange\_rate = f"{row['Курс']:.2f}"

                line = f"{year:<5} | {avg\_price:<30} | {avg\_exchange\_rate:<20}\n"

                f.write(line)

            f.write('-' \* len(header) + '\n')

        return report\_path

    def generate\_annual\_minmax\_report(self):

        df = self.current\_data.dates.copy()

        df['Дата'] = pd.to\_datetime(df['Дата'], format='%d.%m.%Y').dt.year

        pivot\_table = pd.pivot\_table(df, index=['Дата'], values=['Цена', 'Курс'], aggfunc={'Цена': ['min', 'max'], 'Курс': ['max', 'min']})

        pivot\_table.columns = ['Максимальная цена за баррель', 'Минимальная цена за баррель', 'Максимальный курс доллара', 'Минимальный курс доллара']

        report\_path = os.path.join(self.path, 'annual\_minmax\_report.txt')

        with open(report\_path, 'w') as f:

            header = f"{'Год':<5} | {'Мин. цена за баррель':<20} | {'Макс. цена за баррель':<20} | {'Макс. курс доллара':<20} | {'Мин. курс доллара':<20}\n"

            f.write(header)

            f.write('-' \* len(header) + '\n')

            for year, row in pivot\_table.iterrows():

                min\_price = f"{row['Минимальная цена за баррель']:.2f}"

                max\_price = f"{row['Максимальная цена за баррель']:.2f}"

                max\_rate = f"{row['Максимальный курс доллара']:.2f}"

                min\_rate = f"{row['Минимальный курс доллара']:.2f}"

                line = f"{year:<5} | {min\_price:<20} | {max\_price:<20} | {max\_rate:<20} | {min\_rate:<20}\n"

                f.write(line)

            f.write('-' \* len(header) + '\n')

        return report\_path

    def generate\_pivot\_table(self):

        df = pd.merge(self.current\_data.daily\_production, self.current\_data.countries, on='country\_id')[['Добыча', 'Страна']]

        pivot\_table = pd.pivot\_table(df, index=['Страна'], values=['Добыча'], aggfunc='sum')

        report\_path = os.path.join(self.path, 'total\_production.txt')

        with open(report\_path, 'w') as f:

            header = f"{'Страна':<20} | {'Суммарная добыча':<10}"

            f.write(header + '\n')

            f.write('-' \* len(header) + '\n')

            for index, row in pivot\_table.iterrows():

                line = f"{index:<20} | {row['Добыча']:<10}"

                f.write(line + '\n')

            f.write('-' \* len(header) + '\n')

        return report\_path

    def generate\_pivot\_table\_for\_country(self, country\_name):

        df = pd.merge(self.current\_data.daily\_production, self.current\_data.dates, on='date\_id')[['Дата', 'Добыча', 'country\_id']]

        df = pd.merge(df, self.current\_data.countries, on='country\_id')[['Дата', 'Добыча', 'Страна']]

        df['Дата'] = pd.to\_datetime(df['Дата'], format='%d.%m.%Y').dt.year

        pivot\_table = pd.pivot\_table(df[df['Страна'] == country\_name], index=['Дата'], values=['Добыча'], aggfunc='sum')

        report\_path = os.path.join(self.path, f'total\_production\_{country\_name}.txt')

        with open(report\_path, 'w') as f:

            header = f"{'Год':<5} | {'Суммарная добыча':<10}\n"

            f.write(header)

            f.write('-' \* len(header) + '\n')

            for year, row in pivot\_table.iterrows():

                line = f"{year:<5} | {row['Добыча']:<10}\n"

                f.write(line)

            f.write('-' \* len(header) + '\n')

        return report\_path

**Листинг config.py**

import configparser

class Config:

    def \_\_init\_\_(self, config):

        self.report\_dir = config['directories']['report\_dir']

        self.personal\_dir = config['directories']['personal\_dir']

        self.databases = config['databases']

        self.personal\_databases = config['personal\_databases']

        self.graphics = config['graphics']

def Load():

    config = configparser.ConfigParser()

    config.read('Scripts/config.ini')

    return Config(config)

**Листинг views.py**

import flet as ft

from flet.matplotlib\_chart import MatplotlibChart

from Library.components import Page, TitleBar

class Home(Page):

    def \_\_init\_\_(self, page: ft.Page):

        super().\_\_init\_\_(page)

        self.page = page

        self.page.window\_max\_width = self.page.window\_width

        self.controls = [

            TitleBar(page),

            self.initialize\_body(page)

        ]

    def initialize\_body(self, page):

        return ft.Container(

                content = ft.Column(

                    [

                        ft.Row(

                            [

                                ft.Text("Навигация", size=30),

                                ft.Icon(name=ft.icons.HOME, size=30),

                            ],

                            alignment=ft.MainAxisAlignment.CENTER,

                        ),

                        ft.Row(

                            [

                                ft.ElevatedButton("Графики", icon=ft.icons.STACKED\_LINE\_CHART, on\_click= lambda \_: page.go("/graphics"),width=250,height=75)

                            ],

                            alignment=ft.MainAxisAlignment.CENTER,

                        ),

                        ft.Row(

                            [

                                ft.ElevatedButton("Просмотр данных", icon=ft.icons.FILE\_UPLOAD, on\_click=lambda \_: page.go("/datatables"),width=250,height=75)

                            ],

                            alignment=ft.MainAxisAlignment.CENTER

                        ),

                        ft.Row(

                            [

                                ft.ElevatedButton("Просмотр отчетов", icon=ft.icons.MENU\_BOOK\_ROUNDED, on\_click=lambda \_: page.go("/reports"),width=250,height=75)

                            ],

                            alignment=ft.MainAxisAlignment.CENTER

                        ),

                        ft.Row(

                            [

                                ft.ElevatedButton("О проекте", icon=ft.icons.MENU\_BOOK\_ROUNDED, on\_click=lambda \_: page.go("/info"),width=250,height=75)

                            ],

                            alignment=ft.MainAxisAlignment.CENTER

                        ),

                    ],

                    alignment=ft.MainAxisAlignment.CENTER,

                    horizontal\_alignment=ft.CrossAxisAlignment.CENTER,

                    expand=True,

                ),

                alignment=ft.alignment.center,

                expand=True

            )

class Loading(Page):

    def \_\_init\_\_(self, page: ft.Page):

        super().\_\_init\_\_(page)

        self.page = page

        self.controls = [

            ft.Container(

                content=ft.Row(

                    [

                        ft.Column(

                            [

                                ft.ProgressRing(),

                                ft.Text("Загрузка данных"),

                            ],

                            alignment=ft.MainAxisAlignment.CENTER,

                            horizontal\_alignment=ft.CrossAxisAlignment.CENTER,

                            expand=True,

                        )

                    ],

                    alignment=ft.MainAxisAlignment.CENTER,

                    vertical\_alignment=ft.CrossAxisAlignment.CENTER,

                    expand=True,

                ),

                alignment=ft.alignment.center,

                expand=True,

            )

        ]

class Graphics(Page):

    def \_\_init\_\_(self, page: ft.Page, generator):

        super().\_\_init\_\_(page, scroll=True)

        self.page = page

        self.graphGenerator = generator

        self.graphGenerator.setup\_data()

        self.dashboard\_slider = ft.Container()

        self.dashboard\_boxes = ft.Container()

        self.dashboard\_button = ft.Container()

        self.graph = ft.Container()

        self.save\_button = ft.Container()

        self.controls = [

            TitleBar(page),

            self.get\_navbar(),

            ft.Container()

        ]

    def update\_body(self):

        self.controls[-1] = ft.Container(

            content=ft.Row(

                [

                    ft.Column(

                        [

                            ft.Container(

                                content = ft.Column(

                                    [

                                        self.dashboard\_slider,

                                        self.dashboard\_boxes,

                                        self.dashboard\_button

                                    ],

                                    alignment = ft.MainAxisAlignment.CENTER,

                                    horizontal\_alignment = ft.CrossAxisAlignment.CENTER

                                ),

                                padding = 10,

                            )

                        ],

                        alignment=ft.MainAxisAlignment.START,

                        horizontal\_alignment=ft.CrossAxisAlignment.CENTER,

                        width = self.page.window\_max\_width / 4,

                    ),

                    ft.Column(

                        [

                            self.graph

                        ],

                        alignment=ft.MainAxisAlignment.CENTER,

                        horizontal\_alignment=ft.CrossAxisAlignment.CENTER,

                        width = self.page.window\_max\_width / 2,

                    ),

                    ft.Column(

                        [

                            self.save\_button

                        ],

                        alignment=ft.MainAxisAlignment.CENTER,

                        horizontal\_alignment=ft.CrossAxisAlignment.CENTER,

                        width = self.page.window\_max\_width / 4,

                    ),

                ]

            ),

            alignment=ft.alignment.center,

            width = self.page.window\_max\_width,

        )

        self.page.update()

    def get\_navbar(self):

        return ft.Row(

            [

                ft.MenuBar(

                    style = ft.MenuStyle(

                        bgcolor=ft.colors.TRANSPARENT,

                        shadow\_color=ft.colors.TRANSPARENT,

                    ),

                    controls=[

                        ft.MenuItemButton(

                            content=ft.Text("Назад"),

                            on\_click=lambda \_: self.page.go('/home'),

                        ),

                        ft.SubmenuButton(

                            content=ft.Text("Графики изменения"),

                            controls=[

                                ft.MenuItemButton(

                                    content=ft.Text("Курс рубля"),

                                    on\_click = lambda \_: self.open\_configure\_window(self.graphGenerator.plot\_graph, 'Курс', countries\_disabled=True)

                                ),

                                ft.MenuItemButton(

                                    content=ft.Text("Цена на нефть"),

                                    on\_click = lambda \_: self.open\_configure\_window(self.graphGenerator.plot\_graph, 'Цена', countries\_disabled=True)

                                ),

                            ]

                        ),

                        ft.SubmenuButton(

                            content=ft.Text("Box & Whiskers"),

                            controls=[

                                ft.MenuItemButton(

                                    content=ft.Text("Курс рубля"),

                                    on\_click = lambda \_: self.open\_configure\_window(self.graphGenerator.plot\_boxwhiskers, 'Курс', countries\_disabled=True)

                                ),

                                ft.MenuItemButton(

                                    content=ft.Text("Цена на нефть"),

                                    on\_click = lambda \_: self.open\_configure\_window(self.graphGenerator.plot\_boxwhiskers, 'Цена', countries\_disabled=True)

                                ),

                                ft.MenuItemButton(

                                    content=ft.Text("Добыча нефти"),

                                    on\_click = lambda \_: self.open\_configure\_window(self.graphGenerator.plot\_boxwhiskers, 'Добыча')

                                ),

                            ]

                        ),

                        ft.MenuItemButton(

                            content=ft.Text("Гистограмма"),

                            on\_click=lambda \_: self.open\_configure\_window(self.graphGenerator.hist)

                        ),

                        ft.MenuItemButton(

                            content=ft.Text("Диаграмма"),

                            on\_click=lambda \_: self.open\_configure\_window(self.graphGenerator.diag)

                        ),

                        ft.SubmenuButton(

                            content=ft.Text("Рассеивание"),

                            controls=[

                                ft.MenuItemButton(

                                    content=ft.Text("По цене"),

                                    on\_click=lambda \_: self.open\_configure\_window(self.graphGenerator.plot\_scatter, 'Цена')

                                ),

                                ft.MenuItemButton(

                                    content=ft.Text("По курсу"),

                                    on\_click=lambda \_: self.open\_configure\_window(self.graphGenerator.plot\_scatter, 'Курс')

                                ),

                            ]

                        )

                    ],

                )

            ],

            alignment=ft.MainAxisAlignment.CENTER,

        )

    def open\_configure\_window(self, plot\_func, atr='', countries\_disabled=False):

        years = self.graphGenerator.current\_data.years

        min\_year, max\_year = min(years), max(years)

        start, end = min\_year, max\_year

        def change\_period(e):

            nonlocal start, end

            start = e.control.start\_value

            start = round(float(start))

            end = e.control.end\_value

            end = round(float(end))

        def build\_new(e):

            self.graphGenerator.clear()

            true\_countries = [el.label for el in countries\_checkboxes if el.value]

            if not countries\_disabled:

                if atr == '':

                    fig = plot\_func(start, end, true\_countries)

                else:

                    fig = plot\_func(atr, start, end, true\_countries)

            else:

                fig = plot\_func(atr, start, end)

            self.put\_graph(fig)

        countries\_checkboxes = []

        countries = self.graphGenerator.current\_data.countries\_list

        for c in countries:

            countries\_checkboxes.append(ft.Checkbox(label=c, disabled=countries\_disabled, value=True))

        self.dashboard\_slider =  ft.Column(

            [

                ft.Text("Период",  weight=ft.FontWeight.BOLD,),

                ft.RangeSlider(

                    min=min\_year,

                    max=max\_year,

                    start\_value=2006,

                    divisions=17,

                    end\_value=2022,

                    inactive\_color=ft.colors.BLUE\_100,

                    active\_color=ft.colors.BLUE\_500,

                    overlay\_color=ft.colors.BLUE\_200,

                    label="{value}",

                    on\_change=change\_period,

                ),

            ]

        )

        self.dashboard\_boxes =  ft.Column(

            [ft.Text("Страны",  weight=ft.FontWeight.BOLD)] + countries\_checkboxes

        )

        self.dashboard\_button = ft.ElevatedButton(

            "Построить",

            on\_click=build\_new,

            disabled=True

        )

        self.save\_button = ft.ElevatedButton("Сохранить график", disabled=True)

        build\_new(None)

    def put\_graph(self, fig):

        file\_picker = ft.FilePicker(on\_result=lambda e: self.graphGenerator.save\_graph(fig, e.path))

        self.page.overlay.append(file\_picker)

        self.graph = ft.ProgressRing()

        self.update\_body()

        self.graph = MatplotlibChart(fig)

        self.save\_button.on\_click = lambda \_: file\_picker.save\_file(allowed\_extensions=['png'])

        self.save\_button.disabled=False

        self.dashboard\_button.disabled=False

        self.update\_body()

class DataTables(Page):

    def \_\_init\_\_(self, page: ft.Page, data, personal\_data, uploader):

        super().\_\_init\_\_(page)

        self.page = page

        self.data = data

        self.uploader = uploader

        self.personal\_data = personal\_data

        self.file\_saver = ft.FilePicker(on\_result=self.on\_save\_result)

        self.file\_uploader = ft.FilePicker(on\_result=self.on\_upload\_result)

        self.page.overlay.append(self.file\_saver)

        self.page.overlay.append(self.file\_uploader)

        self.file\_name = None

        self.controls = [

            TitleBar(page),

            self.get\_navbar(),

            ft.Row(

                [

                    self.get\_table('ДАТА', 420, 500, self.data.generate\_datatable(self.data.dates), self.get\_save\_button('dates')),

                    self.get\_table('СТРАНЫ', 250, 500, self.data.generate\_datatable(self.data.countries), self.get\_save\_button('countries')),

                    self.get\_table('ДОБЫЧА', 350, 500, self.data.generate\_datatable(self.data.daily\_production), self.get\_save\_button('daily\_production'))

                ],

                alignment=ft.MainAxisAlignment.CENTER,

                visible = True,

            ),

            ft.Row(

                [

                ],

                alignment=ft.MainAxisAlignment.CENTER,

                visible = False,

            ),

            ft.Row(

                [

                ],

                alignment=ft.MainAxisAlignment.CENTER,

                visible = False,

            )

        ]

        self.update\_personal\_table()

    def switch\_data(self, e):

        if self.data.is\_in\_priority:

            self.data.remove\_priority()

            self.personal\_data.set\_priority()

        else:

            self.personal\_data.remove\_priority()

            self.data.set\_priority()

    def update\_personal\_table(self):

        self.controls[-2].controls =  [

            self.get\_table('ДАТА', 420, 500, self.personal\_data.generate\_datatable(self.personal\_data.dates), self.get\_upload\_button('dates')),

            self.get\_table('СТРАНЫ', 250, 500, self.personal\_data.generate\_datatable(self.personal\_data.countries), self.get\_upload\_button('countries')),

            self.get\_table('ДОБЫЧА', 350, 500, self.personal\_data.generate\_datatable(self.personal\_data.daily\_production), self.get\_upload\_button('daily\_production'))

        ]

        self.controls[-1].controls = [

            ft.ElevatedButton('Очистить данные', on\_click=self.clear\_data, disabled=self.personal\_data.is\_empty),

            ft.Switch(label="Использовать пользовательские данные", on\_change=self.switch\_data, disabled = self.personal\_data.is\_empty, value = self.personal\_data.is\_in\_priority),

        ]

        self.page.update()

    def get\_save\_button(self, name):

        return ft.ElevatedButton('Экспортировать данные', data = name, on\_click=self.save\_data)

    def get\_upload\_button(self, name):

        return ft.ElevatedButton('Загрузить данные', data = name, on\_click=self.upload\_data)

    def get\_table(self, name, width, height, table, button):

        return ft.Column(

            [

                ft.Text(name, weight = ft.FontWeight.BOLD),

                ft.Container(

                    width=width,

                    height=height,

                    content=ft.Column(

                        controls=[

                            ft.Container(

                                width=width,

                                content=ft.Column(

                                    controls=[

                                        table

                                    ]

                                ),

                            )

                        ],

                        scroll="auto",

                    ),

                    border=ft.border.all(1, "blue"),

                ),

                button

            ]

        )

    def get\_navbar(self):

        return ft.Row(

            [

                ft.MenuBar(

                    style = ft.MenuStyle(

                        bgcolor=ft.colors.TRANSPARENT,

                        shadow\_color=ft.colors.TRANSPARENT,

                    ),

                    controls=[

                        ft.MenuItemButton(

                            content=ft.Text("Назад"),

                            on\_click=lambda \_: self.page.go('/home'),

                        ),

                        ft.MenuItemButton(

                            content=ft.Text("Мировые данные"),

                            on\_click=lambda \_: self.open\_main\_page(),

                        ),

                        ft.MenuItemButton(

                            content=ft.Text("Собственные данные"),

                            on\_click=lambda \_: self.open\_personal\_page(),

                        ),

                    ],

                )

            ],

            alignment=ft.MainAxisAlignment.CENTER,

        )

    def on\_save\_result(self, e):

        if self.file\_name:

            self.data.save\_data(self.file\_name, e.path)

            self.file\_name = None

    def on\_upload\_result(self, e):

        if self.file\_name:

            personal\_path = self.uploader.upload\_data(self.file\_name, e.files[0].path)

            self.personal\_data.read\_data({self.file\_name: personal\_path})

            self.file\_name = None

            self.update\_personal\_table()

    def save\_data(self, e):

        self.file\_name = e.control.data

        self.file\_saver.save\_file(allowed\_extensions=['xlsx'])

    def upload\_data(self, e):

        self.file\_name = e.control.data

        self.file\_uploader.pick\_files(allow\_multiple=False)

        self.update\_personal\_table()

    def clear\_data(self, e):

        self.personal\_data.destroy()

        self.update\_personal\_table()

    def open\_main\_page(self):

        self.controls[-1].visible = False

        self.controls[-2].visible = False

        self.controls[-3].visible = True

        self.page.update()

    def open\_personal\_page(self):

        self.controls[-3].visible = False

        self.controls[-2].visible = True

        self.controls[-1].visible = True

        self.page.update()

class Reports(Page):

    def \_\_init\_\_(self, page: ft.Page, generator):

        super().\_\_init\_\_(page)

        self.page = page

        self.reportGenerator = generator

        self.reportGenerator.setup\_data()

        self.file\_picker = ft.FilePicker(on\_result=self.on\_file\_picker\_result)

        self.page.overlay.append(self.file\_picker)

        self.country\_combobox = self.get\_combobox()

        self.save\_button = ft.ElevatedButton(

            "Скачать отчеты",

            on\_click=lambda \_: self.file\_picker.get\_directory\_path(),

            width=250,

            height=75,

            disabled=True

        )

        self.controls = [

            TitleBar(page),

            ft.ElevatedButton("Домой", on\_click=lambda \_: page.go("/home"), icon=ft.icons.ARROW\_BACK),

            ft.Container(

                content=ft.Column(

                    [

                        ft.Row(

                            [

                                ft.Text("Выберите страну:", size=20),

                                self.country\_combobox

                            ],

                            alignment=ft.MainAxisAlignment.CENTER,

                        ),

                        ft.Row(

                            [

                                ft.ElevatedButton(

                                    "Создать отчеты",

                                    on\_click=self.on\_create\_reports\_click,

                                    width=250,

                                    height=75

                                ),

                                self.save\_button

                            ],

                            alignment=ft.MainAxisAlignment.CENTER,

                        )

                    ],

                    alignment=ft.MainAxisAlignment.CENTER,

                    horizontal\_alignment=ft.CrossAxisAlignment.CENTER,

                    expand=True,

                ),

                alignment=ft.alignment.center,

                expand=True

            )

        ]

    def get\_combobox(self):

        countries = self.reportGenerator.current\_data.countries\_list

        return ft.Dropdown(

            options=[ft.dropdown.Option(text=country) for country in countries]

        )

    def on\_file\_picker\_result(self, e):

        save\_dir = e.path

        if e.path != None:

            self.reportGenerator.save\_reports(save\_dir)

    def on\_create\_reports\_click(self, e):

        self.save\_button.disabled = False

        self.page.update()

        country\_name = self.country\_combobox.value

        if not country\_name:

            self.show\_error\_dialog()

        else:

            reports = self.reportGenerator.run\_generator(country\_name)

            self.open\_reports\_dialog(e.page, reports)

    def show\_error\_dialog(self):

        def close\_error\_dialog(e):

            error\_dialog.open = False

            self.page.update()

        error\_dialog = ft.AlertDialog(

            title=ft.Text("Ошибка"),

            content=ft.Text("Пожалуйста, выберите страну."),

            actions=[

                ft.TextButton("ОК", on\_click=close\_error\_dialog)

            ]

        )

        self.page.dialog = error\_dialog

        error\_dialog.open = True

        self.page.update()

    def open\_reports\_dialog(self, page: ft.Page, reports):

        report\_contents = []

        for report in reports:

            with open(report, "r") as file:

                content = file.read()

                report\_contents.append(ft.Text(content, size=14, font\_family="Courier New"))

        def close\_dialog(e):

            dialog.open = False

            page.update()

        dialog = ft.AlertDialog(

            title=ft.Text("Отчеты"),

            content=ft.Container(

                content=ft.Column(

                    report\_contents,

                    scroll=True,

                ),

                width=1000,

                height=400,

                padding=20

            ),

            actions=[

                ft.TextButton("Закрыть", on\_click=close\_dialog)

            ],

        )

        page.dialog = dialog

        dialog.open = True

        page.update()

class Info(Page):

    def \_\_init\_\_(self, page: ft.Page):

        super().\_\_init\_\_(page)

        self.page = page

        self.controls = [

            TitleBar(page),

            ft.ElevatedButton("Домой", on\_click= lambda \_: page.go("/home"),icon=ft.icons.ARROW\_BACK),

            ft.Container(

                content = ft.Column(

                    [

                        ft.Row(

                            [

                                ft.Text(

                                    spans = [

                                        ft.TextSpan(

                                            "В данном приложении вы можете создать собственые графики,\nиспользуя удобный интерфейс, "

                                            "а также просмотреть графики, уже созданные нами.\nАвторы приложения:\n Наумов Виталий\n Куров Егор\n"

                                            " Мирумян Артём\n Рахматуллин Айгиз",

                                        )

                                    ],

                                    color="white", size=20, text\_align="CENTER"

                                ),

                            ],

                            alignment=ft.MainAxisAlignment.CENTER,

                        ),

                    ],

                    alignment=ft.MainAxisAlignment.CENTER,

                    horizontal\_alignment=ft.CrossAxisAlignment.CENTER,

                    expand=True,

                ),

                alignment=ft.alignment.center,

                expand=True

            )

        ]